Chapter II - Problem Solving

Chapter II: Problem Solving

I. Problem Solving

2.1 Problem-Solving Agents

A **problem-solving agent** is an AI agent that takes an initial state, a goal state, and a set of possible actions, and then works through a sequence of actions to reach the goal state. It acts as a decision maker that chooses actions to transition from one state to another, eventually solving the problem.

Components of a Problem-Solving Agent:

1. **Initial State**: The starting point or condition from which the agent begins.
2. **Goal State**: The desired outcome or objective that the agent aims to reach.
3. **Actions**: A set of permissible operations or moves that the agent can perform to change its state.
4. **State Space**: The set of all possible states the agent can reach from the initial state.
5. **Transition Model**: A description of what happens when an agent takes an action in a given state, resulting in a new state.
6. **Solution**: A sequence of actions that lead the agent from the initial state to the goal state.

Example:

- **Problem**: The classic 8-puzzle, where the goal is to slide the tiles on a 3x3 grid to reach a specific configuration.
- **Agent**: The AI agent moves the tiles in search of the goal configuration.

2.2 Examples of Problems

AI problem-solving can be applied to a wide variety of situations. Here are some examples of common problem types:

1. **Navigation Problems**: Finding the shortest path between two points (e.g., GPS systems finding routes from one location to another).
2. **Puzzles**: Solving puzzles like the 8-puzzle, 15-puzzle, or the traveling salesman problem.
3. **Game Playing**: Determining optimal moves in a game, such as chess, checkers, or tic-tac-toe.

4. **Scheduling**: Allocating resources efficiently, such as in timetabling classes or assigning tasks to workers.
5. **Planning Problems**: Creating a sequence of actions to achieve a goal, such as in robotics or manufacturing.

2.3 Searching for Solutions

Searching is a central technique in problem-solving, where the agent explores the space of possible states to find a solution.

Steps in Searching:

1. Start at the **initial state**.
2. Explore neighboring states by applying the possible **actions**.
3. Continue expanding the state space until the **goal state** is found or all possibilities have been exhausted.

Types of Search:

- **State-space search** involves systematically exploring the set of possible states.
- The search process can be categorized into two broad approaches:
    - **Uninformed Search**: No additional information is used to guide the search.
    - **Informed Search**: Uses additional knowledge or heuristics to guide the search more efficiently.

2.4 Uninformed Search Strategies

Uninformed search strategies, also known as **blind search**, explore the state space without any additional information beyond the problem's definition.

Common uninformed search strategies:

1. **Breadth-First Search (BFS)**: Expands all nodes at the present depth level before moving on to nodes at the next level. It guarantees the shortest path but can be memory-intensive.
    - **Time Complexity**: O(bd)O(b^d), where bb is the branching factor and dd is the depth of the shallowest solution.
2. **Depth-First Search (DFS)**: Explores as far as possible down one branch of the state space before backtracking. It can be more memory-efficient but does not guarantee the shortest path.
    - **Time Complexity**: O(bm)O(b^m), where mm is the maximum depth of the state space.
3. **Uniform Cost Search**: Expands the node with the lowest cost path to reach that node. It is optimal when the cost of each step is non-negative.

- **Time Complexity**: O(bd)O(b^d), similar to BFS, but more dependent on the path costs.

2.5 Informed Search Strategies (Heuristic)

Informed search strategies use **heuristics** or additional problem-specific knowledge to guide the search towards the goal more efficiently. Heuristics are functions that estimate the cost or distance to the goal from a given state.

Common informed search strategies:

1. **Greedy Search**: Selects the node that appears to be the closest to the goal based on a heuristic function h(n)h(n).
   - **Time Complexity**: O(bd)O(b^d), similar to BFS, but it may not always lead to the optimal solution.
2. _*A* Search_*: Combines the strengths of both uniform cost search and greedy search. It evaluates nodes based on f(n)=g(n)+h(n)f(n) = g(n) + h(n), where:
   - g(n)g(n): The cost of the path from the start node to the current node.
   - h(n)h(n): The heuristic estimate of the cost from the current node to the goal.
   - A* search is optimal if the heuristic is admissible (never overestimates the true cost).
3. **Best-First Search**: Similar to greedy search, but considers both the heuristic and cost. It is not guaranteed to find the optimal solution like A* but is more efficient in some cases.

2.6 Heuristic Functions

A **heuristic function** is a crucial element in informed search algorithms, providing an estimate of the cost or distance from the current state to the goal.

Properties of a Good Heuristic:

1. **Admissibility**: The heuristic should never overestimate the cost to reach the goal.
2. **Consistency**: The heuristic should satisfy the condition that for every node nn and its successor n′n', the estimated cost from nn to the goal should not exceed the cost of getting from nn to n′n' plus the cost of getting from n′n' to the goal.

Example of Heuristic:

- In the case of the **8-puzzle**, a common heuristic is the **Manhattan Distance**, which calculates the total number of moves required to move each tile from its current position to its target position.

Exercises

1. **Problem-Solving Agent**: Describe how a problem-solving agent would solve the 8-puzzle problem. What are the initial state, goal state, and possible actions?
2. **State-Space Exploration**: Draw a simple state-space diagram for a problem where the agent needs to find the shortest path from a start point to a goal point in a grid (like a maze). Show how the agent would expand the state space.
3. **Uninformed Search**: Compare and contrast the strengths and weaknesses of **Breadth-First Search** and **Depth-First Search**. In what scenarios would you choose one over the other?
4. **Heuristic Search**: Given a problem where you need to travel between cities, propose a suitable heuristic for a search algorithm. Justify why your heuristic would be effective.
5. **Heuristic Functions**: Given a problem where an agent needs to arrange books on a shelf in a specific order, design a heuristic function that could guide a search algorithm in solving this problem efficiently.