Overview

- Last time
  - Intelligent agents and environments

- Today:
  - introduce problem solving and problem formulation
  - show how problems can be stated as state space search
  - show the importance and role of abstraction
  - define main performance measures for search

- Learning outcome covered today:
  Identify, contrast and apply to simple examples the major search techniques that have been developed for problem-solving in AI

Problem Solving

- 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 goal

- The difficulty is deciding what order to carry out the operations

- Solution will be a sequence of operations leading from initial state to goal state (plan)

Problem Formulation

More precisely, a problem can be defined by the following items:

States - a set;
- Initial state - a particular state where the agent starts off;
- Actions - are applicable in states;
- Transition model - specified by a function that returns the state that results from performing action a in state s;

Goal test - determines whether a given state is a goal state (may be explicit or implicit);
- Path cost - a function that assigns a numeric cost to each path.
- A solution is a sequence of actions leading from the initial state to a goal state.
Example: Romania

On holiday in Romania; currently in Arad.
Flight leaves tomorrow from Bucharest.

Romania: Problem Formulation

- States: various cities
- Initial state: in Arad
- Actions: drive between cities
  - e.g. In Arad: drive to Sibiu, drive to Zerind, drive to Timisoara, etc.
- Transitions: cities lead to
  - e.g. When in Arad, driving to Zerind results in being in Zerind, etc.
- Goal: be in Bucharest (- an explicit goal state)
- Path cost: Sum of step costs, i.e. distances between cities, in kilometres

Example: Vacuum World

- Consider an agent designed to vacuum clean.
- The world it inhabits has just two locations, squares A and B.
- The agent can perceive its location (which square it is in) and whether there is dirt in the square.
- The agent can choose to move left, move right, suck up the dirt, or do nothing.
Vacuum World Problem Formulation

- **States:** integer dirt and robot location
- **Initial state:** any can be designated
- **Actions:** Left, Right, Suck
- **Transition model:** actions have expected effects, though some result in no effect, e.g. sucking in a clean square
- **Goal test:** no dirt at all locations
- **Path cost:** step costs 1 per action, so path is number of steps

State Space Graph

- When taken together, the initial state, actions and transition model implicitly define the **state space** of the problem
  - This is the set of all states reachable from the initial state by any sequence of actions.
- The state space forms a directed network or graph in which the nodes are states and the links between nodes are actions.
- For the Romania example: the previous map can be interpreted as a state space graph if each road is viewed as standing for two driving actions, one in each direction.
- For the vacuum world, the state space graph is as follows...

Vacuum World State Space Graph

More Examples of Real World Problems

- Game playing
- Route finding - routing in computer networks, rail travel, air travel
- Touring and travelling - find a route between Aberdeen and Glasgow; travelling salesperson problem
- Assembly sequencing
- VLSI layout
- Robot navigation
  - . . . .
Toy Problems:
The n-Queens Problem

- This is a problem from chess.
- In the 8-queens version, place 8 queens on chess board so that no queen can be taken by another.
- A queen attacks any piece in the same row, column or diagonal.
- Has served as a useful test scenario for search algorithms.

Exercise
Add two more queens to this board to provide a solution to the 8-queens problem.

Exercise
A program that solves the problem.
And another one.

n-Queens as a Search Problem

- **States**: Any arrangement of 0-8 queens on the board.
- **Initial state**: empty chess board.
- **Actions**: place queen in empty square.
  - Place queens anywhere
    - For the 8-queens problem $64^8$ possibilities.
  - Place queens only where they are not attacked already.
    - Around 2,000 possible sequences to check.
- **Transition model**: returns the board with a queen added to the specified square.
- **Goal test**: n queens on chess board so that none can take any other.
**Toy Problems: The 8 Puzzle**

**States:** 3 × 3 grid filled with numbers 1–8 and a blank.

**Initial state:** as shown on the left.

**Goal test:** as shown on the right.

**Actions:** $a_i$ - move any tile to left of empty square to right; ....

**Transition model:** the states resulting from actions in a given state.

**Path cost:** number of steps, with each step cost = 1.

---

**Exercise**

For the 8 puzzle, add the next level to the search space showing all possible states reachable from the initial state.

---

**Search Space**

Again we can map out all possible states reachable from the initial state to give the full search space.

---

**8 Puzzle Search Space**

Again we can map out all possible states reachable from the initial state to give the full search space.
Search Tree

The search tree shows nodes explored by a search algorithm to solve the problem. Root is the initial state: successor nodes found by applying operations (expanding nodes). Stops when goal is reached.

Search Strategy Performance

- A search strategy is defined by picking the order of node expansion
- Completeness: does it always find a solution if one exists?
- Time complexity: number of nodes generated/expanded.
- Space complexity: maximum number of nodes in memory.
- Optimality: does it always find a least-cost solution?
- Time and space complexity: are measured in terms of
  - \( b \): maximum branching factor of the search tree
  - \( d \): depth of the least-cost solution
  - \( m \): maximum depth of the state space (may be infinite)

Tree Search Algorithms

- General description:

  ```
  function TREE-SEARCH(problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the corresponding solution
    else expand the node and add the resulting nodes to the search tree
  end loop
  ```

Abstraction

- The real world is absurdly complex
  - therefore state space must be abstracted for problem solving.
- (Abstract) state = set of real states.
- (Abstract) action = complex combination of real actions
  - “Arad to Zerind” represents a complex set of possible routes, detours, rest stops, etc.
- For guaranteed realisability, any real state “in Arad” must get to some real state “in Zerind”.
- (Abstract) solution
  - a set of real paths that are solutions in the real world.
- Abstraction should be “easier” than the original problem.
Right Level of Abstraction

• Example: driving from city A to city B.
  – Some possible actions. . .
    • depress clutch;
    • turn steering wheel right 10 degrees;
  – inappropriate level of abstraction; too much irrelevant detail.
• Better level of abstraction:
  – follow A143 to Colchester for 4 miles;
  – turn right onto M12;
  – . . . and so on.
• Getting abstraction level right lets you focus on the specifics of the problem and combats the combinatorial explosion.

Solution Cost

• For most problems, some solutions are better than others:
  – 8 puzzle: number of moves to get to solution;
  – chess: number of moves to checkmate;
  – route planning: length of distance (or time) to travel.
• Mechanism for determining cost of solution is the path cost function.
• This is the length (cost) of the path through the state space from the initial state to the goal state.

Summary

• Today
  – Some search problems
  – Representing a search problem
  – Search trees
  – Evaluating search strategy performance

• Next time
  – Recursion in Prolog