PROBLEM SOLVING AGENTS

Problem Solving Agents

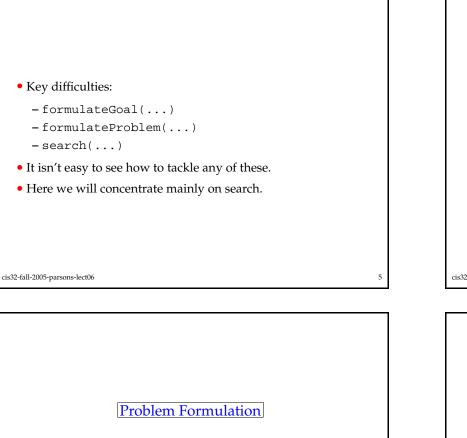
- Lecture 1 introduced *rational agents*.
- Now consider agents as *problem solvers*: Systems which set themselves *goals* and find *sequences of actions* that achieve these goals.
- What is a problem?
- A goal and a means for achieving the goal.
- The goal specifies the state of affairs we want to bring about.
- The means specifies the operations we can perform in an attempt to bring about the means.
- The difficulty is deciding what *order* to carry out the operations.

```
cis32-fall-2005-parsons-lect06
```

Overview Aims of the this lecture: • introduce problem solving; • introduce goal formulation; • show how problems can be stated as state space search; • show the importance and role of abstraction; • introduce undirected search: • breadth 1st search; • depth 1st search. • define main performance measures for search.

```
• Operation of problem solving agent:
/* s is sequence of actions */
repeat {
    percept = observeWorld();
    state = updateState(state, p);
    if s is empty then {
        goal = formulateGoal(state);
        prob = formulateProblem(state,p);
        s = search(prob);
    }
    action = recommendation(s);
    s = remainder(s, state);
}
until false; /* i.e., forever */
```

cis32-fall-2005-parsons-lect06



- Once goal is determined, formulate the problem to be solved.
- First determine set of possible states *S* of the problem.
- Then problem has:
 - *initial state —* the starting point, *s*₀;
 - *operations* the actions that can be performed, $\{o_1, \ldots, o_n\}$.
 - *goal* what you are aiming at subset of *S*.

Goal Formulation

- Where do an agent's goals come from?
 - Agent is a *program* with a *specification*.
 - Specification is to maximise performance measure.
 - Should *adopt goal* if achievement of that goal will maximise this measure.
- Goals provide a *focus* and *filter* for decision-making:
 - *focus*: need to consider how to achieve them;
 - *filter*: need not consider actions that are incompatible with goals.

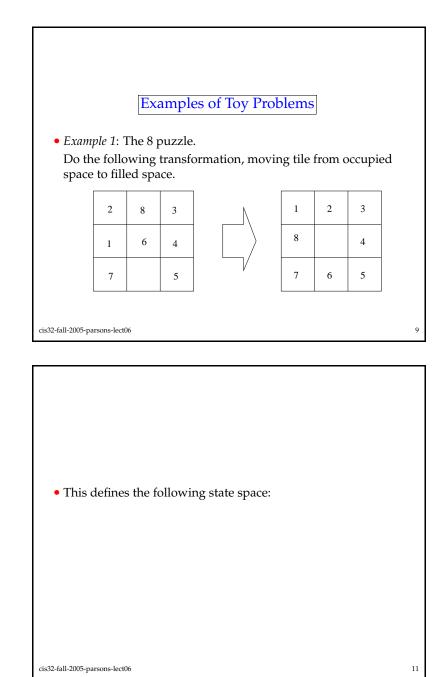
cis32-fall-2005-parsons-lect06

- The initial state together with operations determines *state space* of problem.
- Operations cause *changes* in state.
- Solution is a sequence of actions such that when applied to initial state *s*₀, we have goal state.
- Pictorially:

cis32-fall-2005-parsons-lect06

7

cis32-fall-2005-parsons-lect06



- Initial state as shown above.
- Goal state as shown below.
- Operations:
 - o_1 : move any tile to left of empty square to right;
 - $-o_2$:
 - $-o_3$:
- $-o_4$:

cis32-fall-2005-parsons-lect06

- Example 2: The *n* queens problem from chess.
- Place *n* queens on chess board so that no queen can be taken by another.
- Initial state: empty chess board.
- Goal state: *n* queens on chess board, one occupying each space, so that none can take others.
- Operations: place queen in empty square.

Solution Cost		• As an example, consider the follo
• For most problems, some solutions are better than othe	ers:	2 8
 – in 8 puzzle, number of moves to get to solution; – number of moves to checkmate; – length of distance to travel. 		1 6
• Mechanism for determining <i>cost</i> of solution is <i>path cost</i>	function.	
• This is the length of the path through the state-space fr initial state to the goal state.	-	• How many moves are there to the
cis32-fall-2005-parsons-lect06	13	cis32-fall-2005-parsons-lect06
 Obviously there are four moves: 1. 2. 		 Problem Solvin In the state space view of the wo
3. 4.		a path through the state space.
 And the path through the solution space looks like: 		 When we solve a problem like th of what constitutes the next best
		• It is hard to program this kind of
		 Instead we start by programming computers are good at.
		• A <i>brute force</i> approach to problem <i>searching</i> through the space of <i>all</i> one that achieves goal.
is32-fall-2005-parsons-lect06	15	cis32-fall-2005-parsons-lect06

lowing state in the 8-puzzle:

2	8	3
1	6	4
7		5

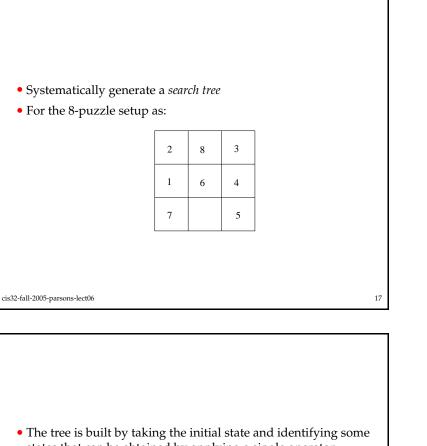
14

16

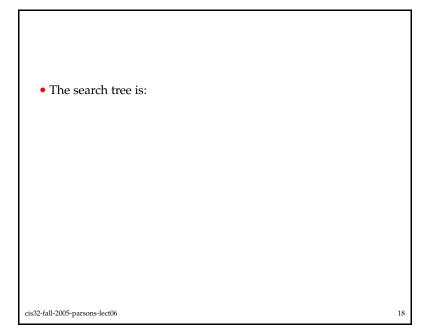
he solution?

ig as Search

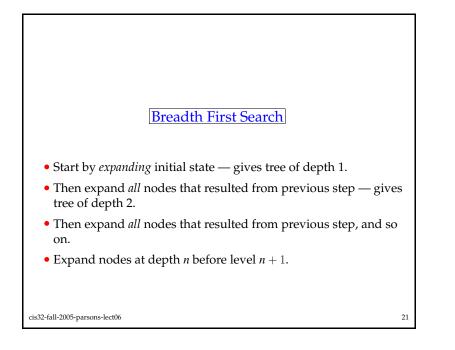
- orld, finding a solution is finding
- ne 8-puzzle we have some idea move.
- f approach.
- ng the kind of repetitive task that
- m solving involves *exhaustively* possible action sequences to find



- states that can be obtained by applying a single operator.These new states become the *children* of the initial state in the
- tree.
- These new states are then examined to see if they are the goal state.
- If not, the process is repeated on the new states.
- We can formalise this description by giving an algorithm for it.



```
General algorithm for search:
agenda = initial state;
while agenda not empty do{
    pick node from agenda;
    new nodes = apply operations to state;
    if goal state in new nodes
    then {
        return solution;
    }
    add new nodes to agenda;
}
Question: How to pick states for expansion?
Two obvious solutions:
    - depth first search;
    - breadth first search.
```



/* Breadth first search */
agenda = initial state;
<pre>while agenda not empty do { pick node from front of agenda; new nodes = apply operations to state; if goal state in new nodes then { return solution; } </pre>
, APPEND new nodes to END of agenda; }

22

24

```
• Time for breadth first search:
                         Depth Nodes
                                                 Time
                                               1 msec
                               0
                                        1
                               1
                                      11
                                               .01 sec
                               2
                                     111
                                                .1 sec
                               4
                                  11,111
                                              11 secs
                               6
                                     10^{6}
                                              18 mins
                              8
                                     10^{8}
                                            31 hours
                                     10^{10}
                             10
                                            128 days
                                     10^{12}
                             12
                                             35 years
                                     10<sup>14</sup> 2500 years
                             14
                             20
                                     10^{20}
                                            3^{15} years
  • Combinatorial explosion!
cis32-fall-2005-parsons-lect06
```

cis32-fall-2005-parsons-lect06

23

• Advantage: *guaranteed* to reach a solution if one exists.

• If all solutions occur at depth *n*, then this is good approach.

- Disadvantage: time taken to reach solution!
- Let *b* be *branching factor* average number of operations that may be performed from any level.
- If solution occurs at depth *d*, then we will look at

 $1+b+b^2+\cdots+b^d$

nodes before reaching solution — *exponential*.

cis32-fall-2005-parsons-lect06



- When formulating a problem, it is crucial to pick the right level of *abstraction*.
- Example: Given the task of driving from New York to Boston.
- Some possible actions...
 - depress clutch;
 - turn steering wheel right 10 degrees;
 - ... inappropriate level of *abstraction*. Too much *irrelevant detail*.

cis32-fall-2005-parsons-lect06

- Better level of abstraction:
 - Take the Henry Hudson Parkway north
 - Take the Cross County turnoff
- ... and so on.
- Getting abstraction level right lets you focus on the specifics of problem and is one way to combat the combinatorial explosion.

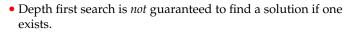
26

28

• (Tell that to Mapquest).

cis32-fall-2005-parsons-lect06

```
/* Depth first search */
                                                                                 agenda = initial state;
                     Depth First Search
                                                                                 while agenda not empty do
  • Start by expanding initial state.
                                                                                      pick node from front of agenda;
                                                                                      new nodes = apply operations to state;
  • Pick one of nodes resulting from 1st step, and expand it.
                                                                                      if goal state in new nodes then
  • Pick one of nodes resulting from 1nd step, and expand it, and so
    on.
                                                                                            return solution;
  • Always expand deepest node.
  • Follow one "branch" of search tree.
                                                                                 put new nodes on FRONT of agenda;
                                                                           cis32-fall-2005-parsons-lect06
cis32-fall-2005-parsons-lect06
                                                              27
```



- However, if it *does* find one, amount of time taken is much less than breadth first search.
- *Memory requirement* is much less than breadth first search.
- Solution found is *not* guaranteed to be the best.

Completeness: Is the search technique *guaranteed* to find a solution if one exists? *Time complexity*: How many computations are required to find solution? *Space complexity*:

Performance Measures for Search

- How much memory space is required?
- *Optimality*: How good is a solution going to be w.r.t. the path cost function.

30

```
cis32-fall-2005-parsons-lect06
```

29

31

cis32-fall-2005-parsons-lect06

Summary

- This lecture introduced the basics of problem solving.
- In particular it discussed *state space* models and looked at the basic techniques for solving them.
 - Search for the goal.
 - Path through state space is the solution.
- We also looked at two techniques for search:
 - Breadth first.
 - Depth first.

cis32-fall-2005-parsons-lect06