Chapter III - Beyond Classical Search

Chapter II: Beyond Classical Search

II. Beyond Classical Search

3.1 Local Search Algorithms and Optimization Problems

Local search algorithms are used for solving optimization problems by iteratively moving towards a better solution, typically by exploring neighboring solutions. Unlike classical search, which explores the entire search space, local search focuses on finding solutions within a local region and does not necessarily guarantee the discovery of an optimal solution.

Key Features:

- **No need for a complete search space representation**: Local search algorithms often work directly with the solution rather than a full state space.
- **Iterative process**: They start with an initial solution and make local improvements based on some criterion (usually the objective function).

Types of Optimization Problems:

1. **Combinatorial Problems**: Problems where the solution involves selecting a combination of discrete objects (e.g., the traveling salesman problem).
2. **Continuous Optimization Problems**: Problems where solutions involve continuous values, such as finding the minimum value of a function.

Example:

- **8-puzzle problem**: A state in the 8-puzzle might be considered a solution, and local search algorithms would attempt to find the arrangement that minimizes the number of misplaced tiles or moves.

Algorithms:

1. **Hill Climbing**: A simple local search algorithm where the agent starts with an initial solution and continuously moves to the neighbor with the highest value (or best fit to the goal). The process stops when no better neighbor can be found.
    - **Problem**: Local optima, plateau, and steepest descent issues can hinder the effectiveness of hill climbing.

2. **Simulated Annealing**: A probabilistic technique that allows for occasional moves to worse solutions to avoid getting stuck in local optima, inspired by the annealing process in metallurgy.
   - **Temperature parameter** controls the likelihood of accepting worse solutions: It gradually decreases over time to focus on improving the solution.
3. **Genetic Algorithms**: These algorithms use principles of natural evolution (selection, crossover, mutation) to explore the search space and find good solutions to optimization problems.

3.2 Local Search in Continuous Spaces

In many real-world problems, solutions are not discrete but continuous. In these cases, local search algorithms must adapt to work within **continuous solution spaces**. Instead of discrete actions or states, the agent explores real-valued variables.

Challenges:

1. **Continuous Variables**: The state space is uncountably large, making it difficult to enumerate or discretize.
2. **Function Optimization**: The objective function can be non-linear, noisy, and multimodal, leading to challenges in finding the global optimum.

Algorithms for Continuous Spaces:

1. **Gradient Descent**: A local search algorithm used for optimization problems where the solution space is continuous. It iteratively moves in the direction of the negative gradient of the objective function to find the minimum.
   - **Problem**: It may converge to local minima or saddle points rather than the global minimum.
2. **Nelder-Mead Simplex Method**: A popular optimization method that does not require the gradient of the objective function and works well for many practical problems with continuous variables.
3. **Particle Swarm Optimization (PSO)**: A global optimization algorithm that simulates the social behavior of birds flocking or fish schooling. Each "particle" in the swarm explores the search space and adjusts its position based on its own experience and the experiences of its neighbors.

3.3 Search with Non-Deterministic Actions

In many real-world situations, actions do not always lead to the same result. Non-deterministic actions are those where, after performing an action, the resulting state is not guaranteed and can vary.

Characteristics of Non-Deterministic Actions:

- **Uncertainty**: The agent cannot predict the exact outcome of its actions, which complicates the search process.
- **Stochastic Elements**: Randomness may be involved in the transition between states.

Example:

- In **robotics**, a robot may attempt to pick up an object, but the action might fail due to factors like slippery surfaces or obstacles.

Search Strategies:

1. **Partially Observable Markov Decision Processes (POMDPs)**: An extension of the Markov decision process (MDP) that allows for partial observability of the environment. This model helps agents plan under uncertainty.
2. **Monte Carlo Tree Search (MCTS)**: This algorithm is widely used for decision-making in games like Go, where the action outcomes are non-deterministic. MCTS simulates many random playthroughs and uses statistical analysis to guide the search.

3.4 Search with Partial Observations

In real-world scenarios, agents often operate with incomplete information about the environment, leading to **partial observations**. In such cases, agents must make decisions based on the limited information they have available.

Challenges:

- **Limited Perception**: The agent cannot fully observe the environment and may only see part of the state at any given time.
- **Increased Uncertainty**: The agent must make decisions while dealing with uncertainty about the unobserved aspects of the state.

Solutions:

1. **Partially Observable Environments**: Agents in such environments need to plan under uncertainty by maintaining a belief about the unobserved part of the world.
2. **Hidden Markov Models (HMMs)**: A statistical model used for decision-making in partially observable environments, where the system's state is not directly observable but can be inferred from observations.

Example:

- **Autonomous vehicles**: These vehicles must navigate with partial information, such as incomplete sensor readings or uncertain road conditions, requiring techniques like belief propagation to handle partial observations.

3.5 Online Search Agents and Unknown Environments

An **online search agent** operates in an environment where the agent does not have access to the complete problem description and must act based on its immediate observations. In such environments, the agent needs to make decisions in real-time while exploring the state space and learning about it dynamically.

Characteristics:

- **Limited Information**: The agent must make decisions without having a complete map or model of the environment.
- **Real-time Action**: The agent must act quickly and continually refine its strategy based on new information.

Approach:

1. **Exploration vs. Exploitation**: The agent must balance exploring the environment (gathering information) and exploiting what it has already learned (using known strategies).
2. **Model-free Algorithms**: These algorithms do not require a model of the environment and instead learn from direct interactions, such as Q-learning in reinforcement learning.

Example:

- **Robotic Exploration**: A robot exploring an unknown environment must make decisions based on partial and evolving information, such as when navigating through an uncharted room.

Exercises

1. **Local Search in Optimization**: For the traveling salesman problem, design a local search algorithm using the hill-climbing method. How would the algorithm deal with local optima?
2. **Gradient Descent**: Implement a simple gradient descent algorithm to minimize a quadratic function f(x)=x2+4x+4f(x) = x^2 + 4x + 4. Plot the optimization process.
3. **Non-Deterministic Search**: Imagine you are controlling a drone, and there is uncertainty in the flight control system. How could you adapt a search algorithm to handle non-deterministic actions?
4. **Search with Partial Observations**: Design a simple agent for a maze environment where some walls are hidden. How would the agent act if it can only sense its immediate surroundings?
5. **Online Search**: Suppose you are designing an agent that must make decisions while exploring an unknown terrain. How would you implement an exploration strategy that balances both exploration and exploitation?