

## PROBLEM SOLVING AGENTS

### 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.

### 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.

- 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 */
```

- Key difficulties:

- `formulateGoal(...)`
- `formulateProblem(...)`
- `search(...)`

- It isn't easy to see how to tackle any of these.
- Here we will concentrate mainly on search.

## 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.

## Problem Formulation

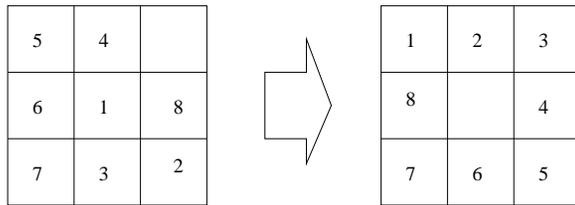
- Once goal is determined, formulate the problem to be solved.
- First determine set of possible states  $S$  of the problem.
- Then problem has:
  - *initial state* — the starting point,  $s_0$ ;
  - *operations* — the actions that can be performed,  $\{o_1, \dots, o_n\}$ .
  - *goal* — what you are aiming at — subset of  $S$ .

- 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:

## Examples of Toy Problems

- *Example 1:* The 8 puzzle.

Do the following transformation, moving tile from occupied space to filled space.



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

- This defines the following state space:

- 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

- For most problems, some solutions are better than others:
  - in 8 puzzle, number of moves to get to solution;
  - number of moves to checkmate;
  - length of distance to travel.
- Mechanism for determining *cost* of solution is *path cost function*.
- This is the length of the path through the state-space from the initial state to the goal state.

- As an example, consider the following state in the 8-puzzle:

5	4	
6	1	8
7	3	2

- How many moves are there to the solution?

- Obviously there are four moves:
  - 1.
  - 2.
  - 3.
  - 4.
- And the path through the solution space looks like:

## Problem Solving as Search

- In the state space view of the world, finding a solution is finding a path through the state space.
- When we solve a problem like the 8-puzzle we have some idea of what constitutes the next best move.
- It is hard to program this kind of approach.
- Instead we start by programming the kind of repetitive task that computers are good at.
- A *brute force* approach to problem solving involves *exhaustively searching* through the space of *all possible* action sequences to find one that achieves goal.

- Systematically generate a *search tree*
- For the 8-puzzle setup as:

5	4	
6	1	8
7	3	2

- The search tree is:

- The tree is built by taking the initial state and identifying some 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

- Start by *expanding* initial state — gives tree of depth 1.
- Then expand *all* nodes that resulted from previous step — gives tree of depth 2.
- Then expand *all* nodes that resulted from previous step, and so on.
- Expand nodes at depth  $n$  before level  $n + 1$ .

```
/* Breadth first search */

agenda = initial state;

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;
    }

    APPEND new nodes to END of agenda;
}
```

- 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

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

nodes before reaching solution — *exponential*.

- Time for breadth first search:

Depth	Nodes	Time
0	1	1 msec
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
12	$10^{12}$	35 years
14	$10^{14}$	2500 years
20	$10^{20}$	$3^{15}$ years

- *Combinatorial explosion!*

## Importance of ABSTRACTION

- 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*.

- 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.
- (Tell that to Mapquest).

## Depth First Search

- Start by expanding initial state.
- Pick one of nodes resulting from 1st step, and expand it.
- Pick one of nodes resulting from 1nd step, and expand it, and so on.
- Always expand *deepest* node.
- Follow one “branch” of search tree.

```
/* Depth first search */  
agenda = initial state;  
  
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;  
    }  
  
    put new nodes on FRONT of agenda;  
}
```

- Depth first search is *not* guaranteed to find a solution if one exists.
- 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.

## Performance Measures for Search

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

## 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.