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
To Do:

- Python practice (PS0)
  - Won’t be graded
- Check out PS1 in the webpage
  - Start ASAP
  - Submission: Canvas

- Website:
  - Do readings for search algorithms
  - Try this search visualization tool
    - http://qiao.github.io/PathFinding.js/visual/
Recap: Search
Search problem:
- States (abstraction of the world)
- Actions (and costs)
- Successor function (world dynamics):
  - \( \{ s' \mid s, a \rightarrow s' \} \)
- Start state and goal test
Depth-First Search
Depth-First Search

Strategy: expand a deepest node first

Implementation:
Fringe is a LIFO stack
Search Algorithm Properties
Search Algorithm Properties

- **Complete:** Guaranteed to find a solution if one exists?
  - Return in finite time if not?
- **Optimal:** Guaranteed to find the least cost path?
- **Time complexity?**
- **Space complexity?**

- **Cartoon of search tree:**
  - \( b \) is the branching factor
  - \( m \) is the maximum depth
  - Solutions at various depths

- **Number of nodes in entire tree?**
  - \( 1 + b + b^2 + \ldots + b^m = O(b^m) \)
**Depth-First Search (DFS) Properties**

- **What nodes DFS expand?**
  - Some left prefix of the tree.
  - Could process the whole tree!
  - If $m$ is finite, takes time $O(b^m)$

- **How much space does the fringe take?**
  - Only has siblings on path to root, so $O(bm)$

- **Is it complete?**
  - $m$ could be infinite, so only if we prevent cycles (more later)

- **Is it optimal?**
  - No, it finds the “leftmost” solution, regardless of depth or cost
Breadth-First Search
Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue
**Breadth-First Search (BFS) Properties**

- **What nodes does BFS expand?**
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be \( s \)
  - Search takes time \( O(b^s) \)

- **How much space does the fringe take?**
  - Has roughly the last tier, so \( O(b^s) \)

- **Is it complete?**
  - \( s \) must be finite if a solution exists, so yes!
    (if no solution, still need depth \( \neq \infty \))

- **Is it optimal?**
  - Only if costs are all 1 (more on costs later)
Video of Demo Maze Water DFS/BFS (part 1)
Video of Demo Maze Water DFS/BFS (part 2)
Iterative Deepening

- Idea: get DFS’s space advantage with BFS’s time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..

- Isn’t that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!
Cost-Sensitive Search
BFS finds the shortest path in terms of number of actions. It does not find the least-cost path. We will now cover a similar algorithm which does find the least-cost path.
Uniform Cost Search
Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)
Uniform Cost Search (UCS) Properties

- **What nodes does UCS expand?**
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs $C^*$ and arcs cost at least $\varepsilon$, then the “effective depth” is roughly $C^*/\varepsilon$
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)

- **How much space does the fringe take?**
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$

- **Is it complete?**
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes! (if no solution, still need depth $\neq \infty$)

- **Is it optimal?**
  - Yes! (Proof via A*)
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!
Video of Demo Empty UCS
Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 1)
Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 2)
Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 3)
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
Up next: Informed Search

- Uninformed Search
  - DFS
  - BFS
  - UCS

- Informed Search
  - Heuristics
  - Greedy Search
  - A* Search
  - Graph Search
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
Example: Heuristic Function

$h(x)$
Greedy Search
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 (Empty)
Video of Demo Contours Greedy (Pacman Small Maze)
A* Search
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!
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
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 $f(A)$

$$f(n) = g(n) + h(n)$$ Definition of f-cost
$$f(n) \leq g(A)$$ Admissibility of $h$
$$g(A) = f(A)$$

$h = 0$ 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 $f(B)$

$g(A) < g(B)$  B is suboptimal
$f(A) < f(B)$  h = 0 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 $f(B)$
  3. $n$ expands before B
- All ancestors of A expand before B
- A expands before B
- A* search is optimal

$$f(n) \leq f(A) < f(B)$$
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*
Video of Demo Contours (Pacman Small Maze) – A*
Which algorithm?
Which algorithm?
A* in Recent Literature

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

- Diagram Understanding (ECCV’17)
Video of Demo Empty Water Shallow/Deep – Guess Algorithm

Pydev-Eclipse

1. search -- plan tiny a* star
2. search -- plan tiny ucs
3. search demo empty
4. search -- contours greedy vs ucs (greedy)
5. search -- contours greedy vs ucs (ucs)
6. search -- contours greedy vs ucs (astar)
7. search -- greedy bad
8. search -- greedy good
9. search demo maze
search time costs

Run As
Run Configurations...
Organize Favorites...

<terminated> 15
Total cost: 27
Number of nodes expanded: 182
Number of unique nodes expanded: 182
Pacman emerges victorious! Source: 673
{'numKills': [5], 'results': ['Min'], 'numMoves': [27], 'moore': [673]}
Creating Heuristics
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?

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]

Start State

![Goal 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>$3.6 \times 10^6$</td>
</tr>
<tr>
<td>TILES</td>
<td>13</td>
<td>39</td>
<td>227</td>
</tr>
</tbody>
</table>

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 \textit{Manhattan} distance

- Why is it admissible?

- $h(\text{start}) = 3 + 1 + 2 + ... = 18$

<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>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
Example: Pancake Problem

- Action: Flip over top \( n \) pancakes
- Cost: Number of pancakes
Example: Pancake Problem

BOUNDS FOR SORTING BY PREFIX REVERSAL

William H. GATES
Microsoft, Albuquerque, New Mexico

Christos H. PAPADIMITRIOU*†
Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.

Received 18 January 1978
Revised 28 August 1978

For a permutation $\sigma$ of the integers from 1 to $n$, let $f(\sigma)$ be the smallest number of prefix reversals that will transform $\sigma$ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all $\sigma$ in (the symmetric group) $S_n$. We show that $f(n) \leq (5n + 5)/3$, and that $f(n) \geq 17n/16$ for $n$ a multiple of 16. If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function $g(n)$ is shown to obey $3n/2 - 1 \leq g(n) \leq 2n + 3$. 
Pancake Problem

- State graph with costs as weights
Uniform Cost Search

Action: flip top two
Cost: 2

Action: flip all four
Cost: 4

Path to reach goal:
Flip four, flip three
Total cost: 7
Example: Heuristic Function

Heuristic?

E.g. the number of the largest pancake that is still out of place

\[ h(x) \]
Graph Search
Failure to detect repeated states can cause exponentially more work.
Graph Search

- 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 $\leq$ actual costs
  - **Admissibility:** heuristic cost $\leq$ actual cost to goal
    \[ h(A) \leq \text{actual cost from A to G} \]
  - **Consistency:** heuristic “arc” cost $\leq$ 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* Search

- With a admissible heuristic, Tree A* is optimal.
- With a consistent heuristic, Graph A* is optimal.
  - See slides, also video lecture from past years for details.
- With h=0, the same proof shows that UCS is optimal.
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?
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
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(problem, STATE[node]) then return node

    for child-node in EXPAND(STATE[node], problem) do
        fringe ← INSERT(child-node, fringe)
    end
end
function \textsc{Graph-Search}(\textit{problem, fringe}) return a solution, or failure

closed \leftarrow \text{an empty set}
fringe \leftarrow \text{INSERT(MAKE-NODE(INITIAL-STATE[\textit{problem}]), fringe)}

loop do

\textbf{if} fringe is empty \textbf{then} return failure

node \leftarrow \text{REMOVE-FRONT(fringe)}
\textbf{if} GOAL-TEST(\textit{problem, state[node]}) \textbf{then} return node

\textbf{if} STATE[node] is not in closed \textbf{then}

\quad add STATE[node] to closed

\quad \textbf{for} child-node in EXPAND(STATE[node], \textit{problem}) \textbf{do}

\quad \quad fringe \leftarrow \text{INSERT(child-node, fringe)}

\textbf{end}

\textbf{end}