Announcements

- Projects:
  - The Python tutorial (Project 0) is due tomorrow
  - Project 1 (Search) is out, due next Wednesday
  - You don’t need to submit answers the project’s discussion questions
  - Use Python 2.5 (on EECS instructional machines)
  - 5 slip days for projects; up to two per deadline
  - Try pair programming, not divide-and-conquer

Today

- A* Search
- Heuristic Design
- Local Search

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 paths / plans
  - Paths have costs (sum of action costs)
- Search Algorithm:
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)

Example: Pancake Problem

State space graph with costs as weights
General Tree Search

- Action: flip top two
  - Cost: 2
- Action: flip all four
  - Cost: 4
- Path to reach goal: Flip four, flip three
  - Total cost: 7

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

Best First (Greedy)

- 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

Example: Heuristic Function

Heuristic: the largest pancake that is still out of place

Combining UCS and Greedy

- Uniform-cost orders by path cost, or backward cost \( g(n) \)
- Best-first 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

- A heuristic $h$ is **admissible** (optimistic) if:
  \[ h(n) \leq h^*(n) \]
  where $h^*(n)$ is the true cost to a nearest goal
- E.g. Euclidean distance in a Pacman maze
- Coming up with admissible heuristics is most of what’s involved in using A* in practice.

Optimality of A*: Blocking

**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 $n$
- $G^*$: the lowest cost goal node
- $G$: another goal node

**Proof:**
- What could go wrong?
- We’d have to have to pop a suboptimal goal $G$ off the fringe before $G^*$
- This can’t happen:
  - Imagine a suboptimal goal $G$ is on the queue
  - Some node $n$ which is a subpath of $G^*$ must be on the fringe (why?)
  - $n$ will be popped before $G$

Properties of A*

**Uniform-Cost**

**A***

UCS vs A* Contours

- Uniform-cost expanded in all directions
- A* expands mainly toward the goal, but does hedge its bets to ensure optimality
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 (examples coming up).
- Inadmissible heuristics are often useful too (why?)
- Very common hack: use $\alpha \times h(n)$ for admissible $h$, $\alpha > 1$ to generate a faster but less optimal, inadmissible $h'$ from admissible $h$.

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

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!

Trivial Heuristics, Dominance

- 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
5 Minute Break

Graph Search, Reconsidered

- Idea: never expand a state twice
- How to implement:
  - Tree search + list of expanded states (closed list)
  - Expand the search tree node-by-node, but...
  - Before expanding a node, check to make sure its state is new
- Python trick: store the closed list as a set, not a list
- Can graph search wreck completeness? Why/why not?
- How about optimality?

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) has the lowest f value, so it must get expanded first

There's a problem with this argument. What are we assuming is true?

Consistency

- Wait, how do we know we expand in increasing f value?
- Couldn't we pop some node \( n \), and find its child \( n' \) to have lower f value?
- YES:
- What can we require to prevent these inversions?
- Consistency: \( c(n,a,n') \geq h(n) - h(n') \)
- Real cost must always exceed reduction in heuristic