EPE472 ARTIFICIAL INTELLIGENCE & DATA MINING

KNAPSACK PROBLEM

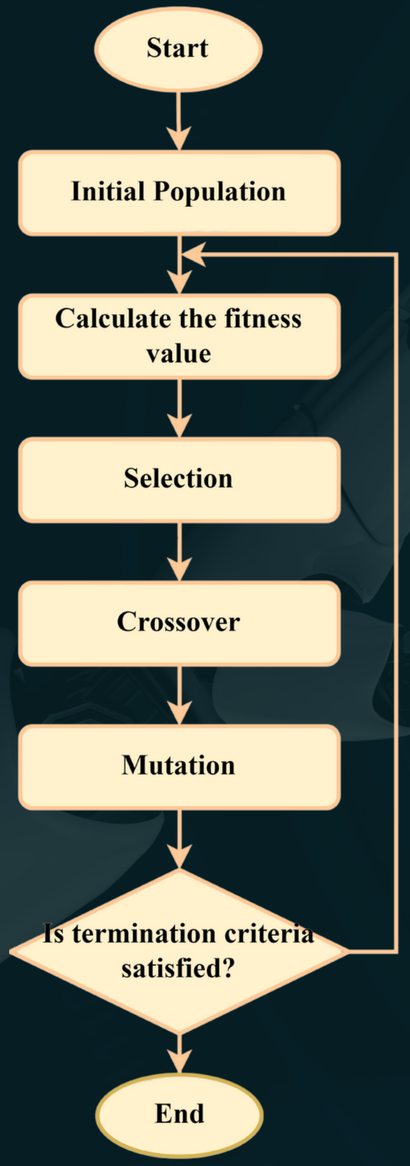
Genetic Algorithm Assignment

Lan Yi Xian	151589
Lee Jia Min	151272
Loke Yan Kuang	152255

USM UNIVERSITI SAINS MALAYSIA

APEX

GENETIC ALGORITHM



- ***Genetic algorithms*** are inspired by Charles Darwin's theory of natural evolution. A genetic algorithm (GA) is an adaptive heuristic search algorithm based on the principles of natural evolution. It employs techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover to find approximate or exact solutions to optimization and search problems. In essence, GAs are used to evolve solutions to problems over successive generations, aiming to improve the quality of solutions incrementally. There are 5 main phases in a genetic algorithm:

- ***Initial Population***: The population consists of a group of individuals, each representing a potential solution. These individuals are defined by a set of variables called genes, which together form a chromosome when combined into a sequence.

- ***Fitness Function***: This function measures how well an individual performs by assigning a fitness score to each one, indicating its suitability as a solution.

- ***Selection***: The process selects the top two pairs of individuals, known as parents, based on their fitness scores. Individuals with higher fitness scores have a greater chance of being chosen to contribute their genes to the next generation.

- ***Crossover***: In this critical phase, a random crossover point is chosen within the genes of each pair of parents. The genes are then exchanged between the parents up to the crossover point, creating new offspring.

- ***Mutation***: Occasionally, genes will undergo mutation, where some bits in the gene sequence are flipped randomly, but this occurs with a low probability.

PROBLEM DESCRIPTION

The Knapsack problem is an example of a combinatorial optimization problem. This problem is also commonly known as the "Rucksack Problem." The knapsack problem can be classified into the following types: Fractional Knapsack Problem, 0/1 Knapsack Problem, Bounded Knapsack Problem, and Unbounded Knapsack Problem. In our case, we are solving the 0/1 knapsack problem. In the knapsack problem, items are not allowed to be broken down into smaller portions. Therefore, we must find the most valuable subset of items that satisfies the constraints. In the 0/1 knapsack problem, we can take objects in integer values. As stated in the question, the knapsack problem must consist of 10 items with a minimum of 3 constraints.

Objective Function

The objective of the knapsack problem in this context is to maximize the total value of selected items while adhering to the specified constraints. The objective function can be formulated as:

$$Maximise \quad \sum_{i=1}^{n} u_i x_i,$$

where u is value of item i, x is a binary variable indicating whether item i is selected (1) or not (0)

Constraints

Weight Constraint

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

Volume Constraint

$$\sum_{i=1}^{n} v_i x_i \leq V,$$

Item Count Constraint

$$\sum_{i=1}^{n} x_i \leq N,$$

where w is weight of item i, v is volume of item i, x is a binary variable indicating whether item i is selected (1) or not (0), W is max. weight limit, V is max. volume limit, N is max. selected items

RELATED EXISTING INDUSTRIAL PROBLEMS

Logistics

Genetic algorithms (GA) applied to the 0/1 knapsack problem help in determining the optimal selection of items to be transported in a way that maximizes the total value without exceeding the capacity constraints. This approach allows logistics companies to improve their cargo loading strategies by selecting the most valuable combination of items that fit within their transport capacities. This helps in reducing transportation costs while maximizing profit.

Ekon, N. (2022, December 26). Optimizing the Loading of Items onto Trucks using Genetic Algorithms in an Improved Knapsack Problem. Medium. https://medium.com/@nahidekon314/optimizing-the-loading-of-items-onto-trucks-using-genetic-algorithms-in-an-improved-knapsack-ec1f11cc5dad

Production Planning

The goal is to minimize production costs while efficiently planning the furnace loading and the subsequent production of items. Optimizing production planning involves two key stages: selecting the alloys to be merged and determining the lots to be produced from these alloys. The knapsack problem is used to model the selection of items to be produced from the alloys loaded into the furnace. Each item has a specific weight (resource requirement) and value (profit or cost saving). The objective is to maximize the total value (minimize the cost) without exceeding the furnace capacity.

Camargo, V. C., Mattiolli, L., & Toledo, F. M. (2012). A knapsack problem as a tool to solve the production planning problem in small foundries. Computers & Operations Research, 39(1), 86–92. https://doi.org/10.1016/j.cor.2010.10.023

Capital Budgeting

Selecting the optimal set of projects to invest in, given budget constraints and the need to maximize returns on investment. Genetic algorithms are used to solve the knapsack problem in project selection and capital budgeting by evaluating various combinations of projects to identify the set that maximizes returns while staying within budget limits. This method allows organizations to make data-driven decisions on which projects to prioritize for funding [3].

Pendharkar, P. C., & Rodger, J. A. (2006). Information technology capital budgeting using a knapsack problem. International Transactions in Operational Research, 13(4), 333–351. https://doi.org/10.1111/j.1475-3995.2006.00551.x

FITNESS FUNCTION

The ***knapsackFitness*** function will evaluate how good each solution is. It needs to account for the total weight, volume, and value of the items selected by each chromosome. Solutions that exceed weight, volume, or item count constraints should be penalized.

The function takes several input parameters for fitness evalution, as listed below:

- ***population***: A matrix where each row represents an individual solution encoded as a binary string. Each column corresponds to an item's inclusion (1) or exclusion (0) in the knapsack. 100 populations are randomly generated in our case.
- **weights**: A 1 x 10 array that stores the weight of each item.
- **volumes**: A 1 x 10 array that stores the volume of each item.
- **values**: A 1 x 10 array that stores the value of each item.
- **maxWeightLimit**, **maxVolumeLimit**, **maxItemsSelected**: Define the constraints for the knapsack problem, limiting the total weight, volume, and number of items that can be selected in the case.

The fitness values are used during the selection process to choose individuals for reproduction. Solutions with higher fitness have a higher chance of passing their genes to the next generation, driving the population towards optimal solutions over successive generations.

```
%% Function for Fitness Evaluation
function fitness = knapsackFitness(population, weights, volumes, ...
    values, maxWeightLimit, maxVolumeLimit, maxItemsSelected)

    % Initialize the array to store fitness values
    fitness = zeros(size(population, 1), 1);

    % Calculate the sum of weights, volumes and values
    for i = 1:size(population, 1)
        totalWeight = sum(population(i, :) .* weights);
        totalVolume = sum(population(i, :) .* volumes);
        totalValue = sum(population(i, :) .* values);

        % Crosscheck the sum with the problem constraints
        if totalWeight <= maxWeightLimit && totalVolume <= ...
            maxVolumeLimit && sum(population(i, :)) <= maxItemsSelected
            fitness(i) = totalValue;
        else
            % Penalize the infeasible solutions
            fitness(i) = 0;
        end
    end
end
```

SOLUTION ENCODING

In this case of ***0/1 Knapsack Problem***, the items picked is non-replaceable and non-fractional, indicates one full items must be picked with respects to the weight and volume.

The demonstration of ***binary encoding*** is shown below:

- For this case, the knapsack problem consists of 10 items. The individual values for each arrays as shown above, represented the properties of each item.
- Then a population matrix is created using the command ***"randi([0, 1], populationSize, nItems)"***. Each item's inclusion in the knapsack is represented as a binary string, where '1' means included and '0' means excluded. The example of binary encoding for this case is shown on the right.
- Taking one binary encoded individual chromosome for further explanation. Since the population size is 100, we have 100 chromosomes with 10 genes, building up the initial population of 100 x 10 matrix.
- Using the result for Case 2 as explanation for solution encoding, we can that the best solution is ***[1, 1, 0, 1, 0, 0, 0, 1, 0, 0]***. Multiplying the individual genes with their respective properties (weight & values). We can get the fitness value of ***20kg*** and ***50 m^3*** respectively.

```
% Define data for each items
weights = [5, 1, 9, 9, 6, 5, 2, 5, 6, 7];
volumes = [12, 12, 17, 14, 15, 12, 19, 12, 20, 13];
values = [62.30, 32.53, 59.38, 65.12, 48.67, 14.36, 69.52, 84.71, 6.58, 46.99];
nItems = length(weights);
```

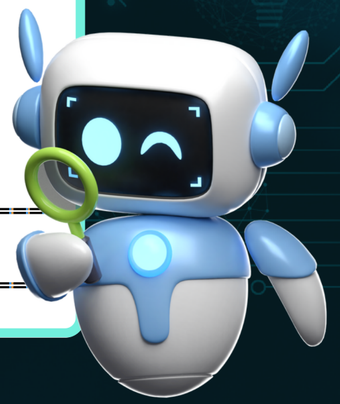
population = ***binary encoding (0 - Not Selected / 1 - Selected)***

0	0	1	0	0	1	0	0	0	1
1	0	0	1	1	0	0	1	1	1
0	1	0	0	1	0	1	0	0	0
1	0	1	1	0	1	0	0	0	1

```
================================================
     KNAPSACK PROBLEM (GENETIC ALGORITHM)
================================================

Best solution       : [1 1 0 1 0 0 0 1 0 0]
Total weight        : 20 kg
Total volume        : 50 m^3

================================================
Maximised Value     : RM 244.66
================================================
```

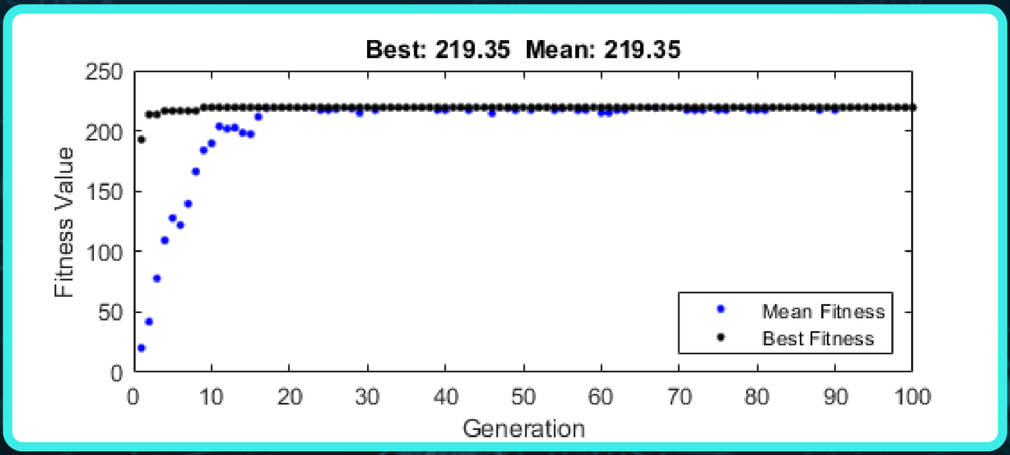


```
% Define general parameters
populationSize = 100;
maxGeneration = 100;
tournamentSize = 2;
crossoverRate = 0.7;
mutationRate = 0.01;
eliteCount = 0.05 * populationSize;
```

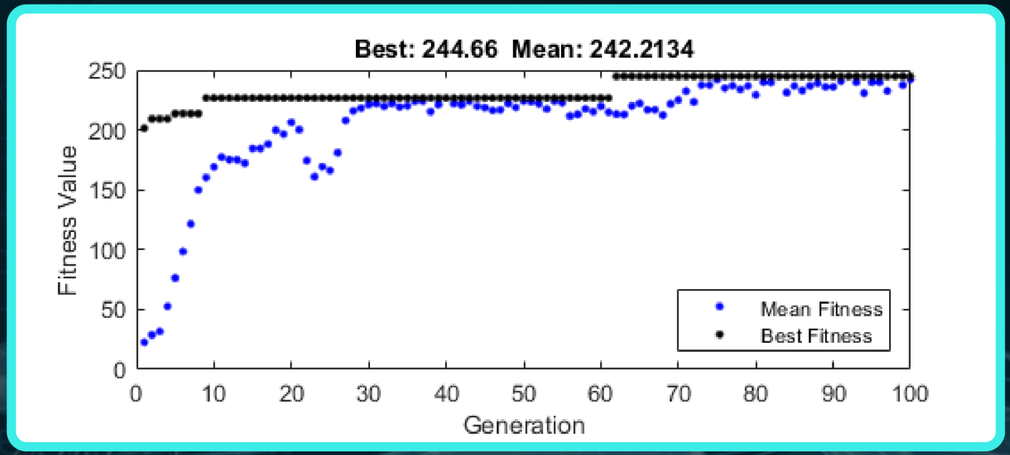
COMPARISON OF 2 CASES

Pc = Crossover Rate
Pm = Mutation Rate

Case 1 (One Point Crossover, N = 100, Pc = 0.7, Pm = 0.001)

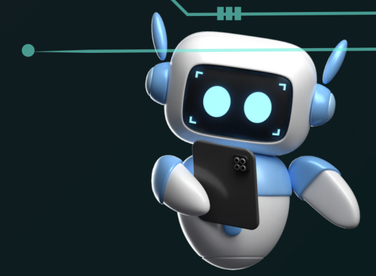


Case 2 (Two Point Crossover, N = 100, Pc = 0.7, Pm = 0.01)



- The comparison between the one-point and two-point crossover mechanisms reveals significant differences in their effectiveness and outcomes.
- With a crossover rate of 0.7 and a mutation rate of 0.001, the ***one-point crossover*** achieved a ***local maxima*** *fitness score* of 219.35, indicating stability but a tendency to get stuck in suboptimal solutions.
- On the other hand, the ***two-point crossover***, with the similar crossover rate but a higher mutation rate of 0.01, achieved a higher fitness score of 244.66. This suggests that the two-point crossover mechanism, coupled with a higher mutation rate, allows for greater genetic diversity and exploration of the solution space, ultimately leading to better optimization performance and the achievement of ***global maxima***.
- Thus, the two-point crossover mechanism is superior due to its enhanced ability to explore and exploit potential solutions effectively.

DISCUSSION



Case 1

```
================================================
     KNAPSACK PROBLEM (GENETIC ALGORITHM)
================================================

Best solution       : [0 0 0 1 0 0 1 1 0 0]
Total weight        : 16 kg
Total volume        : 45 m^3

================================================
Maximised Value     : RM 219.35
================================================
```

Case 2

```
================================================
     KNAPSACK PROBLEM (GENETIC ALGORITHM)
================================================

Best solution       : [1 1 0 1 0 0 0 1 0 0]
Total weight        : 20 kg
Total volume        : 50 m^3

================================================
Maximised Value     : RM 244.66
================================================
```

Effectiveness of Crossover Mechanisms

- The two-point crossover mechanism proved to be more effective than the one-point crossover in achieving higher fitness scores. This indicates that using two crossover points allows for a more diverse combination of genetic material, enhancing the exploration of the solution space.

Impact of Mutation Rates

- The mutation rate plays a crucial role in the optimization process. The higher mutation rate in the two-point crossover mechanism introduced more variability, which helped in avoiding local optima and reaching a global maxima. In contrast, the lower mutation rate in the one-point crossover resulted in more stable but potentially suboptimal solutions.

Optimization Performance

- The overall optimization performance was significantly better with the two-point crossover mechanism. The combination of a higher mutation rate and a more complex crossover mechanism allowed the algorithm to explore a broader range of solutions, leading to a higher fitness score. This demonstrates the importance of fine-tuning both crossover and mutation parameters to achieve optimal results in genetic algorithms.