Solving problems by searching

Chapter 3
Search

• We will consider the problem of designing goal-based agents in fully observable, deterministic, discrete, known environments

• Example:
Search

• We will consider the problem of designing goal-based agents in fully observable, deterministic, discrete, known environments
  – The solution is a fixed sequence of actions
  – Search is the process of looking for the sequence of actions that reaches the goal
  – Once the agent begins executing the search solution, it can ignore its percepts (open-loop system)
Search problem components

- **Initial state**
- **Actions**
- **Transition model**
  - What is the result of performing a given action in a given state?
- **Goal state**
- **Path cost**
  - Assume that it is a sum of nonnegative *step costs*

- The **optimal solution** is the sequence of actions that gives the lowest path cost for reaching the goal
Example: Romania

- On vacation in Romania; currently in Arad
- Flight leaves tomorrow from Bucharest

- **Initial state**
  - Arad

- **Actions**
  - Go from one city to another

- **Transition model**
  - If you go from city A to city B, you end up in city B

- **Goal state**
  - Bucharest

- **Path cost**
  - Sum of edge costs
State space

• The initial state, actions, and transition model define the state space of the problem
  – The set of all states reachable from initial state by any sequence of actions
  – Can be represented as a directed graph where the nodes are states and links between nodes are actions

• What is the state space for the Romania problem?
Example: Vacuum world

- **States**
  - Agent location and dirt location
  - How many possible states?
  - What if there are $n$ possible locations?

- **Actions**
  - Left, right, suck

- **Transition model**
Vacuum world state space graph
Example: The 8-puzzle

- **States**
  - Locations of tiles
    - 8-puzzle: 181,440 states
    - 15-puzzle: 1.3 trillion states
    - 24-puzzle: $10^{25}$ states

- **Actions**
  - Move blank left, right, up, down

- **Path cost**
  - 1 per move

- **Finding the optimal solution of n-Puzzle is NP-hard**
Example: Robot motion planning

- **States**
  - Real-valued coordinates of robot joint angles

- **Actions**
  - Continuous motions of robot joints

- **Goal state**
  - Desired final configuration (e.g., object is grasped)

- **Path cost**
  - Time to execute, smoothness of path, etc.
Search

• Given:
  – Initial state
  – Actions
  – Transition model
  – Goal state
  – Path cost

• How do we find the optimal solution?
  – How about building the state space and then using Dijkstra’s shortest path algorithm?
    • The state space may be huge!
    • Complexity of Dijkstra’s is $O(E + V \log V)$, where $V$ is the size of the state space
Tree Search

• Let’s begin at the start node and expand it by making a list of all possible successor states
• Maintain a fringe or a list of unexpanded states
• At each step, pick a state from the fringe to expand
• Keep going until you reach the goal state
• Try to expand as few states as possible
Search tree

- “What if” tree of possible actions and outcomes
- The root node corresponds to the starting state
- The children of a node correspond to the **successor states** of that node’s state
- A path through the tree corresponds to a sequence of actions
  - A solution is a path ending in the goal state
- Nodes vs. states
  - A state is a representation of a physical configuration, while a node is a data structure that is part of the search tree
Tree Search Algorithm Outline

- Initialize the **fringe** using the **starting state**
- While the fringe is not empty
  - Choose a fringe node to expand according to **search strategy**
  - If the node contains the **goal state**, return solution
  - Else **expand** the node and add its children to the fringe
Tree search example
Tree search example
Tree search example
Search strategies

• A search strategy is defined by picking the order of node expansion

• Strategies are evaluated along the following dimensions:
  – Completeness: does it always find a solution if one exists?
  – Optimality: does it always find a least-cost solution?
  – Time complexity: number of nodes generated
  – Space complexity: maximum number of nodes in memory

• 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 length of any path in the state space (may be infinite)
Uninformed search strategies

- Uninformed search strategies use only the information available in the problem definition
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Iterative deepening search