OPEN ENDED LAB

CS-323

ARTIFICIAL INTELLIGENCE

THIRD YEAR COMPUTER SYSTEM ENGINEERING

BATCH: 2022

GROUP MEMBERS	
IRZA SALAHUDDIN	CS-22001
ESBAH SOHAIL	CS-22008
SUMMIAYA FATIMAH	CS-22014

SUBMITTED TO: MISS HAMEEZA AHMED

DEPARTMENT OF COMPUTER & INFORMATION SYSTEMS ENGINEERING

Course Code: CS-323

Course Title: Artificial Intelligence

Open Ended Lab

TE Batch 2022, Fall Semester 2024

Grading Rubric

Group Members:

Student No.	Name	Roll No.
S1	Irza Salahuddin	CS-22001
S2	Esbah Sohail	CS-22008
S3	Summiaya Fatimah	CS-22014

CRITERIA AND SCALES				Marks Obtained		
				S1	S2	S3
Criterion 1: Has the student appropriately simulated the working of the genetic algorithm?						
0	1	2	-			
The explanation is too basic.	The algorithm is explained well with an example.	The explanation is much more comprehensive.				
Criterion 2: How well is the student's understanding of the genetic algorithm?						
0	1	2	3			
The student has no understanding.	The student has a basic understanding.	The student has a good understanding.	The student has an excellent understanding.			
Criterion 3: How good is the programming implementation?						
0	1	2	3			
The project could not be implemented.	The project has been implemented partially.	The project has been implemented completely but can be improved.	The project has been implemented completely and impressively.			
Criterion 4: How good is the selected application?						
0	1	2	-			
The chosen application is too simple.	The application is fit to be chosen.	The application is different and impressive.				
Criterion 5: How well-written is the report?						
0	1	2	-			
The submitted report is unfit to be graded.	The report is partially acceptable.	The report is complete and concise.				
			Total Marks:			

ABSTRACT

The knapsack problem, a classic combinatorial optimization issue, challenges us to select items that maximize value within a defined weight constraint. Solving this problem with traditional methods becomes increasingly computationally intensive as item count and complexity rise. This study explores an alternative approach by applying a genetic algorithm (GA) to efficiently approximate optimal solutions for the knapsack problem. Genetic algorithms mimic natural evolutionary processes, using mechanisms like selection, crossover, and mutation to iteratively refine solutions. This paper begins with an introduction to GAs and their core operations, and then demonstrates how these principles are adapted to the knapsack problem, covering problem setup, code structure, data handling, and functional flow. Through simulation, we examine the progression of solution quality across generations, with performance analysis underscoring the GA's effectiveness in achieving high-value solutions over time. Ultimately, this research highlights the suitability of GAs for complex optimization challenges and suggests potential improvements to further enhance solution accuracy and efficiency in future work.

Table of Contents

1. INTRODUCTION:

The **Knapsack Problem** is a well-known problem in combinatorial optimization. It involves selecting a set of items with given weights and values to maximize the total value while staying within a weight limit. This problem can be approached using various optimization techniques, and one of the most powerful methods for solving it is through **Genetic Algorithms (GAs).** This report explores the application of Genetic Algorithms to solve the Knapsack Problem, providing an overview of both concepts and explaining how they work together to deliver an optimal or near-optimal solution.

2. THE KNAPSACK PROBLEM:

Problem Description: The basic form of the Knapsack Problem is defined as follows:

- Set of items are provided, each with a specific weight and value.
- A container is provided (the knapsack) with a fixed weight capacity.
- The goal is to select a subset of items that fit within the knapsack's capacity, such that the total value of the selected items is maximized.
- Formally, let 'n' be the number of items,'wi', the weight of item i ,'vi', the value of item i, and W the total weight capacity of the knapsack. The problem can be formulated as:

$$\text{Maximize} \quad \sum_{i=1}^{n} v_i x_i$$

subject to:

$$\sum_{i=1}^{n} w_i x_i \leq W$$

where xi is a binary decision variable that takes the value 1 if item i is included in the knapsack and 0 otherwise.

3. GENETIC ALGORITHM

Overview: A Genetic Algorithm is a search heuristic that mimics the process of natural selection. It is used to find approximate solutions to optimization and search problems. The idea is to simulate the process of **Evolution**, where candidates (solutions) in a population evolve over time to solve a problem optimally.

Working Principle of Genetic Algorithms: Genetic Algorithms work by creating an initial population of possible solutions (chromosomes), selecting individuals based on their fitness, and generating new offspring through crossover and mutation. Over several generations, the population evolves towards better solutions.Key steps in the algorithm included are**:**

- **Initialization:** Start with a random population of solutions.
- **Selection:** Select individuals for reproduction based on their fitness (how well they solve the problem).
- **Crossover (Recombination):** Combine pairs of individuals (parents) to produce offspring.
- **Mutation:** Introduce random changes to an offspring to maintain diversity in the population.
- **Evaluation:** Assess the fitness of the individuals.
- **Termination:** Stop when a satisfactory solution is found or a certain number of generations are reached.

Key Operations in Genetic Algorithms:

- **Selection:** The process of choosing individuals based on their fitness. Common selection methods include Tournament Selection and Roulette Wheel Selection.
- **Crossover:** The process of combining two parent solutions to create one or more child solutions. A common crossover method is one-point crossover
- **Mutation:** A random alteration of a solution to maintain diversity in the population and prevent premature convergence.

4. GENETIC ALGORITHM FOR SOLVING THE KNAPSACK PROBLEM:

Problem Setup: In the context of solving the Knapsack Problem, the Genetic Algorithm is used to find the best subset of items that maximize value while respecting the weight constraint. Each individual in the population represents a potential solution, where:

- A solution is encoded as a binary vector (chromosome), with each element representing whether an item is included (1) or excluded (0).
- The fitness of a solution is determined by its total value, subject to the constraint that the total eight does not exceed the knapsack's capacity.

Explanation of the Code: The provided Python code uses a **Genetic Algorithm** to solve the Knapsack Problem with the following key components:

- Items:
  This class stores the properties of each item (id, weight, and value).

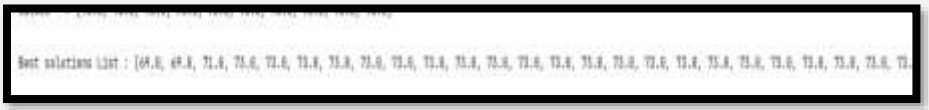
```
class Item:
    def __init__(self, id, weight, value):
        self.id = id
        self.weight = weight
        self.value = value
```

- Functions:
  - **create_random_solution**: Generates a random binary vector representing a potential solution.
  - **valid_solution**: Checks if a solution respects the weight constraint.
  - **calculate_value**: Computes the total value of a solution based on the items selected (those corresponding to 1s in the binary vector).
  - **check_duplicate_solutions**: Ensures no two identical solutions are included in the population.
  - **initial_population:** Generates an initial population of random valid solutions.
  - **tournament_selection**: Implements tournament selection to choose the best individuals for reproduction.
  - **crossover:** Performs one-point crossover between two parent solutions to generate offspring.
  - **mutation:** Mutates a solution by swapping two items in the binary vector.
  - **create_generation:** Creates a new generation using selection, crossover, and mutation.
  - **best_solution:** Finds the solution with the highest value in a generation.
- Genetic Algorithm Flow: The main function, genetic_algorithm, runs the GA, starting with an initial population, iterating through several generations to evolve better solutions.

5.SIMULATION RESULT

- The output of the genetic_algorithm includes the generation number, population, and values. Each generation is labeled, showing the progress of the algorithm. The population is displayed as binary solutions representing item combinations, and the values (fitness) of these solutions are calculated and printed to track improvement over generations

- The best value list tracks the highest value (fitness) achieved in each generation. It stores the most optimal solution found at each step, showing how the best solution evolves and whether the algorithm is improving or stabilizing. By the end, it reflects the progression of the highest values over all generations.



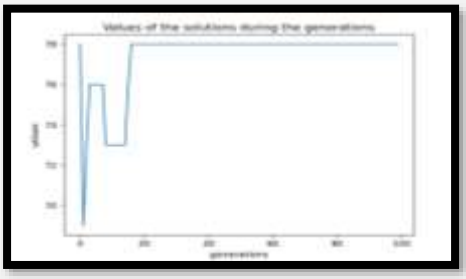
6. RESULTS AND VISUALIZATION:

The GA produces solutions over a number of generations, and the best solution in each generation is evaluated. The progress is visualized through a plot that shows how the solution value evolves over time.

Solution Progress Over Generations: As the GA runs, the value of the best solution typically increases as the population evolves this graph shows how the fitness of the solutions improves as the genetic algorithm progresses through multiple generations, demonstrating the convergence toward an optimal or near-optimal solution.The plot generated by the code illustrates this progression:

```
plt.plot(v_list)
plt.xlabel('generations')
plt.ylabel('values')
plt.title("Values of the solutions during the generations")
plt.show()
```

The Graphical representation of the values of the solutions during the generations will be as follows:



Performance Analysis: The performance of the Genetic Algorithm can be affected by:

- **Population Size**: Larger populations can explore more solutions but take longer to converge.
- **Crossover Rate:** A higher rate can result in more diverse offspring but may also increase the risk of losing good solutions.
- **Mutation Rate:** A higher mutation rate increases diversity but may also disrupt good solutions.
- **Number of Generations:** More generations can lead to better solutions but at the cost of increased computational time.

7. CONCLUSION:

The Genetic Algorithm provides an effective approach to solving the Knapsack Problem, especially when the problem size is large and traditional methods (like dynamic programming) become computationally expensive. The use of selection, crossover, and mutation operations allows the algorithm to explore a broad solution space, and through iterative refinement, it converges on a solution that maximizes value within the weight constraint. This report demonstrated the application of a Genetic Algorithm to the Knapsack Problem, showing how the algorithm evolves potential solutions over several generations. The provided code offers a clear implementation of the GA, and the results validate the effectiveness of this approach in solving optimization problems