PS1 due Thursday!
Last Time: Uniformed Search

- **General-Purpose**
  - Require only the problem definition itself:
    - $state_0$
    - $successors(state)$
    - $is\text{-}goal(state)$
    - $cost(path)$

- **Powerful**
  - Several Complete and Optimal algorithms to choose from!

General Tree Search

```
function Tree-search(problem) returns a solution, or failure

fringe = new Queue();
fringe.put(problem.initialState)

loop do
  if fringe.isEmpty() then return failure
  node ← fringe.get()
  if problem.isGoalState(node)
    then return node;
  fringe.putAll(problem.expand(node))
```

Which node in the fringe does `get()` return?
### Analysis Summary

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Uniform-Cost</th>
<th>Depth-First</th>
<th>Depth-Limited</th>
<th>Iterative Deepening</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Time complexity</td>
<td>(O(b^{d+1}))</td>
<td>(O(b^{C^*+1}))</td>
<td>(O(b^m))</td>
<td>(O(b^L))</td>
<td>(O(b^d))</td>
</tr>
<tr>
<td>Space complexity</td>
<td>(O(b^{d+1}))</td>
<td>(O(b^{C^*+1}))</td>
<td>(O(bm))</td>
<td>(O(bL))</td>
<td>(O(bd))</td>
</tr>
</tbody>
</table>

- Linear in depth!
- Exponential in depth!

### Uninformed Search

- All we have is:
  - problem = \{state\_0, is-goal(), cost(), successors() \}

- We have no idea how well we’re doing until we suddenly find a goal!
Informed Search

- We still need to know the problem
  - problem = \{ state_0, is-goal(), cost(), successors() \}

- We get some additional information at each node
  - “Hot” or “Cold”
  - Which states are better than others?
  - Information about distance to goal

  - Formulate as a real-valued metric \( f(node) \).

Informed Search

- Exploit additional information of \( f() \) such that:
  - We find the goal faster when \( f() \) is accurate

- With the right algorithm:
  - We still (eventually) find a goal when \( f() \) is wrong
  - We can find the \textit{optimal} goal if \( f() \) is only wrong in certain ways!
    - Generally easy to satisfy conditions too!
Best-First Search

- It’s really just TreeSearch again, except we allow more types of Queue-Get functions

- For each node, define an *evaluation function* \( f(node) \).
  - Having \( f() \) is how we “inform” the search!
  - Queue-Get: Expand the node with the smallest value of \( f() \)

- Watch out for “BFS”
  - Breadth-first search or best-first search?

Greedy Search

- Simple informed search with \( f() = \text{cost-to-go}(n) \)

- Complete?

- Optimal?
Greedy Search: Example

- Queue-Get: returns the node with minimum \( f(n) = \text{cost-to-go}(n) \)
- Assume we avoid repeated states.

A (8)
Expand A:
AB (7)
AI (9)
Expand AB:
ABC (6)
AI (9)
Expand ABC:
ABCD (5)
AI (9)
Expand ABCD:
ABCDE (4)
AI (9)
Expand ABCDE:
dead end
AI (9)
Expand AI:
AI*

A*

It’s a Best-First Search!

Thrilling. Yet Another TreeSearch. What’s Queue-Get this time?

Glad you asked! Queue-Get returns the node with the minimum value of:

\[ f(n) = \text{cost-so-far}(n) + h(n), \]
where \( h(n) \) has some special properties.

What’s it good for?
A*:

- Provided the heuristic is *admissible*:
  - A* is complete
  - A* is optimal
  - A* is *optimally efficient*

**Optimally Efficient?**

No algorithm can expand fewer nodes than A* and still be guaranteed to find the optimal answer.

A*: Admissible heuristics

- Admissible:

h(n) \(\leq\) the minimum achievable cost from n to the goal.
**A*: Proof of Optimality

- **Strategy:**
  - Let C* be cost of optimal solution
  - Show that a node on the path to the optimal solution will always be selected for expansion before a sub-optimal goal node.
    - Eventually, that node will be the goal node and we’ll be done.

- **Proof**
  - Suppose a suboptimal goal G’ in a node on fringe
    - G’ is suboptimal \(\iff g(G’) > C^*\)
    - \(f(G’)=g(G’)+h(G’)\)
    - \(f(G’) > C^*\).
  - There must be a node \(n\) on fringe that is on optimal path, and because
    - \(h(n)\) can’t overestimate, \(f(n) < C^*\)
  - Since \(f(n) < f(G’)\), we’ll expand \(n\) before \(G’\).

---

**Admissible Heuristics**

- **Objectives:**
  1. Accurate estimate of distance to goal
  2. Never overestimate (admissible)
  3. Easy to compute

- **Approaches:**
  - Just think of something
  - Relax constraints
  - Learn from experience
A*: Example

What’s a good heuristic?
A* Example: Solution

1: \{AA(265)\}

3: \{ AA-D(269), AA-T(279), AA-K(470) \}

7: \{ AA-T(279), AA-D-AA(341), AA-D-F(391), AA-D-N(451), AA-D-L(456), AA-K(470) \}


Your turn!

- 1. Which algorithm do we get if we do Best First Search (BFS) with:
  - A. \( f() = \text{num-actions}(\text{node}) \)
  - B. \( f() = \text{cost-so-far}(\text{node}) \)
  - C. \( f() = \text{cost-to-go}(\text{node}) \)
  - D. \( f() = \text{cost-so-far}(\text{node}) + \text{cost-to-go}(\text{node}) \)
  - E. \( f() = \text{time-in-fringe}(\text{node}) \)
  - F. \( f() = -\text{time-in-fringe}(\text{node}) \)

- 2. Where will BFS “back up” to when it gets stuck?

- 3. What is BFS’s CPU runtime?

- 4. What is BFS’s memory usage?

- 5. Is BFS Complete? Optimal?
Graph Search and A*

- Tree search A* has same problem as other tree searches... (what problem is it?)
  - Can re-expand same states many times over.

- Graph Search was the answer...
  - Don’t add paths to the fringe when we already know how to get to that state.

Graph Search A*

- Use GraphSearch/A*

Sub-optimal!
(why did this happen?)
A stronger heuristic: Consistency

- **Admissibility**
  - $h(n) \leq \text{true cost from } n \text{ to goal}$

- **Consistency (Monotonicity)**
  - $h(n) \leq c(n, n') + h(n')$
  - (Implies admissibility. Why?)
    - $h(n) \leq c(n, n') + [c(n', n'') + h(n'')]$

- **A* optimal if:**
  - Tree: $h()$ is admissible
  - Graph search: $h()$ is consistent

Consistency

- **Where is this function non-consistent?**
  - **Consistency property:** $h(n) \leq c(n, a, n') + h(n')$
Consistency and Tree Search

- Why *don’t* we need consistency for tree search?

Constructing Admissible Heuristics

- Relax constraints
- Sub-problems
- Pattern databases
Generating Heuristics by Relaxation

Calculate exact distance for relaxed version of problem

Allow tile to be moved to any location
h1: #misplaced tiles

Allow move to adjacent square even if occupied
h2: Manhattan distance

Generating heuristics with sub-problems

- $h_3(n) =$ How many moves to get tiles 1-4 into the correct position?

- $h_4(n) =$ How many moves to get tiles 7-8 into the correct position?
Pattern Database

- Stanford Parking Planner
  - Precompute distance to adjacent cells assuming no obstacles

Admissible Heuristics

- Which heuristic is better?
  - The one that produces the larger values.

- Are there admissible $h_1, h_2$ such that $h_1(n_i) > h_2(n_i)$ and $h_1(n_j) < h_2(n_j)$?

- Domination
  - $h_1(n) \geq h_2(n)$ for all $n$. 
Combining Multiple Heuristics

- Suppose we have multiple admissible heuristics. What is the optimal combination of them?
- What if the heuristics are disjoint? I.e., progress on one heuristic can not affect progress of another heuristic

Inadmissible Heuristics

- Learning heuristics based on features
  - $H(n) = Ax(n) + By(n)$
  - Where A,B are parameters fit to observed data.
- Are these useful?
A*: Not a panacea

- Avoiding exponential search size requires heuristic error to grow as \( \log(\text{cost}) \).
- This is hard to do: error often proportional to cost.
  - Consider: Straight-line distance?

- Otherwise, memory/CPU are \( O(b^n) \)
  - Memory will generally be the limiting problem
  - Sound familiar?
Recursive Best First Search (RBFS)

- RBFS is a scheme to reduce the memory requirements.

- Each node knows the f() of the best alternative path from one of its ancestors.
  - If the current f() value exceeds this limit, we unwind back to the common ancestor, then re-expands the tree down the alternate path.

- Two (or more) paths can wrestle control back and forth:
  - Constantly re-expanding nodes

- Space:
  - $O(d)$

IDA*

- IDA* is a scheme to reduce the memory requirements.

- Virtually identical to IDS:
  - Instead of Depth-Limited Search, use f()-limited search
  - Upon failure, f()-limit is increased to the lowest f() value that was previously pruned.

- Time:
  - $O(b^{C^*/e})$

- Space:
  - $O(bd)$
SMA*

- RBFS and IDA* suffer from using too little memory.

- Idea: Remember as much of the search tree as we can. (Delay the onset of thrashing as long as we can!)
  - 1. When the queue is not too big
    - Expand the leaf in the search tree with the minimum f()
  - 2. When memory runs out:
    - Find the leaf in the search tree whose f() is the worst and remove it.
    - Back up its f() to its parent. Note that the parent might now become a leaf

Next time

- Local Search