CS 188: Artificial Intelligence

Search Continued

Instructors: Anca Dragan, Sergey Levine

University of California, Berkeley

[These slides adapted from Dan Klein and Pieter Abbeel; ai.berkeley.edu]
Recap: Search
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
Tree 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
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!

- **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!

[Demo: empty grid UCS (L2D5)]
[Demo: maze with deep/shallow water DFS/BFS/UCS (L2D7)]
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)
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

[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 $< \text{ estimated good goal cost}$
- 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
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:
1. Imagine B is on the fringe
2. Some ancestor $n$ of A is on the fringe, too (maybe A!)
3. Claim: $n$ will be expanded before B
   1. $f(n)$ is less or equal to $f(A)$

- Definition of $f$-cost
  $f(n) = g(n) + h(n)$
- Admissibility of $h$
  $f(n) \leq g(A)$
- $g(A) = f(A)$
- $h = 0$ 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 $f(B)$

$g(A) < g(B)$  \hspace{1cm} B is suboptimal
$f(A) < f(B)$  \hspace{1cm} 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

[Demo: contours UCS / greedy / A* empty (L3D1)]
[Demo: contours A* pacman small maze (L3D5)]
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*
Comparison

Greedy

Uniform Cost

A*
A* Applications
A* Applications

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

[Demo: UCS / A* pacman tiny maze (L3D6,L3D7)]
[Demo: guess algorithm Empty Shallow/Deep (L3D8)]
Video of Demo Pacman (Tiny Maze) – UCS / A*
Video of Demo Empty Water Shallow/Deep – Guess Algorithm
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

Statistics from Andrew Moore

<table>
<thead>
<tr>
<th></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>
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>13</td>
<td>39</td>
<td>227</td>
<td></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.
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
    \[ h(A) - h(C) \leq \text{cost(A to C)} \]
- **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
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 proofs shows that UCS is optimal.
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
Graph Search Pseudo-Code

function \textsc{Graph-Search}(\textit{problem, fringe}) return a solution, or failure
\begin{itemize}
  \item \textit{closed} $\leftarrow$ an empty set
  \item \textit{fringe} $\leftarrow$ \textsc{Insert} (\textsc{make-node}(\textsc{initial-state}[\textit{problem}]), \textit{fringe})
\end{itemize}
\begin{itemize}
  \item loop do
    \begin{itemize}
      \item if \textit{fringe} is empty then return failure
      \item \textit{node} $\leftarrow$ \textsc{Remove-Front}(\textit{fringe})
      \item if \textsc{goal-test}(\textit{problem}, \textsc{state}[\textit{node}]) then return \textit{node}
      \item if \textsc{state}[\textit{node}] is not in \textit{closed} then
        \begin{itemize}
          \item add \textsc{state}[\textit{node}] to \textit{closed}
          \item \textit{fringe} $\leftarrow$ \textsc{Insert}(\textit{child-node}, \textit{fringe})
        \end{itemize}
    \end{itemize}
\end{itemize}
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.
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?