Recap: Search

- Search problem:
  - States (configurations of the world)
  - Successor function: a function from states to lists of (state, action, cost) triples; drawn as a graph
  - 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)

Example: Pancake Problem

Action: Flip over the top \( n \) pancakes

Cost: Number of pancakes flipped

Example: Pancake Problem

State space graph with costs as weights
### General Tree Search

- **Action:** flip top two
- **Cost:** 2
- **Path to reach goal:** Flip four, flip three
  - **Total cost:** 7

### Example: Heuristic Function

- Heuristic: the largest pancake that is still out of place

### What is a Heuristic?

- **An estimate** of how close a state is to a goal
- **Designed for a particular search problem**

- **Examples:** Manhattan distance: 10 + 5 = 15
  - Euclidean distance: \( \sqrt{10^2 + 5^2} = 11.2 \)

### 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
Greedy Search

- Expand the node that seems closest...

- What can go wrong?

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 path cost
- We need estimates to be less than or equal to actual 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:
- 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 \( f(B) \)

\[ g(A) < g(B) \]
\[ f(A) < f(B) \]

\( h = 0 \) at a goal

\( f(n) = g(n) + h(n) \)
\( f(n) \leq g(A) \)
\( g(A) = f(A) \)

All ancestors of A expand before B
- A expands before B
- A* search is optimal

**UCS vs A* Contours**

- Uniform-cost expanded in all directions
- A* expands mainly toward the goal, but hedges its bets to ensure optimality

**Which Algorithm?**

- Uniform cost search (UCS):

![UCS Contours](image)

![A* Contours](image)
Which Algorithm?

- A*, Manhattan Heuristic:

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

Creating Heuristics

- 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
  - h(start) = 8
  - Is it admissible?

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:
  - h(start) = 3 + 1 + 2 + 2 + 3 + 3 + 5 + 8 = 22
  - Admissible?
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!

Trivial Heuristics, Dominance

- Dominance: $h_a(n) \geq h_c(n)$
- 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

A* Applications

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

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

- 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
  - Hint: in python, 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
    \( h(A) \leq \text{actual cost from } A \text{ to } G \)
  - 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
      \[ f(A) \leq g(A) + h(A) \leq g(A) + \text{cost}(A \text{ to } C) + h(C) \]
    - A* graph search is optimal

Optimality of A* Graph Search

- Sketch: consider what A* does with a consistent heuristic:
  - Nodes are popped with non-decreasing f-scores: for all \( n, n' \) with \( n' \) popped after \( n \):
    \( f(n') \geq f(n) \)
  - Proof by induction: (1) always pop the lowest f-score from the fringe, (2) all new nodes have larger (or equal) scores, (3) add them to the fringe, (4) repeat!
  - For every state \( s \), nodes that reach \( s \) optimally are expanded before nodes that reach \( s \) sub-optimally
  - Result: A* graph search is optimal

Optimality

- Tree search:
  - A* 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, natural admissible heuristics tend to be consistent, especially if from relaxed problems

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