Today

- Informed search
  - Heuristics
  - Greedy search
  - A* search

- Graph search
Recap: Search

- **Search problem:**
  - States (configurations of the world)
  - Actions and costs
  - Successor function: a function from states to lists of (state, action, cost) triples (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

State space graph with costs as weights
function Tree-Search(problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end

Action: flip top two
Cost: 2

Path to reach goal:
Flip four, flip three
Total cost: 7
Recall: 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
A heuristic is:

- A function that estimates how close a state is to a goal
- Designed for a particular search problem
Example: Heuristic Function

$h(x)$
Example: Heuristic Function

Heuristic: the largest pancake that is still out of place

$h(x)$
• How to use the heuristic?
• What about following the “arrow” of the heuristic? .... Greedy search
Example: Heuristic Function

\[ h(x) \]
Best First / Greedy Search

- Expand the node that seems closest...

- What can go wrong?
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
Enter: 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* \( h(n) \)

\[ f(n) = g(n) + h(n) \]

Example: Teg Grenager
When should \( A^* \) terminate?

- Should we stop when we enqueue a goal?
  
  - No: only stop when we dequeue a goal

\[ \begin{align*}
S & \xrightarrow{2} A & A & \xrightarrow{2} G \\
 & \quad \quad h = 3 & h = 2 & \quad \quad h = 0 \\
S & \xrightarrow{2} B & B & \xrightarrow{3} G \\
 & \quad \quad h = 1 & h = 1 & \quad \quad h = 0
\end{align*} \]
Is A* Optimal?

- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!
Inadmissible (pessimistic): break optimality by trapping good plans on the fringe

Admissible (optimistic): slows down bad plans but never outweigh true costs
Admissible Heuristics

- A heuristic $h$ is *admissible* (optimistic) if:
  \[ h(n) \leq h^*(n) \]
  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*

Notation:

- $g(n) =$ cost to node $n$
- $h(n) =$ estimated cost from $n$ to the nearest goal (heuristic)
- $f(n) = g(n) + h(n) =$ estimated total cost via $n$

- A: a lowest cost goal node
- B: another goal node

Claim: A will exit the fringe before B.
Claim: A will exit the fringe before B.

Optimality of A*

- Imagine B is on the fringe.
- Some ancestor n of A must be on the fringe too (maybe n is A)
- Claim: n will be expanded before B.

1. \( f(n) \leq f(A) \)

- \( f(n) = g(n) + h(n) \) // by definition
- \( f(n) \leq g(A) \) // by admissibility of h
- \( g(A) = f(A) \) // because h=0 at goal
Optimality of A*

- Imagine B is on the fringe.
- Some ancestor n of A must be on the fringe too (maybe n is A)
- Claim: n will be expanded before B.
  1. \( f(n) \leq f(A) \)
  2. \( f(A) < f(B) \)

Claim: A will exit the fringe before B.

- \( g(A) < g(B) \) // B is suboptimal
- \( f(A) < f(B) \) // \( h=0 \) at goals
Optimality of A*

Claim: A will exit the fringe before B.

- Imagine B is on the fringe.
- Some ancestor n of A must be on the fringe too (maybe n is A)
- Claim: n will be expanded before B.
  1. $f(n) \leq f(A)$
  2. $f(A) < f(B)$
  3. n will expand before B

\[ f(n) \leq f(A) < f(B) \]  // from above
\[ f(n) < f(B) \]
Claim: A will exit the fringe before B.

Optimality of A*

- Imagine B is on the fringe.
- Some ancestor n of A must be on the fringe too (maybe n is A)
- Claim: n will be expanded before B.
  1. $f(n) \leq f(A)$
  2. $f(A) < f(B)$
  3. n will expand before B
- All ancestors of A expand before B
- A expands before 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

[demo: countours UCS / A*]
A* applications

- Pathing / routing problems
- Video games
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …
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?)
Example: 8 Puzzle

- What are the states?
- How many states?
- What are the actions?
- What states can I reach 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](image1)
![Goal State](image2)

<table>
<thead>
<tr>
<th>Average nodes expanded when optimal path has length...</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>
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 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>12</td>
<td>25</td>
<td>73</td>
<td></td>
</tr>
</tbody>
</table>

Average nodes expanded when optimal path has length...
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 admissible?

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

\[ = 18 \]
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!
Today

- Informed search
  - Heuristics
  - Greedy search
  - A* search

- Graph search
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 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 is new
  - If not new, skip it

- Important: store the closed set as a set, not a list

- Can graph search wreck completeness? Why/why not?

- How about optimality?

Warning: 3e book has a more complex, but also correct, variant
A* Graph Search Gone Wrong?

State space graph

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

- **Admissibility**: heuristic cost $\leq$ actual cost to goal
  - $h(A) \leq$ actual cost from A to G
Consistency of Heuristics

- Stronger than admissibility
- Definition:
  - heuristic cost \( \leq \) actual cost per arc
  - \( h(A) - h(C) \leq \) cost(A to C)

- Consequences:
  - The f value along a path never decreases
  - A* graph search is optimal
Optimality

- **Tree search:**
  - A* is optimal if heuristic is admissible (and non-negative)
  - 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
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
Summary: A*

- 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