Announcements

- **Homework 1: Search**
  - Has been released! Due Tuesday, Sep 4th, at 11:59pm.
  - Electronic component: on Gradescope, instant grading, submit as often as you like.
  - Written component: exam-style template to be completed (we recommend on paper) and to be submitted into Gradescope (graded on effort/completion)

- **Project 1: Search**
  - Has been released! Due Friday, Sep 7th, at 4pm.
  - Start early and ask questions. It’s longer than most!

- **Sections**
  - Started this week
  - You can go to any, but have priority in your own
  - Section webcasts

---

CS 188: Artificial Intelligence

Informed Search

Instructors: Pieter Abbeel & Dan Klein
University of California, Berkeley

---

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
  - 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

---

Example: Pancake Problem

Cost: Number of pancakes flipped
Example: Pancake Problem

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

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
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

$h(x)$

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**

- **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: $f(n) = g(n) + h(n)$

**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 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

Proof:
1. $f(n)$ is less or equal to $f(A)$
2. $f(A)$ is less than $f(B)$

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 $f(A)$
   2. $f(A)$ is less than $f(B)$

B is suboptimal $h = 0$ 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 itself)
- 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

Properties of A*

- Uniform-cost expands equally in all "directions"
- A* expands mainly toward the goal, but does hedge its bets to ensure optimality

UCS vs A* Contours

Video of Demo Contours (Empty) -- UCS

Video of Demo Contours (Empty) -- Greedy
Comparison

- Greedy
- Uniform Cost
- A*

A* Applications

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

Video of Demo Pacman (Tiny Maze) – UCS / A*

A* Applications

[Demo: UCS / A* pacman tiny maze (L3D6,L3D7)]
[Demo: guess algorithm Empty Shallow/Deep (L3D8)]
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?
- \( h(\text{start}) = 8 \)
- This is a relaxed-problem heuristic.

Average nodes expanded when the optimal path has:

<table>
<thead>
<tr>
<th>( .4 \text{ steps} )</th>
<th>( .8 \text{ steps} )</th>
<th>( .12 \text{ steps} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCS</td>
<td>112</td>
<td>6,300</td>
</tr>
<tr>
<td>TILES</td>
<td>13</td>
<td>39</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?
- \( h(\text{start}) = 3 + 1 + 2 + ... = 18 \)

Average nodes expanded when the optimal path has:

<table>
<thead>
<tr>
<th>( .4 \text{ steps} )</th>
<th>( .8 \text{ steps} )</th>
<th>( .12 \text{ steps} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>TILES</td>
<td>13</td>
<td>39</td>
</tr>
<tr>
<td>MANHATTAN</td>
<td>12</td>
<td>25</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

Semi-Lattice of Heuristics

- Dominance: $h_a \geq h_b$ if
  $$\forall n : h_a(n) \geq h_b(n)$$
- 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

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?)

Trivial Heuristics, Dominance

- Exact
  - 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

Graph Search

- In BFS, for example, we shouldn't bother expanding the circled nodes (why?)

Tree Search: Extra Work!

- Failure to detect repeated states can cause exponentially more work.
**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
    \[ h(s) \leq \text{actual cost from } s \text{ to goal} \]
  - 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* uses both backward costs and (estimates of) forward costs
- A* is optimal with admissible / consistent heuristics
- Heuristic design is key: often use relaxed problems

---

**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?*

---

**Tree Search Pseudo-Code**

```plaintext
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
end
```

---

**Graph Search Pseudo-Code**

```plaintext
function GRAPH-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
        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
end
```

---

**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!