Announcements

- Projects:
  - Project 1 (Search) is out, due next Monday 9/14
  - You don’t need to submit answers the project’s discussion questions
  - Use Python 2.5 (on EECS instructional machines), 2.6 interpreter is backwards compatible, so also OK
  - 5 slip days for projects; up to two per deadline
  - Try pair programming, not divide-and-conquer

- Newsgroup:
  - Seems like WebNews wants your original password (inst is aware), or use Thunderbird, etc.
Today

- A* Search
- Heuristic Design

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

**Example: Pancake Problem**

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

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

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
Example: Heuristic Function

Heuristic: the largest pancake that is still out of place

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

[demo: countours greedy]
Example: Heuristic Function

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

- Examples:

- 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 $n$
- $G^*$: a lowest cost goal node
- $G$: another goal node
Optimality of A*: Blocking

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 also be on the fringe (why?)
  - n will be popped before G

\[ f(n) = g(n) + h(n) \]
\[ g(n) + h(n) \leq g(G^*) \]
\[ g(G^*) < g(G) \]
\[ g(G) = f(G) \]
\[ f(n) < f(G) \]

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

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

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>UCS</th>
<th>Average nodes expanded when optimal path has length…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tiles misplaced</td>
<td>112</td>
<td>…4 steps</td>
</tr>
<tr>
<td>Tiles</td>
<td>13</td>
<td>39</td>
</tr>
</tbody>
</table>
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?

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

<table>
<thead>
<tr>
<th></th>
<th>TILES</th>
<th>MANHATTAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average nodes expanded when optimal path has length...</td>
<td>...4 steps</td>
<td>...8 steps</td>
</tr>
<tr>
<td>4 steps</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>8 steps</td>
<td>39</td>
<td>25</td>
</tr>
<tr>
<td>12 steps</td>
<td>227</td>
<td>73</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!
Trivial Heuristics, Dominance

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

Other A* Applications

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

[demo: plan tiny UCS / A*]
### Tree Search: Extra Work!

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

![Graph Search Example](image1)

### Graph Search

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

![Graph Search Example](image2)
Graph Search

- 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?

---

Graph Search

- Very simple fix: never expand a state twice

```python
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? Optimality?
Optimality of A* Graph Search

Proof:
- New possible problem: nodes on path to G* that would have been in queue aren’t, because some worse n’ for the same state as some n was dequeued and expanded first (disaster!)
- Take the highest such n in tree
- Let p be the ancestor which was on the queue when n’ was expanded
- Assume f(p) < f(n)
- f(n) < f(n’) because n’ is suboptimal
- p would have been expanded before n’
- So n would have been expanded before n’, too
- Contradiction!

Consistency

- Wait, how do we know parents have better f-values than their successors?
- 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’) ≥ h(n) − h(n’)
- Real cost must always exceed reduction in heuristic
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

Summary: A*

- A* uses both backward costs and (estimates of) forward costs

- A* is optimal with admissible heuristics

- Heuristic design is key: often use relaxed problems