Search

Burr H. Settles
CS-540, UW-Madison
www.cs.wisc.edu/~cs540-1
Summer 2003

1

Announcements

- Homework #1 is out today (6/18) and will be due next Friday (6/27)
- Please hand in Homework #0 (the info card/sheet about *you*) as soon as you can
- Read Chapter 4 in *AI: A Modern Approach* for next time

2

Clarifications from Last Time

- We talked about the features of agents (situatedness, autonomy, etc.) and of environments (discrete/continuous, deterministic/stochastic, etc.)
- These are *formalisms* that we use to help in modeling a problem:
  - We as the programmers get to draw these lines!
  - Can be sometimes arbitrary
  - Generally used at a single level of abstraction

3

Recap of Last Time

- The different types of agent programs:
  - Lookup table
  - Simple reflex agent
  - Model-based reflex agent
  - Goal-based agent
  - Utility-based agent

} ***Let's talk about these***

4

Goal-Based Agents

- We are interested in how we can design goal-based agents to solve problems
- There are three major questions to consider:
  - What goal does the agent need to achieve?
  - What knowledge does the agent need?
  - What actions does the agent need to do?

5

Goal-Based Agents

- ✱ *What goal does the agent need to achieve?*
- How do you describe the goal?
  - A situation to be reached
  - The answer to a question
  - A set of properties to be acquired
- How do you know when the goal is reached?
  - With a goal test that defines what it means to have achieved/satisfied the goal

6

Goal-Based Agents

- ✱ *What knowledge does the agent need?*
- The information needs to be:
  - Sufficient to describe everything relevant to reaching the goal
  - Adequate to describe the world state/situation
- We'll use a closed world assumption:
  - All necessary information about a problem domain is observable in each percept so that each state is a complete description of the world
    - *i.e.* There is never any hidden information

7

Goal-Based Agents

- ✱ *What actions does the agent need to do?*
- Given:
  - An set of available actions
  - A description of the current state of the world
- Determine:
  - Which actions can be applied (those applicable/legal?)
  - What the exact state of the world will be (or likely be) after an action is performed in the current state
    - No history information needed to compute the new world state
  - What is the action is likely to lead me to my goal?

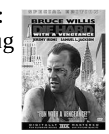
8

Motivation

- We want to design a goal-based agent to solve a puzzle called the water jug problem
- What better motivation is there than to keep from *being blown up!?!?!*

9

Case Study: Die Hard III



- Imagine you are given two containers: a 3-gallon water jug, and a 4-gallon jug
- Initially both jugs are *empty*
- You have 3 actions available to you:
  - Fill a jug completely
  - Dump a jug completely
  - Pour as much water as possible from one of the jugs into the other
- You must end up with a jug having exactly 2 gallons of water in order to disarm the bomb!!

10

How Can a Machine Do This?

- We want our agent to search for a sequence of actions that lead to a solution
- To do this we formalize the problem as a search task, considering our three questions:
  - What *goal* does the agent need to achieve?
  - What *knowledge* does the agent need?
  - What *actions* does the agent need to do?

11

Formalizing a Search

- State: specific description of the world
  - Combination of jug volumes
- State space: set of all possible states in our problem environment
  - $4 \times 5 = 20$ water jug states
- Search node: data structure where the state information is stored for the search
- Search tree: directed graph $G = (V,E)$
  - V is a set of nodes (states)
  - E is a set of edges (actions turning one state to another)

12

Formalizing a Search

- Initial state: designated start state
  - 2 empty jugs
- Successor states: collection of states generated by applying actions to a particular state
- Goal state: state which satisfies to the object of our search task
  - One jug with 2 gallons
- Goal test: way of deciding a goal state

13

Formalizing a Search

- Open list: list of states which are *waiting* to be considered in the search
- Closed list: list of states which have *already* been considered in the search
- Solution: path from the initial state to a goal

14

Water Jug Search Setup

- State: ordered pair "AB"
  - A is the 3 gallon jug; B is the 4 gallon jug
- State space: our 14 possible states
- Edges: generated by our six possible actions:
  - fill(A), dump(A), pour(A,B)
  - fill(B), dump(B), pour(B,A)
- Initial state: the state "00"
- Goal state: any state "*n*2" or "2*n*"
  - Where *n* can be any number

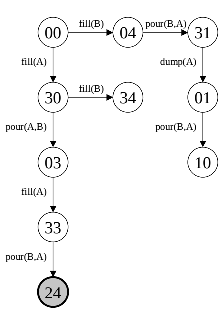
15

Water Jug Search Algorithm

```
OPEN = { 00 }     // states we're considering
CLOSED = { }      // states we've already seen
while OPEN is not empty {
   X = state removed from OPEN
   if X is (n2) or (2n) then
        return solution
   else {
        add X to CLOSED
        generate successor states via actions on X
        ignore successors already in OPEN or CLOSED
        add successors to OPEN
   }
}
return FAILURE    // no solution found
```

16

Water Jug Search Tree



State	Open	Closed
--	00	--
00	30, 04	00
30	04, 03, 34	00, 30
04	03, 34, 31	00, 30, 04
03	34, 31, 33	00, 30, 04, 03
34	31, 33	00, 30, 04, 03, 34
31	33, 01	00, 30, 04, 03, 34, 31
33	01, 24	00, 30, 04, 03, 34, 31, 33
01	24, 10	00, 30, 04, 03, 34, 31, 33, 01
24	**Goal!!**	

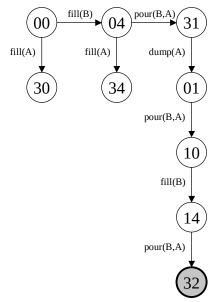
17

The Open List

- In this example, we used a FIFO scheme (or a queue) to track states in our open list. This is called a breadth-first search (BFS)
- There are several other ways to manage states in the open list… we call each method a different search strategy
- If we use a LIFO scheme (a stack), we perform what is called depth-first search (DFS)

18

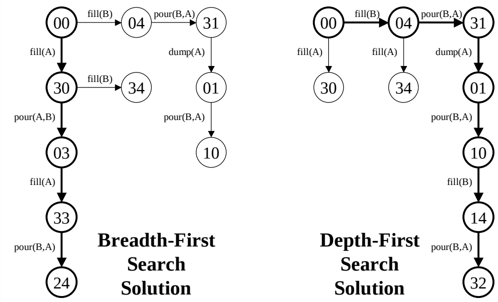
Water Jug Search with DFS



State	Open	Closed
--	00	--
00	04, 30	00
04	31, 34, 30	00, 04
31	01, 34, 30	00, 04, 31
01	10, 34, 30	00, 04, 31, 01
10	14, 34, 30	00, 04, 31, 01, 10
14	32, 34, 30	00, 04, 31, 01, 10, 14
32	**Goal!!**	

19

Water Jug Search Comparison



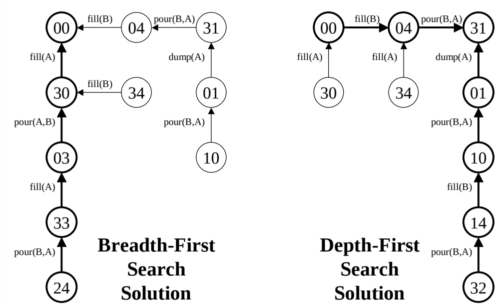
20

A Note on Finding Solutions

- In this problem, our solution is a sequence of actions, or path from the initial state to a goal state
- Search nodes are irresponsible, because they don't *keep track of their children!*
  - In practice, we make each node *remember its parent* instead, then backtrack from the goal to the initial state

21

Responsible Children



22

Utility-Based Agents

- Recall that utility-based agents are goal-based agents that can determine which solutions are *best*
- What solution is best in the water jug problem example?
- What if actions have costs?

23

Searching with Costs

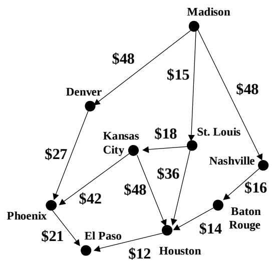
- For the water jug problem, the costs to fill, dump, or pour a jug are *all the same*
- Uniform-cost search (UCS) is a strategy we use if there are costs associated with actions (edges), and we want the *least expensive* solution to a goal
- We use a *priority queue* for the open list, where states are ranked by the total cost of the path from the initial state

24

Example with Costs

Suppose we want to travel by train to the Armadillo Convention in El Paso, and we want to find the least expensive series of tickets from Madison

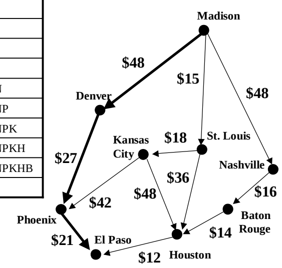
Cities are our *states*, and *tickets* are *actions*. Let's try comparing BFS, DFS, and UCS strategies…



BFS Solution

State	Open	Closed
--	M	--
M	DSN	M
D	SNP	MD
S	NPKH	MDS
N	PKHB	MDSN
P	KHBE	MDSNP
K	HBE	MDSNPK
H	BE	MDSNPKH
B	E	MDSNPKHB
E	**Goal!!**	

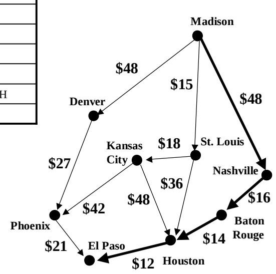
Total Cost: $96



DFS Solution

State	Open	Closed
--	M	--
M	NSD	M
N	BSD	MN
B	HSD	MNB
H	ESD	MDBH
E	**Goal!!**	

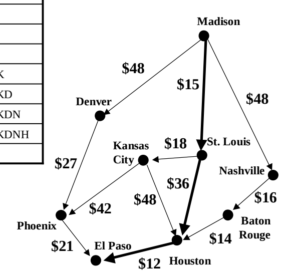
Total Cost: $90



UCS Solution

State	Open	Closed
--	M:0	--
M	S:15, D:48, N:48	M
S	K:33, D:48, N:48, H:51	MS
K	D:48, N:48, H:51, P:75	MSK
D	N:48, H:51, P:75	MSKD
N	H:51, B:64, P:75	MSKDN
H	E:63, B:64, P:75	MSKDNH
E	**Goal!!**	

Total Cost: $63

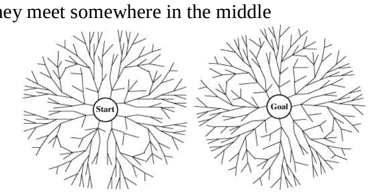


Other Search Strategies

- Depth-limited search (DLS)
  - If memory space is a major concern, one can conduct a simple DFS with a fixed depth limit *l*
- Iterative deepening search (IDS)
  - Conduct a depth-limited search at increasing depth limits until a solution is found

Other Search Strategies

- Bi-directional search (BDS)
  - If we want to find a particular goal node, we can search from *both ends* of the search space
  - Conducts a BFS from both the start and goal states until they meet somewhere in the middle



Evaluating Search Strategies

- Completeness
  If a solution exists, will it be found?
  - A *complete* algorithm will find a solution
- Optimality
  If a solution is found, is it guaranteed to be the best one? (remember *utility*-based agents)
  - An *optimal* algorithm will find a solution with the minimum cost

31

Evaluating Search Strategies

- Time Complexity
  How long does it take to find a solution?
  - Measured for worst or average case
  - Measured in number of states expanded/tested (*i.e.* the size of the closed list)
- Space Complexity
  How much space is used by the algorithm?
  - Measured in terms of the states generated (*i.e.* the size of the open + closed lists)

32

Evaluating Search Strategies

Strategy	**Depth-first**	**Breadth-first**	**Uniform-cost**	**Depth-limited**	**Iterative deepening**	**Bi-directional**
Complete?	No	Yes	Yes	No	Yes	Yes
Optimal?	No	Yes (if all costs are equal)	Yes	No	Yes (if all costs are equal)	Yes (if all costs are equal)
Time Complexity	$O(b^m)$	$O(b^d)$	$O(b^d)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space Complexity	$O(bm)$	$O(b^d)$	$O(b^d)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$

b is the branching factor of the problem, *d* is the depth of the shallowest solution, *m* is the maximum depth of the search tree, and *l* is the depth limit

33

Uninformed Search

- All the strategies discussed so far are called uninformed search strategies because there is no information provided other than the problem definition
- Next we'll discuss informed or heuristic search strategies that try to speed things up by using domain knowledge to guide the search

34