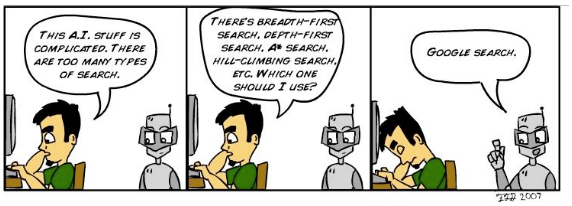


http://blog.lib.umn.edu/torre107/si/pics/superficialintelligence2.jpg

Uninformed Search

CS311
David Kauchak
Spring 2013

Adapted from notes from:
Sara Owsley Sood, Eric Eaton

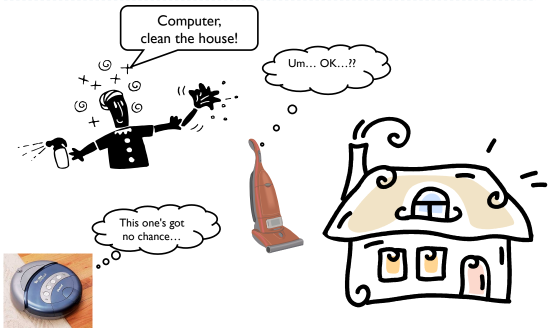
Administrative

- Send me videos!
- Written problems will be posted today
- Programming assignment 1 due before class on Tue.
  - Anyone started?
- My office hours posted:
  - Mon/Wed 1-2:30pm
  - Fri 11am-12
  - and by appointment

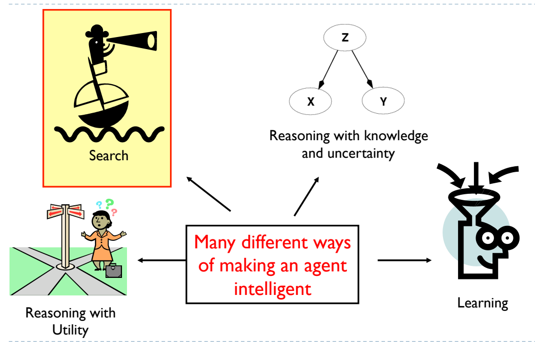
Python

- Whether importing or running, python executes code from the top down
  - Be careful about calling functions before they're defined
- Comments and docstrings
- Don't mix tabs and spaces! (setup your text editor to only use spaces)

How do we make a computer "smart?"



Fundamental problem of AI



Today: search

- Brute force approach
- Very unlikely how humans do it

Think like a human Cognitive Modeling	**Think rationally** Logic-based Systems
Act like a human Turing Test	**Act rationally** Rational Agents

- Enumerate out possibilities in a reasonable order

What is an "agent"?

"anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators"

- Human agent
  - sensors = eyes, ears, etc
  - actuators = hands, legs, mouth, etc
- Software agent
  - sensors = any input devices - keyboard gives it keystrokes, commands over the network, files give it text or data
  - actuators = any output devices - using the screen to display things, pass things over the network, write things to files, etc

search agents

- Search agent is an agent that approaches *problem solving* via *search*
- To accomplish a task:
  1. Formulate problem and goal
  2. Search for a sequence of actions that will lead to the goal (the policy)
  3. Execute the actions one at a time

done offline!

Formulating the problem:

What information does a search agent need to know to plan out a solution?

Formulating the problem:

Initial state: where are we starting from
- **what are the states?**

Actions: what are the possible actions

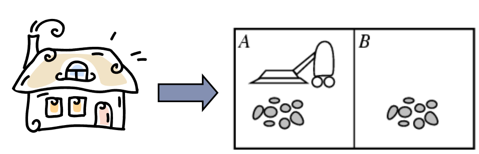
Transition model: aka state-space, mapping from action x state to state

Goal/goal test: what is the end result we're trying to achieve?

Cost: what are the costs of the different actions

Let's start with our vacuum cleaner example

State space
- Just two possible spaces in the house (though this generalizes easily to more)
- each space can either be dirty or clean
- vaccum is in one space at a time



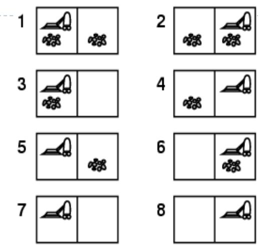
Let's start with our vacuum cleaner example

State space

- Just two possible spaces in the house (though this generalizes easily to more)
- each space can either be dirty or clean
- vaccum is in one space at a time

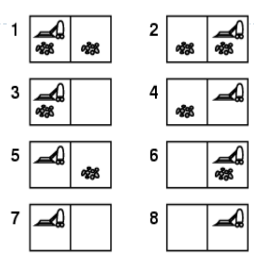
How many states?

Vacuum world



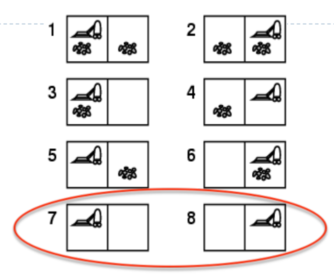
Only 8 states

Vacuum world



goal state(s)?

Vacuum world

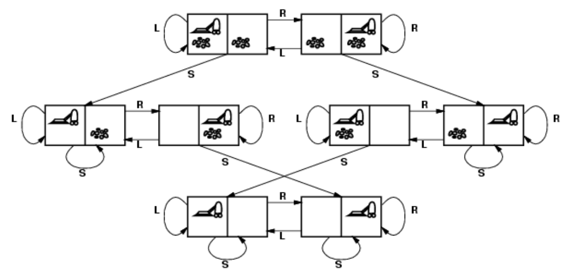


Vacuum world

Actions?

- move left
- move right
- suck
- no-op

Vacuum world: state space/transition model



Problem characteristics

Fully observable vs. partially observable

- do we have access to all of the *relevant* information
- noisy information, inaccurate sensors, missing information

Deterministic vs. non-deterministic (stochastic)

- outcome of an action are not always certain
- probabilistic sometimes

Known/unknown environment

- Do we know a priori what the problem space is like (e.g. do we have a map)

Search problem types

Deterministic, fully observable

- Agent knows exactly which state it will be in
- solution is a sequence of actions

Non-observable → sensorless problem

- Agent may have no idea where it is
- solution is still a sequence

Non-deterministic and/or partially observable → contingency problem

- percepts provide new information about current state
- often interleave search, execution

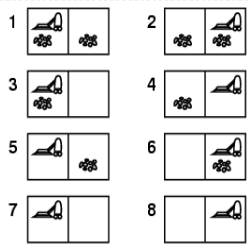
Unknown state space → exploration problem

- this is how roomba works

Example: vacuum world

Deterministic, fully observable

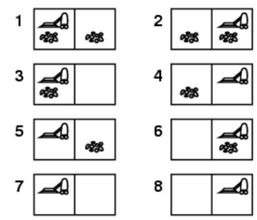
start in #5. Solution?



Example: vacuum world

Sensorless

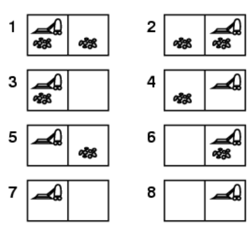
start in *{1,2,3,4,5,6,7,8}*
Solution?



Example: Vacuum world

Non-deterministic and/or partially observable

- Nondeterministic: *Suck* may dirty a clean carpet
- Partially observable: location, dirt at current location.
- Percept: *[L, Clean]*, i.e., start in #5 or #7
  Solution?



Vacuum world

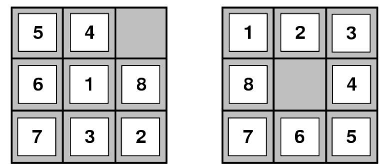
Cost?

Some example problems

Toy problems and micro-worlds
- 8-Puzzle
- Missionaries and Cannibals
- Cryptarithmetic
- Remove 5 Sticks
- Water Jug Problem

Real-world problems

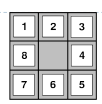
Another problem: 8-Puzzle



Start State Goal State

8-puzzle

goal



Goal State

states?

actions?

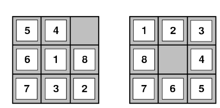
path cost?

8-Puzzle

state:
- all 3 x 3 configurations of the tiles on the board

actions:
- Move Blank Square Left, Right, Up or Down.
- This is a more efficient encoding than moving each of the 8 distinct tiles

path cost:
- +1 for each action



Start State Goal State

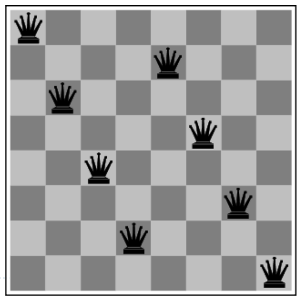
The 8-Queens Problem

State transition:?

Initial State?

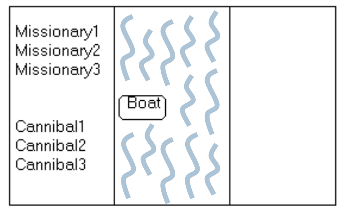
Actions?

Goal: Place eight queens on a chessboard such that no queen attacks any other!



Missionaries and Cannibals

Three missionaries and three cannibals wish to cross the river. They have a small boat that will carry up to two people. Everyone can navigate the boat. If at any time the Cannibals outnumber the Missionaries on either bank of the river, they will eat the Missionaries. Find the smallest number of crossings that will allow everyone to cross the river safely.



Cryptarithmetic

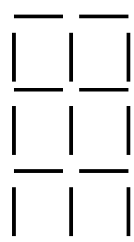
Find an assignment of digits (0, ..., 9) to letters so that a given arithmetic expression is true. examples:

SEND + MORE = MONEY

```
 FORTY     Solution:  29786
+  TEN                  850
+  TEN                  850
 -----                -----
 SIXTY                31486
F=2, O=9, R=7, etc.
```

Remove 5 Sticks

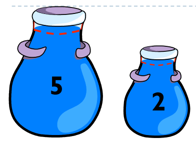
Given the following configuration of sticks, remove exactly 5 sticks in such a way that the remaining configuration forms exactly 3 squares.



Water Jug Problem

Given a full 5-gallon jug and a full 2-gallon jug, fill the 2-gallon jug with exactly one gallon of water.

Water Jug Problem



State = (x,y), where x is the number of gallons of water in the 5-gallon jug and y is # of gallons in the 2-gallon jug

Initial State = (5,2)

Goal State = (*,1), where * means any amount

Operator table

Name	Cond.	Transition	Effect
Empty5	–	(x,y)→(0,y)	Empty 5-gal. jug
Empty2	–	(x,y)→(x,0)	Empty 2-gal. jug
2to5	$x \leq 3$	(x,2)→(x+2,0)	Pour 2-gal. into 5-gal.
5to2	$x \geq 2$	(x,0)→(x-2,2)	Pour 5-gal. into 2-gal.
5to2part	$y < 2$	(1,y)→(0,y+1)	Pour partial 5-gal. into 2-gal.

Some real-world problems

Route finding
- directions, maps
- computer networks
- airline travel

VLSI layout

Touring (traveling salesman)

Agent planning

Search algorithms

We've defined the problem

Now we want to find the solution!

Use search techniques
- offline, simulated exploration of state space by generating successors of already-explored states (a.k.a. expanding states)
- Start at the initial state and search for a goal state

What are candidate search techniques?
- BFS
- DFS
- Uniform-cost search
- Depth limited DFS
- Depth-first iterative deepening

Finding the path: Tree search algorithms

Basic idea:

- keep a set of nodes to visit next (frontier)
- pick a node from this set
- check if it's the goal state
- if not, expand out adjacent nodes and repeat

```
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
```

BFS and DFS

How do we get BFS and DFS from this?

```
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
```

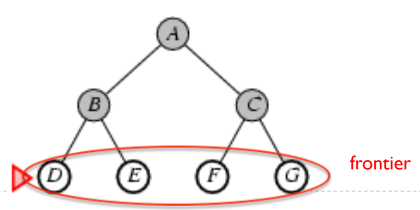
Breadth-first search

Expand shallowest unexpanded node

Nodes are expanded a level at a time (i.e. all nodes at a given depth)

Implementation:

- *frontier* is a FIFO queue, i.e., new successors go at end

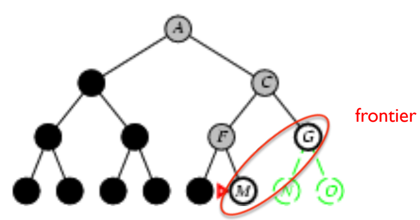


Depth-first search

Expand deepest unexpanded node

Implementation:

- *frontier* = LIFO queue, i.e., put successors at front



Search algorithm properties

Time (using Big-O)

Space (using Big-O)

Complete
- If a solution exists, will we find it?

Optimal
- If we return a solution, will it be the best/optimal solution

A divergence from algorithms/data structures
- we generally won't use V and E to define time and space. Why?
  - Often V and E are infinite!
  - Instead, we often use the branching factor (*b*) and depth (*d*)

Activity

Analyze DFS and BFS according to the criteria time, space, completeness and optimality

(for time and space, analyze in terms of *b*, *d*, and *m* (max depth); for complete and optimal - simply YES or NO)
- Which strategy would you use and why?

Brainstorm improvements to DFS and BFS

BFS

Time: $O(b^d)$

Space: $O(b^d)$

Complete: YES

Optimal: YES if action costs are fixed, NO otherwise

b = branching factor
d = depth
m = max depth of tree

Time and Memory requirements for BFS

Depth	Nodes	Time	Memory
2	1100	.11 sec	1 MB
4	111,100	11 sec	106 MB
6	10^7	19 min	10 GB
8	10^9	31 hours	1 terabyte
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3,523 years	1 exabyte

BFS with b=10, 10,000 nodes/sec; 10 bytes/node

DFS

Time: $O(b^m)$

Space: $O(bm)$

b = branching factor
d = depth
m = max depth of tree

Complete: YES, if space is finite (and no circular paths), NO otherwise

Optimal: NO

Problems with BFS and DFS

BFS

- doesn't take into account costs
- memory! ☹

DFS

- doesn't take into account costs
- not optimal
- can't handle infinite spaces
- loops

Uniform-cost search

Expand unexpanded node with the smallest ***path*** cost, g(x)

Implementation?

Uniform-cost search

Expand unexpanded node with the smallest ***path*** cost, g(x)

Implementation:

- *frontier* = priority queue ordered by **path** cost
- similar to Dijkstra's algorithm

How does it relate to bfs?

equivalent if costs are fixed

Uniform-cost search

Time? and Space?

- dependent on the costs and optimal path cost, so cannot be represented in terms of *b* and *d*
- Space will still be expensive (e.g. take uniform costs)

Complete?

- YES, assuming costs > 0

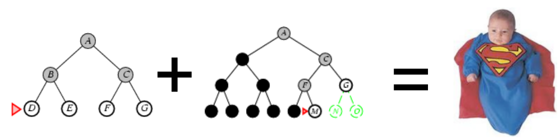
Optimal?

- Yes, assuming costs > 0

This helped us tackle the issue of costs, but still going to be expensive from a memory standpoint!

Ideas?

Can we combined the optimality and completeness of BFS with the memory of DFS?



Depth limited DFS

DFS, but with a depth limit **L** specified

- nodes at depth **L** are treated as if they have no successors
- we only search down to depth **L**

Time?

- $O(b^L)$

Space?

- O(bL)

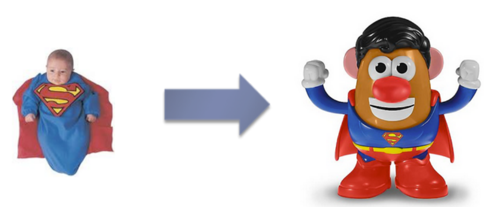
Complete?

- NO, if solution is longer than L

Optimal

- NO, for same reasons DFS isn't

Ideas?



Iterative deepening search

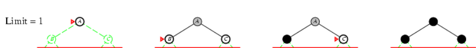
For depth 0, 1, …., ∞
    run depth limited DFS
    if solution found, return result

Blends the benefits of BFS and DFS
- searches in a similar order to BFS
- but has the memory requirements of DFS

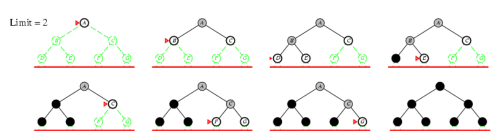
Will find the solution when **L** is the depth of the shallowest goal

Iterative deepening search L =0

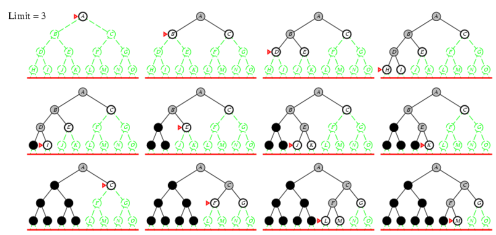
Iterative deepening search L =1



Iterative deepening search L =2



Iterative deepening search L =3



Time?

L = 0: 1

L = 1: 1 + b

L = 2: 1 + b + b^2

L = 3: 1 + b + b^2 + b^3

…

L = d: 1 + b + b^2 + b^3 + … + b^d

Overall:

- $d(1) + (d-1)b + (d-2)b^2 + (d-3)b^3 + \ldots + b^d$
- $O(b^d)$
- the cost of the repeat of the lower levels is subsumed by the cost at the highest level

Properties of iterative deepening search

Space?

- *O(bd)*

Complete?

- YES

Optimal?

- YES, if step size = 1

Missionaries and Cannibals Solution

	Near side			Far side
0 Initial setup:	MMMCCC	B		-
1 Two cannibals cross over:	MMMC		B	CC
2 One comes back:	MMMCC	B		C
3 Two cannibals go over again:	MMM		B	CCC
4 One comes back:	MMMC	B		CC
5 Two missionaries cross:	MC		B	MMCC
6 A missionary & cannibal return:	MMCC	B		MC
7 Two missionaries cross again:	CC		B	MMMC
8 A cannibal returns:	CCC	B		MMM
9 Two cannibals cross:	C		B	MMMCC
10 One returns:	CC	B		MMMC
11 And brings over the third:	-		B	MMMCCC