EECS 391: Introduction to AI

Soumya Ray

Website: http://engr.case.edu/ray_soumya/eecs391_sp15/
Email: sray@case.edu
Office: Olin 516
Office hours: F 11:30-2pm or by appointment
Announcements

- HW due next Thursday
Today

• Goal-Directed Search (Chapter 3)
Environment Type

• We’ll assume the environment is:
  – Fully observable (to track the state)
  – Static (shouldn’t change while agent is searching)
  – Deterministic (agent needs to be able to precisely predict states resulting after each action)

• Some search algorithms also need discrete environments
Problem Setup

• Our agent is currently in some state of the world
  – 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
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
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)
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 min cost & 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 space
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?
  • $O(b^m)$
Uninformed vs. Informed Search

• Previous search methods use various methods to pick which node to expand

• But which node would we *really* like to expand, path costs being equal?
  – The one that is *really the closest to the goal*

• But we don’t know this
  – If we did, wouldn’t need to search
“Informed” Search

• Uninformed or Blind Search
  – These algorithms only use the information in the problem setup to find a solution

• Informed or Heuristic Search
  – These algorithms also use an extra function, a search heuristic, to find a solution
  – The search heuristic is not part of the problem definition---it is up to the agent designer to specify it (some recent work on learning the heuristic)