EECS 391: Introduction to AI

Soumya Ray

Website: http://engr.case.edu/ray_soumya/eecs391_sp16/
Email: sray@case.edu
Office: Olin 516
Office hours: M 12:30-2pm or by appointment
Announcements

• HW1 out
• Programming 1 out later today
• Monday 25\textsuperscript{th} office hours 1-2pm
• Start reading Chapter 4 of textbook
Today

• Goal-Directed Search (Chapter 3)
Overview

• We saw that an intelligent agent needs to be flexible

• So we can’t give it solutions to specific problems, we need to give it *problem solving strategies*

• The most basic general purpose problem solving strategy is called “Goal-Directed Search”
Example: Search Problem

Time taken to go from C to D

Start here

Want to go here
Exact Solution

• If the search space is very small, we could easily find the exact solution
  – How?
  – Then we wouldn’t need to search

• In real life, though, search is used to solve problems of very large size, for which the search space cannot be stored
Problem Setup

• Our agent is currently in some state of the world, generally represented atomically
  – Call this the *initial state*

• It wants to get to a different state of the world
  – Call this the *goal state*
  – In general, the desired target may be defined by a logical predicate (*goal test*) that encompasses a set of goal states
  – In this case the agent wants to get to *any* goal state satisfying the goal test
Problem Setup (2)

• To change the state of the world, the agent has actions
  – These will be called “search operators”
  – Also called “successor functions”, same thing

• The search operators applied successively to the initial state generate a sequence of states
  – This is the “search space”
Problem Setup (3)

• Each search operator has an associated cost

• The search objective is to discover a sequence of operators that takes the agent from the initial state to any goal state with minimum cost
“Offline” Problem Solving

• The entire search operation is part of the “agent function”---it is internal to the agent

• After the search is complete and the solution found, the agent can apply them to the world to execute the solution
## Constraints on Environment

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Type</th>
<th>Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observability</td>
<td>Full*</td>
<td>Need to track the state</td>
</tr>
<tr>
<td>Static</td>
<td>Yes</td>
<td>State should not change while searching</td>
</tr>
<tr>
<td>Deterministic</td>
<td>Yes*</td>
<td>Need to predict next state after action</td>
</tr>
<tr>
<td>Discrete</td>
<td>Maybe</td>
<td>Algorithm-dependent</td>
</tr>
</tbody>
</table>

*: possible to relax it, assume for now
When to use Goal-Directed Search

• Environment criteria are satisfied
  – Agent can distinguish different states
    • In particular, goal situations from non-goal situations
    • It can tell how the state will change if it takes an action (e.g. “state 5 will become state 43 if I go left”)

• It wants the least cost path to get to the goal

• It does not possess a structured state representation and/or it is not feasible to acquire one
  – (also possible to relax, assume for now)
Checklist

- Initial state
- Goal Test
- Search Operator (agent “action”/successor fn)
- Operator/Step Cost
- Objective: Find a sequence of actions that get from initial state to goal with minimum cost
Route Finding

- Suppose an agent wants to get from location A to location B
  - Initial state?
  - Goal Test?
  - Search Operators?
  - Operator/Step Cost?
Game Playing

- Solving puzzles

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>
Nodes are world states; edges are operators. Edges could be directed or undirected.
Basic Steps of Search Algorithms

- Add initial state to open list (list of unvisited states)
- While open list is not empty
  - Remove node from the open list
  - If this is the goal, solution found, return path
  - Else
    - Find the successors of this node ("expanding a state")
    - Add the current state to the list of visited (expanded) states (closed list)
    - Add the new states to the open list (if they do not appear in the closed list)
      - State could already be in open list, set parent pointer correctly (based on minimum path cost)

Search algorithms differ in order of node removal.
Blind Search 1: Breadth First Search

• Shallowest (lowest depth) nodes on open list are expanded first

• First expand successors of initial state, then their successors, etc

• Usually, the open list will be implemented via a queue
Example

Merge in open list, update \textit{min} cost & \textit{min} depth
Uniform Cost Search

• Expand the node on the open list with the lowest path cost
  – otherwise same as BFS

• Can be implemented using a priority queue
Depth First Search

• Expands deepest nodes on open list first

• Can be implemented using a stack
Example

Merge in open list, update **min** cost & **max** depth
Iterative Deepening (ID-DFS)

• DFS may go down long, useless paths
  – We can parameterize it with a depth limit
  – If limit is reached, it will not expand further

• We can iteratively increase the depth parameter
  – Start with zero
  – If DFS with current depth fails, increment depth and restart
Bidirectional Search

• Why not search from both the initial state and the goal state?

• Idea: run simultaneous searches, checking for intersections between the open lists
  – If nonempty, path found
  – If using BFS at each end, and goal depth is \( d \), time and space complexity will be reduced to \( O(b^{d/2}) \)
But...

• What if the goal is defined in terms of a predicate?
  – Could have lots of satisfying states, or could be hard to satisfy (SAT problem)

• Also constrains the operators
  – Note when searching from goal, need to find “predecessors”---could be tricky!
    • “All states that led to this checkmate configuration”
Search Algorithm Analysis

• As it searches, the agent generates a “search tree”
• Each node in the search tree represents some state in the search space, with extra bookkeeping information
  – The parent node
  – The action that was applied at the parent
  – Path cost
  – Depth

• NOTE: The search tree is only meant to visualize the flow of computation. It does not correspond to a data structure you would maintain in practice.
Different search algorithms can be compared using characteristics of the search tree they generate.
Characteristics of the Search Tree

• Branching factor \( b \)
  – Maximum number of successors of any node

• Goal depth \( d \)
  – Depth of the shallowest goal in the search tree

• Max Path Length \( m \)
  – Maximum length of a path in the search tree
Algorithm Performance

• Completeness
  – Algorithm always finds a solution if it exists?

• Optimality
  – Algorithm always finds minimum cost solution?

• Time Complexity
  – Time taken to find a solution?

• Space Complexity
  – Memory required to find a solution?
BFS Performance

• Complete? • Yes

• Optimal? • Sometimes (when?)

• Time Complexity? • \(O(b^{d+1})\)

• Space Complexity? • \(O(b^{d+1})\)
DFS Performance

• Complete?
  • Yes, assuming no infinite paths

• Optimal?
  • Sometimes

• Time Complexity?
  • $O(b^m)$

• Space Complexity?
  • $O(b^m)$
Summary

• We learned about:
  – When to use search (environment constraints etc)
  – How to set up a search problem
  – Uninformed/blind search algorithms
    • BFS, UCS, DFS, ID-DFS
  – How to analyze search engine performance
  – Difference between uninformed and informed search