Informed search

Lirong Xia

Rensselaer

Spring, 2017
Last class

Search problems

• state space graph: modeling the problem
• search tree: scratch paper for solving the problem

Uninformed search

• BFS
• DFS
Today’s schedule

➢ More on uninformed search
  • iterative deepening: BFS + DFS
  • uniform cost search (UCS)
  • searching backwards from the goal

➢ Informed search
  • best first (greedy)
  • A*
Combining good properties of BFS and DFS

- Iterative deepening DFS:
  - Call limited depth DFS with depth 0;
  - If unsuccessful, call with depth 1;
  - If unsuccessful, call with depth 2;
  - Etc.

- Complete, finds shallowest solution
- Flexible time-space tradeoff
- May seem wasteful timewise because replicating effort
Costs on Actions
Uniform-Cost Search (UCS)

- BFS: finds shallowest solution
- Uniform-cost search (UCS): work on the lowest-cost node first
  - so that all states in the fringe have more or less the same cost, therefore “uniform cost”
- If it finds a solution, it will be an optimal one
- Will often pursue lots of short steps first
- Project 1 Q3
Example
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>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pq.push(key, value)</code></td>
<td>Inserts (key, value) into the queue.</td>
</tr>
<tr>
<td><code>pq.pop()</code></td>
<td>returns the key with the lowest value, and removes it from the queue</td>
</tr>
</tbody>
</table>

• Insertions aren’t constant time, usually
• We’ll need priority queues for cost-sensitive search methods
The good: UCS is

- complete: if a solution exists, one will be found
- optimal: always find the solution with the lowest cost
- really?
  - yes, for cases where all costs are positive

The bad

- explores every direction
- no information about the goal location
Searching backwards from the goal

- Switch the role of START and GOAL
- Reverse the edges in the problem graph
- Need to be able to compute *predecessors* instead of successors
Informed search

- So far, have assumed that no non-goal state looks better than another

- Unrealistic
  - Even without knowing the road structure, some locations seem closer to the goal than others

- Makes sense to expand seemingly closer nodes first
Search Heuristics

• Any estimate of how close a state is to a goal
• Designed for a particular search problem
• Examples: Manhattan distance, Euclidean distance
Best First (Greedy)

- Strategy: expand a node that you think is closest to a goal state
  - **Heuristic function**: estimate of distance to nearest goal for each state
  - \( h(n) \)
- A common case:
  - Best-first takes you straight to the (wrong) goal
- Worst-case: like a badly-guided DFS
h(b) = h(c) = h(d) = h(e) = h(f) = h(g) = 1
h(a) = 2
A*
A*: Combining UCS and Greedy

• **Uniform-cost** orders by path cost $g(n)$

• **Greedy** orders by goal proximity, or forward cost $h(n)$

• **A* search** orders by the sum: $f(n) = g(n) + h(n)$
When should A* terminate?

• Should we stop when we add a goal to the fringe?

• No: only stop when a goal pops from the fringe.
Is A* Optimal?

- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual cost!
Admissible Heuristics

- A heuristic $h$ is admissible (optimistic) if:
  \[ h(n) \leq h^*(n) \]
  where $h^*(n)$ is the true cost to a nearest goal.

- Examples:

- Coming up with admissible heuristics is most of what’s involved in using A* in practice.
Creating Admissible Heuristics

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics.
- Often, admissible heuristics are solutions to relaxed problems, where new actions are available.
Example: 8 Puzzle

- What are the states?
- How many states?
- What are the actions?
- What states can I reach from the start state?
- What should the costs be?
8 Puzzle I

- Heuristic: Number of tiles misplaced

- Why is it admissible?

- \( h(\text{start}) = 8 \)

- This is a relaxed-problem heuristic

---

### Average nodes expanded when optimal path has length:

<table>
<thead>
<tr>
<th>Path Length</th>
<th>UCS</th>
<th>Tiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 steps</td>
<td>112</td>
<td>13</td>
</tr>
<tr>
<td>8 steps</td>
<td>6,300</td>
<td>39</td>
</tr>
<tr>
<td>12 steps</td>
<td>3,600,000</td>
<td>227</td>
</tr>
</tbody>
</table>
8 Puzzle II

• What if we had an easier 8-puzzle where any tile could slide any direction at any time ignoring other tiles?

• Total Manhattan distance

• Why admissible?

• \( h(\text{start}) = 3 + 1 + 2 + ... = 18 \)
8 Puzzle III

• How about using the actual cost as a heuristic?
  • Would it be admissible?
  • Would we save on nodes expanded?
  • What’s wrong with it?
• With A*: a trade-off between quality of estimate and work per node!
Other A* Applications

- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- ...

25
Features of A*

- A* uses both backward costs and (estimates of) forward costs

- A* is optimal with admissible heuristics

- Heuristic design is key: often use relaxed problems
Project 1

- You can start to work on UCS and A* parts
- Start early!
- Ask questions on Piazza