CS 232: Artificial Intelligence

Informed Search

Sep 10, 2015

Today

- Informed Search
  - Heuristics
  - Greedy Search
  - A* Search
- Graph Search

Recap: Search

- **Search problem:**
  - States (configurations of the world)
  - Actions and costs
  - Successor function (world dynamics)
  - Start state and goal test

- **Search tree:**
  - Nodes: represent plans for reaching states
  - Costs: sum of action costs

- **Search algorithm:**
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: finds least-cost plans

Example: Pancake Problem

Cost: Number of pancakes flipped
Example: Pancake Problem

General Tree Search

The One Queue

- All these search algorithms are the same except for fringe strategies
- Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)
- Practically, for DFS and BFS, you can avoid the $\log(n)$ overhead from an actual priority queue, by using stacks and queues
- Can even code one implementation that takes a variable queuing object
Uninformed Search

Strategy:
- expand lowest path cost

The good:
- UCs is complete and optimal!

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

Uniform Cost Search

Video of Demo Contours UCS Empty

Video of Demo Contours UCS Pacman Small Maze

Spots with brighter color are visited earlier in the search, darker color later.
Informed Search

Search Heuristics

- A heuristic is:
  - A function that estimates how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance for pathing

Example: Heuristic Function

Heuristic: the number of the largest pancake that is still out of place
Greedy Search

- **Expand the node that seems closest...**
- What can go wrong?

Example: Heuristic Function

- **Strategy:** expand a node that you think is closest to a goal state
  - Heuristic: estimate of distance to nearest goal for each state
- **A common case:**
  - Best-first takes you straight to the (wrong) goal
- **Worst-case:** like a badly-guided DFS

[Demo: contours greedy empty (L3D1)]
[Demo: contours greedy pacman small maze (L3D4)]
Video of Demo Contours Greedy (Empty)

Video of Demo Contours Greedy (Pacman Small Maze)

A* Search

Combining UCS and Greedy

- **Uniform-cost** orders by path cost, or **backward cost** \( g(n) \)
- **Greedy** orders by goal proximity, or **forward cost** \( h(n) \)

- **A* Search** orders by the sum: \( f(n) = g(n) + h(n) \)

Example: Teg Grenager
When should $A^*$ terminate?

- Should we stop when we enqueue a goal?
  - No: only stop when we dequeue a goal

Is $A^*$ Optimal?

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

Admissible Heuristics

Idea: Admissibility

Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe

Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs
Admissible Heuristics

- A heuristic $h$ is **admissible** (optimistic) if:
  \[ 0 \leq h(n) \leq h^*(n) \]
  
  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.

---

Optimality of A* Tree Search

Assume:
- $A$ is an optimal goal node
- $B$ is a suboptimal goal node
- $h$ is admissible

Claim:
- $A$ will exit the fringe before $B$

Optimality of A* Tree Search: Blocking

Proof:
- Imagine $B$ is on the fringe
- Some ancestor $n$ of $A$ is on the fringe, too (maybe $A$!)
- Claim: $n$ will be expanded before $B$
  1. $f(n)$ is less or equal to $f(A)$

\[
 f(n) = g(n) + h(n) \\
 f(n) \leq g(A) \\
 g(A) = f(A) \\
 h = 0 \text{ at a goal}
\]
**Optimality of A* Tree Search: Blocking**

**Proof:**
- Imagine $B$ is on the fringe
- Some ancestor $n$ of $A$ is on the fringe, too (maybe $A$!)
- Claim: $n$ will be expanded before $B$
  1. $f(n)$ is less or equal to $f(A)$
  2. $f(A)$ is less than $f(B)$

\[ g(A) < g(B) \quad \text{B is suboptimal} \]
\[ f(A) < f(B) \quad \text{h = 0 at a goal} \]

**Properties of A***

**Uniform-Cost**

**A***

**UCS vs A*** Contours

- Uniform-cost expands equally in all “directions”
- $A^*$ expands mainly toward the goal, but does hedge its bets to ensure optimality
Video of Demo Contours (Empty) – A*

Video of Demo Contours (Pacman Small Maze) – A*

Comparison

A* Applications

Greedy	Uniform Cost	A*
A* Applications

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

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
- Inadmissible heuristics are often useful too

Example: 8 Puzzle

- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

8 Puzzle I

- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8
- This is a relaxed-problem heuristic

Statistics from Andrew Moore
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 is it admissible?
- \( h(\text{start}) = 3 + 1 + 2 + \ldots = 18 \)

<table>
<thead>
<tr>
<th>Tiles</th>
<th>4 steps</th>
<th>8 steps</th>
<th>12 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiles</td>
<td>13</td>
<td>39</td>
<td>227</td>
</tr>
<tr>
<td>Manhattan</td>
<td>12</td>
<td>25</td>
<td>73</td>
</tr>
</tbody>
</table>

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
  - As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

Graph Search

- Failure to detect repeated states can cause exponentially more work.

Tree Search: Extra Work!
In BFS, for example, we shouldn't bother expanding the circled nodes (why?)

Graph Search

- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states ("closed set")
  - Expand the search tree node-by-node, but...
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to closed set
- Important: store the closed set as a set, not a list
- Can graph search wreck completeness? Why/why not?
- How about optimality?

A* Graph Search Gone Wrong?

State space graph

Search tree

Consistency of Heuristics

- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal
    \[ h(A) \leq \text{actual cost from } A \text{ to } G \]
  - Consistency: heuristic "arc" cost ≤ actual cost for each arc
    \[ h(A) - h(C) \leq \text{cost}(A \text{ to } C) \]
- Consequences of consistency:
  - The f value along a path never decreases
    \[ h(A) \leq \text{cost}(A \text{ to } C) + h(C) \]
  - A* graph search is optimal
Optimality of A* Graph Search

Sketch: consider what A* does with a consistent heuristic:

- Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)
- Fact 2: For every state s, nodes that reach s optimally are expanded before nodes that reach s suboptimally
- Result: A* graph search is optimal

Optimality

- Tree search:
  - A* is optimal if heuristic is admissible
  - UCS is a special case (h = 0)

- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal (h = 0 is consistent)

- Consistency implies admissibility

- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems

A*: Summary
A*: Summary

- A* uses both backward costs and (estimates of) forward costs
- A* is optimal with admissible / consistent heuristics
- Heuristic design is key: often use relaxed problems

Tree Search Pseudo-Code

```java
function TREE-SEARCH(problem, fringe) return a solution, or failure
fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
loop do
  if fringe is empty then return failure
  node ← REMOVE-FRONT(fringe)
  if GOAL-TEST(node) then return node
  for child-node in EXPAND(node, problem) do
    fringe ← INSERT(child-node, fringe)
end
```

Informal description

Graph Search Pseudo-Code

```java
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
closed ← an empty set
fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
loop do
  if fringe is empty then return failure
  node ← REMOVE-FRONT(fringe)
  if GOAL-TEST(node) then return node
  if STATE[node] is not in closed then
    add STATE[node] to closed
  for child-node in EXPAND(node, problem) do
    fringe ← INSERT(child-node, fringe)
end
```

Informal description

Initialize the fringe using the initial state of problem
Loop do
  if the fringe is empty then return failure
  if the node contains a goal state then return the corresponding solutions
  if the fringe is not in the closed set then
    add the node to the closed set
  add the node to the closed set
End loop