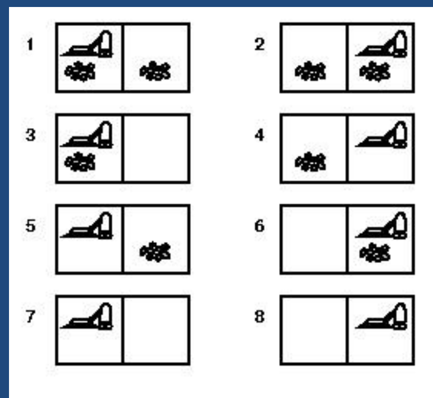


PROBLEM SOLVING AS SEARCH

EECS 492
January 11, 2011

On Problem Solving

- This course is about computing the "right thing to do" when faced with a problem.
- We start with very general purpose algorithms that:
  - … can be used on virtually any problem
  - … make very few assumptions
  - … very easy to implement
  - … are often hopelessly slow

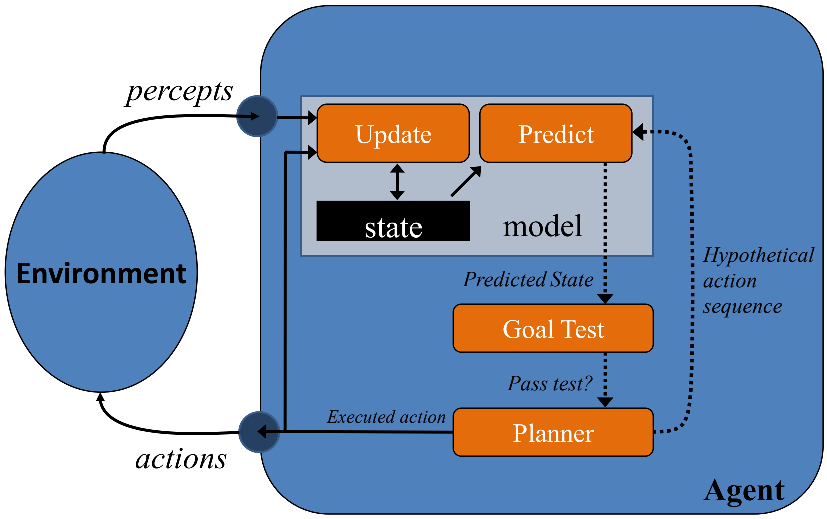
Exploiting Problem Structure

- The reason these methods are sometimes slow is because they don't exploit structure in the problem
  - Exploiting problem structure leads to fantastically better methods
  - Much of this course is about how to exploit particular types of structure (probabilistic, constraints, logic)
  - … but sometimes there is no structure (where is the Ark?)

And so, today…

- We'll talk about general-purpose problem solving
  - Very useful
  - More complex methods are based on these simple methods
  - Methods based on *search*
    - As opposed to what?

Goal-Based Agent

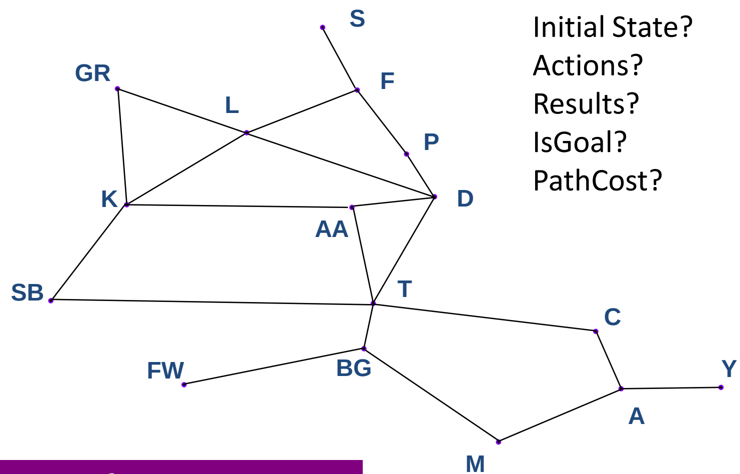


Problem Solving

- What do we need to describe a problem?
  - Initial state
    - What is state?
  - What actions can we take from state s?
    - {jump, forward, wait} = Actions(s)
  - What is the result of action a in state s?
    - s' = Result(s, a)
  - Is ‘s’ a goal state?
    - IsGoal(s)
  - What is the cost of a path?
    - PathCost(a_1, a_2, a_3, …)

Notice the limitations of this formulation!

Motivating example: Route Planning



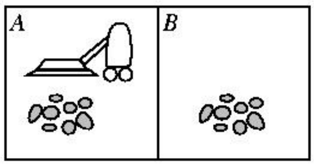
Initial State?
Actions?
Results?
IsGoal?
PathCost?

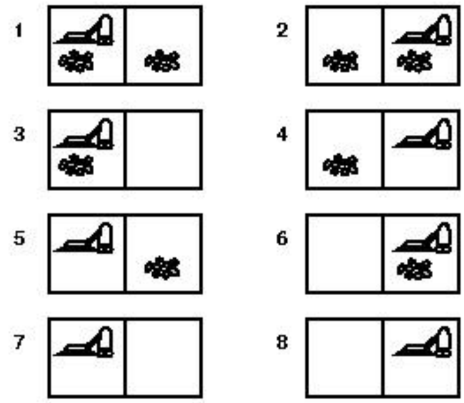
Goal: Get from AA to SB

State space graph interpretation

- States are "places"
  - In route planning example, states are *literally* places.
  - In general, states record all relevant properties of environment.
- Actions are "moves"
  - Move from one *situation* (*state*) to another
- Solution:
  - A path from the initial state to a goal state
- Optimal solution
  - The shortest possible path from the initial state to a goal state.

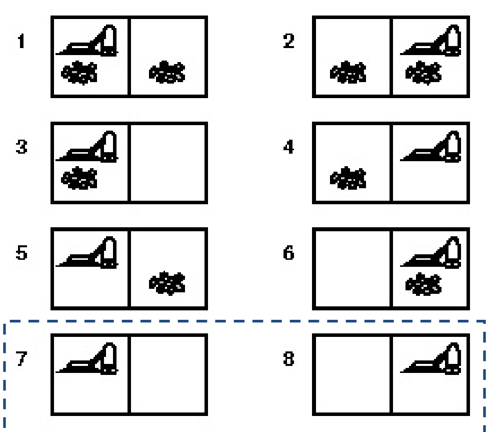
Vacuum World





Action Space:
L – move **Left**
R – move **Right**
S – **Suck** up the dirt

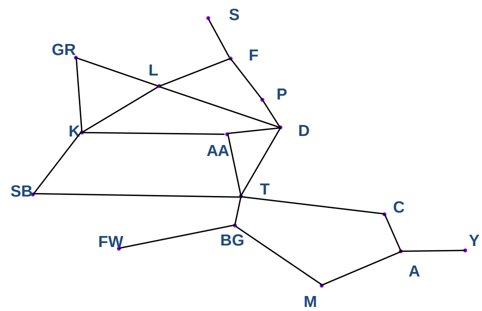
Goal States



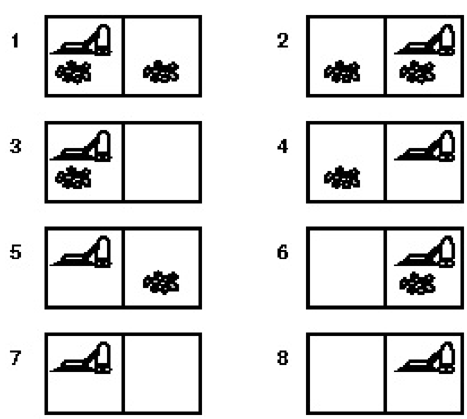
Goal: No dirt!

State Space Graph

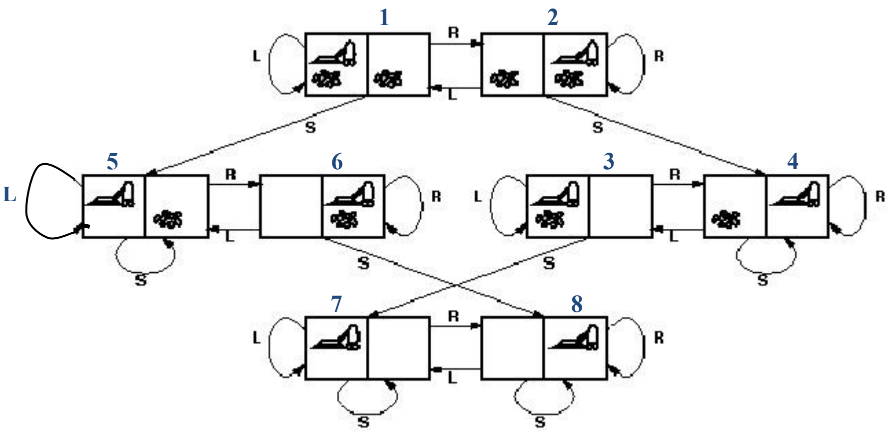
- We can connect states with actions, just like we did with the route planning example…



Your turn: Draw the State Space Graph

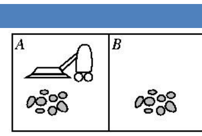


State Space Graph (solution)



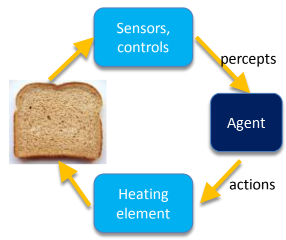
Q: How to get from 5 to 8? A: [Right, Suck]

Mini-quiz: Environment Properties



	Vacuum World	Roomba
Full or Partial Observability	Full	Partial
Determinstic or Stochastic	Deterministic	Stochastic
Static or Dynamic	Static	Dynamic
Discrete or Continuous	Discrete	Continuous
Single or multi agent	Single	?

Designing a toaster (Agent types review)



Planning \ Model	Stateless	Fixed Model	Learning Model
Reflexive			
Predictive			

How can we generalize our approach?

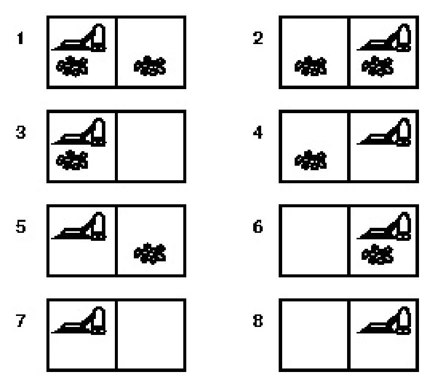
- Stochastic environments
  - Probabilistic reasoning
  - Decision processes (later in the course)
- Partial observable environments
  - Don’t know which state we’re in…
  - Keep track of which states we *might* be in.

Belief states: a preview

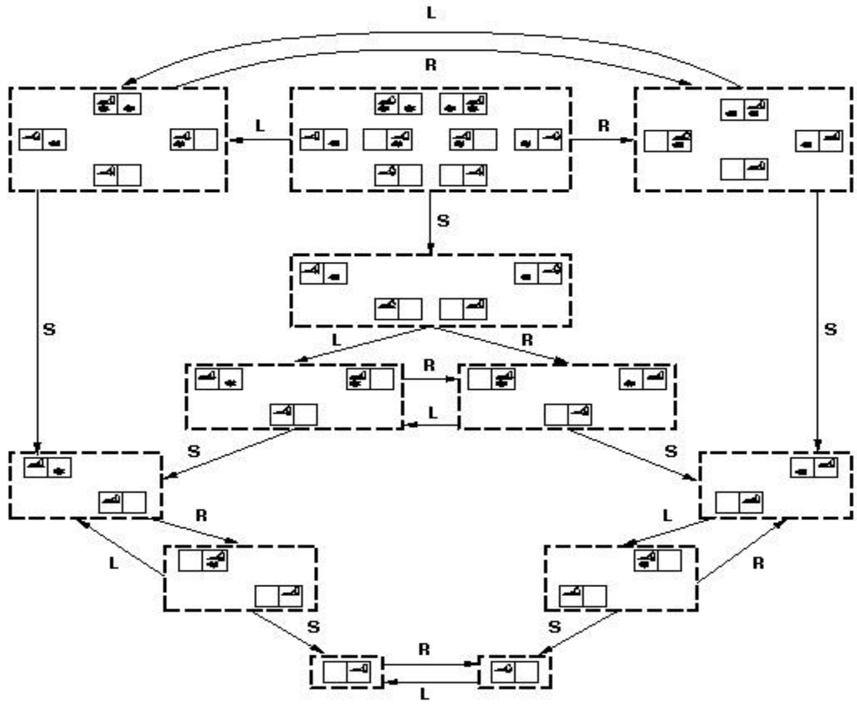
- We can convert partially-observable problems into fully observable problems!
- Create a new search problem with a different state space:
  - In new problem, states correspond to sets of states that we might be in.
  - How do we handle actions and results?
  - How do we handle goal test?
  - How do we handle path cost?

Belief States: Try it!

- Draw the state space diagram for the vacuum world where:
  - Robot is lost.
  - Location of dirt is unknown.
  - Robot has *no* sensors.
  - Goal: no dirt.



- Can we clean the house?



Analogy to NFAs and DFAs

- Tokenizing files using regular expressions:
  - `SYMBOL:  [a-zA-Z0-9_]`
  - `KEYWORD: if | while | for`
  - The actual states we care about are the type of token (e.g., SYMBOL or KEYWORD)
  - Our belief state is the set of parsing states that we might be in
- If we receive characters "w", "h", "i", etc., our belief state consists of both SYMBOL and KEYWORD; future characters will resolve our belief state until we reach a production state.

Non-deterministic actions

- Now, suppose that:
  - Robot doesn't move reliably: sometimes attempting to move results in no movement.
  - Robot doesn't clean reliably: half the time, it misses some dirt in the room.
- Is there a (finite) action sequence that cleans the house?
  - With 100% probability? No.
  - LLL…. SSS… RRR… SSS…
  - *Synchronizing Sequence. (Maze example).*
- Suppose that rooms become dirty again with probability P each turn and we get $1 if both rooms are clean, we pay $1 if one room is dirty, and we pay $5 if both rooms are dirty.
  - For this, we'll have to wait for better methods later in the course.

Limits of state space graph

- Not all problems have simple state space graph:
  - State space can be infinite (including continuous)
  - Results function can be very complicated
    - Could be a simulation of the laws of physics…
  - In these cases, it is important to allow Actions() and Results() to be generalized functions

Search

- We have not actually described a way of solving these problems…. Until now!
- Search
  - Any systematic way of traversing the graph of states in order to find a sequence of actions leading to a goal state
  - General approach, can be applied to any well-defined problem

Search Trees

- Set of all paths, starting at initial state
- Search node
  - corresponding state in state space
  - predecessor in path, or parent
  - action applied to parent to generate node
  - path cost from root
- Elaborate paths in a search tree by expanding search nodes
  - Which nodes do we expand?

Search Strategy

- Dictates which node to expand in any particular search situation
- Candidates are nodes at fringe: leaves of partial search tree
- Maintain fringe in generalized queue

Queue ADT

new Queue()	creates a queue
q.isEmpty()	emptiness predicate
q.get()	returns/removes next elt
q.put(elt)	inserts elt into q
q.putAll(elts)	puts all elts into q

General Tree Search

```
function Tree-search(problem)
   returns a solution, or failure

   fringe = new Queue();
   fringe.put(problem.initialState)

   loop do
      if fringe.isEmpty() then return failure
      node ← fringe.get()
      if problem.isGoalState(node)
         then return node;
      fringe.putAll(problem.expand(node))
```

Which node in the fringe does get() return?

Why is a goal node a solution?

Measuring Search Performance

- Completeness
  - Is the algorithm guaranteed to find a solution if it exists?
- Optimality
  - Does the strategy find the minimum path cost solution?
- Time Complexity
  - How long does it take to find a solution?
- Space Complexity
  - How much memory is needed to perform the search?

Our search strategy will affect all of the above!

Next time: Uninformed Search

- Search strategies based only on structure of search tree
- Questions:
  - How do alternative approaches compare wrt our performance measures?
  - What are key tradeoffs?
  - What would constitute *informed* search?