Solving problems by searching

Chapter 3
Types of agents

**Reflex agent**
- Consider how the world IS
- Choose action based on current percept
- Do not consider the future consequences of actions

**Planning agent**
- Consider how the world WOULD BE
- Decisions based on (hypothesized) consequences of actions
- Must have a model of how the world evolves in response to actions
- Must formulate a goal

Source: D. Klein, P. Abbeel
Search

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

• We will consider the problem of designing goal-based agents in fully observable, deterministic, discrete, known environments
  – The agent must find a sequence of actions that reaches the goal
  – The performance measure is defined by (a) reaching the goal and (b) how “expensive” the path to the goal is
  – We are focused on the process of finding the solution; while executing the solution, we assume that the agent can safely ignore its percepts (open-loop system)
Search problem components

- **Initial state**
- **Actions**
- **Transition model**
  - What state results from 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 (total distance traveled)
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?
    - The size of the state space grows exponentially with the “size” of the world!

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

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

• **States**
  – Locations of tiles
    • 8-puzzle: 181,440 states (9!/2)
    • 15-puzzle: ~10 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 joint parameters (angles, displacements)

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

- **Goal state**
  - Configuration in which 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?
    • Complexity of Dijkstra’s is $O(E + V \log V)$, where $V$ is the size of the state space
    • The state space may be huge!
Search: Basic idea

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

• “What if” tree of sequences of 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 the world, while a node is a data structure that is part of the search tree
    • Node has to keep pointer to parent, path cost, possibly other info
Tree Search Algorithm Outline

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

Start: Arad
Goal: Bucharest
Tree search example

Start: Arad
Goal: Bucharest
Tree search example

Start: Arad
Goal: Bucharest
Tree search example

Start: Arad  
Goal: Bucharest
Start: Arad
Goal: Bucharest
Tree search example

Start: Arad
Goal: Bucharest
Tree search example

Start: Arad
Goal: Bucharest
Handling repeated states

- Initialize the **frontier** using the **starting state**
- While the frontier is not empty
  - Choose a frontier node according to **search strategy** and take it off the frontier
  - If the node contains the **goal state**, return solution
  - Else **expand** the node and add its children to the frontier

- To handle repeated states:
  - Every time you expand a node, add that state to the **explored set**; do not put explored states on the frontier again
  - Every time you add a node to the frontier, check whether it already exists in the frontier with a higher path cost, and if yes, replace that node with the new one
Search without repeated states

Start: Arad
Goal: Bucharest

Straight-line distance
Arad  366
Bucharest  160
Craiova  242
Dobrogea  101
Fagaras  176
Giurgiu  151
Iasi  126
Lugoj  244
Mehadia  341
Neamt  234
Oradea  382
Pitesti  193
Rimnicu Vilcea  233
Sibiu  198
Timisoara  325
Urziceni  86
Vaslui  195
Zerind  374
Search without repeated states

Start: Arad
Goal: Bucharest
Search without repeated states

Start: Arad
Goal: Bucharest
Search without repeated states

Start: Arad
Goal: Bucharest
Search without repeated states

Start: Arad
Goal: Bucharest
Search without repeated states

Start: Arad
Goal: Bucharest
Search without repeated states

Start: Arad  
Goal: Bucharest