CSE 473: Introduction to Artificial Intelligence

Hanna Hajishirzi
Search
(Un-informed, Informed Search)

slides adapted from
Dan Klein, Pieter Abbeel ai.berkeley.edu
And Dan Weld, Luke Zettelmoyer
Announcements

- HW1 is released
  - Due: Friday 6pm

- PS1 is due: Next Wednesday (April 14th)
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

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

- Uninformed Search
  - DFS
  - BFS
  - UCS

- Informed Search
  - Heuristics
  - Greedy Search
  - A* Search
  - Graph Search
Uniform Cost Issues

- Remember: UCS explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every “direction”
  - No information about goal location
- We’ll fix that soon!
Search Heuristics

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

○ Expand the node that seems closest...

○ Is it optimal?
  ○ No. Resulting path to Bucharest is not the shortest!
Greedy Search

- **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
Video of Demo Contours Greedy (Pacman Small Maze)
A* Search
A* Search

UCS

Greedy
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
Questions

- Should we stop when we enqueue a goal?
- Is A* optimal?
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!
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.
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
Comparison

Greedy

Uniform Cost

A*
Video of Demo Contours (Empty) -- UCS
Video of Demo Contours (Empty) -- Greedy
Video of Demo Contours (Empty) – A*
UCS vs. A*
Video of Demo Empty Water Shallow/Deep
– Guess Algorithm
CSE 473: Introduction to Artificial Intelligence

Hanna Hajishirzi
Search
(Un-informed, Informed Search)

slides adapted from
Dan Klein, Pieter Abbeel ai.berkeley.edu
And Dan Weld, Luke Zettelmoyer
A*: Summary
A*: Summary

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

YOU GOT
HEURISTIC
UPGRADE!
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?

Look at the image:
- Start State
- Actions
- Goal State

Admissible heuristics?
8 Puzzle I

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

Start State

Goal State

<table>
<thead>
<tr>
<th>Average nodes expanded when the optimal path has...</th>
<th>...4 steps</th>
<th>...8 steps</th>
<th>...12 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCS</td>
<td>112</td>
<td>6,300</td>
<td>3.6 x 10^6</td>
</tr>
<tr>
<td>TILES</td>
<td>13</td>
<td>39</td>
<td>227</td>
</tr>
</tbody>
</table>

Statistics from Andrew Moore
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?

\[ h(\text{start}) = 3 + 1 + 2 + \ldots = 18 \]

Average nodes expanded when the optimal path has...

<table>
<thead>
<tr>
<th>TILES</th>
<th>4 steps</th>
<th>8 steps</th>
<th>12 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>39</td>
<td>227</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MANHATTAN</th>
<th>4 steps</th>
<th>8 steps</th>
<th>12 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>25</td>
<td>73</td>
<td></td>
</tr>
</tbody>
</table>
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.
Example: Pancake Problem

- Action: Flip over top $n$ pancakes
- Cost: Number of pancakes
Semi-Lattice of Heuristics
Trivial Heuristics, Dominance

- Dominance: \( h_a \geq h_c \) if
  \[
  \forall n : h_a(n) \geq h_c(n)
  \]

- Heuristics form a semi-lattice:
  - Max of admissible heuristics is admissible
    \[
    h(n) = \max(h_a(n), h_b(n))
    \]

- Trivial heuristics
  - Bottom of lattice is the zero heuristic (what does this give us?)
  - Top of lattice is the exact heuristic
Optimality of A* Tree Search
Optimality of A* Tree Search

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

Proof Sketch:
- All ancestors of A will exit the fringe before B
  - Because $f(n) < f(B)$
- A will exit the fringe before B
Graph Search
Tree Search: Extra Work!

- Failure to detect repeated states can cause exponentially more 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

Closed Set: S B C A
Consistency of Heuristics

- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal
    \[ h(A) \leq \text{actual cost from A to G} \]
  - Consistency: heuristic “arc” cost ≤ actual cost for each arc
    \[ h(A) - h(C) \leq \text{cost(A to C)} \]

- Consequences of consistency:
  - The \( f \) value along a path never decreases
    \[ h(A) \leq \text{cost(A to C)} + h(C) \]
  - \( A^* \) graph search is optimal
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 of A* Search

- With a admissible heuristic, Tree A* is optimal.
- With a consistent heuristic, Graph A* is optimal.
- With h=0, the same proof shows that UCS is optimal.
Pseudo-Code

**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($problem$, STATE[$node$]) then return $node$
  for $child-node$ in EXPAND(STATE[$node$], $problem$) do
    $fringe$ ← INSERT($child-node$, $fringe$)
  end
end

**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($problem$, STATE[$node$]) then return $node$
  if STATE[$node$] is not in $closed$ then
    add STATE[$node$] to $closed$
    for $child-node$ in EXPAND(STATE[$node$], $problem$) do
      $fringe$ ← INSERT($child-node$, $fringe$)
    end
  end
end
A* Applications

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

...
A* in Recent Literature

- Joint A* CCG Parsing and Semantic Role Labeling (EMLN’15)

- Diagram Understanding (ECCV’17)
Search and Models

- Search operates over models of the world
  - The agent doesn’t actually try all the plans out in the real world!
  - Planning is all “in simulation”
  - Your search is only as good as your models…
Search Gone Wrong?