Informed Search

CS311
David Kauchak
Spring 2013

Some material borrowed from:
Sara Owsley Sood and others

CMU Snake Robot

http://www-cgi.cs.cmu.edu/afs/cs.cmu.edu/Web/People/biorobotics/projects/modsnake/index.html

Administrative

• Assignment 1 was due before class
  – how’d it go?
  – come talk to me earlier than later!
• Written problems?
• Assignment 2
  – Mancala (game playing)
  – will be out later today or tomorrow
  – < 2 weeks to complete
  – Can work with a partner
  – tournament!
• Lectures slides posted on the course web page

Uninformed search strategies

Uninformed search strategies use only the information available in the problem definition

– Breadth-first search
– Uniform-cost search
– Depth-first search
– Depth-limited search
– Iterative deepening search
Summary of algorithms

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Uniform-Cost</th>
<th>Depth-Limited</th>
<th>Depth-First</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>Time</td>
<td>$O(b^{d+1})$</td>
<td>$O(b^{d+1})$</td>
<td>$O(h)$</td>
<td>$O(h)$</td>
<td>$O(h)$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(b^{d+1})$</td>
<td>$O(b^{d+1})$</td>
<td>$O(bh)$</td>
<td>$O(bh)$</td>
<td>$O(bh)$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

A few subtleties...

What is the difference between a state and a node?

Be Careful! states vs. nodes

A state is a (representation of) a physical configuration.

A node is a data structure constituting part of a search tree includes state, parent node, action, path cost $g(x)$, depth.

Repeated states

What is the impact of repeated states?

def treeSearch(start):
    add start to the frontier
    while frontier isn’t empty:
        get the next node from the frontier
        if node contains goal state:
            return solution
        else:
            expand node and add resulting nodes to frontier
Can make problems seem harder

Can make problems seem harder

What will this look like for treeSearch?

Solution?

Graph search

Keep track of nodes that have been visited (explored)

Only add nodes to the frontier if their state has not been seen before

```
def graphSearch(start):
    add start to the frontier
    set explored to empty
    while frontier isn’t empty:
        get the next node from the frontier
        if node contains goal state:
            return solution
        else:
            add node to explored set
            expand node and add resulting nodes to frontier,
            if they are not in frontier or explored
```

Graph search implications?

We’re keeping track of all of the states that we’ve previously seen

For problems with lots of repeated states, this is a huge time savings

The tradeoff is that we blow-up the memory usage

  - Space graphDFS?
    - O(b^m)

Something to think about, but in practice, we often just use the tree approach

8-puzzle revisited

The average depth of a solution for an 8-puzzle is 22 moves

What do you think the average branching factor is?

  - ~3 (center square has 4 options, corners have 2 and edges have 3)

An exhaustive search would require ~3^{22} = 3.1 x 10^{10} states

  - BFS: 10 terabytes of memory
  - DFS: 8 hours (assuming one million nodes/second)
  - IDS: ~9 hours

Can we do better?

1 3 8
4 7
6 5 2
from: Claremont to: Rowland Heights

What would the search algorithms do?

from: Claremont to: Rowland Heights

DFS

from: Claremont to: Rowland Heights

BFS and IDS

We’d like to bias search towards the actual solution

Ideas?
Informed search

Order the frontier based on some knowledge of the world that estimates how "good" a state is
- \( f(n) \) is called an evaluation function

Best-first search
- rank the frontier based on \( f(n) \)
- take the most desirable state in the frontier first
- different approaches depending on how we define \( f(n) \)

```python
def treeSearch(start):
    add start to the frontier
    while frontier isn't empty:
        get the next node from the frontier
        if node contains goal state:
            return solution
        else:
            expand node and add resulting nodes to frontier
```

Heuristic function: \( h(n) \)

An estimate of how close the node is to a goal
Uses domain-specific knowledge

Examples
- Map path finding?
  - straight-line distance from the node to the goal ("as the crow flies")
- 8-puzzle?
  - how many tiles are out of place
- Missionaries and cannibals?
  - number of people on the starting bank

Greedy best-first search

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

rank nodes by how close we think they are to the goal

Arad to Bucharest
Greedy best-first search

Is this right/optimal?
Problems with greedy best-first search

Time?
– \(O(b^m)\) – but can be much faster

Space?
– \(O(b^m)\) – have to keep them in memory to rank

Complete?
– Graph search, yes
– Tree search, no
Problems with greedy best-first search

Optimal?
– no, as we just saw in the map example

Sometimes it’s too greedy

Shortest path from a to g?
What is the problem?

A* search

Idea:
– don’t expand paths that are already expensive
– take into account the path cost!

\[ f(n) = g(n) + h(n) \]
– \( g(n) \) is the path cost so far
– \( h(n) \) is our estimate of the cost to the goal

\( f(n) \) is our estimate of the total path cost to the goal through \( n \)
A* search

Admissible heuristics

A heuristic function is *admissible* if it never overestimates
- if $h^*(n)$ is the actual distance to the goal
- if $h(n) \leq h^*(n)$

An admissible heuristic is optimistic (it always thinks the goal is closer than it actually is)

Is the straight-line distance admissible?

A* properties

- **Time?**
  - depends on heuristic, but generally exponential
- **Space?**
  - exponential (have to keep all the nodes in memory/frontier)
- **Complete?**
  - YES
- **Optimal?**
  - YES, if the heuristic is admissible
    - Why?
      - If we could overestimate, then we could find (that is remove from the queue) a goal node that was suboptimal because our estimate for the optimal goal was too large
A point of technicality

Technically if the heuristic isn’t admissible, then the search algorithm that uses \( f(n) = g(n) + h(n) \) is called “Algorithm A”

A* algorithm requires that the heuristic is admissible

That said, you’ll often hear the later referred to as A*

Algorithm A is not optimal

Admissible heuristics

8-puzzle

- \( h_1(n) = \) number of misplaced tiles?
- \( h_2(n) = \) manhattan distance?

\[
\begin{array}{c|c|c}
1 & 3 & 8 \\
4 & 7 & \\
6 & 5 & 2 \\
\end{array}
\quad
\begin{array}{c|c|c}
1 & 2 & 5 \\
4 & 8 & \\
3 & 6 & 7 \\
\end{array}
\quad
\begin{array}{c|c|c}
1 & 2 \\
3 & 4 & 5 \\
6 & 7 & 8 \\
\end{array}
\]

\( h_1 = 7 \)
\( h_2 = 12 \)

\( h_1 = 8 \)
\( h_2 = 8 \)

Which is better?
Dominance

Given two admissible heuristic functions
- if \( h_i(n) \geq h_j(n) \) for all \( n \)
- then \( h_i(n) \) dominates \( h_j(n) \)

A dominant function is always better. Why?
- It always gives a better (i.e., closer) estimate to the actual path cost, without going over

What about?
- \( h_1(n) \) = number of misplaced tiles
- \( h_2(n) \) = manhattan distance

\[ h_2(n) \text{ dominates } h_1(n) \]

Relaxed problems

A problem with fewer restrictions on the actions is called a relaxed problem.

The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.

8-puzzle: relaxed problems?
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then \( h_1(n) \) gives the shortest solution.
- If the rules are relaxed so that a tile can move to any adjacent square, then \( h_2(n) \) gives the shortest solution.

Combining heuristics

Sometimes, we have multiple admissible heuristics, but none dominates.

What then?
- We can take the max of all the heuristics!

Why?
- Since they're all admissible, we know none overestimate.
- Taking the max gives us a closer/better estimate.
- Overall, a better heuristic function.
Using A* in Planning

Creating Heuristics

- 8-Puzzle
- Missionaries and Cannibals
- Remove 5 Sticks
- N-Queens
- Water Jug Problem
- Route Planning