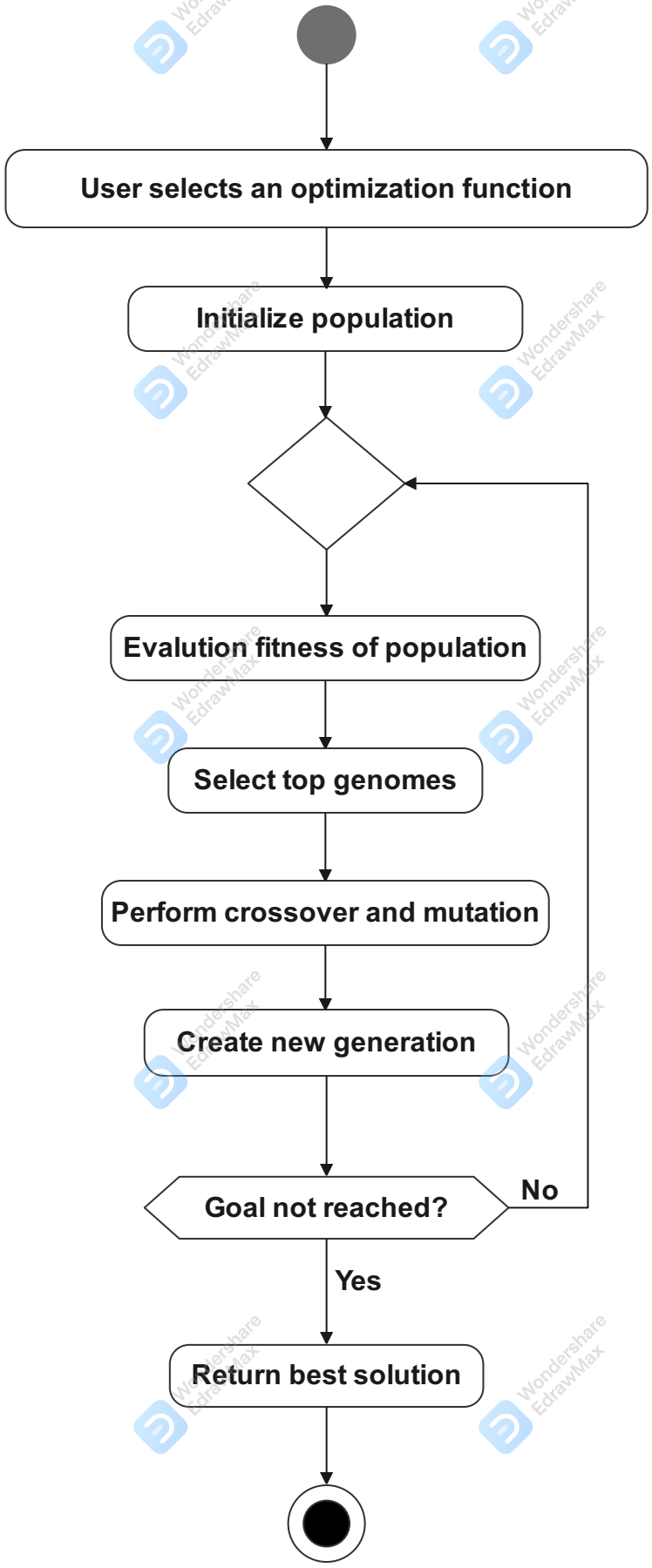
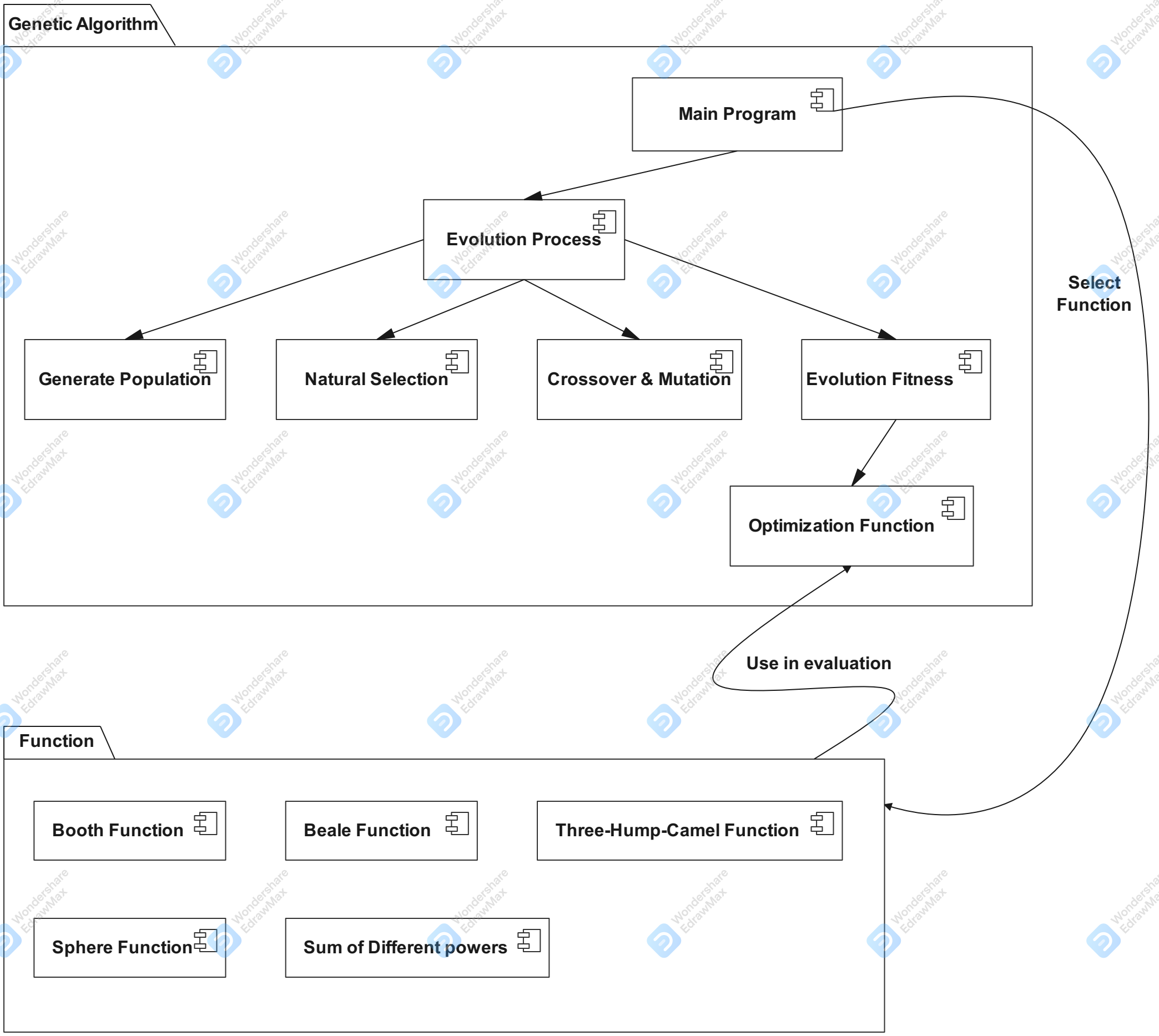
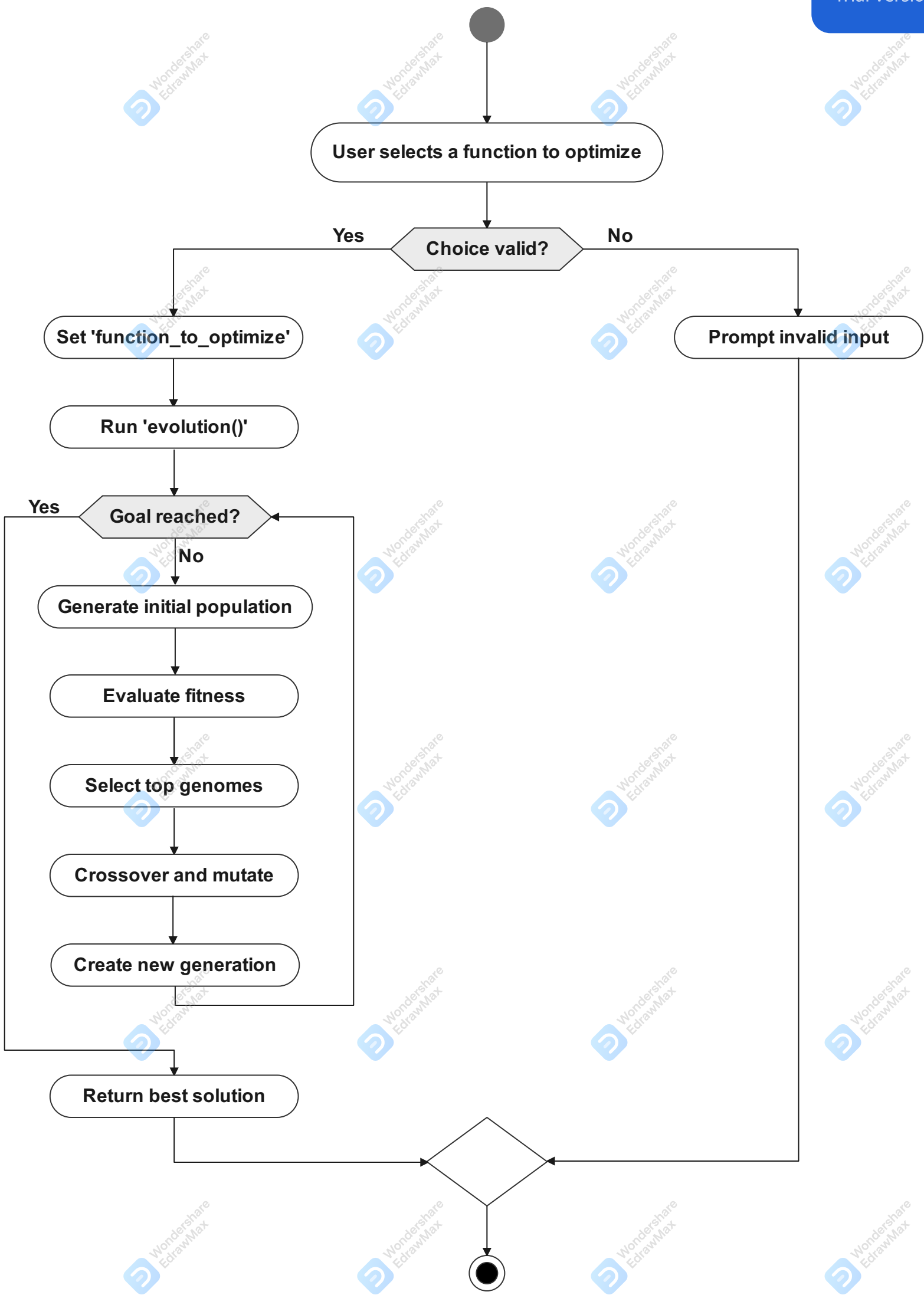
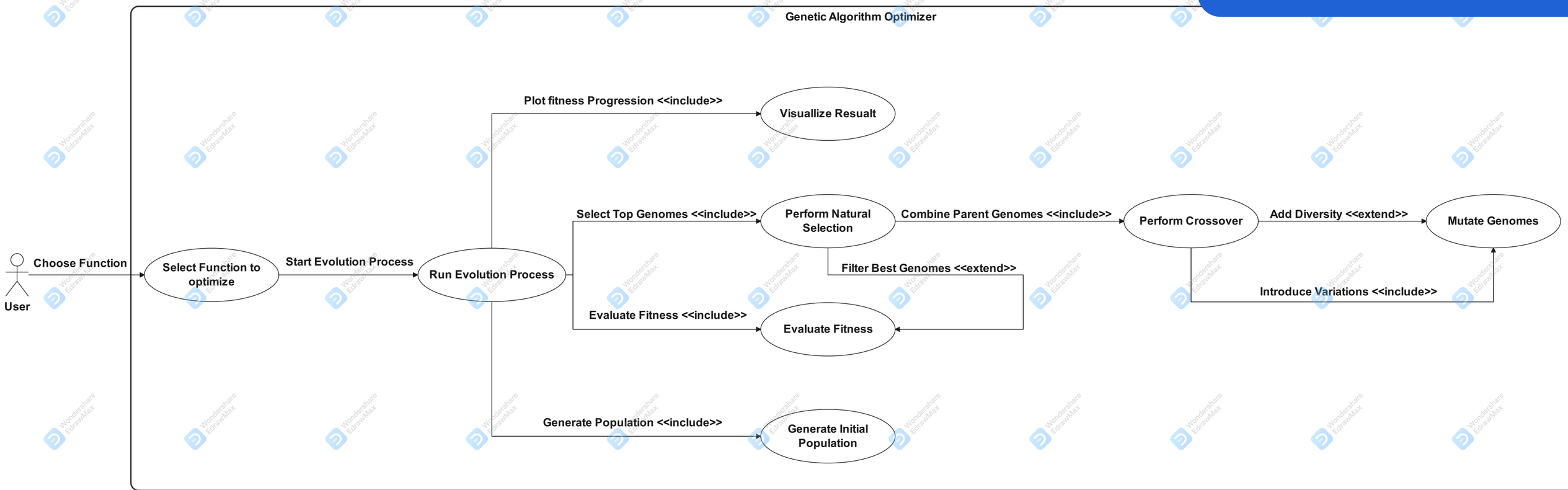








AI310: Artificial Intelligence – Fall 2024

Course Project Documentation – Team No. 14

Members:

- Marwan Ahmed Samir – مروان احمد سمير – ID: 20220456
- Omar Abdelnasser Sayed - عمر عبدالناصر السيد – ID: 20220321
- Mohamed Ezzat Mostafa - محمد عزت مصطفى – ID: 20220417
- Mohamed Rashed Ramah - محمد راشد رماح – ID: 20220397
- Mohamed Ali Mohamed - محمد علي محمد – ID: 20220421
- Youssef Mohamed Abdulazim - يوسف محمد عبدالعظيم – ID: 20220576

Project Description

Our project is project #20 – "Genetic Algorithms for Function Optimisation". We implemented a genetic algorithm to find the global minimum for 5 benchmark optimisation functions. We used the following 5 functions:

1. Booth function
2. Three-hump-camel function
3. 2-D Sum of Different Powers function
4. 2-D Sphere function
5. Beale function

We obtained these functions from https://www.sfu.ca/~ssurjano/optimization.html. Functions 3 and 4 are "D-dimensional" so we acquired their equations by setting D = 2.

The genetic algorithm works as follows:

1. Creates an initial population (gen 0) with random values.
2. Selects the best genomes from this population according to a fitness function
3. Perform a crossover (breeding) using the best genomes
4. Mutate each genome for variety & a new generation is obtained based on the previous one.
5. Repeat steps 2-4 until our fitness function obtains a fitness greater than or equal to our goal.

The 'goal' value for the fitness can be changed. Our experiments found that the higher the goal, the more optimised our solution will be, but there is a tradeoff wherein the algorithm will take longer to reach a solution, and more generations are required. We found that a goal value of 200,000 was a good balance between execution time and accuracy of the value (most of the time, the solutions are correct to 3 decimal places)

Once an optimised solution is reached, a scatter plot showing the fitness against the number of generations is displayed, to visualise the progress made by the algorithm.

Experiments on Parameters

We will use the below equation to measure the accuracy of the solution found when compared to the optimal solution, in order to observe the effect that changing certain parameters has on the accuracy of the algorithm.

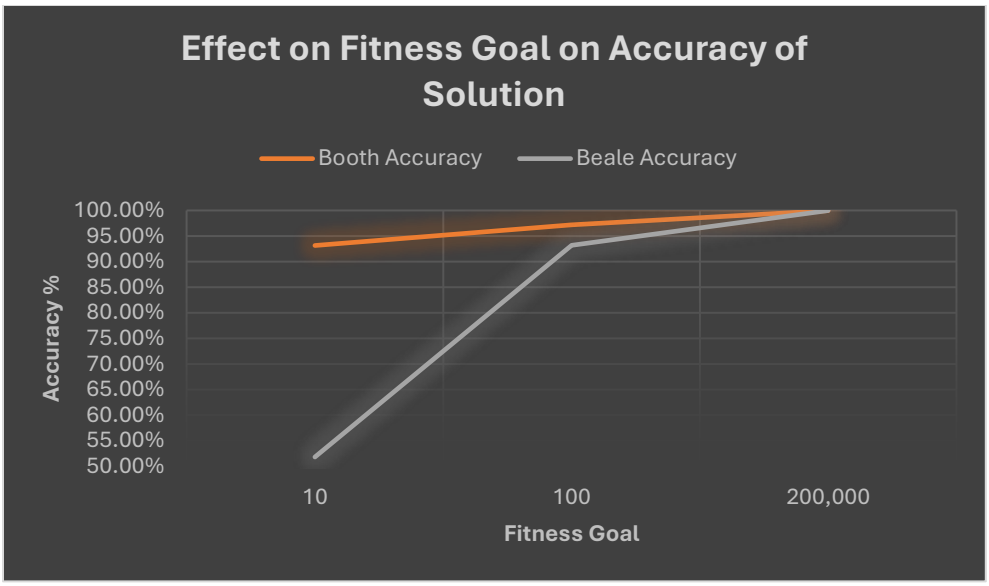
$$Accuracy\ \% = 100\ \times \left(1 - \frac{\|Solution\ Found - Optimal\ Solution\|}{\|Optimal\ Solution\|}\right)$$

Default values of parameters:

- Goal = 200,000
- Population size = 1,200
- Mutation factor range = 0.99 – 1.01 (to minimise effect of mutation)

Goal value experiment

Function	Fitness Goal	Generations	Best fitness	Solution found	Optimal solution	Accuracy of solution found
Booth	10	189	10.56	[1.197..., 2.912...]	[1, 3]	93.16%
	100	367	125.93	[1.0654..., 2,94091....]		97.21%
	200,000	601	282,914	[0.99966, 3.001086]		99.97%
Beale	10	109	10.2	[4.448, 0.71]	[3, 0.5]	51.84%
	100	808	105.97	[2.8016, 0.437]		93.17%
	200,000	1,247	651,390	[2.999, 0.5001]		99.98%

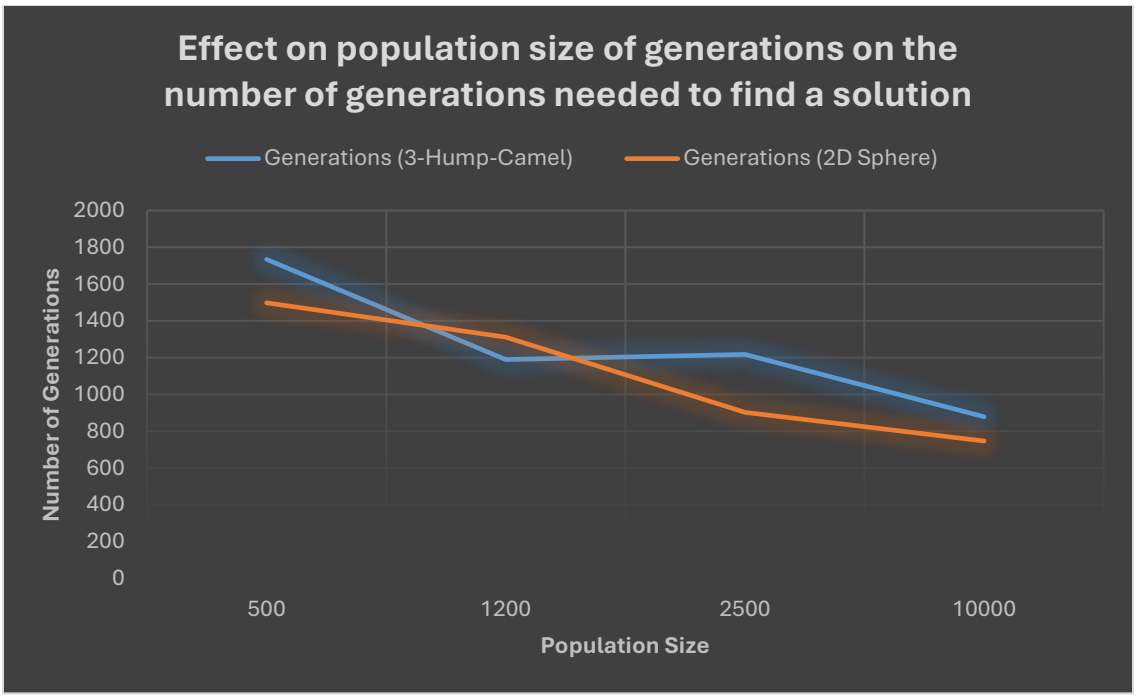


From this experiment, we can therefore conclude that the higher the fitness goal, the more accurate the solution. However, there is a point at which we receive diminishing returns as we increase the fitness goal – the algorithm will take a lot longer and iterate through more generations for a very small increase in the accuracy.

Population size experiment

This experiment involves changing the size of the initial random population (generation 0) as well as the size of every subsequent generation that is based on the previous one.

Function	Population size	Generations needed for a solution
Three-hump-camel	500	1,734
	1,200	1,190
	2,500	1,217
	10,000	879
2D Sphere	500	1,498
	1200	1,311
	2,500	902
	10,000	747



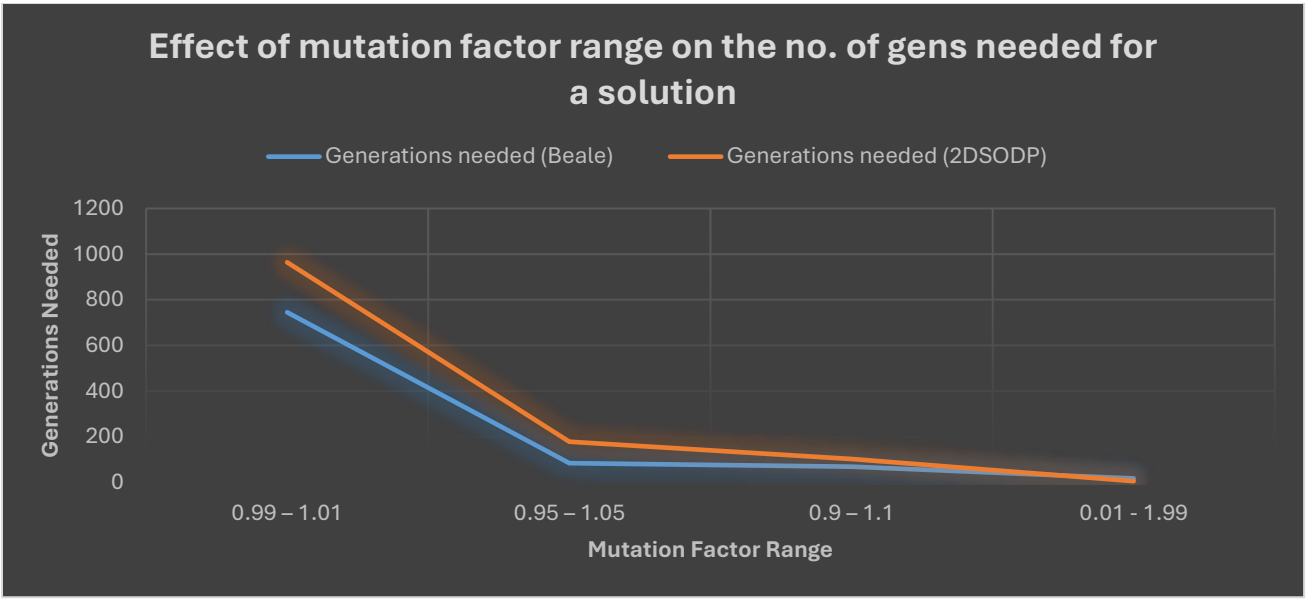
Because the goal stays constant at 200,000, the solution in each case is almost exactly the same. The significant observation we can make is that a higher population size results in fewer generations needed to find a solution. This, in turn, affects the execution time of the algorithm. But just like the previous experiment, there will be a point at which we receive diminishing returns and any further increase in the population size won't be worth it. At a population size of 10,000, even though it took fewer generations to find a solution, each generation took a longer time and the time to find a solution was significantly longer.

Mutation factor experiment

When a new genome is generated, it is multiplied by a random float between a certain range of numbers. This means that it can slightly decrease, stay the same, or slightly increase. This helps achieve variety in the so-called "gene pool". In this experiment we changed the range of numbers the mutation factor can be to observe what effect it had on the number of generations needed to find a solution.

Function	Mutation factor range	Generations needed for a solution
Beale	0.01 – 1.99	17
	0.9 – 1.1	68
	0.95 – 1.05	84
	0.99 – 1.01	745
2D Sum of Diff. Powers	0.01 – 1.99	5
	0.9 – 1.1	102
	0.95 – 1.05	178
	0.99 – 1.01	964

From this data, we can conclude that the larger the mutation factor range, the fewer generations needed for a solution. This is likely because the larger variety among genomes meant some genomes get "luckier" than others and the mutation factor they were multiplied by, brings them much closer to the optimal solution than a narrower range would. A narrower range such as 0.99 – 1.01 means that mutation has a far less significant effect on the evolution process. However, this is not the case for every function. More experimentation would be needed to further prove this relation.



Similar Applications in the Market

Genetic algorithms are often used where traditional algorithms fail due to non-linearity, multiple local optima & high computational cost. The following are some more advanced applications of GAs that exist in the market:

Name	Description
Engineering Design Optimisation	• Designing lightweight yet strong structures in aerospace and civil engineering • Optimising parameters for gears, turbines, engines etc to achieve the best possible efficiency
Machine Learning/AI	• Optimising parameters for ML models like neural networks, SVMs & decision trees
Economics & Finance	• Optimising a portfolio by selecting an optimal combination of financial assets to balance risk & return
Energy Systems	• Optimising the location of wind turbines/solar panels to maximise energy production
Game Development	• Training AI for NPCs in games

	• Generating game levels that adapt to player abilities

Literature review

The following 6 papers are relevant to Genetic Algorithms and/or global optimisation problems:

1. Genetic Algorithm – an Approach to Solve Global Optimization Problems (Pratibha Bajpai et al., 2010)
    a. This paper highlights genetic algorithms (GAs) as robust, nature-inspired methods for solving global optimization problems. GAs use populations, selection, crossover, and mutation to explore search spaces effectively, avoiding local optima. The authors discuss their adaptability, wide applications, and factors influencing performance, emphasizing their utility for complex optimization tasks.
2. Genetic Algorithms for Function Optimisation (Brindle, 1980)
    a. This study introduces genetic algorithms for function optimization, focusing on their ability to solve complex, non-linear, and multi-modal problems. The paper explores key components of GAs, such as selection, crossover, and mutation, emphasizing their flexibility and efficiency in identifying optimal solutions in challenging search spaces.
3. Study of Genetic Algorithm for Optimization Problems (Azevedo, 2020)
    a. Examines genetic algorithms for solving optimization problems, emphasizing their adaptability and effectiveness in handling complex, high-dimensional, and multi-modal challenges. The paper discusses the role of selection, crossover, and mutation in driving solution convergence while maintaining diversity in the search space.
4. A review on genetic algorithm: past, present, and future (Katoch, Chauhan & Kumar, 2020)
    a. This review explores the evolution of genetic algorithms, highlighting advancements in their design and applications. The paper examines core components like selection, crossover, and mutation, discusses current trends in hybridization and parallelism, and outlines future directions for improving GAs' efficiency and scalability in optimization problems.
5. Applying Genetic Algorithms to Optimization Problems in Economics (Nicoară, 2015)
    a. This study explores the application of genetic algorithms (GAs) to economic optimization problems, demonstrating their ability to handle complex, dynamic, and multi-criteria scenarios. The paper highlights the adaptability of GAs in economic modeling, focusing on their efficiency in finding optimal solutions in non-linear and constrained environments.

6. A genetic algorithm for solving large scale global optimization problems (M L Shahab et al 2021)
    a. This study presents a genetic algorithm (GA) tailored for large-scale global optimization problems. The paper emphasizes modifications to traditional GA components, such as selection and mutation, to enhance scalability and efficiency. The proposed approach demonstrates effectiveness in solving high-dimensional and computationally intensive problems.

Dataset employed

This project did not require a dataset, instead we utilised benchmark optimisation test functions that are publicly available at https://www.sfu.ca/~ssurjano/optimization.html

- Booth function: $f(x,y) = (x + 2y - 7)^2 + (2x + y - 5)^2$
- Three-hump-camel function: $f(x,y) = 2x^2 - 1.05x^4 + x^6 + xy + y^2$
- 2-D Sum of Different Powers function: $f(x,y) = x^2 + y^3$
- 2-D Sphere function: $f(x,y) = x^2 + y^2$
- Beale function: $f(x,y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$

Development platform

This project was solely developed on Colab using Python 3.9.13 and the following libraries:

- 'random' – For random numbers, used to generate an initial population, crossover parent genomes & mutate genomes.
- 'typing' – To define what a Genome & Population are in more readable terms, as well as make the Fitness function as well as the benchmark optimisation functions usable to other functions
- 'matplotlib.pyplot' – To plot our results (generation # plotted against fitness of best genome of that generation)
- 'os' – Solely used to clear the terminal of previous function's results for better usability.