EECS 391: Introduction to AI

Soumya Ray
http://vorlon.case.edu/~sray/eecs391_sp12/index.html
Email: sray@case.edu
Office: Olin 516

Office hours: Tue 2:30-4 or by appointment

Announcements

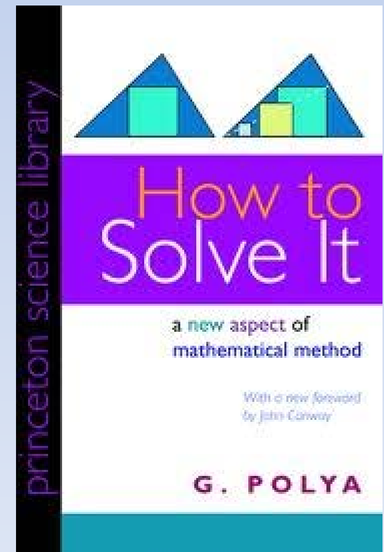
- TA: Feng Cao
  - Office: Olin 806
  - Office hours: Th 2:30-4
- HW1 and PA1 posted last week
- SimpleRTS tutorial sessions here, Thursday/Friday 4:30pm

Today

- Solving Problems Using Search (Chapter 3)
  - Idea
  - How to set up the problem
  - Basic algorithms
  - Characteristics of algorithms

Overview

- We saw last time 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 “Search”



When to use Search

- The agent *fully perceives* the current state of the world and wants to achieve a certain goal
  – It 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

“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
  – World needs to be static

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

Examples

- Route Finding
  - Suppose an agent wants to get from location A to location B
  - Same techniques used by mapping software e.g. Google Maps
- Solving puzzles
  - 8-puzzle
  - Sudoku

Search

- A fundamental technique in AI
- A strategy when the agent has limited/no idea about the detailed structure of a problem to allow more complex reasoning
- Very easy to implement
- Will show up often when we study the more complex algorithms later

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 transitively to the initial state generate a sequence of states
  - Call this 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

Route Finding

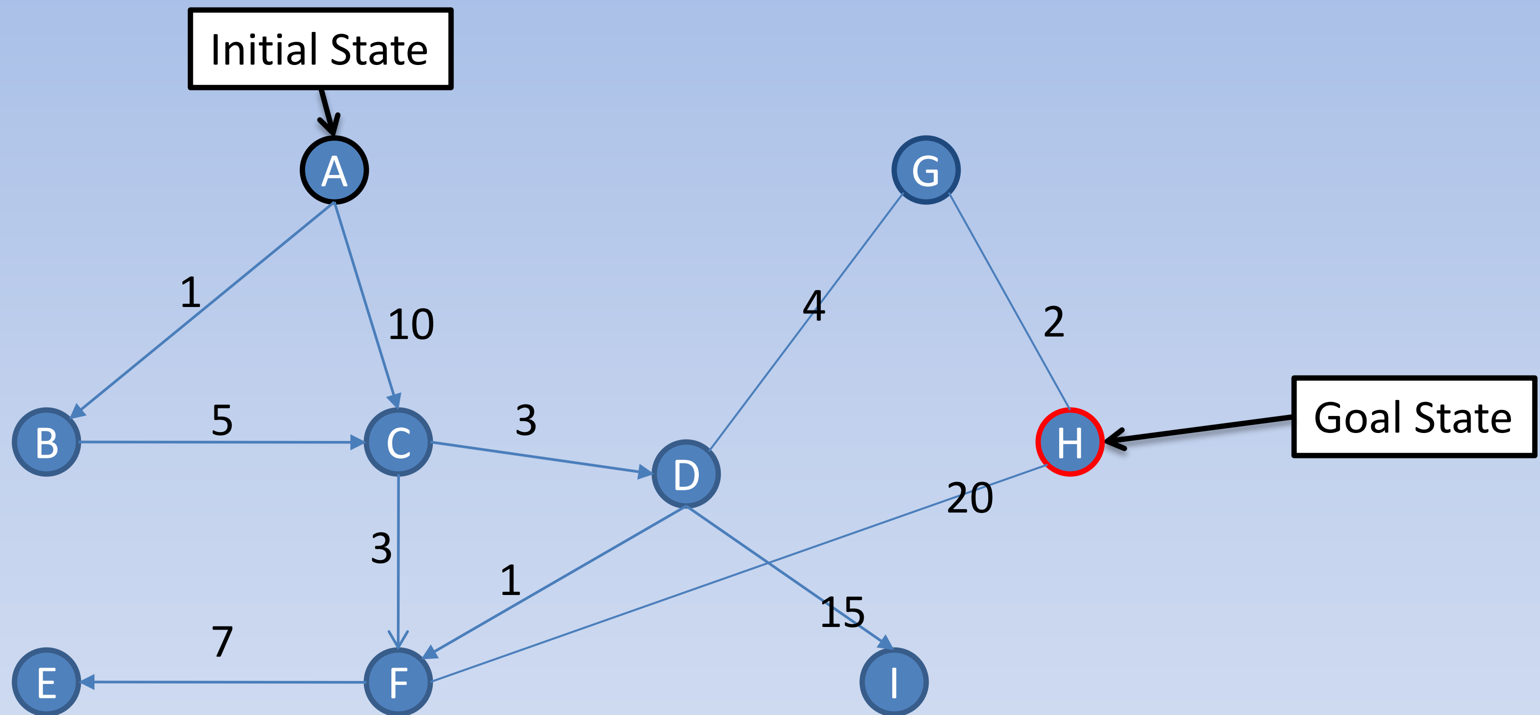
- Suppose an agent wants to get from location A to location B
  - Initial state?
  - Goal Test?
  - Search Operators?
  - Operator/Step Cost?
- Same techniques used by mapping software e.g. Google Maps

Game Playing

- Solving puzzles

	3	5
7	8	1
2	6	4

Example: Search Problem



Nodes are world states; edges are operators. Edges could be directed or undirected.

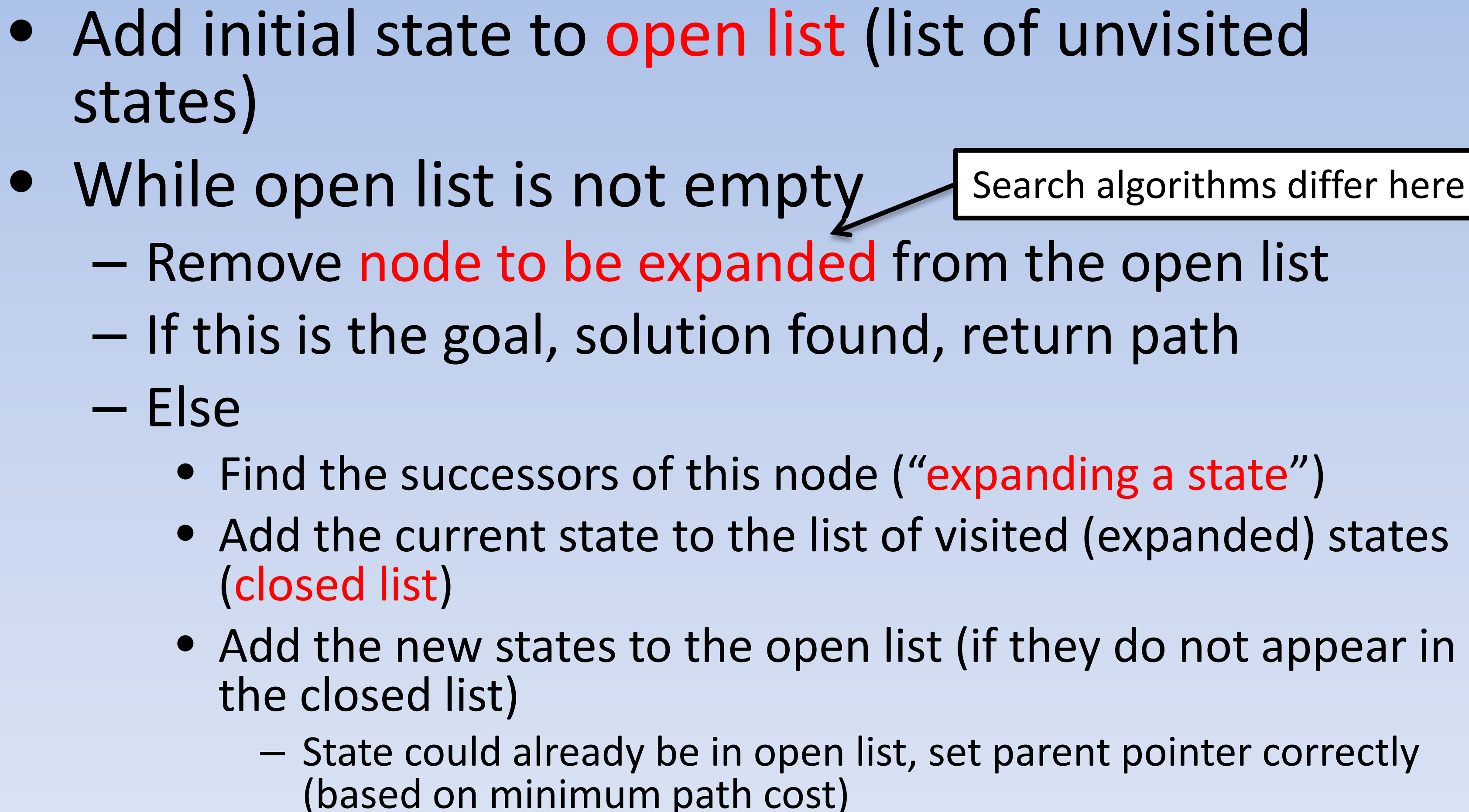
Exact Solution

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

Dijkstra's algorithm

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

Basic Steps of Search Algorithms



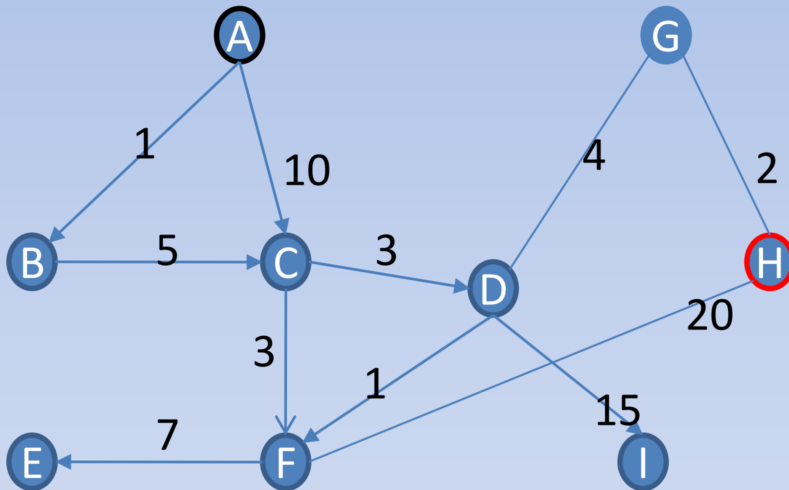
- Add initial state to open list (list of unvisited states)
- While open list is not empty
  - Remove node to be expanded from the open list
  - If this is the goal, solution found, return path
  - Else
    - Find the successors of this node ("expanding a state")
    - Add the current state to the list 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)

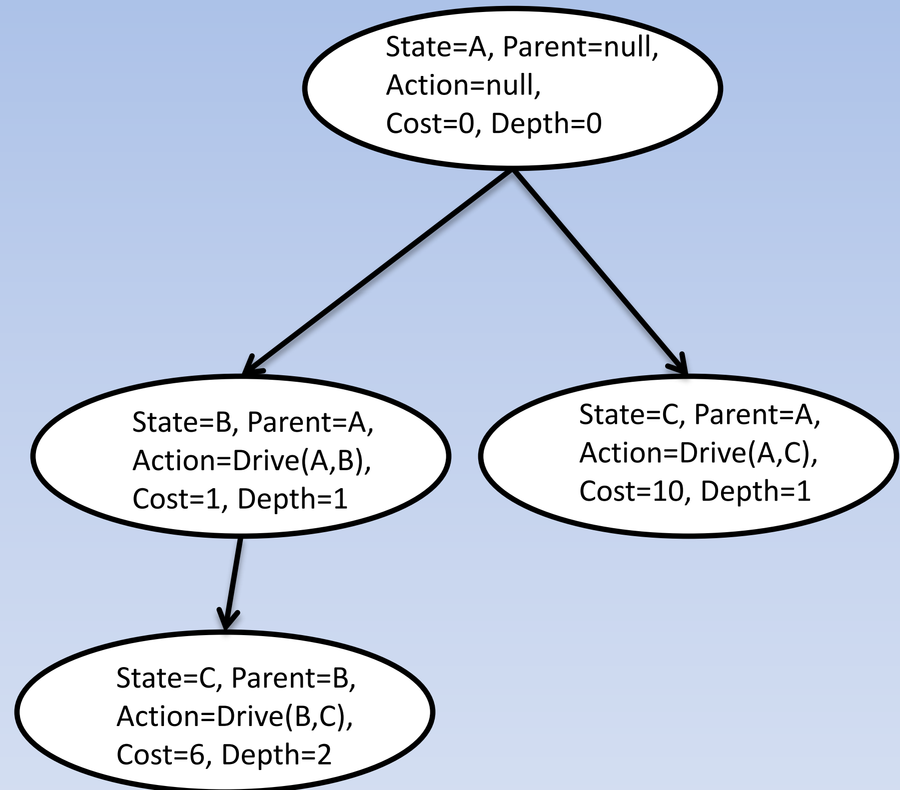
Search Tree

- 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 $g(n)$
  - 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.

Example: Search Tree





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
- Fringe
  - Current set of nodes on our "unvisited" list
  - These are (a subset of) the leaf nodes in the current search tree

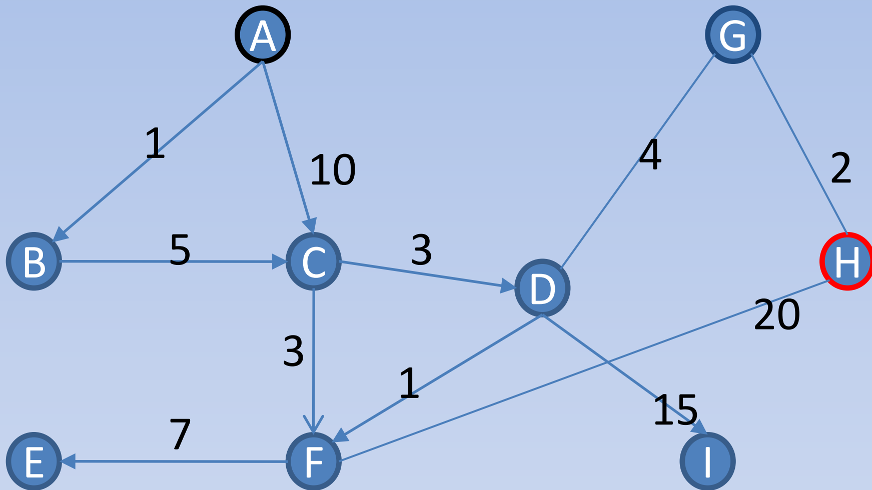
Search Algorithm Classes

- 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)

Blind Search 1: Breadth First Search

- Shallowest 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

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?
- Optimal?
- Time Complexity?
- Space Complexity?

- Yes
- Sometimes (when?)
- $O(b^{d+1})$
- $O(b^{d+1})$

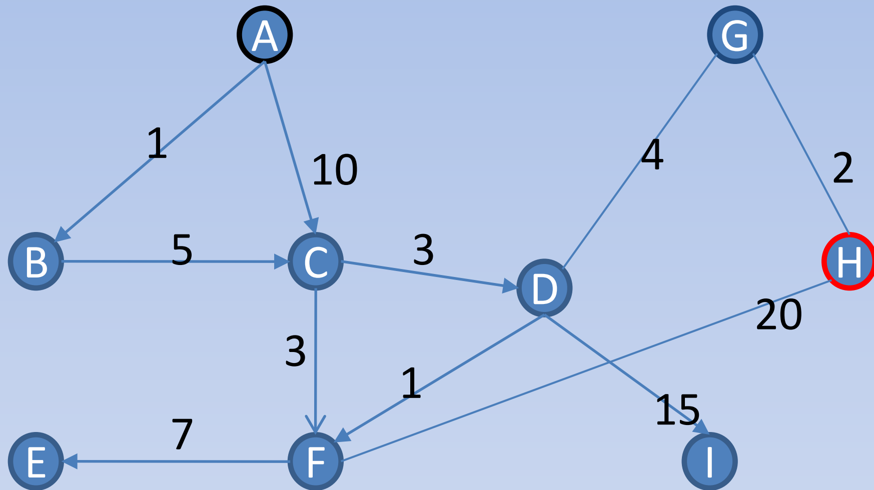
Uniform Cost Search

- BFS ignores the path costs, seems silly
- Uniform Cost search expands 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

- Deepest nodes on open list are expanded first
- Keep going until goal found or nodes have no (unvisited) successors, then backtrack
- Can be implemented using a stack

Example



A
B C
C
D F
F G I
E H

Merge in open list, update **min** cost & **max** depth

DFS Performance

- Complete?
- Optimal?
- Time Complexity?
- Space Complexity?

- Yes, assuming no infinite paths
- Sometimes
- $O(b^m)$
- $O(b^m)$

Depth Limited Search (DL-DFS)

- DFS runs into problems when m is large or infinite
- To prevent it going down long, useless paths, we can set a depth limit
- If limit is reached, returns no solution

Iterative Deepening (ID-DFS)

- Depth Limited Search may fail if depth limit is too low

- To resolve this, we can iteratively increase the depth parameter
  - Start with zero
  - If DL-DFS with current depth fails, increment depth and restart

Why is this a good idea?

- ID-DFS seems wasteful
- The key is in the observation that the number of nodes in the search tree is an exponential function of depth

- Suppose shallowest goal depth is d. Then:

$$N(ID\text{-}DFS) = d \cdot b + (d-1)b^2 + \ldots + b^d = O(b^d)$$

$$N(BFS) = b + b^2 + \ldots + (b^{d+1} - b) = O(b^{d+1})$$

ID-DFS Performance

- Complete?
- Optimal?
- Time Complexity?
- Space Complexity?

- Yes
- Sometimes
- $O(b^d)$
- $O(b^d)$

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"

Uninformed vs. Informed Search

- Uninformed 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

Search Heuristic

- Suppose we had a function that *estimated* the distance to the goal for a node n
  - Call it $h(n)$ (remember path cost is denoted $g(n)$)

- This is called a search heuristic
  - This function depends on the specific problem and is not externally specified; has to be designed by us or possibly learned by the agent
  - Informed search algorithms use a search heuristic in addition to problem specification

Summary

- We learned about:
  - Setting up a search problem
  - Search Trees and their characteristics
  - Breadth First, Uniform Cost, Depth First Search
  - Depth Limited and Iterative Deepening Search
  - Bidirectional Search
  - Intro to informed search
- Next: Informed Search