Problem Description

The Knapsack Problem is a classic optimization challenge given items with specific weights and values, select a subset to fit into a container (knapsack) with a limited weight capacity, maximizing the total value without exceeding the limit.

Used Algorithms

1. **Hill Climbing Algorithm**
   - Type: Local search (heuristic optimization)
   - Starts with a random or empty solution
   - Iteratively moves to a neighboring solution with a higher value
   - Stops when no better neighbor exists (local optimum)

2. **Uniform Cost Search (UCS)**
   - Type: Uninformed search
   - Expands the node with the lowest cumulative cost
   - Guarantees an optimal solution if costs are non-negative

3. **A* Search Algorithm**
   - Type: Informed search
   - Heuristic Used: Fractional knapsack upper bound heuristic

4. **Genetic Algorithm (GA)**
   - Type: Evolutionary optimization algorithm
   - Population-based search
   - Each chromosome is a binary string representing item selection

Evaluation Methods

To evaluate and compare the algorithms, the following metrics are used:

1. **Solution Quality**
- Total profit achieved
- Validity (weight ≤ capacity)

2. **Execution Time**
- Time taken to reach a solution
- Measured in milliseconds

3. **Optimality**
- Comparison with the known optimal or best-known solution

4. **Scalability**
- Performance with an increasing number of items

5. **Memory Usage**
- Especially important for UCS and A*

Comparison Between the Used Algorithms

The four algorithms used to solve the Knapsack Problem differ significantly in terms of search strategy, optimality, execution time, and scalability.

Uniform Cost Search (UCS)

guarantees finding the optimal solution because it explores the state space based on the cumulative path cost. However, this guarantee comes at a high computational cost. UCS expands many nodes, making it slow and memory-intensive, especially as the number of items increases. As a result, while UCS is theoretically optimal, it is impractical for large knapsack instances.

A*

search improves upon UCS by incorporating heuristic information that estimates the remaining benefit of unexplored items. This guidance allows A* to reach the optimal solution faster than UCS while still maintaining optimality, provided that the heuristic is admissible. Despite this improvement, A* still consumes considerable memory because it stores many nodes during the search process. Its performance is highly dependent on the quality of the heuristic used.

Hill Climbing

follows a completely different approach from a local search algorithm. Instead of exploring the entire state space, it iteratively improves a single solution by moving to neighboring solutions with higher value. This makes Hill Climbing very fast and memory efficient. However, it does not guarantee an optimal solution because it can easily get stuck in local maxima. Its performance also depends on the initial starting state.

Genetic Algorithm (GA)

uses an evolutionary approach that searches through a population of solutions rather than a single path. By applying selection, crossover, and mutation, GA is capable of escaping local optima and exploring a wide solution space. This makes it highly suitable for large-scale knapsack problems where traditional search algorithms become inefficient. However, GA does not guarantee optimality and requires careful tuning of parameters such as population size and mutation rate.

Expected Outcomes

- UCS and A* will find optimal solutions but may struggle with large inputs
- Hill Climbing will be fast but may return to suboptimal solutions
- Genetic Algorithm will provide near-optimal solutions efficiently for large problem sizes
- A* is expected to outperform UCS due to heuristic guidance

Team Contributions

Name	Section	Role
Lojain Elshazly	C11	• Problem Description • Comparison Between Algorithms
Wegdan Mosad	C5	• Genetic Algorithm
Jana Sameh	C2	• Hill Climbing Algorithm
Rana Mabrouk	C2	• A* Search Algorithm
Mariam Mostafa	C11	• Uniform Cost Search Algorithm (UCS)