CIS 471/571 (Winter 2020): Introduction to Artificial Intelligence

Lecture 3: Informed Search

Thanh H. Nguyen

Most slides are by Pieter Abbeel, Dan Klein, Luke Zettlemoyer, John DeNero, Stuart Russell, Andrew Moore, or Daniel Lowd
Source: http://ai.berkeley.edu/home.html
Reminder

- Homework 1: Search
  - Deadline: Jan 19th, 2020

- Project 1: Search
  - Deadline: Jan 20th, 2020
Today

- Uninformed Search
  - Uniform Cost Search

- 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
  - Plans have 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
Uninformed Search
Uniform-Cost Search (UCS)

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)
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!
Uniform Cost 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
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

\[ h(x) \]
Example: Pancake Problem

Cost: Number of pancakes flipped
Example: Pancake Problem

State space graph with costs as weights
Example: Heuristic Function

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

- Expand the node that seems closest…

Straight-line distance to Bucharest

<table>
<thead>
<tr>
<th>Place</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
</tr>
<tr>
<td>Dobroța</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Făgăraș</td>
<td>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hirșova</td>
<td>151</td>
</tr>
<tr>
<td>Iași</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Neamț</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitești</td>
<td>98</td>
</tr>
<tr>
<td>Râmnicu Vlăcăea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timișoara</td>
<td>329</td>
</tr>
<tr>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Vâlcea</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Greedy Search

- Expand the node that seems closest...

- What can go wrong?
Greedy Search

- A common case:
  - Best-first takes you straight to the (wrong) goal

- Worst-case: like a badly-guided DFS
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: $f(n) = g(n) + h(n)$
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 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

- Heuristic function $h$ is admissible
- Claim: A* tree search is optimal
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 $f(A)$

\[
\begin{align*}
  f(n) &= g(n) + h(n) & \text{Definition of f-cost} \\
  f(n) &\leq g(A) & \text{Admissibility of h} \\
  g(A) &= f(A) & h = 0 \text{ at a goal}
\end{align*}
\]
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 B \text{ is suboptimal}

$f(A) < f(B)$  \quad 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 \( f(B) \)
  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*
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
A* Applications

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- ...
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?
- $h(\text{start}) = 8$
- This is a relaxed-problem heuristic

Start State

Goal State

Average nodes expanded when the optimal path has...

<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>

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?

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

<table>
<thead>
<tr>
<th></th>
<th>Average nodes expanded when the optimal path has...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>...4 steps</td>
</tr>
<tr>
<td>TILES</td>
<td>13</td>
</tr>
<tr>
<td>MANHATTAN</td>
<td>12</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
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 (why?)
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
Tree Search: Extra Work!

- Failure to detect repeated states can cause exponentially more work. Why?
Graph Search

- In BFS, for example, we shouldn’t bother expanding some nodes (which, and why?)
Graph Search

- Very simple fix: never expand a state type twice

```
function GRAPH-SEARCH(problem, fringe) returns 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
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
    end
```

- Can this wreck completeness? Why or why not?
- How about optimality? Why or why not?
**A* Graph Search Gone Wrong**

**State space graph**

- **S** (h=2) → **A** (h=4) → **C** (h=1)
- **B** (h=1) → **C** (h=1)
- **G** (h=0)

**Search tree**

- **S** (0+2) → **A** (1+4) → **C** (2+1) → **G** (5+0)
- **B** (1+1) → **C** (3+1) → **G** (6+0)
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
    \[ h(A) \leq \text{cost(A to C) + h(C)} \]
    \[ f(A) = g(A) + h(A) \leq g(A) + \text{cost(A to C) + h(C)} \leq f(C) \]
  - A* graph search is optimal
Optimality of A* Graph Search
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

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}, \textit{fringe}) return a solution, or failure
   \textit{closed} ← an empty set
   \textit{fringe} ← \textsc{Insert}(\textsc{make-node}(\textit{initial-state}[^{\textit{problem}}]), \textit{fringe})
   loop do
      if \textit{fringe} is empty then return failure
      \textit{node} ← \textsc{remove-front}(\textit{fringe})
      if \textsc{goal-test}(\textit{problem}, \textit{state}[\textit{node}]) then return \textit{node}
      if \textit{state}[\textit{node}] is not in \textit{closed} then
         add \textit{state}[^{\textit{node}}] to \textit{closed}
         for \textit{child-node} in \textsc{expand}(\textit{state}[^{\textit{node}}], \textit{problem}) do
            \textit{fringe} ← \textsc{Insert}(\textit{child-node}, \textit{fringe})
         end
      end
   end
Optimality of A* Graph Search

Consider what A* does:
- Expands nodes in increasing total f value (f-contours)
  Reminder: $f(n) = g(n) + h(n) = \text{cost to } n + \text{heuristic}$
- Proof idea: the optimal goal(s) have the lowest f value, so it must get expanded first

There’s a problem with this argument. What are we assuming is true?
Optimality of A* Graph Search

Proof:

- New possible problem: some \( n \) on path to \( G^* \) isn’t in queue when we need it, because some worse \( n' \) for the same state dequeued and expanded first (disaster!)
- Take the highest such \( n \) in tree
- Let \( p \) be the ancestor of \( n \) that was on the queue when \( n' \) was popped
- \( f(p) < f(n) \) because of consistency
- \( f(n) < f(n') \) because \( n' \) is suboptimal
- \( p \) would have been expanded before \( n' \)
- Contradiction!