PERFORMANCE COMPARISON OF GENETIC ALGORITHM WITH PARTICLE SWARM OPTIMISATION USING BENCHMARK FUNCTIONS

By

Brandon Ting En Junn

Ling Ji Xiang

Loh Chia Heung

Yeap Chun Hong

A PROPOSAL

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

MAY 2024

ACKNOWLEDGEMENTS

We would like to express our sincere thanks and appreciation to our supervisor, Ts Dr Lim Seng Poh who has given us this bright opportunity to engage in an evolutionary computation project. It is our first step to establish a career in the soft computing field. A million thanks to you.

ABSTRACT

Written By: Brandon Ting En Junn 2101751

Optimisation problems are real world problems that require extensive calibration and fine-tuning in order to achieve maximum efficiency for maximising the throughput. An example of this can be related to the manufacturing industry. Evolutionary computation algorithms such as the Genetic Algorithm (GA) and Particle Swarm Optimisation (PSO) are more efficient in solving optimisation problems rather than traditional algorithms such as gradient descent, linear programming, and dynamic programming. The research is to provide a comprehensive comparison of the performance between GA and PSO models using 10 standardised benchmarks functions as the minimisation optimisation problems. The models were evaluated with the performance metrics of average lowest fitness and average time. Four techniques of each selection, crossover, mutation, and replacement operation techniques were implemented to produce 256 unique combinations of GA models. The results of the experiment showed that the best GA model out of the 256 GA models had the operation techniques combination of Dynamic Tournament Selection, Uniform Crossover, Hybrid Comparing Mutation, and Combined Replacement. It had outperformed on a total of three benchmark functions with the best average lowest fitness out of all the 256 GA models. The PSO model that consisted of the parameter settings such as constant inertia weight, *w* and velocity clamping, *vc* was to be compared with the best GA model. The comparison results showed that the PSO model outperformed the best GA model by dominating on seven benchmark functions for the best average lowest fitness, while the best GA model dominated the PSO model on all 10 benchmark functions for the lowest average time. In conclusion, PSO performed better than GA due to its significance in continuous search spaces.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

w	Inertia Weight
vc	Velocity Clamping
c_1, c_2	Acceleration Constant
r_1, r_2	Random Numbers
P_{best}	Personal Best Position
G_{best}	Group Best Position
P1, P2	Parent Chromosome
T1, T2, T3, T4	Temporary Chromosome
C1, C2	Child Chromosome

LIST OF ABBREVIATIONS

GA	Genetic Algorithm
PSO	Particle Swarm Optimisation
OPF	Optimal Power Flow
ORPD	Optimized Reactive Power Dispatch
TS-GA	Tournament Selection Genetic Algorithm
TSP	Traveling Salesman Problems
OBX	Order Based Crossover
MOC	Modified Order Crossover
PMX	Partially Mapped Crossover
UX	Uniform Crossover
UNDX	Uniform Normal Distribution Crossover
EMGG	Enhanced Minimal Generation Gap
MISC	Mutual Information and Shuffle Crossover
SGA	Simple Genetic Algorithms
MSE	Mean Square Error
DRS	Dual Response System
NHH	Hopfield Neural Network
DHM/ILC	Dynamic Decreasing of High Mutation Ratio/Dynamic Increasing of Low Crossover Ratio
ILM/DHC	Dynamic Increasing of Low Mutation/Dynamic Decreasing of High Crossover
SIM	Simple Inversion Mutation
SSGAs	Steady-State GAs
PWR	Pressurized Water Reactor
S, C, M, R	Selection, Crossover, Mutation, Replacement
NSGA-II	Non-Dominated Sorting Genetic Algorithm II

CHAPTER 1

Introduction

Written By: Brandon Ting En Junn 2101751

This chapter includes the problem statement and motivation, project scopes and objectives, contributions to the field of evolutionary computation, and the overall report organisation.

1.1 Problem Statement and Motivation

Written By: Loh Chia Heung 2301684

Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) are widely recognized tools for solving difficult optimization problems across many fields. For example, GA is widely applied in chemistry, mathematics, economics, and bioinformatics [2]. Meanwhile, PSO has proven effective in solving complex problems such as the Optimal Power Flow problem [1], Min-Max problem, Dynamic Tracking, and Multi Objective Optimization, which are relevant in data mining, signal processing, telecommunication and many more [2].

Despite their widespread use, previous studies on GA and PSO have often been limited in scope or focused on specific applications, rather than offering a comprehensive performance comparison. For instance, according to [3], it outlines the nature of GA and PSO, their handling of discrete and continuous problems, their efficiency in dealing with complexity, and their tendencies in terms of convergence. However, these comparisons lack a thorough performance evaluation across diverse optimization tasks, particularly using standardized benchmark functions. Such benchmark functions are important as they are often used to test the performance of any optimization problem as they represent various real-world scenarios and are commonly used to assess algorithmic performance [4].

Additionally, Papazoglou and Biskas [1] explored the strengths and weaknesses of GA and PSO in the context of the Optimal Power Flow (OPF) problem. While their research highlights the performance of these algorithms in a specific scenario, it does not extend to a generalized performance evaluation across different problem types using standardized benchmark functions. As mentioned before, such functions are critical for generalizing performance outcomes and offer deeper insights into the strengths and limitations of the optimization algorithms being tested [5].

Therefore, this project aims to provide valuable insights through a comprehensive performance comparison of GA and PSO using 10 standardised benchmark functions to minimize the optimisation problem and address the research gap. The GA and PSO models will be tested with these benchmark functions to evaluate their performance based on the fitness values. This analysis will investigate the relative strengths and weaknesses of these algorithms, offering guidance for their application in various optimization problems.

1.2 Objectives

Written By: Ling Ji Xiang 2104584

The goal of this project is to evaluate and compare the performance comparison between best performed GA with best combination of S, C, M, R techniques and PSO using 10 benchmark functions. The experiment will be conducted at least 10 times to obtain more accurate results to do the analyzing. The outcome will provide an idea of effectiveness of both GA and PSO to optimize the benchmark functions. The objectives of this project are:

1. To implement GA operators to optimize benchmark functions:
    - Conduct a comprehensive literature review for researching the given operation techniques in previous work.
    - Develop and implement a GA model with operator techniques such as selection, crossover, mutation and replacement.
    - Apply the GA component to a set of benchmark functions to evaluate its optimization performance.
2. To evaluate and compare the performance between GA and PSO:
    - Implement a PSO model with constant inertia weight (*w*) and velocity clamping (*vc*) to optimize the same set of benchmark functions.

- Conduct a minimum of 10 experiments for each benchmark function using best performed GA and PSO models to get the experiment result.
- Analyze and compare the performance of GA and PSO based on metrics such as average fitness value and average computational time taken to run the benchmark functions.
- Provide insights and conclusions from the comparative analysis to highlight the strengths and weaknesses of each optimization technique

1.3 Project Scope

Written By: Ling Ji Xiang 2104584

In this project, PSO with constant weight (*w*) and constant velocity clamping (*vc*) was used to do the performance comparison with the best performed GA operation techniques using 10 benchmark functions. The system will be developed using C++ programming language and each member contributes the task together using Microsoft Visual Studio Code as programming platform and GitHub to synchronize work progress.

The component of GA in the system will implement operation techniques which are provided and further research through a thorough literature review from previous work articles. Techniques such as Selection, Crossover, Mutation and Replacement will be applied in GA. In the previous group contribution, 2 methods for each technique have been delivered and tested. In this project, another 2 new methods for each technique will be further research and develop. One develops through research and gets the idea from previous work and the other is each member's own idea. So, in total 4 methods will be developed for each technique, only 1 method for each technique will be used in final comparison. Each combination will be tested to obtain the best GA in the form of best S, C, M, R combinations to have best performance.

The PSO component will utilize a model with a constant inertia weight (*w*) = 0.7 and velocity clamping (*vc*) = 2. This will include implementing the PSO algorithm to optimize the same set of benchmark functions to provide the data for comparison with GA.

The program will run and record the experiment results. GA and PSO will be tested to optimize 10 benchmark functions such as Sphere function, Ackley function, Rastrigin function and so on. Furthermore, both GA and PSO algorithms will be run on benchmark functions with each experiment conducted at least 10 times to ensure robust and reliable results. In this project, the termination condition is set to 2000 iterations. The best solution found, G_{best}, is returned as the optimal solution. The performance metrics such as average fitness value and average computational time taken will be collected and analyzed.

In conclusion, this project focus on the detailed performance comparison between best performed GA operation techniques combinations with PSO using 10 benchmark functions. The experiment result will offer insights into which optimization algorithm is more effective under various problem.

1.4 Contributions

Written By: Yeap Chun Hong 2206352

This project had provided a comprehensive comparison of GA and PSO performance by using different existing operation techniques and one proposed technique for each operator in GA to compared with PSO with constant weight and constant velocity clamping. The future researchers can use as a reference point to apply the best method found in this study to solve the real-world problem. Furthermore, the study seeks to identify the optimal GA model by analysing the fitness value of each GA models so that the findings can increase the efficiency for future researchers to obtain the appropriate GA models that is used in the research work later. For example, applying the best performed GA model in this study for cost optimising problem in logistic sector.

The use of 10 standard benchmark functions makes this project significant as the results are robust, reliable and comparable with other studies. Therefore, this study will contribute new insights into the behaviour and optimisation of evolutionary computation.

1.5 Report Organisation

Written By: Brandon Ting En Junn 2101751

The report is organised as follows:

- **Chapter 2 – Literature Review**

  The chapter covers the literature review and discussion of previous works related to the research.

- **Chapter 3 – System Design**

  The chapter elaborates the proposed design regarding the methodologies, techniques, and parameter settings that were implemented to conduct the experiments.

- **Chapter 4 – System Evaluation and Discussion**

  The chapter shows the results that had been obtained from the experiments to conduct further discussion and analysis.

- **Chapter 5 – Conclusion and Recommendation**

  The chapter summarises our findings and final remarks of the research.

CHAPTER 2

Literature Review

Written By: Brandon Ting En Junn 2101751

This chapter covers the reviews of previous works that contribute to the research by discussing the strengths and weaknesses of each different operation techniques being used.

2.1 Genetic Algorithm

2.1.1 Selection Operation

Written By: Yeap Chun Hong 2206352

According to [6], selection operation is also known as reproduction operation. It is the crucial stage where two of the chromosomes are selected as parents from the existing population in order to breed their offsprings in the following stages. The selection imposes the survival of the fittest where the chromosome that have a higher fitness value are more likely to be selected for maximisation while lower fitness for minimisation. Therefore, the selection pressure affects the convergence rate of GA, where a higher selection pressure leads to higher convergence rate. In this study, the selection techniques used are Roulette Wheel Selection, Tournament Selection and Linear Ranking Selection.

Roulette Wheel Selection

Written By: Yeap Chun Hong 2206352

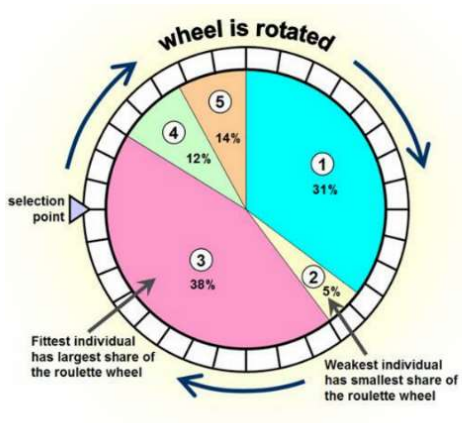

wheel is rotated
selection point
1
31%
2
5%
3
38%
4
12%
5
14%
Fittest individual has largest share of the roulette wheel
Weakest individual has smallest share of the roulette wheel

Figure 2.1.1 Roulette Wheel Selection [57]

Roulette Wheel Selection is a traditional GA selection technique. All of the chromosomes are mapped on the roulette wheel and each of them is given a segment of the wheel based on their fitness value. The wheel is then spined randomly and the individual corresponding to the selection point will be selected when the roulette wheel stops [7]. The selection technique is easy to implement and preserve the diversity of the chromosome. However, this technique will have a risk of premature convergence to a local optimum as the individual that have a larger portion will have a higher chance to be select and lead to low diversity of the population [8]. It can possibly miss the best individuals into the next generation. There is no assurance for the best individuals to be selected due to the probabilistic nature of random spin in roulette wheel.

According to the research [9], the researchers proposed a new optimized reactive power dispatch (ORPD) strategy for power systems with integrated wind power generation. It uses an improved genetic algorithm combining the elitist non-dominated sorting genetic algorithm with inheritance (i-NSGA-II) and roulette wheel selection to solve the multi-objective ORPD problem. The paper mentions that the algorithm can converge faster to the optimal solutions within 15 generations while maintaining same level of accuracy as compared to the ordinary NSGA-II algorithm which required 100 generations. However, the researchers only use one selection techniques to combine with the i-NSGA-II algorithm, where the possible of higher performance with other selection techniques might be overlooked in the experiment. For example, tournament selection with i-NSGA-II algorithms might produce a higher convergence rate as compared to the roulette wheel selection.

Tournament Selection

Written By: Yeap Chun Hong 2206352

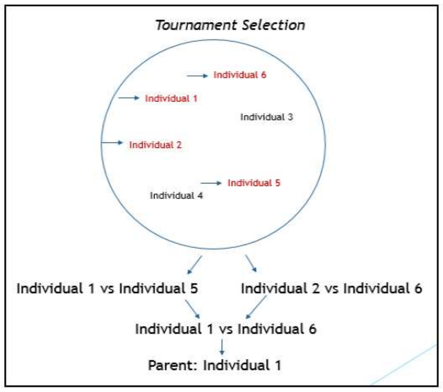


Figure 2.1.2 Tournament Selection [37]

The Tournament Selection technique selects individuals from the current population to perform a tournament competition within a common size of 2, 4 and 7 [10]. The optimal tournament size to be used is depends on the problem domain [11]. This is because the study in [11] shows that a larger tournament size in Gallagher's Gaussian 21-hi Peaks Function produces a better result while a smaller tournament size in Discus Function produces a better result. The individual with the best fitness will be chosen as the parent. Tournament Selection consist of selection pressure that improve the overall population fitness after several generations. The selection pressure is defined as the degree to which the better individuals are favoured in [12]. A high selection pressure means that the tournament size is large and the selection favours the best individuals more strongly. It acts as a significant role in determining a GA's convergence rate, larger selection pressures lead to higher convergence rates and thus require a lower number of generations to reach the global optima. However, if the selection pressure is too low, the convergence rate will be reduced and the GA will take more generation to discover the optimal solution. On the other hand, an extremely high selection pressure will lead to loss of diversity [7].

In [13], the authors proposed a modified GA called Tournament Selection Genetic Algorithm (TS-GA) to minimize task completion time and cost while maximizing the resource utilization in cloud computing environment. Two individuals are randomly selected from the population. Then, a random number between 0-1 is generated and compare to the selection parameter. If the random number is lesser than the parameter, the fitter individual will be selected as parent and vice versa. The TS-GA outperformed the default GA and Round Robin in terms of reducing the completion time and cost and improving the resource utilization, speedup and efficiency in the meantime. However, the researchers have chosen a specific implementation of binary tournament and thus there is a need to redesign the tournament selection algorithm when there is a change for tournament size.

Linear Ranking Selection

Written By: Yeap Chun Hong 2206352

Linear Ranking Selection is a technique that was introduced to address the limitations of Roulette Wheel Selection, which is the weighted portion of wheel based on the fitness value of the individual [14]. In this technique, the individuals are rank according to the fitness value in descending order due to the minimization scenario. After that, the rank is assigned to the individual and the selection probability is computed below based on the formula in 2.1 [14]:

$$P_i = \frac{1}{N}(\min + (\max - \min)\frac{i-1}{N-1};\ i \in \{1, \ldots, N\}) \tag{2.1}$$

where P_i is the selection probability of ith individual, *min* and *max* is the selection probability for minimum and maximum respectively.

The conditions for max and min are $1 \leq max \leq 2$ and $min = 2 - max$ respectively [15]. In addition, the author does also suggest the *max* to be set as 1.1.

According to [16], the researchers had applied various selection techniques in GA for optimising the Travelling Salesman Problem (TSP) and provide a comparative study among the selection techniques. The selection techniques used are Linear Ranking, Roulette Wheel and Tournament selection techniques in the study. The techniques are conducted with different size of cities which is 20, 40 and 60 and the mean, standard deviation and standard error mean of respective techniques are taken from the average of 10 test. Linear Ranking Selection had achieved the best results in terms of distance and followed by Roulette Wheel and Tournament Selection. However, the computational time required by different selection methods is not considered in the research and it might be an important factor for larger cities.

2.1.2 Crossover

Written By: Loh Chia Heung 2301684

In genetic algorithms, the crossover operation plays a crucial role in generating new solutions, or offspring, by interchanging information from parent solutions. This helps the algorithm explore different regions of the search space to discover better solutions [19]. Also, it serves both as an exploration and exploitation mechanism [17], [18]. A good balance between exploration and exploitation capabilities of genetic algorithm is proven to speed up the searching process and achieving higher-quality solutions in Genetic Algorithm [19]. These operations are generally divided into two categories: application-independent and application-dependent.

Application-independent crossover operations are standard crossover technique that can be used in a wide variety of problems without the customization for specific applications. Examples include single-point crossover, uniform crossover and two-point crossover [20]. Application-dependent crossover operations are specialized crossover methods tailored to the unique characteristics or constraints of a specific problem. For instance, some of the crossover methods that used to solve Traveling Salesman Problems (TSP) are Order-Based Crossover (OBX), Modified Order Crossover (MOC), and Partially Mapped Crossover (PMX) [20]. The effectiveness of these operations is greatly influenced by the encoding method used and the specific application context.

According to [20], there are two main factors to be considered when choosing for the operators, which are global convergence and search space. Global convergence affects how effectively and efficiently the GA can find for the optimal solution. Meanwhile, search space is used to determine the range and diversity of solutions that the GA can explore. As mentioned before, both are important to achieve the right balance between exploration and exploitation to achieve the higher-quality solutions in Genetic Algorithm.

Uniform Crossover

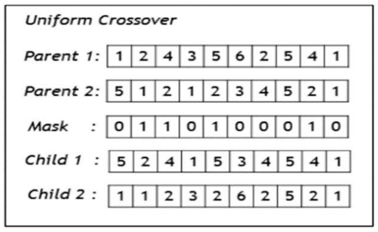
Written By: Loh Chia Heung 2301684

Uniform crossover ensures a balanced combination of bits from both parents by randomly swapping their bits to create offspring. This process involves selecting two parents and generating two children, each inheriting genes uniformly from both parents based on a randomly generated real number ranging from 0 to 1 [19]. The random number helps determine whether each gene in the first child comes from either the first parent or the second parent. [19]. Therefore, the inheritance of uniform crossover does not depend on the position of the parents, and it allows for any combination of parental genes in the offspring, potentially leading to more diverse solutions [20]. It is because it increases the genetic diversity and gene variation, leading to a broader range of solution being explored.

According to [20], research has shown that uniform crossover can outperform two-point crossover, even with smaller population sizes in specific problems, indicating its effectiveness regardless of population size, as proved by an experiment conducted by Lobo [21]. Other than that, uniform crossover has been found to be more effective than two-point crossover and one-point crossover across a wide range of optimization problems, including the Traveling Salesperson, Exponentially Decreasing Sine, Sparse One Max, except for the Shekel's Foxholes [22]. This can be proven by the empirical experiment that was conducted by [22], and all the results showed that uniform crossover outperforms than two-point crossover and one-point crossover except for Shekel's Foxholes [22]. This finding highlighted the robustness of uniform crossover across diverse optimization scenarios. To conclude, uniform crossover's ability to maintain genetic diversity helps prevent premature convergence and allows for a more thorough exploration of the search space [22].

Furthermore, researchers have developed a novel genetic algorithm approach that combines uniform crossover (UX) and unimodal normal distribution crossover (UNDX). This method, named as (UX, UNDX) + Enhanced Minimal Generation Gap

(EMGG), features its self-adaptive system that regulates the usage of each crossover technique throughout the optimization. The results also demonstrate that this hybrid approach outperforms algorithms utilizing either UX or UNDX exclusively, successfully optimizing a broader range of functions [23].



Uniform Crossover

Parent 1:	1	2	4	3	5	6	2	5	4	1
Parent 2:	5	1	2	1	2	3	4	5	2	1
Mask :	0	1	1	0	1	0	0	0	1	0
Child 1 :	5	2	4	1	5	3	4	5	4	1
Child 2 :	1	1	2	3	2	6	2	5	2	1

Figure 2.1.3 Uniform Crossover [20]

Shuffle Crossover

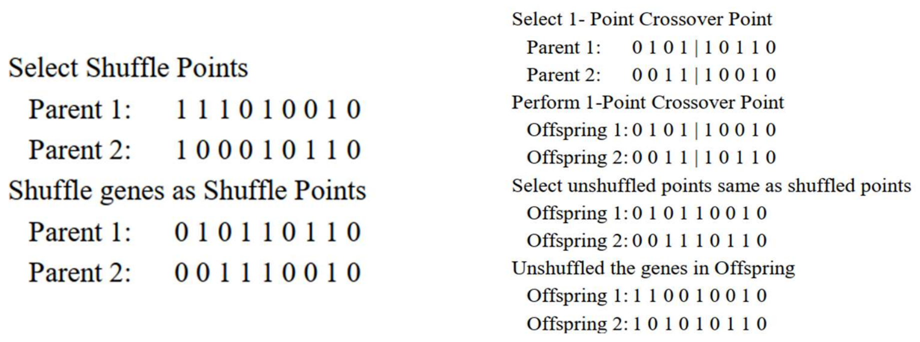
Written By: Loh Chia Heung 2301684

Shuffle Crossover starts by selecting two parents for crossover. Initially, it randomly shuffles the genes of both parents in the same manner. It means that both parent genes are subjected to the same random sequence or arrangement. For instance, if the shuffle order for Parent A is [1, 0, 3, 2], then the same shuffling sequence will be applied to Parent B. Subsequently, a 1-point crossover technique is applied by randomly selecting a crossover point and then combining the parents to produce two offspring. After the 1-point crossover is performed, the genes in the offspring are unshuffled in the same way they were shuffled initially [19]. Consequently, a fixed positional bias is eliminated as the gene positions are randomly reassigned with each crossover event in the shuffle crossover. This process involves random shuffling based on shuffle points, performing a 1-point crossover, and then restoring the order.

Initially, according to [24], the paper has shown that the search performance is influenced by the position-dependent bias of traditional crossover, which is one-point crossover through testing with Plateau and Trap functions. They demonstrated that this bias is negatived by shuffle crossovers, resulting in performance improvement. This improvement is proven in the experiment conducted in [24], where shuffle crossover achieved lower mean and standard error compared to traditional one-point crossover. The results, as illustrated in Figure 2.1.5, clearly demonstrate the enhanced performance of shuffle crossover. To prove it, the author used both adjacent and distributed representations in the Plateau function and the result showed that shuffle crossover performed consistently across representations, while traditional crossover struggled with the distributed representation [24].

Other than that, the experiments with De Jong's test suite demonstrated that shuffle crossover achieved significant improvements over traditional crossover [24]. Specifically, it achieved statistically significant better results on two of the test functions, which are Rosenbrock’s Saddle, Step Function and Quadratic with Noise [26] while performing comparably on the remaining three, which are Parabola and Shekel’s foxholes [26]. These results suggest that shuffle crossover may offer advantages and useful for a wider range of optimization problems [24]. As in the previous case, the same Figure 2.1.5 is used to illustrate these performance gains. The performance was measured using the mean number of evaluations required to find the optimum and the standard error, providing insights into the consistency and reliability of the methods [24].

According to [25], in 1989, a novel approach combining the mutual information and shuffle crossover is proposed by Eshelman, Caruna and Schaffer and it is named as Mutual Information and Shuffle Crossover (MISC). This technique was evaluated against traditional genetic algorithms with one-point and uniform crossover operation. The result indicated that MISC generally outperformed in the performance of GA. The results, as shown in Figure 2.1.6, indicated that MISC generally outperformed these traditional crossover methods in terms of fitness value. A higher fitness value signifies better performance in the following studies. Notably, MISC consistently demonstrated competitive results compared to Simple Genetic Algorithms (SGA) with one-point and uniform crossover [25]. It suggested that it may offer significant advantages in improving search performance in genetic algorithms.



Select Shuffle Points

Parent 1: 1 1 1 0 1 0 0 1 0

Parent 2: 1 0 0 0 1 0 1 1 0

Shuffle genes as Shuffle Points

Parent 1: 0 1 0 1 1 0 1 1 0

Parent 2: 0 0 1 1 1 0 0 1 0

Select 1- Point Crossover Point

Parent 1: 0 1 0 1 | 1 0 1 1 0

Parent 2: 0 0 1 1 | 1 0 0 1 0

Perform 1-Point Crossover Point

Offspring 1: 0 1 0 1 | 1 0 0 1 0

Offspring 2: 0 0 1 1 | 1 0 1 1 0

Select unshuffled points same as shuffled points

Offspring 1: 0 1 0 1 1 0 0 1 0

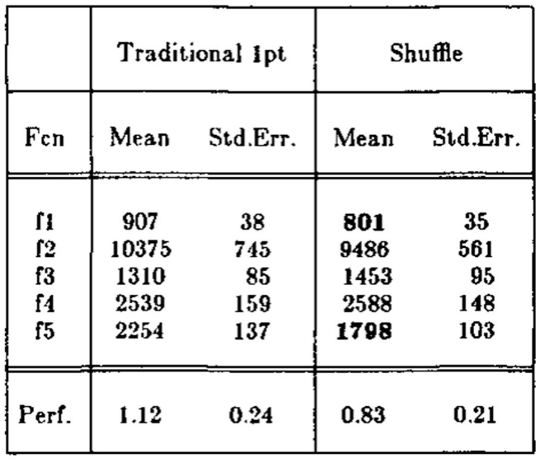
Offspring 2: 0 0 1 1 1 0 1 1 0

Unshuffled the genes in Offspring

Offspring 1: 1 1 0 0 1 0 0 1 0

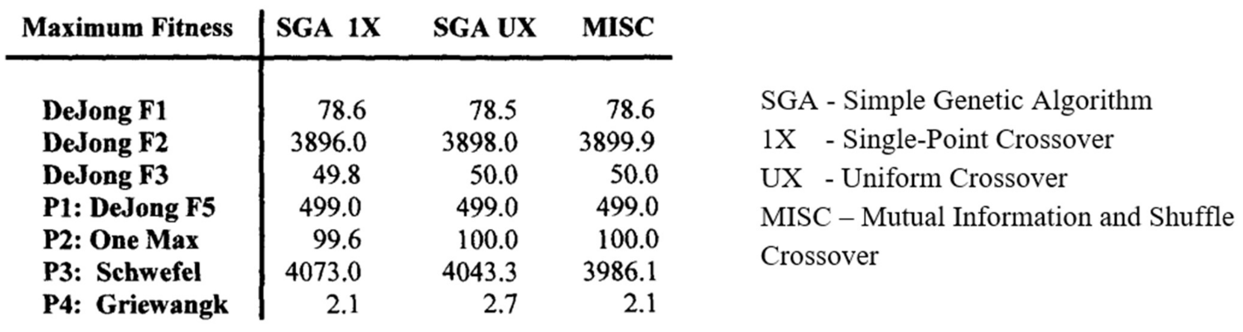
Offspring 2: 1 0 1 0 1 0 1 1 0

Figure 2.1.4 Shuffle Crossover [19]



	Traditional 1pt		Shuffle	
Fcn	Mean	Std.Err.	Mean	Std.Err.
f1	907	38	**801**	35
f2	10375	745	9486	561
f3	1310	85	1453	95
f4	2539	159	2588	148
f5	2254	137	**1798**	103
Perf.	1.12	0.24	0.83	0.21

Figure 2.1.5 Performance of 1-Point Crossover and Shuffle Crossover [24]



Maximum Fitness	SGA 1X	SGA UX	MISC
DeJong F1	78.6	78.5	78.6
DeJong F2	3896.0	3898.0	3899.9
DeJong F3	49.8	50.0	50.0
P1: DeJong F5	499.0	499.0	499.0
P2: One Max	99.6	100.0	100.0
P3: Schwefel	4073.0	4043.3	3986.1
P4: Griewangk	2.1	2.7	2.1

SGA - Simple Genetic Algorithm

1X - Single-Point Crossover

UX - Uniform Crossover

MISC – Mutual Information and Shuffle Crossover

Figure 2.1.6 Maximum Fitness Result of the 3 Different Crossover Method [25]

Arithmetic Crossover

Written By: Loh Chia Heung 2301684

Arithmetic crossover is a crossover operator in genetic algorithms that produces new solutions or offspring by linearly combining the chromosomes of the parents, as shown in the following equations [27].

$$\lambda = \text{rand}(0,1)$$
$$C_1 = (\lambda)P_1 + (1-\lambda)P_2$$
$$C_2 = (1-\lambda)P_1 + (\lambda)P_2$$

Parameter:

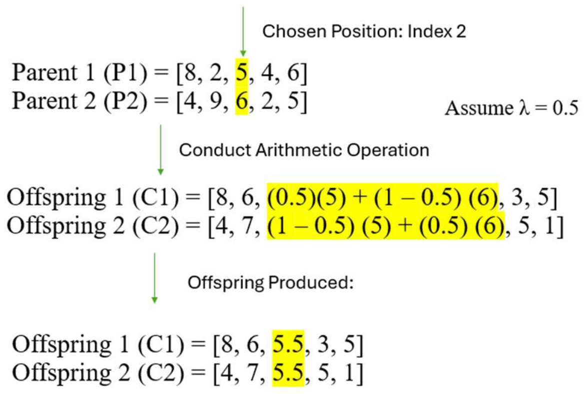
C_1 = Children 1	P_1 = Parent 1
C_2 = Children 2	P_2 = Parent 2

λ = random number generated in the range between 0 and 1

Figure 2.1.7 Formula of Arithmetic Crossover [27]

According to [28], arithmetic crossover consists of three types, which is single, whole and simple.

In **single arithmetic crossover**, a random position is chosen on the chromosome. At this position, the offspring gene is calculated using the formula shown in Figure 2.1.7 [29]. The remaining genes in the offspring are directly copied from the parents.


Chosen Position: Index 2

Parent 1 (P1) = [8, 2, 5, 4, 6]
Parent 2 (P2) = [4, 9, 6, 2, 5]

Assume λ = 0.5

Conduct Arithmetic Operation

Offspring 1 (C1) = [8, 6, (0.5)(5) + (1 – 0.5) (6), 3, 5]
Offspring 2 (C2) = [4, 7, (1 – 0.5) (5) + (0.5) (6), 5, 1]

Offspring Produced:

Offspring 1 (C1) = [8, 6, 5.5, 3, 5]
Offspring 2 (C2) = [4, 7, 5.5, 5, 1]

Figure 2.1.8 Single Arithmetic Crossover

In **simple arithmetic crossover**, a random crossover point is selected along the chromosome, but only the genes after the point are blended using the arithmetic operations and the offspring gene is calculated using the formula shown in Figure 2.1.7 [29]. The genes before the crossover point are directly copied from the parents.

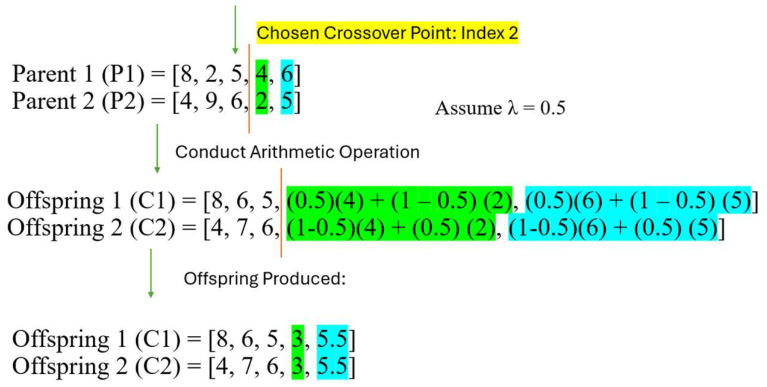

Chosen Crossover Point: Index 2
Parent 1 (P1) = [8, 2, 5, 4, 6]
Parent 2 (P2) = [4, 9, 6, 2, 5]
Assume λ = 0.5
Conduct Arithmetic Operation
Offspring 1 (C1) = [8, 6, 5, (0.5)(4) + (1 – 0.5) (2), (0.5)(6) + (1 – 0.5) (5)]
Offspring 2 (C2) = [4, 7, 6, (1-0.5)(4) + (0.5) (2), (1-0.5)(6) + (0.5) (5)]
Offspring Produced:
Offspring 1 (C1) = [8, 6, 5, 3, 5.5]
Offspring 2 (C2) = [4, 7, 6, 3, 5.5]

Figure 2.1.9 Simple Arithmetic Crossover

For the **Whole Arithmetic Crossover**, the entire chromosome is blended using the arithmetic combination of the parent chromosomes. Every gene in the offspring is a combination of the corresponding genes from both parents and they are calculated using the formula shown in Figure 2.1.7 [29].

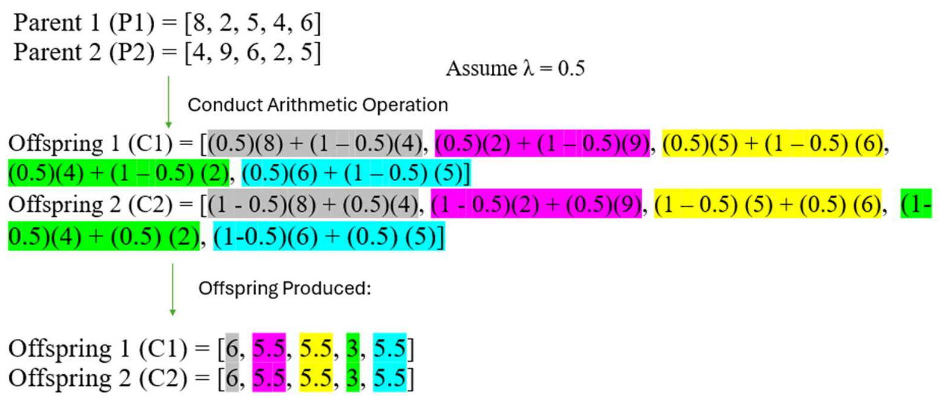
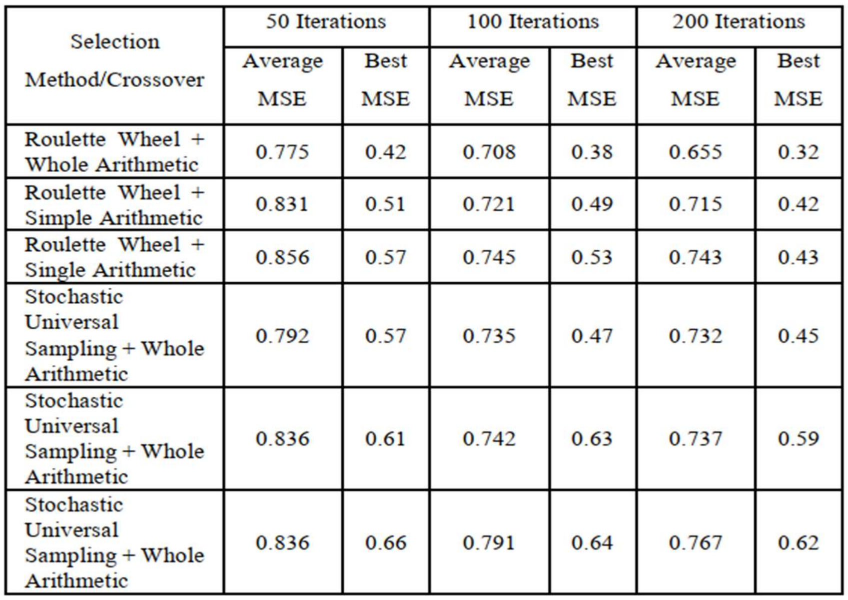

Parent 1 (P1) = [8, 2, 5, 4, 6]
Parent 2 (P2) = [4, 9, 6, 2, 5]
Assume λ = 0.5
Conduct Arithmetic Operation
Offspring 1 (C1) = [(0.5)(8) + (1 – 0.5)(4), (0.5)(2) + (1 – 0.5)(9), (0.5)(5) + (1 – 0.5) (6), (0.5)(4) + (1 – 0.5) (2), (0.5)(6) + (1 – 0.5) (5)]
Offspring 2 (C2) = [(1 - 0.5)(8) + (0.5)(4), (1 - 0.5)(2) + (0.5)(9), (1 – 0.5) (5) + (0.5) (6), (1-0.5)(4) + (0.5) (2), (1-0.5)(6) + (0.5) (5)]
Offspring Produced:
Offspring 1 (C1) = [6, 5.5, 5.5, 3, 5.5]
Offspring 2 (C2) = [6, 5.5, 5.5, 3, 5.5]

Figure 2.1.10 Whole Arithmetic Crossover

According to King Junior et.al. [30], a performance evaluation was conducted on whole arithmetic crossover, simple arithmetic crossover and single arithmetic crossover methods, combined with different selection techniques in the GenClust++ algorithm.

The GenClust++ algorithm is designed to determine the centroid and prevent the local optima problems from occurring in K-means clustering [30].

Their results consistently demonstrated that whole arithmetic crossover outperformed both simple and single crossover across various iterations and selection methods. This is illustrated in Figure 2.1.11, where the whole Arithmetic Crossover achieved the lowest Average and Best Mean Square Error (MSE), indicating higher accuracy compared to other crossover methods that were tested in the experiment. The authors found that whole arithmetic crossover produces the better results because the crossing process occurs across all genes. Therefore, it provides a more comprehensive exploration of the solution space by increasing the genetic diversity [30]. As stated in [31], enhancing genetic diversity enables a population to adapt more quickly to the environmental changes and enables continuous exploration of new potential solutions, reducing the risk of getting trapped in the local optima.



Selection Method/Crossover	50 Iterations		100 Iterations		200 Iterations	
	Average MSE	Best MSE	Average MSE	Best MSE	Average MSE	Best MSE
Roulette Wheel + Whole Arithmetic	0.775	0.42	0.708	0.38	0.655	0.32
Roulette Wheel + Simple Arithmetic	0.831	0.51	0.721	0.49	0.715	0.42
Roulette Wheel + Single Arithmetic	0.856	0.57	0.745	0.53	0.743	0.43
Stochastic Universal Sampling + Whole Arithmetic	0.792	0.57	0.735	0.47	0.732	0.45
Stochastic Universal Sampling + Whole Arithmetic	0.836	0.61	0.742	0.63	0.737	0.59
Stochastic Universal Sampling + Whole Arithmetic	0.836	0.66	0.791	0.64	0.767	0.62

Figure 2.1.11 Overview of Test Results Using Stochastic Universal Sampling and Roulette Wheel with 3 Types of Arithmetic Crossover [30]

In addition, according to [32], the authors proposed a novel approach to solve the Dual Response System (DRS) problem using a genetic algorithm with arithmetic crossover. This method was designed to overcome the limitations of previous approaches by generating multiple alternative solutions. This flexibility allows decision-makers to better explore the trade-offs between the mean and standard deviation responses. The previous approaches to the DRS problem included methods such as Hopfield Neural Network (NHH) by Koksoy and Yalcinoz [33], Composite Objective Function proposed by Lin and Tu [34], and NIMBUS algorithm by Koksoy and Doganaksoy [35].

The authors applied arithmetic crossover to a well-known printing process problem and compared the results with those from the NIMBUS algorithm. The findings indicated that arithmetic crossover not only produced solutions comparable to those of the NIMBUS algorithm, but also offered advantages in computational efficiency and the ability to generate a diverse set of solutions [32].

This novel approach demonstrated the potential of arithmetic crossover in genetic algorithms for solving multi-objective optimization problems like DRS. It provided a flexible framework for balancing various objectives, enabling decision-makers to explore a range of solutions based on their specific needs and preferences.

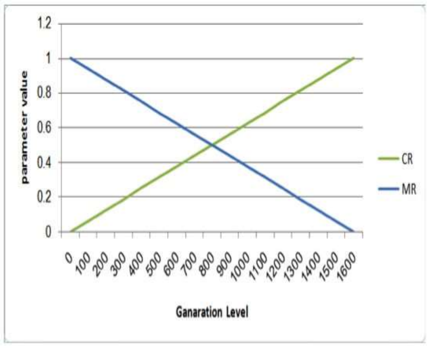
2.1.3 Mutation

Mutation Probability

Written By: Brandon Ting En Junn 2101751

Mutation operators serve an important role in diversifying the existing population to help achieve a good balance between explorative and exploitative strategies in GAs [19]. Objectively, it enables GAs to escape the local of suboptimal solutions to avoid premature convergence [19]. Mutation operators avoid premature convergence by introducing new search spaces to the GA. Depending on the selection of mutation operators, the GA model may perform exceptionally better or worse. Controlling the rate of this diversification is the mutation probability or mutation ratio that serves as a parameter setting for the GA to determine the likelihood of mutating a gene. Mutation probability typically complements the crossover probability that falls in the range of [0, 1]. Commonly, static ratios are the approach in determining the mutation probability. Static ratio is the approach where a value is determined before the experiment and is constant throughout the whole experiment. As a reference, the static ratio used by [36] is 0.01. However, [38] proposed another approach namely the Dynamic Decreasing of high mutation ratio/dynamic increasing of low crossover ratio (DHM/ILC) and

Dynamic Increasing of Low Mutation/Dynamic Decreasing of High Crossover (ILM/DHC). Figure 2.1.12 and Figure 2.1.13 shows the applications of DHM/ILC and ILM/DHC in terms of their crossover and mutation rates during a GA experiment. [38] stated that DHM/ILC and ILM/DHC were more effective when dealing with small and large population sizes respectively. However, it may prove costly to adhere to the expensive computation costs in processing time since they are fine-tuning approaches.



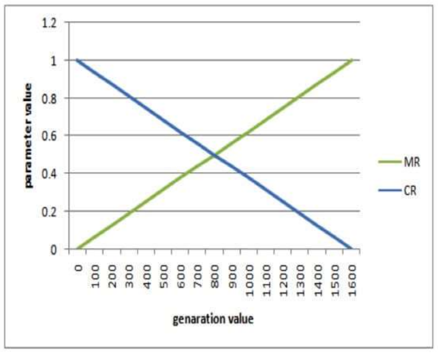


Figure 2.1.12 DHM/ILC [38]

Figure 2.1.13 ILM/DHC [38]

Reversing Mutation

Written By: Brandon Ting En Junn 2101751

As [38] suggested, the Reversing Mutation randomly selects a gene of the chromosome to swap with the gene in front of it. This is also known as reversing the values of the genes. If a special case happens where the first gene is to be selected, the reversing position is to the last gene of the chromosome. Figure 2.1.14 shows the normal case process of the Reversing Mutation. Reversing Mutation may be effective in introducing new local search spaces through the reordering of gene values. Hence, better genes can only be introduced only if the genes are not equally weighted. In this context, the weights of the genes are determined by the mathematical nature of the benchmark function. For example, the Axis Parallel Hyper-Ellipsoid function considers the genes' position as a weight constant and each increasing position contributes more to the fitness value. On the other hand, the Sphere function only considers the genes' values, therefore are equally weighted when producing the fitness value.

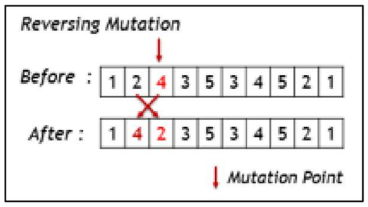


Figure 2.1.14 Reversing Mutation [37]

Random Mutation

Written By: Brandon Ting En Junn 2101751

In [36], they used Random Mutation in one of their GA models that selects two random genes from the chromosome and replaces both of them with randomly generated values between the lower and upper bounds [39]. An illustration of the process is shown in Figure 2.1.15. Random Mutation is very effective in escaping the local minima by introducing new genetic materials, thus drastically increasing the search space at a global scale [36]. However, this may prove ineffective in certain situations whereby the randomly generated genes are worse than the existing genes.

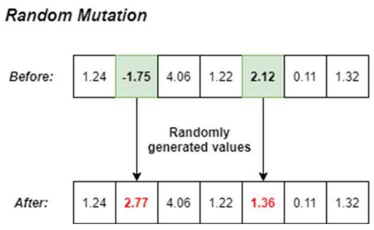


Figure 2.1.15 Random Mutation

Simple Inversion Mutation

Written By: Brandon Ting En Junn 2101751

A similar variant of the previously reviewed Reversing Mutation was mentioned in [6], namely the Simple Inversion Mutation operator (SIM). It randomly selects two gene points in a chromosome and inverts the substring between these two gene points at the middle [40], [41]. The process is shown in Figure 2.1.16. This operator may prove to be more effective than Reversing Mutation due to its substring inversion of multiple genes rather than only two selected genes. However, similar to the reversing mutation, it will prove to be ineffective if the genes are equally weighted depending on the benchmark function.

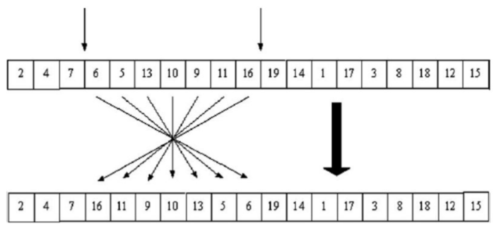


Figure 2.1.16 Simple Inversion Mutation [41]

2.1.4 Replacement

Written By: Ling Ji Xiang 2104584

This paper [42] states that replacement strategy is to enhance population's diversity while improving solution quality. One of the primary goals is to maintain diversity within the population which is vital for preventing premature convergence. Premature convergence occurs when the populations become too homogenous which limits the algorithm's ability to explore more regions of solution space effectively. Replacement strategy used is Steady-State GAs (SSGAs) which replaces only a few individuals per generation instead of entire population. This gradual replacement allows for a continuous but controlled flow of new genetic material into population. This can help to maintain diversity awhile also gradually improving the overall fitness of the population. By avoiding large-scale replacement, SSGAs reduce the risk of losing valuable genetic information that might be crucial for finding the optimal solution.

Weak Parent Replacement

Written By: Ling Ji Xiang 2104584

The idea of Weak Parent Replacement is parent compete with the offspring and the parent who has weaker fitness value compared to the offspring will be replaced by the offspring that has higher fitness value. This means that the population quality will gradually improve by ensuring that only the higher fitness value will survive and reproduce. This method encourages a gradual improvement in the population by ensuring that only individuals with superior fitness are retained. Because of only weaker parents are replaced by stronger child, this strategy maintains a level of diversity that helps prevent algorithm from getting stuck in a local optimum as it avoids letting weaker individuals dominate the population. [42]

Both Parent Replacement

Written By: Ling Ji Xiang 2104584

Both Parent Replacement means that both parents are replaced by the new offspring regardless the fitness value comparison. This potentially introduce new genetic material into the population more rapidly while it also increases the risk of losing genetic diversity. It is possible that the parents with higher fitness value replaced by offspring with lower fitness value, this causes losing the valuable fitness value and lead to premature convergence. However, this strategy can lead to faster convergence by quickly shifting the population towards new regions of solution space. To ensure that this results in improvement of fitness value rather than going into premature convergence, it is important to combine this strategy with mechanisms that preserve valuable fitness value such as weak parent replacement. [43]

Binary Tournament Replacement

Written By: Ling Ji Xiang 2104584

Binary Tournament Replacement is a replacement method used in GA to random choosing 2 individuals from the population, if the 2 selected individual are the same, then it will select another individual to ensure that 2 different individuals are selected. Then the selected individuals will go through the tournament, the individual with better fitness value will get to pass to the next generation. The individual with weaker fitness value will be replaced by the offspring that has the stronger fitness value. This strategy is straightforward and easy to implement. [44]

2.2 Particle Swarm Optimisation

Written By: Loh Chia Heung 2301684

Particle Swarm Optimization (PSO) is a population-based stochastic optimization method was proposed by Kennedy and Eberhart in 1995 and it is inspired by the by the collaborative swarm intelligence observed in groups of animals, such as schools of fish and flocks of birds [45]. These swarms work together in a collaborative way to locate food, with each member constantly adjusting its search strategy based on both its own experience and what it learns from other members [45]. The main concept behind the PSO algorithm is influenced by two areas of research: evolutionary algorithms and artificial life [45]. Similar to evolutionary algorithms, PSO uses a swarm-based approach that enables it to explore wide solution space efficiently [45]. This efficiency can be demonstrated through the five principles outlined later.

Artificial life explores the systems that imitate life-like behaviours by studying social animal interactions through the artificial life theory. According to [46], Millonas proposed five fundamental principles for this in 2001, which are:

1. **Proximity**: The swarm must be capable of performing basic computations related to space and time.

2. **Quality**: The swarm must be able to detect the changes in environment quality and respond to it accordingly.

3. **Diverse** Response: It needs to explore a wide range of strategies for resource acquisition to avoid narrow or restricted approaches.

4. **Stability**: The swarm should maintain state behaviour and not react to every minor environmental change.

5. **Adaptability**: It should adapt its behaviour only when the change is advantageous.

In PSO, particles adjust their positions and velocities based on the change of environment, which satisfies the proximity and quality principles [45]. Moreover, the swam is not limited to a fixed path, it will continuously explore the solution space to find the best solution. While particle maintain this stable search behaviour, they can also adapt their movement strategies in response to environmental changes. As a result, PSO systems align well with all five principles [45].

Written By: Brandon Ting En Junn 2101751

In the context of minimisation optimisation problems, particles are known to be the solution candidates in PSO. The positions of the particles are updated with the velocities of the particles, while the velocities of the particles are updated with the personal best and NSGA-II particles. The following formulae show the update velocity and update position calculations [47].

Update Velocity:

$$v_{id} = w * v_{id} + c_1 r_1 (p_{id} - x_{id}) + c_2 r_2 (p_{gd} - x_{id})$$

Update Position:

$$x_{id} = x_{id} + v_{id}$$

$v_{id}: Velocity\ of\ particle$

$x_{id}: Position\ of\ particle$

$p_{id}: Personal\ best\ position\ of\ particle$

$p_{gd}: Group\ best\ position\ of\ particle$

$c_1, c_2: Acceleration\ constants$

$r_1, r_2: Random\ number$

$w: Inertia\ weight$

A type of PSO model is the inclusion of the constant inertia weight, *w* and velocity clamping, *vc*. [47] mentioned that the inclusion of *w* better controls the exploration and exploitation of the PSO model. Additionally, a study from [48] revealed that a large *w* favours exploration with increased diversity while a small *w* favours local exploitation. A constant *w* will fix the rate of global and local search throughout the algorithm.

The boundary limits for the updated velocities are referred to as *vc*. If the calculated velocity exceeds the boundary limits, it will refer to the upper or lower boundary values as its updated velocity. A study [49] found that *vc* should be implemented alongside the presence of a constant *w* to avoid the case of difficult convergence.

Based on [50], their parameters setting for the constant *w* is 0.7 while the *vc* is half of the range of the dataset. Additionally, the random numbers, r_1 and r_2 are to be in the range of [0, 1] as [50] suggested. Additionally, the study [51] set both of the acceleration constants, c_1 and c_2 as 2 throughout the experiment.

2.3 Comparison of Previous Works

Operation Techniques

Written By: Yeap Chun Hong 2206352

The comparison of previous work of selection operation are listed in Table 2.3.1 with strength and weakness of the model in the previous work.

Table 2.3.1 Strengths and Weaknesses of Reviewed Selection Operation

Author	Selection Techniques	Strength	Limitation
Y. Liu et al. (2022) [9]	Combination of elitist non-dominated sorting genetic algorithm with inheritance (i-NSGA-II) and roulette wheel selection.	Converge faster to the optimal solutions within 15 generations while maintaining same level of accuracy.	Possible of higher performance with other selection techniques might be overlooked as only one selection techniques to combine with the i-NSGA-II algorithm.
S. A. Hamad and F. A. Omara (2016) [13]	Modified Tournament Selection Genetic Algorithm	The TS-GA outperformed the default GA and Round Robin.	A specific implementation of binary tournament and thus there is a need to redesign the tournament selection algorithm when there is a change for tournament size.

H. M. Pandey (2016) [16]	Linear Ranking, Roulette Wheel and Tournament	Comparative analysis for different size of cities and different selection techniques.	Computational time is not measured.

Written By: Loh Chia Heung 2301684

The table of comparison of previous work for **Uniform Crossover** is listed below in Table 2.3.2.

Table 2.3.2 Comparison of Previous Work for Uniform Crossover

Aspect	**Uniform Crossover (UX)**	**Two-Point & One-Point Crossover**	**Hybrid (UX + UNDX + EMGG)**
Optimization Problem	Effective in Travelling Salesperson, Exponentially Decreasing Sine, Sparse One Max [22].	Less effective in these optimization problems such as Travelling Salesperson compared to UX [22].	Effective across multiple optimization problems [23].
Effectiveness	Outperform two-point and one-point crossover in several optimization problems such as Traveling Salesperson [20].	Less effective compared to UX [20].	Outperforms both UX and UNDX alone, optimizes a broader range of functions [23].
Genetic Diversity	Helps maintain genetic diversity, preventing	Lower diversity than UX [22].	Enhanced-MGG system adapts crossover usage

	premature convergence [22].		dynamically for better diversity [23].

The table of comparison of previous work for **Shuffle Crossover** is listed below in Table 2.3.3.

Table 2.3.3 Comparison of Previous Work for Shuffle Crossover

Aspect	**Shuffle Crossover**	**One-Point Crossover**	**Mutual Information + Shuffle Crossover (MISC)**
Effectiveness	Outperforms one-point crossover in Plateau and Trap functions [24].	Position-dependent bias limits its effectiveness [24].	Outperforms one-point and uniform crossover in overall GA performance [25].
Optimization Problems	Performs well in Plateau, Rosenbrock's Saddle, Step Function, and Quadratic with Noise [24].	Struggles with distributed representation [24].	Performs better in distributed and adjacent representations [25].
Performance Metrics	Lower mean number of evaluations and standard error [24].	Higher mean number of evaluations, less consistent [24].	Achieved higher fitness value compared to Simple Genetic Algorithms (SGA) with one-point and uniform [25].
Genetic Diversity	Helps negate position-dependent bias, maintaining better diversity [24].	Suffers from position bias, reducing diversity [24].	Uses mutual information for higher diversity in GA search [25].

Lastly, the table of comparison of previous work for **Arithmetic Crossover** is listed below in Table 2.3.4.

Table 2.3.4 Comparison of Previous Work for Arithmetic Crossover

Aspect	**Whole Arithmetic Crossover**	**Simple Arithmetic Crossover**	**Single Arithmetic Crossover**	**Novel Approach with Arithmetic Crossover (DRS Problem)**
Effectiveness	Outperformed simple and single arithmetic crossover in terms of accuracy and MSE [30].	Less effective compared to whole arithmetic crossover [30].	The least effective compared to simple and whole arithmetic [30].	Offers advantages in multi-objective optimization problems [32]
Optimization Problems	Effective in GenClust++ for improving K-means clustering and avoiding local optima [30].	Less effective for GenClust++ algorithm [30].	Less effective for GenClust++ algorithm [30].	Effective in solving the Dual Response System (DRS) problem [32].
Performance Metrics	Lowest Average and Best Mean Square Error (MSE) [30].	Higher MSE than whole arithmetic crossover [30].	Higher MSE than whole arithmetic crossover [30].	Comparable and better computational efficiency than NIMBUS algorithm [32].
Genetic Diversity	Enhances diversity by crossing across all genes, improving exploration of the solution space [30].	Lower diversity due to limited gene involvement [30].	Limited diversity due to only one gene crossover [30].	Helps explore a diverse set of solutions [32].

Written By: Brandon Ting En Junn 2101751

Various aspects of the mutation operation have been reviewed to comprehend their implementation details for the experiment. Table 2.3.5 and Table 2.3.6 show the main strengths and weaknesses of the mutation probability approaches and mutation operators.

Table 2.3.5 Summary of Reviewed Mutation Probability

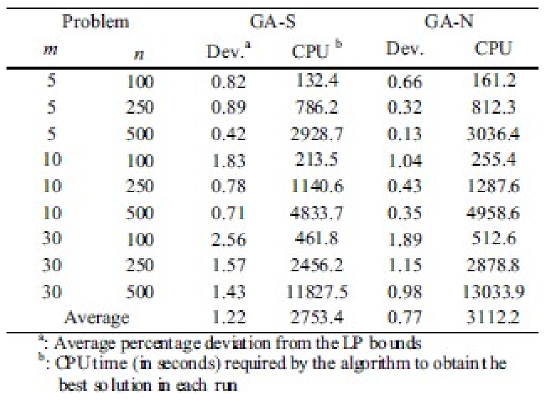
Approach	Short Computational Time
Dynamic	✕
Static	✓

Table 2.3.6 Summary of Reviewed Mutation Operators

Operator	Local Exploitation	Global Exploration
Reversing	✓	✕
Random	✕	✓
Simple Inversion	✓	✕

Written By: Ling Ji Xiang 2104584

A new proposed replacement strategy Steady-State GA (SSGAs) is tested with Generational GA from previous study [51]. The performance of both GA has been tested on two combinational optimization problems which are multiple knapsack problem and capacitated vehicle routing problem. The results demonstrate that the new replacement strategy significantly improved the GA's performance in solving these challenging optimization tasks.



Problem		GA-S		GA-N	
m	*n*	Dev.[a]	CPU [b]	Dev.	CPU
5	100	0.82	132.4	0.66	161.2
5	250	0.89	786.2	0.32	812.3
5	500	0.42	2928.7	0.13	3036.4
10	100	1.83	213.5	1.04	255.4
10	250	0.78	1140.6	0.43	1287.6
10	500	0.71	4833.7	0.35	4958.6
30	100	2.56	461.8	1.89	512.6
30	250	1.57	2456.2	1.15	2878.8
30	500	1.43	11827.5	0.98	13033.9
Average		1.22	2753.4	0.77	3112.2

[a]: Average percentage deviation from the LP bounds
[b]: CPU time (in seconds) required by the algorithm to obtain the best solution in each run

Figure 2.3.1 Computational Result for the Multiple Knap Sack Problems [51]

GA-S = Steady-state; GA-N = Generational

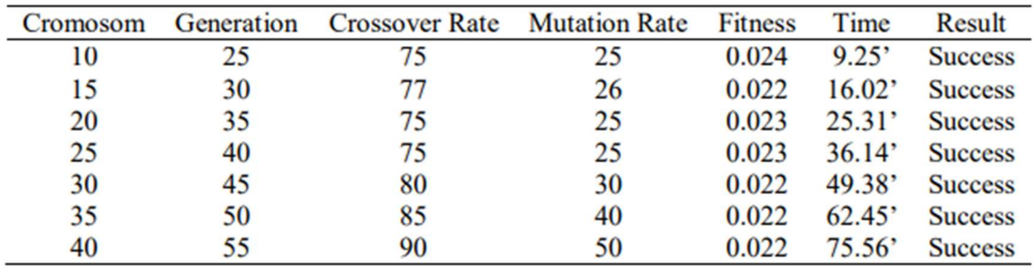
GA and PSO

Written By: Yeap Chun Hong 2206352

The author in [52] compared the performance of GA and PSO in designing a reduced scale model of a typical Pressurized Water Reactor (PWR) core. The performance metrics used in this study are fitness value, convergence rate, data homogeneity and computational time to evaluate which optimization techniques is more effective in minimization context. 10 experiments were run using both PSO and GA with different population sizes of 20, 50 and 100. The results show the PSO fitness value remains consistent in 1.407827513 after 6280, 9100 and 13,200 objective function evaluations with 20, 50 and 100 particles respectively. However, GA only reach the fitness value which is lesser than the fitness value of PSO averagely after 20000 evaluations. Therefore, PSO outperformed GA in relation to data homogeneity around the mean and median. Besides that, PSO converge faster with less computational time to produce the optimal solution. However, the optimization problem involves a continuous search space in this study. Therefore, the PSO method achieved a better minimisation due to its ability to operate in a continuous manner, making it more suitable for finding solutions in this type of search space. The GA, which naturally operates in a discrete search space was observed to have limitations in this study as it might not find the optimal solution in a continuous search space.

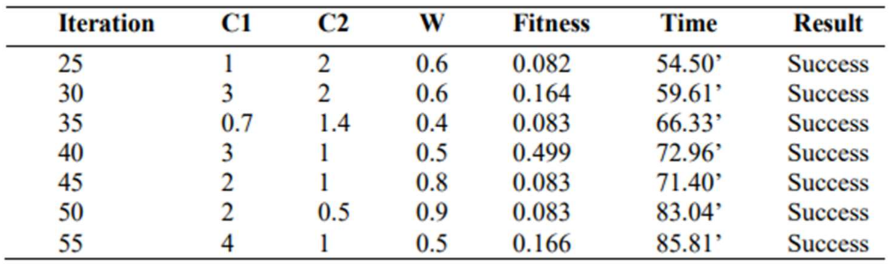
Written By: Ling Ji Xiang 210485

The paper [53], provided comparison of GA and PSO in course scheduling problems. The authors conducted the experiment using a university's course scheduling dataset, focusing on metrics such as converge speed, solution quality and computational efficiency. The results shows that PSO provide faster convergence, quickly reaching near-optimal solution with few iterations. However, the quality of these solutions was slightly lower compared to those produced by GA which effectively satisfied more constraints and achieved better overall schedules. Despite taking more iterations to converge, GA showed a steady improvement in solution quality overtime. When look into computational efficiency, PSO require less time and resources which makes it more suitable for large scale problems. On the other hand, GA require more computational power due to its genetic operations but it provide more robust solutions. The study concluded that while PSO is advantageous for problem that require quick solutions with moderate quality and GA is more suitable for problems that require higher solution quality.



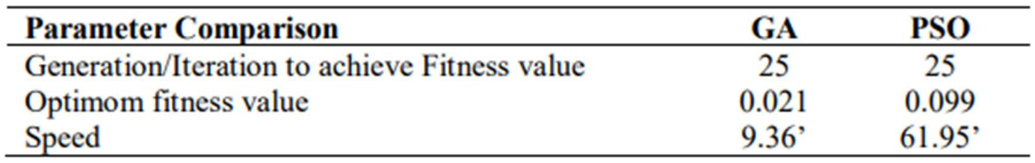
Cromosom	Generation	Crossover Rate	Mutation Rate	Fitness	Time	Result
10	25	75	25	0.024	9.25’	Success
15	30	77	26	0.022	16.02’	Success
20	35	75	25	0.023	25.31’	Success
25	40	75	25	0.023	36.14’	Success
30	45	80	30	0.022	49.38’	Success
35	50	85	40	0.022	62.45’	Success
40	55	90	50	0.022	75.56’	Success

Figure 2.3.2 Accuracy Test Result with GA



Iteration	C1	C2	W	Fitness	Time	Result
25	1	2	0.6	0.082	54.50’	Success
30	3	2	0.6	0.164	59.61’	Success
35	0.7	1.4	0.4	0.083	66.33’	Success
40	3	1	0.5	0.499	72.96’	Success
45	2	1	0.8	0.083	71.40’	Success
50	2	0.5	0.9	0.083	83.04’	Success
55	4	1	0.5	0.166	85.81’	Success

Figure 2.3.3 Accuracy Test Result with PSO



Parameter Comparison	**GA**	**PSO**
Generation/Iteration to achieve Fitness value	25	25
Optimom fitness value	0.021	0.099
Speed	9.36’	61.95’

Figure 2.3.4 Testing the Comparison Results of GA and PSO algorithms

CHAPTER 3

System Design

Written By: Brandon Ting En Junn 2101751

This chapter elaborates the design in terms of the workflows and techniques used to conduct the experiments.

3.1 Genetic Algorithm

Written By: Brandon Ting En Junn 2101751

Figure 3.1.1 illustrates the workflow processes that the research referred from [54] in generating the necessary GA models for the experiments. Each GA model will go through each benchmark function 10 times to obtain the average lowest fitness and average time.

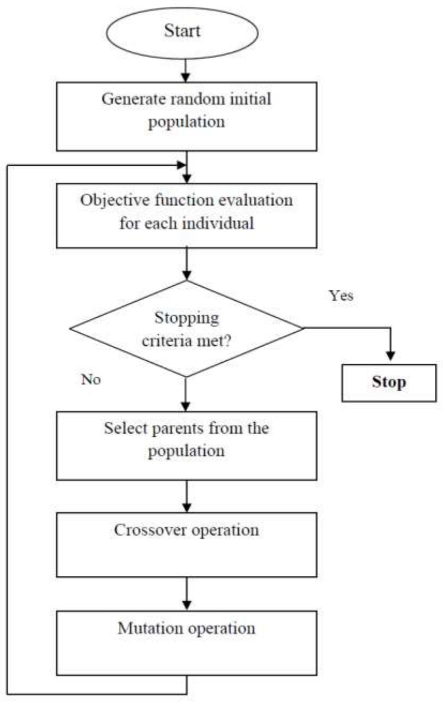

Start
Generate random initial population
Objective function evaluation for each individual
Stopping criteria met?
Yes
Stop
No
Select parents from the population
Crossover operation
Mutation operation

Figure 3.1.1 Flowchart of Genetic Algorithm [54]

- Initially, the population is randomly generated with 40 chromosomes with each a dimension of 30 genes.

- Then, the fitness of each chromosome is evaluated using the benchmark function.

- Firstly, the selection operation is executed to select the parent chromosomes.

- Secondly, a randomly generated probability is generated and compared to the crossover probability. If the randomly generated probability is less than or equal to the crossover probability, the crossover operation is then performed. The crossover operation is performed on the selected parent chromosomes by interchanging a section of the genes between the parent chromosomes. New child chromosomes are produced from the results of this process.

- Thirdly, a randomly generated probability is generated and compared to the mutation probability. If the randomly generated probability is less than or equal to the mutation probability, the mutation operation is then performed. The mutation operation is performed on the child chromosomes by introducing and replacing new genetic material with the existing genes.

- Then, the fitness of the child chromosomes is evaluated.

- The algorithm repeats this cycle back to the selection operation process until 2000 generations. Afterwards, the termination criterion is met and the algorithm stops. The best chromosome with the optimal fitness value is recorded and displayed with its total execution time. In the context of the research, the best chromosome with the optimal fitness value refers to minimisation that favours lower fitness value.

3.2 Operations in Genetic Algorithm

3.2.1 Selection

Written By: Yeap Chun Hong 2206352

The selection techniques used in this study are Roulette Wheel, Tournament, Linear Ranking and Dynamic Tournament Selection. The Roulette Wheel Selection will assign a segment of the wheel based on individual's fitness value and select the individual corresponding to the selection point when the roulette wheel stops. The Tournament Selection method works by randomly selecting a specified number of unique individuals from the population based on the tournament size and then choose the one with the best fitness value as the parent. The individual wit lowest fitness value is considered as the best in this case. The Linear Ranking Selection assigns the selection probability of individual based on the rank to ensure a controlled selection pressure. The details of Roulette Wheel, Tournament and Linear Ranking can be referred in Chapter 2.

A Dynamic Tournament Selection method where the tournament size is adjusted based on the diversity of the population is proposed in this study. The objective of the Dynamic Tournament Selection is to ensure balance selection pressure and diversity in the population. The diversity of population is first calculated by measuring the standard deviation of fitness value in the population [55]. In this study, the parameter of high and low diversity threshold is set as 0.04 and 0.01 respectively while the range of tournament size are capped in the range of 3 to 7. The diversity threshold was fine tune through trial and error and was found that values between 0.01 and 0.04 produced the desired effect of adaptive tournament size. When the diversity is greater than 0.04, the tournament is decreased to reduce selection pressure. On the other hand, the tournament size is increase when the diversity is less than 0.01. Two of the parents are then selected through a typical tournament selection based on the modified tournament size. The dynamic adjustment allows for an adaptive selection process that helps in maintaining the genetic diversity while also driving the population toward optimal solutions. The flowchart of dynamic tournament selection is shown below.

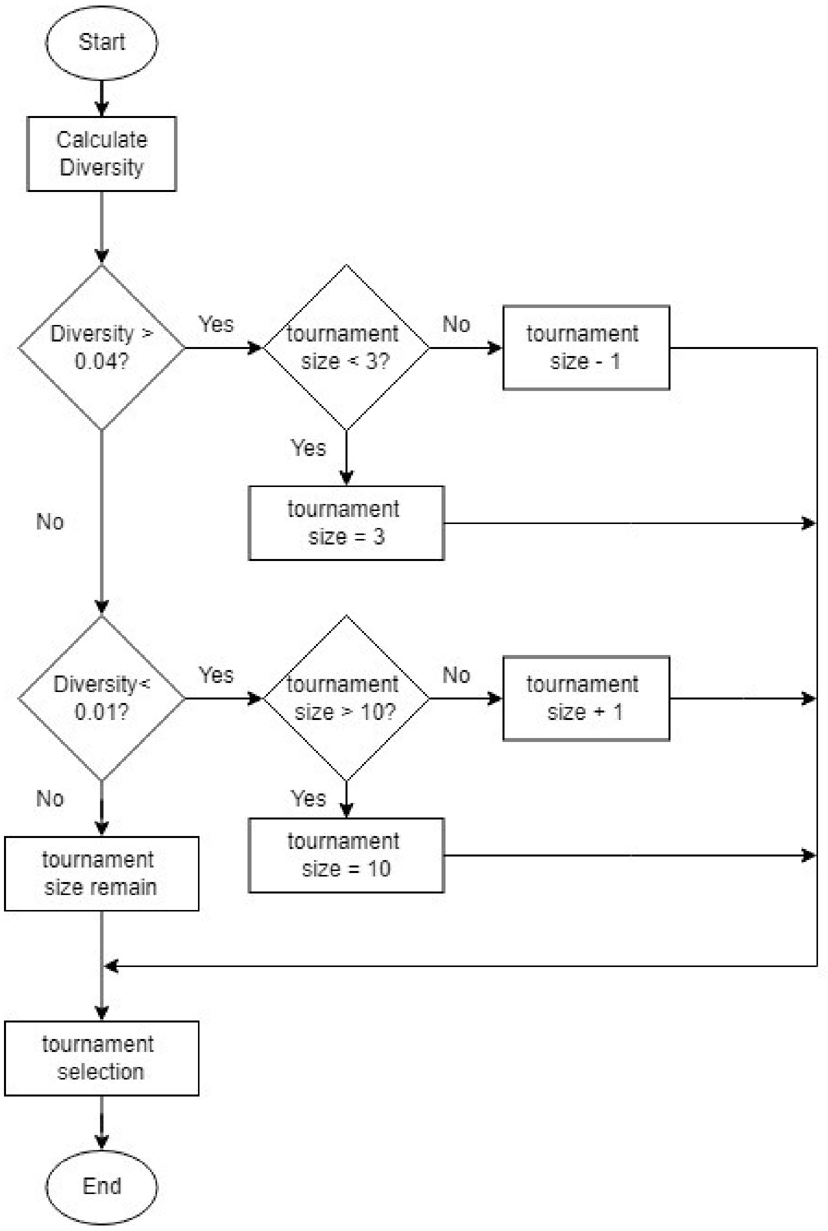


Figure 3.2.1 Flowchart of Dynamic Tournament Selection

3.2.2 Crossover

Written By: Loh Chia Heung 2301684

In genetic algorithms, the crossover operation plays a crucial role in generating new solutions, or offspring, by exchanging genetic information between the parent solutions. This operation helps the algorithm explore different regions of the search space to discover better solutions [19]. The crossover techniques examined in this study include Uniform Crossover, Shuffle Crossover, and Arithmetic Crossover.

In Uniform Crossover, genes from two parents are combined based on a random binary mask (0,1). For each gene position, a random decision is made to assign the gene from either Parent 1 or Parent 2, resulting in offspring with a balanced mix of genes from both parents.

Arithmetic Crossover generates offspring by calculating a weighted average of the parent genes. A random weight (alpha) is applied to each gene position to combine the genes from both parents to create new solutions.

Shuffle Crossover involves randomly shuffling the gene of the parents before performing a crossover operation. After the crossover is completed, then the genes are unshuffled to their original positions.

More detailed descriptions of these 3 crossover techniques can be found in Chapter 2.

The proposed crossover technique, **Combined Crossover** integrates three distinct crossover techniques which are Uniform Crossover, Single-Point Crossover, and Arithmetic Crossover. A detailed explanation of the flow and process of the Combined Crossover is provided below.

Flowchart Of Combined Crossover:

Figure 3.2.2 below illustrates the general process for the Combined Crossover Technique used in the genetic algorithm.

- **Step 1: Select Parents (Results from Selection Operator)**

Two Parents, P1 and P2 are selected for crossover based on the results from the selection operator.

- **Step 2: 1st Crossover – Uniform Crossover**

A crossover probability, ‘dcp’, is compared with a randomly generated value ‘gcp’. If ‘gcp’ is less than or equal to ‘dcp’, Uniform Crossover is performed. For each gene position ‘j’, a random number is generated (0,1).

If the result is 0, gene from Parent 1 (P1) is assigned to Temporary Chromosome 1 (T1) and gene from Parent 2 (P2) is assigned to Temporary Chromosome 2 (T2).

If the result is 1, gene from Parent 2 (P2) is assigned to Temporary Chromosome 1 (T1), and gene from Parent 1 (P1) is assigned to Temporary Chromosome 2 (T2).

If ‘gcp’ is greater than ‘dcp’, The gene are directly copied from the Parents. All genes from Parent 1 (P1) are assigned to Temporary Chromosome 1 (T1) and genes from Parent 2 (P2) are assigned to Temporary Chromosome 2 (T2).

- **Step 3: 2nd Crossover – Single Point Crossover**

A random crossover point is generated between 0 and 1 and converted to integer index with the range of chromosome dimensions. [It is 30 in this project.]

For each gene position, ‘j’:

If ‘j’ is less than the crossover point, assign genes from Temporary Chromosome 1(T1) to Temporary Chromosome 3 (T3) and from Temporary Chromosome 2 (T2) to temporary chromosome 4 (T4).

If ‘j’ is greater than or equal to the crossover point, the assignment is swapped to assign genes from Temporary Chromosome 2 (T2) to Temporary Chromosome 3(T3) and Temporary Chromosome 1 (T1) to Temporary Chromosome 4 (T4).

- **Step 4: 3rd Crossover – Arithmetic Crossover**

A random value alpha is generated between 0 and 1.

For each gene position ‘j’:

The gene value for offspring 1 (C1) is calculated using the formula:

C1 [j] = alpha * T3[j] + (1 - alpha) * T4[j].

The gene value for offspring 2 (C2) is calculated using the formula:

C2[j] = (1 - alpha) * T3[j] + alpha * T4[j]

- **Step 5: End**

Two new offspring chromosomes (C1 and C2) are generated. These offspring will be evaluated for fitness and used in the subsequent evolution process.

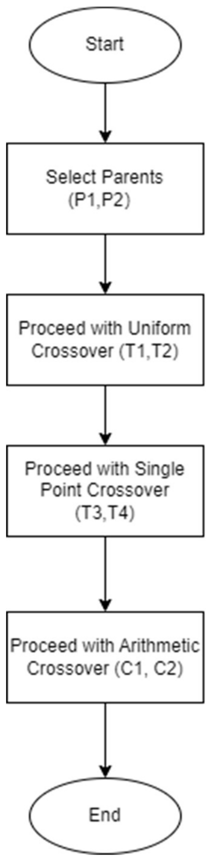

Start
Select Parents (P1,P2)
Proceed with Uniform Crossover (T1,T2)
Proceed with Single Point Crossover (T3,T4)
Proceed with Arithmetic Crossover (C1, C2)
End

Figure 3.2.2 Flowchart of Combined Crossover

This hybrid approach utilizes the strengths of each technique and hope to generate more diverse and potentially optimal offspring.

Firstly, **Uniform Crossover** is performed. This technique allows for any combination of parental genes in the offspring. It potentially leading to a more diverse solutions through increase the genetic diversity and gene variation [20]. This broader range of solution helps explore a wider part of the search space. However, exploitation is limited as gene selection is random [19].

Secondly, **Single Point Crossover** is conducted after the Uniform Crossover. The idea behind Single Point Crossover is to further enhance genetic diversity by combining larger segments of genes from the parents. After Uniform Crossover, which randomly combines the gene from the parents it introduces a more structured approach to recombine the parental genes. Also, single point crossover is valued for its simplicity [20] and efficiency in implementation, which contribute to reduced computational time.

Lastly, **Arithmetic Crossover** is used to balance the gene value between the chromosomes. Specifically, the gene value for an offspring is computed as a weighted average of the corresponding gene values from the two parent chromosomes. It introduces new gene values that are not present in either parent but lie in between the range defined by them, helping to explore the search space more effectively [32]. Additionally, it fine-tunes the offspring towards optima solutions by generating values that are closer to the average of the parents' values, thus facilitating convergence towards high-quality solutions. Therefore, Arithmetic Crossover aims to achieve a balance between exploration and exploitation, as a good balance can speed up the searching process and lead to higher-quality solutions in Genetic Algorithms [19].

3.2.3 Mutation

Written By: Brandon Ting En Junn 2101751

During the mutation operation, a mutation probability is randomly generated to be compared with the determined mutation probability of 0.01 [36]. If the generated mutation probability is less than or equal to the determined mutation probability, the mutation operation executes the mutation operator to perform mutation on the child chromosome. If it is greater, no mutation is performed.

The details of the mutation operators such as Reversing Mutation, Random Mutation, and Simple Inversion Mutation can be referred back to Chapter 2. Reversing Mutation reverses the gene’s position, Random Mutation generates random genes while Simple Inversion Mutation inverses a gene substring.

The proposed mutation operator is the Hybrid Comparing Mutation. It is the combination of the Reversing Mutation and Random Mutation with an additional comparing process. First, a gene of the child chromosome is randomly selected. Next, a new gene is randomly generated (Random Mutation). Then, the newly generated gene compares with the existing selected gene. If the newly generated gene has a value that is closer to zero than the existing selected gene, it replaces the existing selected gene (Comparing). The gene is then reversed to the gene in front of it (Reversing Mutation). Similar to the special case, if the gene is at the first position, it will reverse with the last gene. A flowchart of the Hybrid Comparing Mutation processes is shown in Figure 3.2.3.

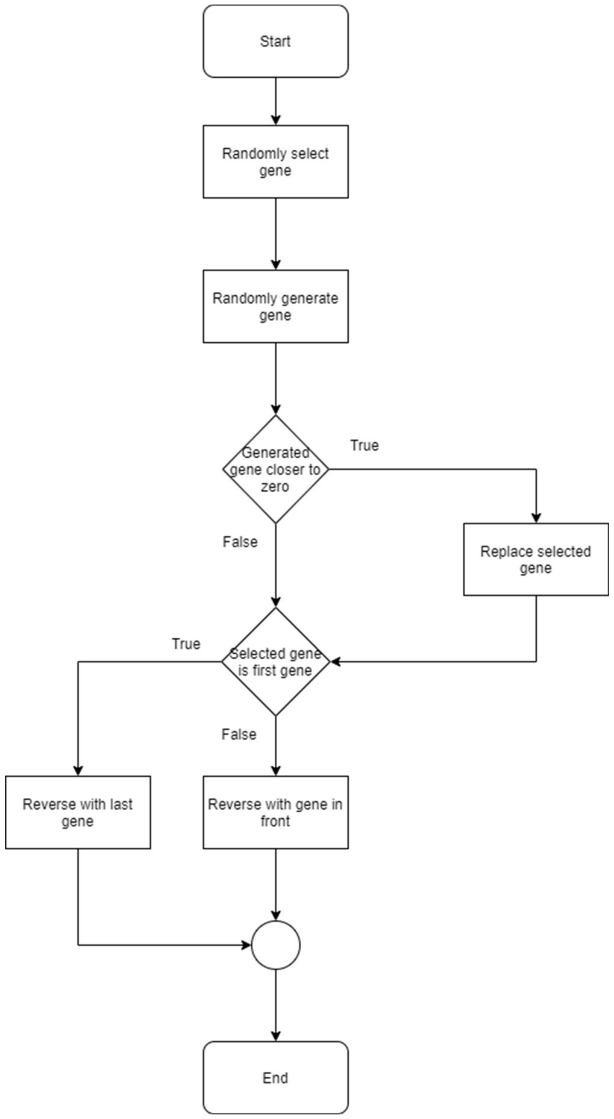


Figure 3.2.3 Hybrid Comparing Mutation Flowchart

3.2.4 Replacement

Written By: Ling Ji Xiang 2104584

The replacement technique is a method use to decide which individuals in a population should be replace by the offspring if the offspring has the stronger fitness value. The goal is to maintain a balance between preserving high-quality solutions and at the mean time also introducing new genetic materials to explore more region of solution space. Therefore, effective replacement strategies help to prevent premature convergence by maintaining diversity within the population.

The replacement techniques that included is Both parent replacement, weak parent replacement and binary tournament replacement. The detail of these three techniques can refer to Chapter 2.

The proposed replacement technique is Combined Replacement Technique. This technique is the hybrid of both parent and weak parent replacement. The idea is using the advantages of weak parent to preserve the stronger individual to next generation to solve the potential of losing valuable genetic materials while using both parent replacement because it might replace the stronger parents with weaker children. This approach can make sure to improve the population diversity while ensuring the weaker individuals are replaced by fitter offspring. The process begins with Both Parent Replacement to replace both parents with offsprings without considering the fitness value. Weak Parent Replacement come after, two parents are compared and choose the weaker, two offsprings are compared and choose the stronger. If the weak parent's fitness worse than strong child's, the replacement process occurs. This step preserves the high-quality genes while eliminating less fir individuals. Additionally, if non-chosen parent has worse fitness than non-chosen child, the replacement also occurs. This approach balances the retention of genetic diversity with pressure to improve overall fitness of the population. The relevant flow of the replacement technique can refer to Figure 3.2.4.

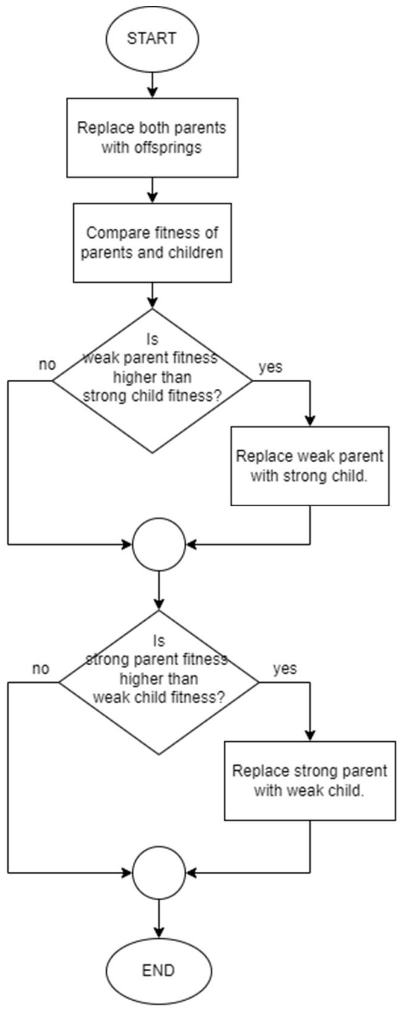

START
Replace both parents with offsprings
Compare fitness of parents and children
Is weak parent fitness higher than strong child fitness?
no
yes
Replace weak parent with strong child.
Is strong parent fitness higher than weak child fitness?
no
yes
Replace strong parent with weak child.
END

Figure 3.2.4 Flowchart of Combined Replacement

3.3 Particle Swarm Optimisation

Written By: Loh Chia Heung 2301684

Figure 3.3.1 illustrates the flowchart for the Particle Swarm Optimization (PSO) algorithm used in this project. The flowchart outlines the key steps involved in the optimization process, including parameter initialization, evaluation of fitness values, updating personal and global bests (P_{best} and G_{best}), and adjusting the velocity and position of the particles.

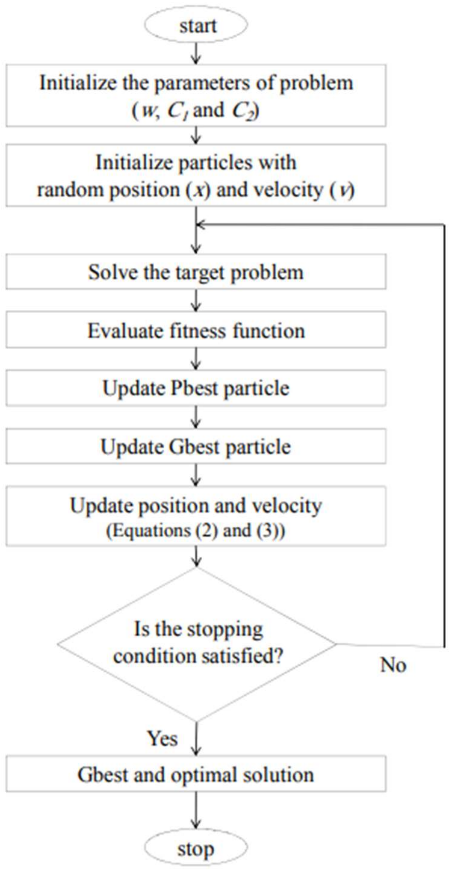

start
Initialize the parameters of problem (w, C_1 and C_2)
Initialize particles with random position (x) and velocity (v)
Solve the target problem
Evaluate fitness function
Update Pbest particle
Update Gbest particle
Update position and velocity (Equations (2) and (3))
Is the stopping condition satisfied?
No
Yes
Gbest and optimal solution
stop

Figure 3.3.1 Flowchart of Particle Swarm Optimization [36]

- **Step 1: Initialize Parameters**

- Set the value for inertia weight, 'w' and the acceleration coefficients 'c1' and 'c2'. These parameters influence the movement of particles within the search space.

- ‘w’ controls the influence of previous velocity, while ‘c1’ and ‘c2’ determine the particle's attraction to its own best-known position (P_{best}) and the swarm's best-known position (G_{best}).

- In this project, the inertia weight, ‘w’ is set to 0.7 and the acceleration coefficients ‘c1’ and ‘c2’ is set to 2.

- **Step 2: Initialize Random Positions and Velocity**

- Initialize the position (x) randomly and velocity is initialized as all 0 at first.

- Additionally, in this project, the number of particles ‘n’ is set to 40, which defines the swarm size, and the dimension size is set to 30, representing the number of variables being optimized.

- **Step 3: Solve the target problem**

- In this project, the target problem is to optimize a benchmark function to find the optimal solution. Therefore, PSO algorithm seeks to **minimize** the function's output based on particle positions in this case.

- **Step 4: Evaluate Fitness of Each Particle**

- The fitness of each particle is evaluated using the benchmark function. Fitness value here reflects how close the particle is to the optimal solution.

- **Step 5: Update Personal Best Position (P_{best})**

- For each particle, compare its current fitness value with its previous personal best position. Initially, it is set to 0 for the first iteration. If the current fitness is better, update P_{best} to the current position. P_{best} is the best position a particle has achieved during its search.

- **Step 6: Update Global Best (G_{best})**

- Group Best Particle, G_{best} is identified among all the particles. Then it is updated with the particle position, which represents the best solution found by the group of swarms so far.

- **Step 7: Update Velocity and Position**

- Velocity is updated based on both P_{best} and G_{best} and position is updated based on the Velocity, which allows the particles to move within the search space, getting closer to better solutions.

- **Step 8: Check Termination Criteria**

- The algorithm will continue iterating until the termination condition is met.

- In this project, the termination condition is set to 2000 iterations. If the number of iterations reaches 2000, the algorithm ends. If the termination condition is met, the algorithm stops, and the best solution found, G_{best}, is returned as the optimal solution.

- If the termination condition is not met, the process returns to solving the target problem. This involves evaluating the fitness of particles again using the benchmark function to continue the optimization process. This iterative approach allows the PSO algorithm to refine and improve the solutions over time.

3.4 Parameters Setting and Combination of GA Models

Written By: Brandon Ting En Junn 2101751

The parameters settings for the GA and PSO models are shown in Table 3.4.1 and Table 3.4.2 respectively. The 10 benchmark functions are also shown in Table 3.4.3.

Table 3.4.1 Parameter Settings for the GA Models [36]

No	Parameters	Value
1.	Number of Generations	2000
2.	Population Size	40
3.	Dimension Size	30
4.	Crossover Probability	0.7
5.	Mutation Probability	0.01
6.	Number of Testing	10

7.	Tournament Size	5
8.	High Diversity Threshold	0.04
9.	Low Diversity Threshold	0.01

Number of generations determine the number of iterations that the GA will go through. Population size and dimension size determine the number of chromosomes in the population and number of genes of each chromosome.

Additionally, the crossover probability and mutation probability determine the rate of crossover and mutation respectively, in which the likelihood that the operation performs. Number of testing indicate the number of experiments being conducted for each benchmark function to obtain the averages. The suggested tournament size of 5 was referred from [56]. The diversity threshold is the measurement of population diversity using standard deviation. A range between 0.01 and 0.04 is applied to ensure the balance between selection pressure and diversity among the population.

Table 3.4.2 Parameter Settings for the PSO Models [50]

No	**Parameters**	**Value**
1.	Number of Generations	2000
2.	Population Size	40
3.	Dimension Size	30
4.	Number of Testing	10
5.	Maximum Position	Range of the dataset
6.	Maximum Velocity	Half of the range of the dataset
7.	Inertia Weight, w	0.7
8.	Random Number (r_1, r_2)	[0, 1]
9.	Acceleration Constant (c_1, c_2)	2

Maximum position and maximum velocity indicate the upper and lower boundaries of the allowed values of the positions and velocities of the particles. Velocity clamping, *vc* is based on the maximum velocity imposed. Inertia weight, *w* controls the rate of exploration and exploitation. Random numbers, r_1, r_2 and acceleration constants, c_1, c_2 contribute to the calculation of the updated velocity for the update of the particles' positions.

Table 3.4.3 Benchmark Function

No.	**Benchmark Function**	**Range**	**Global Minimum**
1.	Sphere	[-5.12, 5.12]	0
2.	Ackley	[-32.768, 32.768]	0
3.	Rastrigin	[-5.12, 5.12]	0
4.	Zakharov	[-5, 10]	0
5.	Axis Parallel Hyper-Ellipsoid	[-5.12, 5.12]	0
6.	Griewank	[-600, 600]	0
7.	Sum of Different Powers	[-1, 1]	0
8.	Rotated Hyper-Ellipsoid	[-65.536, 65.536]	0
9.	Schwefel 2.22	[-5, 5]	0
10.	Exponential	[-1, 1]	-1

A total of 256 GA models were implemented with unique combinations. 4 techniques of each operation technique were uniquely combined and evaluated to find the best GA model combination for the performance comparison of PSO. The table of all 256 GA models with its corresponding operation technique combinations are attached in the appendix.

CHAPTER 4

System Evaluation and Discussion

Written By: Brandon Ting En Junn 2101751

This chapter shows the results of the experiment to conduct further analysis and discussion regarding the performance of the models.

4.1 System Testing and Performance Metrics

Written By: Yeap Chun Hong 2206352

Unit testing is conducted to ensure the correctness and functionality for each selection, crossover, mutation, and replacement technique operates as expected. After that, integration testing is performed to ensure that all of the operation techniques work together as a GA model seamlessly. Defects such as negative fitness value and the negative value generated repeatedly by random function are identified and fixed from component interactions.

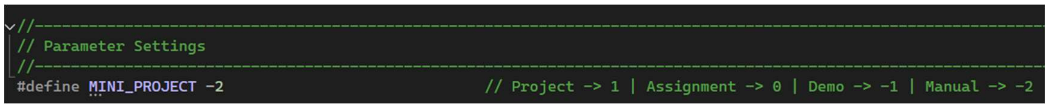
The performance metrics used to evaluate the effectiveness and efficiency of GA models are minimum, maximum and average of fitness value for 10 experiments per benchmark functions. Furthermore, the time taken for each experiment is also recorded to find the average computational time on the GA models.

4.2 Testing Setup and Result

4.2.1 Testing Setup

Written By: Ling Ji Xiang 2104584

For the ease of testing each function is running as expected, the program is designed to have 4 modes of running methods which is Project (1), Assignment (0), Demo (-1), Manual (-2) which can be define at the parameter settings before running the program. Figure 4.2.1 shows the implementation of project mode definition for the experiment.



```
//--------------------------------------------------------------------------------------------------------
// Parameter Settings
//--------------------------------------------------------------------------------------------------------
#define MINI_PROJECT -2                    // Project -> 1 | Assignment -> 0 | Demo -> -1 | Manual -> -2
```

Figure 4.2.1 Project Mode Definition

Project: Total of 256 GA results will produce in this mode, each 4 operation techniques (S, C, M, R) have 4 models. Each model will run through every benchmark function 10 times to produce the results.

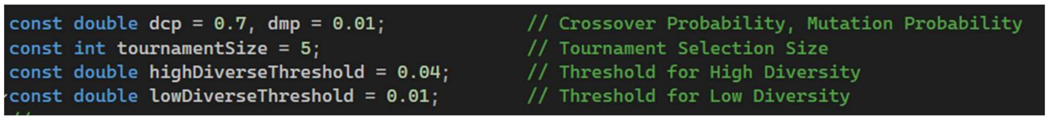
Assignment: Total of 64 GA results will produce in this mode, each 4 operation techniques (S, C, M, R) have 2 models. Each model will run through every benchmark function 10 times to produce the results.

Demo: Demo will show only 1 iteration of techniques being used (S, C, M, R), but will output more details in cmd.

Manual: Manual will show less details but run through the whole 10 benchmarks + 10 experiments.

Unit Testing

Unit test is to test whether the particular function is working as expected. The tournament size of 5 means the selection method only choose 5 individuals to compare their fitness values. Figure 4.2.2 shows the implementation of the parameter settings.



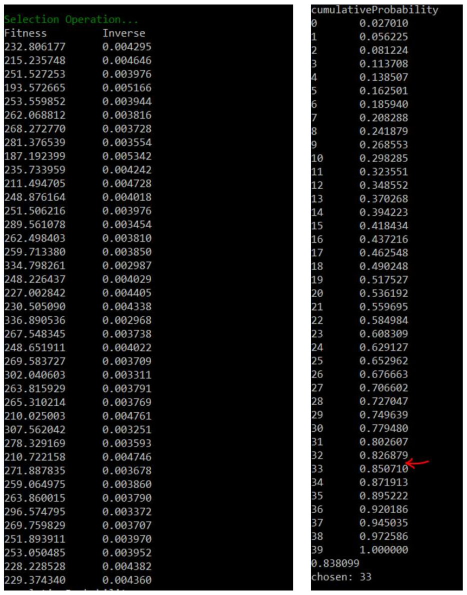
```
const double dcp = 0.7, dmp = 0.01;          // Crossover Probability, Mutation Probability
const int tournamentSize = 5;                // Tournament Selection Size
const double highDiverseThreshold = 0.04;    // Threshold for High Diversity
const double lowDiverseThreshold = 0.01;     // Threshold for Low Diversity
```

Figure 4.2.2 Parameter Settings Implementation

Written By: Yeap Chun Hong 2203652

Roulette Wheel Selection

Figure 4.2.3 shows the operation of Roulette Wheel Selection. The individual with lower fitness value will obtain higher inverse fitness, which indicates it have a larger proportion of roulette. For instance, the first individual with 232.806 fitness is having a lower inverse fitness value compared to the second individual with 215.235 fitness. After that, the inverse fitness value is normalised into range of [0,1] and a random number of 0.838099 is generated. The random number lies between 32nd and 33rd individual and therefore the 33rd individual is chosen as parent.



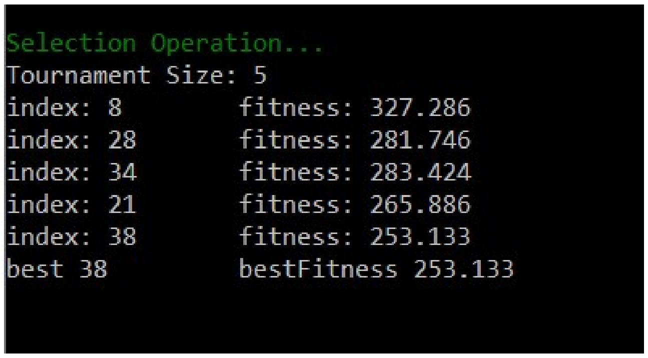
```
Selection Operation...
Fitness         Inverse
232.806177      0.004295
215.235748      0.004646
251.527253      0.003976
193.572665      0.005166
253.559852      0.003944
262.068812      0.003816
268.272770      0.003728
281.376539      0.003554
187.192399      0.005342
235.733959      0.004242
211.494705      0.004728
248.876164      0.004018
251.506216      0.003976
289.561078      0.003454
262.498403      0.003810
259.713380      0.003850
334.798261      0.002987
248.226437      0.004029
227.002842      0.004405
230.505090      0.004338
336.890536      0.002968
267.548345      0.003738
248.651911      0.004022
269.583727      0.003709
302.040603      0.003311
263.815929      0.003791
265.310214      0.003769
210.025003      0.004761
307.562042      0.003251
278.329169      0.003593
210.722158      0.004746
271.887835      0.003678
259.064975      0.003860
263.860015      0.003790
296.574795      0.003372
269.759829      0.003707
251.893911      0.003970
253.050485      0.003952
228.228528      0.004382
229.374340      0.004360
```

```
cumulativeProbability
0       0.027010
1       0.056225
2       0.081224
3       0.113708
4       0.138507
5       0.162501
6       0.185940
7       0.208288
8       0.241879
9       0.268553
10      0.298285
11      0.323551
12      0.348552
13      0.370268
14      0.394223
15      0.418434
16      0.437216
17      0.462548
18      0.490248
19      0.517527
20      0.536192
21      0.559695
22      0.584984
23      0.608309
24      0.629127
25      0.652962
26      0.676663
27      0.706602
28      0.727047
29      0.749639
30      0.779480
31      0.802607
32      0.826879
33      0.850710
34      0.871913
35      0.895222
36      0.920186
37      0.945035
38      0.972586
39      1.000000
0.838099
chosen: 33
```

Figure 4.2.3 Operation of Roulette Wheel Selection

Tournament Selection

Figure 4.2.4 shows the operation of Tournament Selection. The system will select the number of individuals based on the tournament size. For example, tournament size of 5 is used in this study and individual 8, 28, 34, 21, 38 is chosen to compete among each other. Individual 38 wins the tournament and will be selected as parent as it has the lowest fitness value of 253.133.

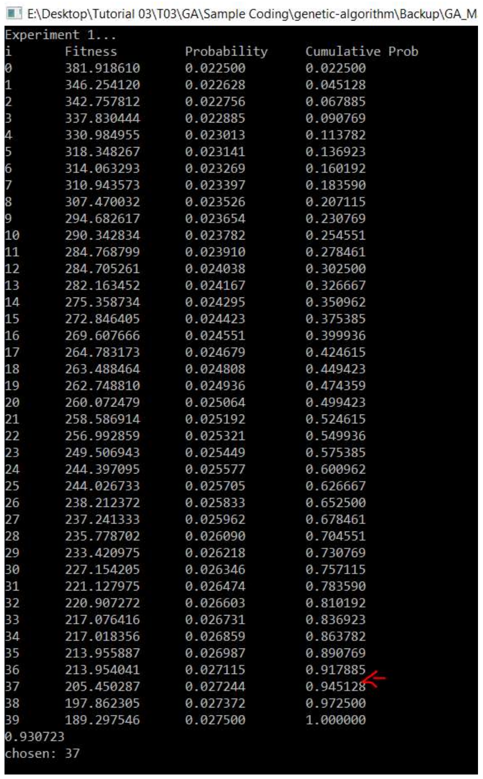


```
Selection Operation...
Tournament Size: 5
index: 8        fitness: 327.286
index: 28       fitness: 281.746
index: 34       fitness: 283.424
index: 21       fitness: 265.886
index: 38       fitness: 253.133
best 38         bestFitness 253.133
```

Figure 4.2.4 Operation of Tournament Selection

Linear Ranking Selection

Figure 4.2.5 shows the operation of Linear Ranking Selection. The individual with lower fitness gets a higher probability as it has a larger number of ranks. For example, individual 39 have probability of 0.0275 to be selected while individual 0 stands a chance of 0.0225 to be selected. A random number of 0.930723 is generated, where it lies between individual 3 and 37 thus individual 37 will be selected as the parent.

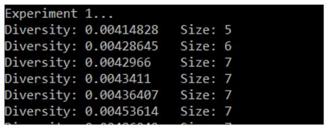


```
E:\Desktop\Tutorial 03\T03\GA\Sample Coding\genetic-algorithm\Backup\GA_M
Experiment 1...
i       Fitness         Probability     Cumulative Prob
0       381.918610      0.022500        0.022500
1       346.254120      0.022628        0.045128
2       342.757812      0.022756        0.067885
3       337.830444      0.022885        0.090769
4       330.984955      0.023013        0.113782
5       318.348267      0.023141        0.136923
6       314.063293      0.023269        0.160192
7       310.943573      0.023397        0.183590
8       307.470032      0.023526        0.207115
9       294.682617      0.023654        0.230769
10      290.342834      0.023782        0.254551
11      284.768799      0.023910        0.278461
12      284.705261      0.024038        0.302500
13      282.163452      0.024167        0.326667
14      275.358734      0.024295        0.350962
15      272.846405      0.024423        0.375385
16      269.607666      0.024551        0.399936
17      264.783173      0.024679        0.424615
18      263.488464      0.024808        0.449423
19      262.748810      0.024936        0.474359
20      260.072479      0.025064        0.499423
21      258.586914      0.025192        0.524615
22      256.992859      0.025321        0.549936
23      249.506943      0.025449        0.575385
24      244.397095      0.025577        0.600962
25      244.026733      0.025705        0.626667
26      238.212372      0.025833        0.652500
27      237.241333      0.025962        0.678461
28      235.778702      0.026090        0.704551
29      233.420975      0.026218        0.730769
30      227.154205      0.026346        0.757115
31      221.127975      0.026474        0.783590
32      220.907272      0.026603        0.810192
33      217.076416      0.026731        0.836923
34      217.018356      0.026859        0.863782
35      213.955887      0.026987        0.890769
36      213.954041      0.027115        0.917885
37      205.450287      0.027244        0.945128 ←
38      197.862305      0.027372        0.972500
39      189.297546      0.027500        1.000000
0.930723
chosen: 37
```

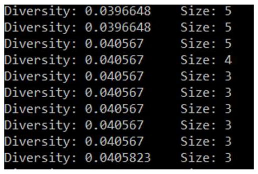
Figure 4.2.5 Operation of Linear Ranking Selection

Dynamic Tournament Selection

Figure 4.2.6 shows the operation of Dynamic Tournament Selection. (a) shows the dynamic tournament selection in Sphere Function while (b) shows in Zakharov function. When the diversity is less than 0.01, the tournament size is increase. On the other hand, the tournament size will decrease when the diversity is greater than 0.04.



```
Experiment 1...
Diversity: 0.00414828   Size: 5
Diversity: 0.00428645   Size: 6
Diversity: 0.0042966    Size: 7
Diversity: 0.0043411    Size: 7
Diversity: 0.00436407   Size: 7
Diversity: 0.00453614   Size: 7
```

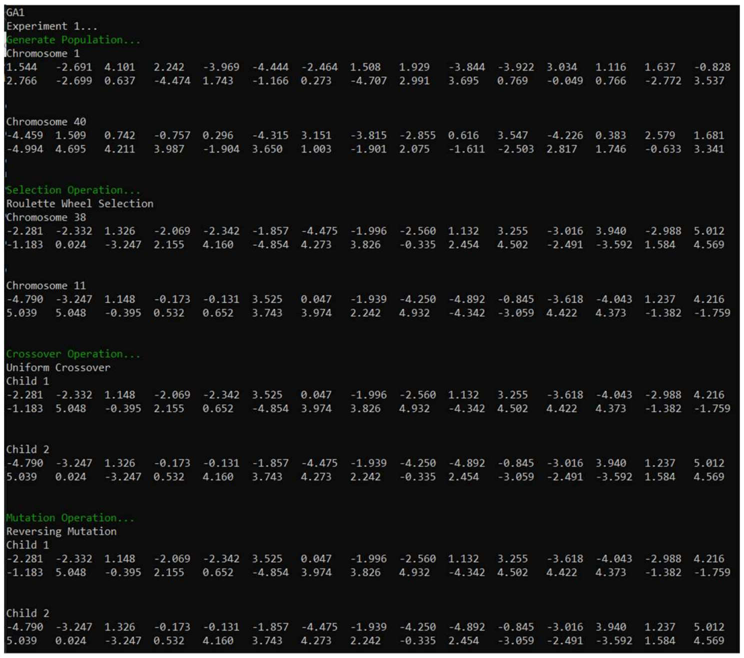


```
Diversity: 0.0396648    Size: 5
Diversity: 0.0396648    Size: 5
Diversity: 0.040567     Size: 5
Diversity: 0.040567     Size: 4
Diversity: 0.040567     Size: 3
Diversity: 0.040567     Size: 3
Diversity: 0.040567     Size: 3
Diversity: 0.040567     Size: 3
Diversity: 0.040567     Size: 3
Diversity: 0.0405823    Size: 3
```

Figure 4.2.6 Operation of Dynamic Tournament Selection

Written By: Ling Ji Xiang 2104584
Integration Testing

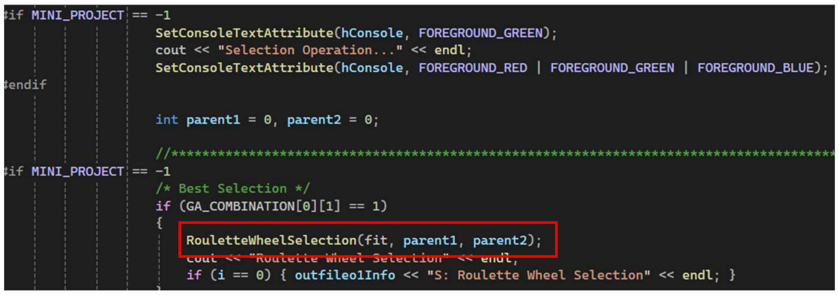
This test uses Demo, which show more detail with only 1 model of each technique being used in this test. This test can ensure the operation techniques work fine together. Coding: Figure 4.2.7 shows the interface for the demo testing.



```
GA1
Experiment 1...
Generate Population...
Chromosome 1
1.544   -2.691  4.101   2.242   -3.969  -4.444  -2.464  1.508   1.929   -3.844  -3.922  3.034   1.116   1.637   -0.828
2.766   -2.699  0.637   -4.474  1.743   -1.166  0.273   -4.707  2.991   3.695   0.769   -0.049  0.766   -2.772  3.537


Chromosome 40
-4.459  1.509   0.742   -0.757  0.296   -4.315  3.151   -3.815  -2.855  0.616   3.547   -4.226  0.383   2.579   1.681
-4.994  4.695   4.211   3.987   -1.904  3.650   1.003   -1.901  2.075   -1.611  -2.503  2.817   1.746   -0.633  3.341


Selection Operation...
Roulette Wheel Selection
Chromosome 38
-2.281  -2.332  1.326   -2.069  -2.342  -1.857  -4.475  -1.996  -2.560  1.132   3.255   -3.016  3.940   -2.988  5.012
-1.183  0.024   -3.247  2.155   4.160   -4.854  4.273   3.826   -0.335  2.454   4.502   -2.491  -3.592  1.584   4.569


Chromosome 11
-4.790  -3.247  1.148   -0.173  -0.131  3.525   0.047   -1.939  -4.250  -4.892  -0.845  -3.618  -4.043  1.237   4.216
5.039   5.048   -0.395  0.532   0.652   3.743   3.974   2.242   4.932   -4.342  -3.059  4.422   4.373   -1.382  -1.759


Crossover Operation...
Uniform Crossover
Child 1
-2.281  -2.332  1.148   -2.069  -2.342  3.525   0.047   -1.996  -2.560  1.132   3.255   -3.618  -4.043  -2.988  4.216
-1.183  5.048   -0.395  2.155   0.652   -4.854  3.974   3.826   4.932   -4.342  4.502   4.422   4.373   -1.382  -1.759


Child 2
-4.790  -3.247  1.326   -0.173  -0.131  -1.857  -4.475  -1.939  -4.250  -4.892  -0.845  -3.016  3.940   1.237   5.012
5.039   0.024   -3.247  0.532   4.160   3.743   4.273   2.242   -0.335  2.454   -3.059  -2.491  -3.592  1.584   4.569


Mutation Operation...
Reversing Mutation
Child 1
-2.281  -2.332  1.148   -2.069  -2.342  3.525   0.047   -1.996  -2.560  1.132   3.255   -3.618  -4.043  -2.988  4.216
-1.183  5.048   -0.395  2.155   0.652   -4.854  3.974   3.826   4.932   -4.342  4.502   4.422   4.373   -1.382  -1.759


Child 2
-4.790  -3.247  1.326   -0.173  -0.131  -1.857  -4.475  -1.939  -4.250  -4.892  -0.845  -3.016  3.940   1.237   5.012
5.039   0.024   -3.247  0.532   4.160   3.743   4.273   2.242   -0.335  2.454   -3.059  -2.491  -3.592  1.584   4.569
```

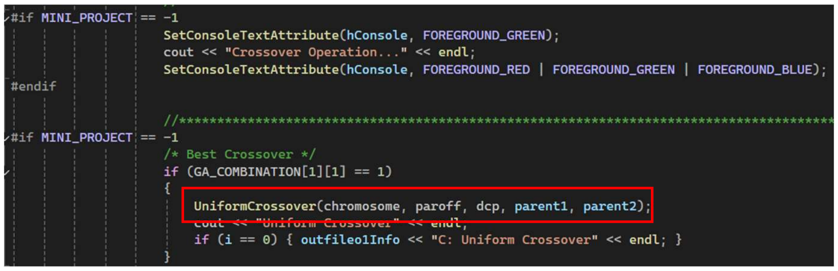
Figure 4.2.7 Demo Testing Interface

Figure 4.2.8 to Figure 4.2.11 show the demo testing of each technique. In order to check other operators, simply just change calling function that highlighted in the Figure 4.2.8 to Figure 4.2.11 to other function that mentioned in Table 4.2.1.



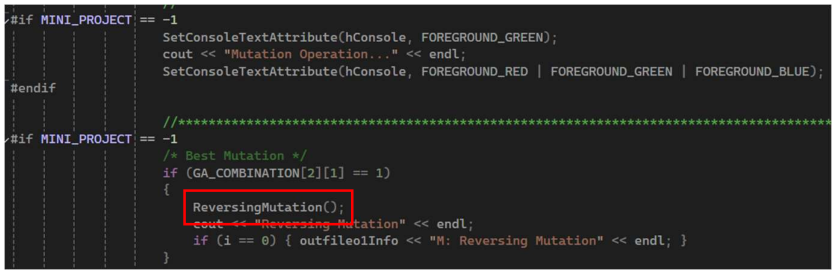
```
#if MINI_PROJECT == -1
                SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
                cout << "Selection Operation..." << endl;
                SetConsoleTextAttribute(hConsole, FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE);
#endif

                int parent1 = 0, parent2 = 0;

                //****************************************************************************************
#if MINI_PROJECT == -1
                /* Best Selection */
                if (GA_COMBINATION[0][1] == 1)
                {
                    RouletteWheelSelection(fit, parent1, parent2);
                    cout << "Roulette Wheel Selection" << endl;
                    if (i == 0) { outfileo1Info << "S: Roulette Wheel Selection" << endl; }
```

Figure 4.2.8 Selection Method



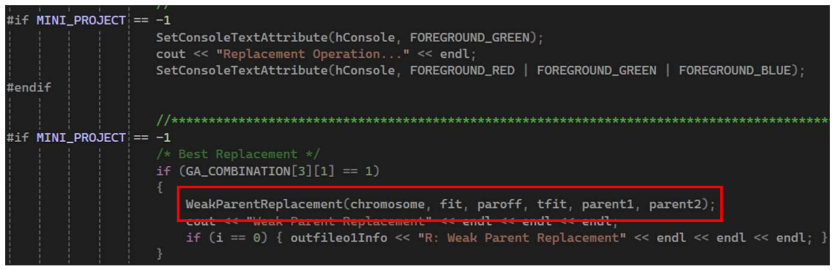
```
#if MINI_PROJECT == -1
                SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
                cout << "Crossover Operation..." << endl;
                SetConsoleTextAttribute(hConsole, FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE);
#endif

                //****************************************************************************************
#if MINI_PROJECT == -1
                /* Best Crossover */
                if (GA_COMBINATION[1][1] == 1)
                {
                    UniformCrossover(chromosome, paroff, dcp, parent1, parent2);
                    cout << "Uniform Crossover" << endl;
                    if (i == 0) { outfileo1Info << "C: Uniform Crossover" << endl; }
                }
```

Figure 4.2.9 Crossover Method



```
#if MINI_PROJECT == -1
                SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
                cout << "Mutation Operation..." << endl;
                SetConsoleTextAttribute(hConsole, FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE);
#endif

                //****************************************************************************************
#if MINI_PROJECT == -1
                /* Best Mutation */
                if (GA_COMBINATION[2][1] == 1)
                {
                    ReversingMutation();
                    cout << "Reversing Mutation" << endl;
                    if (i == 0) { outfileo1Info << "M: Reversing Mutation" << endl; }
                }
```

Figure 4.2.10 Mutation Method



```
#if MINI_PROJECT == -1
                SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
                cout << "Replacement Operation..." << endl;
                SetConsoleTextAttribute(hConsole, FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE);
#endif

                //****************************************************************************************
#if MINI_PROJECT == -1
                /* Best Replacement */
                if (GA_COMBINATION[3][1] == 1)
                {
                    WeakParentReplacement(chromosome, fit, paroff, tfit, parent1, parent2);
                    cout << "Weak Parent Replacement" << endl << endl << endl;
                    if (i == 0) { outfileo1Info << "R: Weak Parent Replacement" << endl << endl << endl; }
                }
```

Figure 4.2.11 Replacement Method

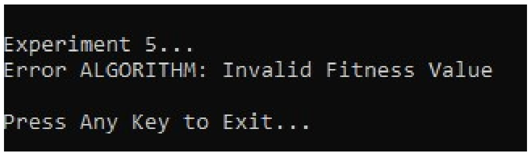
Table 4.2.1 Operation Techniques

Selection	Roulette Wheel	Tournament	Linear Ranking	Dynamic Tournament
Crossover	Uniform	Shuffle	Arithmetic	Combined
Mutation	Reversing	Random	Simple Inversion	Hybrid Comparing
Replacement	Both Parent	Weak Parent	Binary Tournament	Combined

Defect Testing

To ensure the result won't produce negative fitness value, the program will terminate when it detects a negative fitness and shows the output shown in Figure 4.2.12. If the program did not produce any negative fitness value, then it will run until termination criteria meets which is 2000 iterations and show the output shown in Figure 4.2.13.

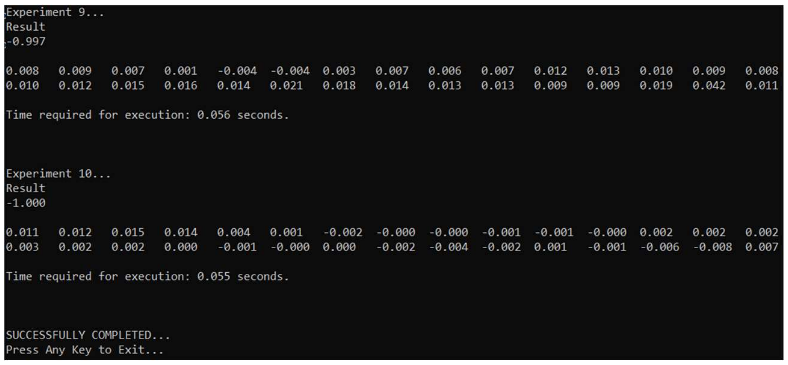
Output when negative fitness detected:



```
Experiment 5...
Error ALGORITHM: Invalid Fitness Value

Press Any Key to Exit...
```

Figure 4.2.12 Error Termination Output

Output when no negative value detected:

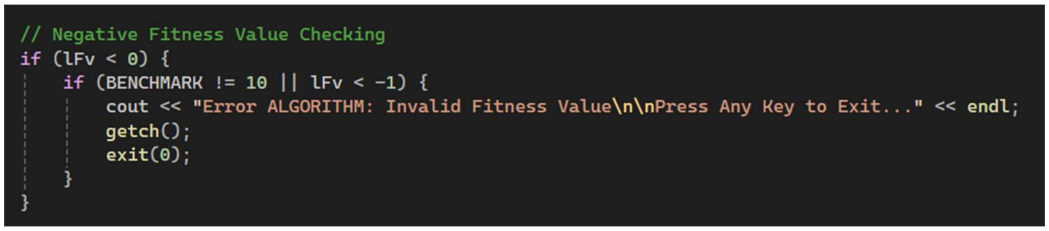


```
Experiment 9...
Result
-0.997

0.008   0.009   0.007   0.001   -0.004  -0.004  0.003   0.007   0.006   0.007   0.012   0.013   0.010   0.009   0.008
0.010   0.012   0.015   0.016   0.014   0.021   0.018   0.014   0.013   0.013   0.009   0.009   0.019   0.042   0.011

Time required for execution: 0.056 seconds.



Experiment 10...
Result
-1.000

0.011   0.012   0.015   0.014   0.004   0.001   -0.002  -0.000  -0.000  -0.001  -0.001  -0.000  0.002   0.002   0.002
0.003   0.002   0.002   0.000   -0.001  -0.000  0.000   -0.002  -0.004  -0.002  0.001   -0.001  -0.006  -0.008  0.007

Time required for execution: 0.055 seconds.



SUCCESSFULLY COMPLETED...
Press Any Key to Exit...
```

Figure 4.2.13 Success Termination Output

Coding:

The Figure 4.2.14 shows the coding of displaying error output when detect negative fitness value.

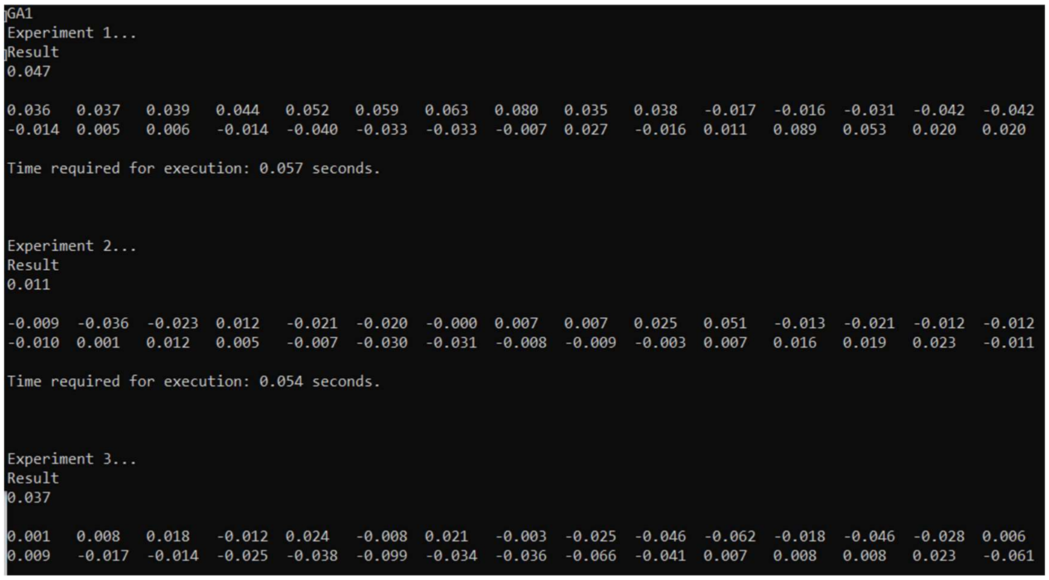


```cpp
// Negative Fitness Value Checking
if (lFv < 0) {
    if (BENCHMARK != 10 || lFv < -1) {
        cout << "Error ALGORITHM: Invalid Fitness Value\n\nPress Any Key to Exit..." << endl;
        getch();
        exit(0);
    }
}
```

Figure 4.2.14 Error Output Coding

Performance Metrics

To check the system performance, the program run in Manual mode which will simulate the final program but to only produce 1 GA result instead of all 256 GA. From the results produced, we can check and discuss the values, to ensure that the result is in the acceptable range. Figure 4.2.15 shows the interface for manual testing.



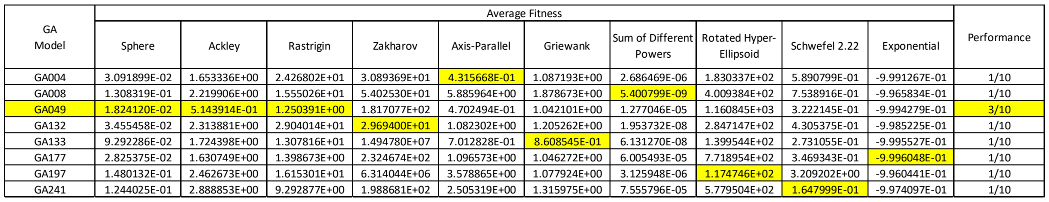
```
GA1
Experiment 1...
Result
0.047

0.036  0.037  0.039  0.044  0.052  0.059  0.063  0.080  0.035  0.038  -0.017 -0.016 -0.031 -0.042 -0.042
-0.014 0.005  0.006  -0.014 -0.040 -0.033 -0.033 -0.007 0.027  -0.016 0.011  0.089  0.053  0.020  0.020

Time required for execution: 0.057 seconds.


Experiment 2...
Result
0.011

-0.009 -0.036 -0.023 0.012  -0.021 -0.020 -0.000 0.007  0.007  0.025  0.051  -0.013 -0.021 -0.012 -0.012
-0.010 0.001  0.012  0.005  -0.007 -0.030 -0.031 -0.008 -0.009 -0.003 0.007  0.016  0.019  0.023  -0.011

Time required for execution: 0.054 seconds.


Experiment 3...
Result
0.037

0.001  0.008  0.018  -0.012 0.024  -0.008 0.021  -0.003 -0.025 -0.046 -0.062 -0.018 -0.046 -0.028 0.006
0.009  -0.017 -0.014 -0.025 -0.038 -0.099 -0.034 -0.036 -0.066 -0.041 0.007  0.008  0.008  0.023  -0.061
```

Figure 4.2.15 Manual Testing Interface

4.2.2 Results

Written By: Ling Ji Xiang 2104584

This section shows the results of GA based on average fitness value and average time. The full table can refer to appendix. From Figure 4.2.16, we highlight that GA004, GA008, GA049, GA132, GA133, GA177, GA197, GA241 achieve the minimum average fitness value in certain benchmark functions. The combination of the outperformed GA based on average fitness value is shown in Table 4.2.2. Additionally, the well performed GA and the respective benchmark functions are summarize in Table 4.2.3.



GA Model	Average Fitness										Performance
	Sphere	Ackley	Rastrigin	Zakharov	Axis-Parallel	Griewank	Sum of Different Powers	Rotated Hyper-Ellipsoid	Schwefel 2.22	Exponential	
GA004	3.091899E-02	1.653336E+00	2.426802E+01	3.089369E+01	4.315668E-01	1.087193E+00	2.686469E-06	1.830337E+02	5.890799E-01	-9.991267E-01	1/10
GA008	1.308319E-01	2.219906E+00	1.555026E+01	5.402530E+01	5.885964E+00	1.878673E+00	5.400799E-09	4.009384E+02	7.538916E-01	-9.965834E-01	1/10
GA049	1.824120E-02	5.143914E-01	1.250391E+00	1.817077E+02	4.702494E-01	1.042101E+00	1.277046E-05	1.160845E+03	3.222145E-01	-9.994279E-01	3/10
GA132	3.455458E-02	2.313881E+00	2.904014E+01	2.969400E+01	1.082302E+00	1.205262E+00	1.953732E-08	2.847147E+02	4.305375E-01	-9.985225E-01	1/10
GA133	9.292286E-02	1.724398E+00	1.307816E+01	1.494780E+07	7.012828E-01	8.608545E-01	6.131270E-08	1.399544E+02	2.731055E-01	-9.995527E-01	1/10
GA177	2.825375E-02	1.630749E+00	1.398673E+00	2.324674E+02	1.096573E+00	1.046272E+00	6.005493E-05	7.718954E+02	3.469343E-01	-9.996048E-01	1/10
GA197	1.480132E-01	2.462673E+00	1.615301E+01	6.314044E+06	3.578865E+00	1.077924E+00	3.125948E-06	1.174746E+02	3.209202E+00	-9.960441E-01	1/10
GA241	1.244025E-01	2.888853E+00	9.292877E+00	1.988681E+02	2.505319E+00	1.315975E+00	7.555796E-05	5.779504E+02	1.647999E-01	-9.974097E-01	1/10

Figure 4.2.16 Outperforming GA Model based on Average Fitness Value

(E-2 = $\times 10^{-2}$, E+2 = $\times 10^{2}$)

Table 4.2.2 Combinations of Outperformed GA based on Average Fitness Value

Model	Selection	Crossover	Mutation	Replacement
GA004	Dynamic Tournament	Combined	Hybrid Comparing	Weak Parent
GA008	Dynamic Tournament	Combined	Simple Inversion	Weak Parent
GA049	Dynamic Tournament	Uniform	Hybrid Comparing	Combined
GA132	Tournament	Combined	Hybrid Comparing	Weak Parent
GA133	Tournament	Combined	Simple Inversion	Combined

GA177	Tournament	Uniform	Hybrid Comparing	Combined
GA197	Roulette Wheel	Combined	Simple Inversion	Combined
GA241	Roulette Wheel	Uniform	Hybrid Comparing	Combined

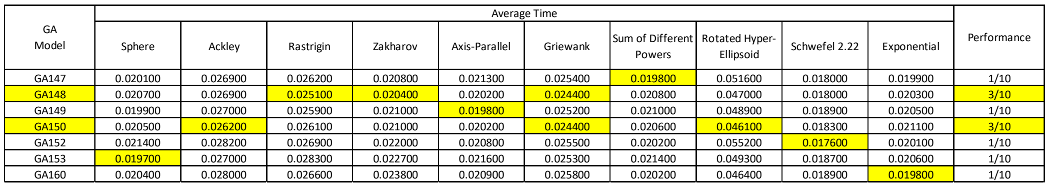
Table 4.2.3 Outperformed GA Models based on Average Fitness with Respective Benchmark Functions

	GA004	GA008	GA049	GA132	GA133	GA177	GA197	GA241
Sphere			✓					
Ackley			✓					
Rastrigin			✓					
Zakhorov				✓				
Axis-Parallel	✓							
Griewank					✓			
Sum of Different Power		✓						
Rotated Hyper-Ellipsoid							✓	
Schwefel 2.22								✓
Exponential						✓		

Performance	1/10	1/10	3/10	1/10	1/10	1/10	1/10	1/10

Average Time

From the results, we highlighted several GA models that contain the well performed results, these GA models include GA147, GA148, GA149, GA150, GA152, GA153, GA160 from Figure 4.2.17. These models having minimum average time compared to other GA models. The combination of the outperformed GA based on average time is shown in Table 4.2.4. Additionally, the well performed GA and the respective benchmark functions are summarize in Table 4.2.5.



GA Model	Average Time										Performance
	Sphere	Ackley	Rastrigin	Zakharov	Axis-Parallel	Griewank	Sum of Different Powers	Rotated Hyper-Ellipsoid	Schwefel 2.22	Exponential	
GA147	0.020100	0.026900	0.026200	0.020800	0.021300	0.025400	0.019800	0.051600	0.018000	0.019900	1/10
GA148	0.020700	0.026900	0.025100	0.020400	0.020200	0.024400	0.020800	0.047000	0.018000	0.020300	3/10
GA149	0.019900	0.027000	0.025900	0.021000	0.019800	0.025200	0.021000	0.048900	0.018900	0.020500	1/10
GA150	0.020500	0.026200	0.026100	0.021000	0.020200	0.024400	0.020600	0.046100	0.018300	0.021100	3/10
GA152	0.021400	0.028200	0.026900	0.022000	0.020800	0.025500	0.020200	0.055200	0.017600	0.020100	1/10
GA153	0.019700	0.027000	0.028300	0.022700	0.021600	0.025300	0.021400	0.049300	0.018700	0.020600	1/10
GA160	0.020400	0.028000	0.026600	0.023800	0.020900	0.025800	0.020200	0.046400	0.018900	0.019800	1/10

Figure 4.2.17 Outperforming GA Model based on Average Time

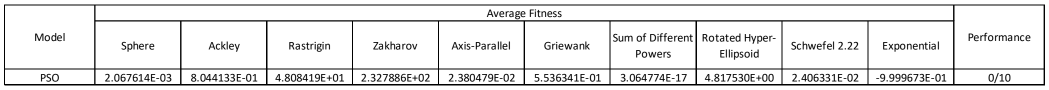
Table 4.2.4 Combinations of Outperformed GA based on Average Time

Model	Selection	Crossover	Mutation	Replacement
GA147	Tournament	Arithmetic	Hybrid Comparing	Weak Parent
GA148	Tournament	Arithmetic	Hybrid Comparing	Weak Parent
GA149	Tournament	Arithmetic	Simple Inversion	Combined
GA150	Tournament	Arithmetic	Simple Inversion	Weak Parent
GA152	Tournament	Arithmetic	Simple Inversion	Combined
GA153	Tournament	Arithmetic	Random	Combined
GA160	Tournament	Arithmetic	Reversing	Combined

Table 4.2.5 Outperformed GA Models based on Average Time with Respective Benchmark Functions

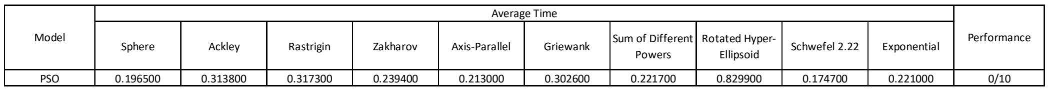
	GA147	GA148	GA149	GA150	GA152	GA153	GA160
Sphere						✓	
Ackley				✓			
Rastrigin		✓					
Zakhorov		✓					
Axis-Parallel			✓				
Griewank		✓		✓			
Sum of Different Power	✓						
Rotated Hyper-Ellipsoid				✓			
Schwefel 2.22					✓		
Exponential							✓
Performance	1/10	3/10	1/10	3/10	1/10	1/10	1/10

From Figure 4.2.18, it shows the results of PSO based on average fitness value and Figure 4.2.19 it shows the results of PSO based on average time.



Model	Average Fitness										Performance
	Sphere	Ackley	Rastrigin	Zakharov	Axis-Parallel	Griewank	Sum of Different Powers	Rotated Hyper-Ellipsoid	Schwefel 2.22	Exponential	
PSO	2.067614E-03	8.044133E-01	4.808419E+01	2.327886E+02	2.380479E-02	5.536341E-01	3.064774E-17	4.817530E+00	2.406331E-02	-9.999673E-01	0/10

Figure 4.2.18 PSO Results based on Average Fitness Value



Model	Average Time										Performance
	Sphere	Ackley	Rastrigin	Zakharov	Axis-Parallel	Griewank	Sum of Different Powers	Rotated Hyper-Ellipsoid	Schwefel 2.22	Exponential	
PSO	0.196500	0.313800	0.317300	0.239400	0.213000	0.302600	0.221700	0.829900	0.174700	0.221000	0/10

Figure 4.2.19 PSO Results based on Average Time

4.3 Objectives Evaluation

4.3.1 Best GA Model

Written By: Ling Ji Xiang 2104584

According to Table 4.2.3, there are 8 GA models produce well performed results. These models achieve the minimum average fitness value among other GA models. The operator combinations used are Dynamic Tournament Selection, Uniform Crossover, Hybrid Comparing Mutation and Combined Replacement. These techniques create a well-balanced GA that is capable of both exploring the search space and exploiting good solutions to improve performance. The dynamic adaption of the tournament size and hybrid comparing strategy are particularly effective in maintaining population diversity and avoiding premature convergence. Additionally, the crossover allows a high level of mixing between parent genes which can potentially create offspring with beneficial traits from both parents. The replacement techniques used in this GA also preserve the better gene to pass to next generation, producing better solutions overtime.

The best GA model is GA049 which having 3 out of 10 benchmarks achieving the minimum average fitness value. The combination of S, C, M, R are shown in Table 4.3.1.

Table 4.3.1 Combination of Best GA Model based on Average Fitness Value

Model	Selection	Crossover	Mutation	Replacement
GA049	Dynamic Tournament	Uniform	Hybrid Comparing	Combined

Written By: Yeap Chun Hong 2206352

Table 4.2.5 shows 8 examples of the average time for each GA models performing 10 benchmark function. The complete table are shown in appendix. The results shows that GA148 and GA150 outperformed the other GA models by achieving fastest computational time for 3 benchmark functions. It is observed that GA148 and GA150 have the common operation techniques which are Tournament Selection and Arithmetic Crossover. The reason of Tournament Selection require less computational time is because it is straightforward as it only involves comparing a few individuals from the population and selecting the best one. This simplicity results in lower computational overhead compared to more complex selection methods like linear ranking or roulette wheel. Besides that, Arithmetic Crossover involves simple arithmetic operations between pairs of parent solutions and the operations are also computational inexpensive. In addition, the models that achieved lowest computational time for 1 benchmark function such as GA147, GA149, GA 152 and GA 153 also used Tournament Selection and Arithmetic Crossover as its operators.

The best GA models are GA148 and GA150 which both having 3 out of 10 benchmarks achieving the minimum average time. The combination of S, C, M, R are shown in Table 4.3.2.

Table 4.3.2 Combination of Best GA Model based on Average Time

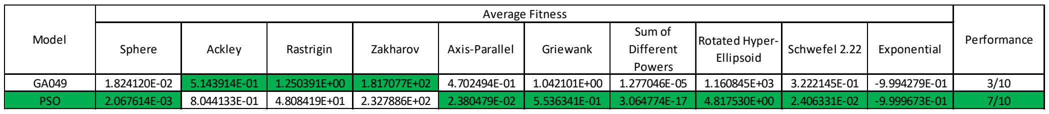
Model	**Selection**	**Crossover**	**Mutation**	**Replacement**
GA148	Tournament	Arithmetic	Hybrid Comparing	Weak Parent
GA150	Tournament	Arithmetic	Simple Inversion	Binary Tournament

In summary, GA049, GA148, and GA150 were the candidates to be selected as the best GA model. GA049 had the best performance in terms of lowest average fitness while GA138 and GA150 had the best performances in terms of average time. However, GA049 was determined to be the best GA model due to the increased significance of lowest average fitness than average time

4.3.2 Best GA Model and PSO Model Comparison

Written By: Loh Chia Heung 2301684

Figure 4.3.1 compares the overall performance of PSO and the best GA model, GA049 based on the average fitness value. The result shows that PSO outperformed GA in 7 out of 10 benchmark functions which are Sphere, Axis-Parallel, Griewank, Sum of Different Power, Rotated Hyper-Ellipsoid, Schwefel 2.22 and Exponential.



Model	Average Fitness										Performance
	Sphere	Ackley	Rastrigin	Zakharov	Axis-Parallel	Griewank	Sum of Different Powers	Rotated Hyper-Ellipsoid	Schwefel 2.22	Exponential	
GA049	1.824120E-02	5.143914E-01	1.250391E+00	1.817077E+02	4.702494E-01	1.042101E+00	1.277046E-05	1.160845E+03	3.222145E-01	-9.994279E-01	3/10
PSO	2.067614E-03	8.044133E-01	4.808419E+01	2.327886E+02	2.380479E-02	5.536341E-01	3.064774E-17	4.817530E+00	2.406331E-02	-9.999673E-01	7/10

Figure 4.3.1 Performance Comparison of GA and PSO Based on Average Fitness Value

One of the main reasons for PSO's better performance is that it works by guiding the particles toward the optimal solutions. Each particle in PSO is influenced by two factors, its own personal best position (P_{best}) and the global best position (G_{best}) found by the entire swarm. This collaboration between the particles helps PSO to converge faster on the optimal solutions. On the other hand, GA more relied on the randomize technique like crossover and mutation to explore the solution space.

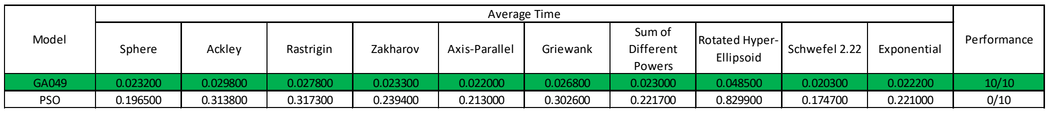
Another key advantage of PSO to outperform GA is that it balances the exploration and exploitation more effectively. This balance is primarily due to the parameter settings in the PSO, particularly the inertia weight and the acceleration coefficients (c1 and c2). The inertia weight is especially important here to influence a particle's previous velocity on its current movement to the new velocity [58] and balance the local and global searching [59]. c1 and c2 determines how strongly a particle is attracted to the P_{best} and G_{best} which can further enhance this balance. On the other hand, GA requires careful tuning of parameters like population size, crossover rate, and mutation rate to manage the exploration and exploitation. Not well-controlled parameter may make it more potentially to getting stuck in local optimal or not exploring the solution space effectively as it doesn't have a guidance mechanism like what PSO does.

In addition, the nature of the benchmark functions used in this study involves continuous search space. PSO is particularly perform well for continuous optimization problems [2] because it allows for smooth adjustments in the particle positions, leading to a more precise convergence. On the other hand, GA is more suited to the discrete or combinatorial problems [2]. Therefore, in continuous space, GA's crossover and mutation operation may not produce the offspring that are well-suited to the problem landscape, resulting in higher fitness value.

In conclusion, based on the results of this research, it is evident that PSO outperforms GA in optimizing benchmark functions, particularly in producing lower fitness values in continuous search spaces.

Written By: Brandon Ting En Junn

Figure 4.3.2 showed the performance comparisons between the best GA model and PSO model in terms of the average time for each benchmark function. It was shown that PSO lost to GA in terms of lower average time on all 10 benchmark functions. Although PSO had outperformed GA in terms of the average fitness, it had performed significantly worse than GA in terms of average time. The PSO had average times that were approximately 10 times slower than GA on all of the benchmark functions.



Model	Average Time										Performance
	Sphere	Ackley	Rastrigin	Zakharov	Axis-Parallel	Griewank	Sum of Different Powers	Rotated Hyper-Ellipsoid	Schwefel 2.22	Exponential	
GA049	0.023200	0.029800	0.027800	0.023300	0.022000	0.026800	0.023000	0.048500	0.020300	0.022200	10/10
PSO	0.196500	0.313800	0.317300	0.239400	0.213000	0.302600	0.221700	0.829900	0.174700	0.221000	0/10

Figure 4.3.2 Performance Comparison of GA and PSO Based on Average Time

PSO is computationally more expensive than GA. In other words, PSO require more computation time to complete its operations than GA. This is due to its heavy mathematical calculation nature. PSO requires the mathematical operators of addition, subtraction, and multiplication as its backbone to calculating the updated values. Additionally, other processes such as generation of random values and replacement of updated values are more frequent than of GA. Each of the particles require its own position, velocity, personal best to be updated, alongside with the group best for each generation. The mathematical equations referred in Chapter 2 are being run extensively to perform updates to the values.

Meanwhile, GA require less computation time than PSO due to its nature of operation techniques such as selection, crossover, mutation, and replacement. The selection process only selects two child chromosomes that the other operation techniques will be performed on. Crossover and mutation perform the operations of gene swapping and random value generation of at most two genes only of each child chromosome. Then, replacement updates the new generation by replacing at most two existing chromosomes due to the steady-state updates. The nature of these processes makes GA a more lightweight algorithm than PSO in terms of computation time.

In conclusion, the PSO model outperformed the GA model in terms of the average lowest fitness, while the GA outperformed the PSO in terms of the average time. In this context, the performance metric of average lowest fitness is more significant than the average time. Hence, it was determined that the overall performance of the PSO outperformed the GA due to the average lowest fitness performance metric, providing better overall optimal solutions for the 10 benchmark functions.

4.3.3 Operation Techniques

Written By: Yeap Chun Hong 2206352

In the selection operation techniques perspective, 5 of the GA models used Dynamic Tournament Selection to achieve the lowest average fitness value while 4 of them used Tournament Selection and 1 for Roulette Wheel Selection.

Dynamic Tournament Selection can outperform the other GA selection techniques due to its ability to adapt to the diversity of the population during the search process. It adjusts the tournament size based on the diversity of the population instead of fixing tournament size in the typical tournament selection. Early in the search, the tournament size can be kept small to maintain exploration and avoid premature convergence as the diversity remain high. As the population converges, the tournament size can be increased to apply more selection pressure, focusing on exploiting the best solutions.

However, the Dynamic Tournament Selection is unable to perform better than PSO in terms of fitness. This may because of the nature of PSO where the constant velocity clamping prevents particles from moving too quickly across the search space, while the weight controls the balance between exploration and exploitation [60]. The balance is often more effective in smoothly navigating the search space and avoiding local optima as compared to Dynamic Tournament Selection.

On the other hand, Roulette Wheel and Linear Ranking Selection does not perform well in the experiments. This is because both selection techniques introduced randomness into the selection process. Although the randomness provides a chance for the individual with higher fitness value, it is however the stochastic nature can cause the algorithm to pick suboptimal individuals by chance, which lead to the population converge slower than other selection techniques. Besides that, the Linear Ranking applies a uniform selection pressure to the individual as it assigns ranks to individuals based on their fitness and then selects them based on these ranks. This will cause the individual with better fitness value loses its advantage to be select as it does not stand a significantly higher chance of being selected compared to average individuals.

Written By: Loh Chia Heung 2301684

Combined Crossover and Uniform Crossover both achieved a high number of best performances, each achieving 5 out of the 10 lowest average fitness values. On the other hand, Shuffle Crossover and Arithmetic Crossover did not perform as well in terms of achieving the lowest average fitness values.

In Uniform Crossover, genes are randomly chosen from their parent with an equal probability of 0.5 for each gene position. This method ensures that a broad exploration of possible solution while avoiding excessive randomness. Therefore, it can strike a balance between exploration through maintain a good level of genetic diversity [22]. This diversity is crucial for preventing premature convergence, where the algorithm might otherwise become trapped in the suboptimal solutions too early. Instead, Uniform Crossover allows the GA to effectively explore the solution space, ensuring a more comprehensive search for high-quality solutions.

In Combined Crossover, it integrates the multiple crossover technique - Uniform Crossover, Single Point Crossover and Arithmetic Crossover. Uniform Crossover provides genetic diversity and explore various regions of the search space [20], Single Point crossover introduced the structured recombination, enhancing genetic diversity by combining larger segments of genes, and Arithmetic Crossover fine-tunes the solutions by averaging gene values from parents, helping to converge towards high-quality solutions. This hybrid approach utilizes the strengths of each technique, resulting in a balanced exploration and exploitation of the search space, which enhances the overall efficiency and effectiveness in finding optimal solutions for benchmark functions.

Shuffle Crossover, which involves shuffling the genes of the parents before performing crossover, may introduced excessive randomness in this case, potentially disrupting the genes combinations, and leading to the suboptimal solutions. On the other hand, Arithmetic Crossover, which is effective for balancing genes value between parents, may have limited its effectiveness in exploring the diverse regions of the search space. It is because this approach tends to generate offspring values that are intermediate between the parent chromosomes, which might not always be advantageous for optimization problems where a broader exploration of the solution space is needed.

In the perspective of average time, Arithmetic Crossover outperforms other crossover method, achieving the result of 10 out of the 10 lowest average time. This superior performance is primarily due to its simplicity and computational efficiency. The Arithmetic Crossover method employs a straightforward formula to blend the genes of parent chromosomes, involving basic arithmetic operations such as addition and multiplication. This simplicity means that the calculations are relatively quick and do not involve complex procedures or additional data handling. In contrast, other crossover methods that may introduce additional computational complexity. For instance, Shuffle Crossover requires shuffling genes before performing the crossover, which adds a layer of complexity and randomness that can increase computation time. Combined Crossover, which integrates multiple crossover techniques like Uniform, Single-Point, and Arithmetic Crossover, inherently involves more complex operations and requires additional computational resources to execute each crossover method effectively.

Written By: Brandon Ting En Junn 2101751

Based on the results, Hybrid Comparing Mutation and Simple Inversion Mutation had the majority in the top outperforming GA models on the average lowest fitness and average time respectively. Hybrid Comparing Mutation outperformed as the majority on seven GA models on average lowest fitness, including the best GA model while Simple Inversion Mutation outperformed as majority on five GA models on average time.

Simple Inversion Mutation was able to provide better average time due to its algorithmic simplicity that allowed lower computation times to be achieved. However, it is notably mentioned that the results may be also influenced by other operation techniques that also contribute to the average time. The mutation operator undergoes a simple local exploitation process by reversing the substring of the genes of the child chromosome. Although it was an extended variant to the Reversing Mutation, it performed shorter computation time due to the decreased probability generations that determine the occurrence of the mutation operation. Hence, it was the best performing mutation operator in terms of the lower computation times.

Hybrid Comparing Mutation was able to provide better average lowest fitness due to its combined approach of local exploitation and global exploration. As it is the combination of Reversing Mutation and Random Mutation, leading to a better search space for the algorithm. Reversing Mutation contributed towards the local exploitation while Random Mutation contributed towards the global exploration. It also introduced better values into the algorithm due to the comparing process. The comparing process improved the global exploration by only allowing new values that were better into the next generation, leading to vastly improved average lowest fitness values.

However, the best GA model with the Hybrid Comparing Mutation was unable to outperform the PSO with its performance on local exploitation and global exploration. The balance between local search and global search is essential in order to find the optimal solution for the benchmark functions. Although the Hybrid Comparing Mutation had the characteristics of local search and global search, it was unable to outperform the PSO due to its unknown exploration to exploitation ratio. This may be its inferior local search and global search balance than the PSO.

PSO had better search spaces due to the inclusion of personal best and group best in finding the optimal solution. In addition to that, PSO with constant inertia weight and velocity clamping contribute to better global exploration search spaces and control on the rate of global exploration to local exploitation [61, 62]. These applications served as pillars of guidance for better searches than GA towards the optimal solution.

Written By: Ling Ji Xiang 2104584

Based on the results, two replacement strategies which is Combined Replacement and Weak Parent Replacement include in the well performing GA models based on the average fitness value and average time. In the results of average fitness value, there are

3 weak parent and 5 combined replacements. In the term of average time, there are 3 weak parent and 3 combined replacements included in the well performing GA model.

Weak Parents Replacement is designed to enhance the overall quality of the population by selectively replacing the weaker of two parents with stronger offspring. This strategy ensures that the population gradually improves as only individuals with lower fitness value are passed to the next generations. This replacement strategy was able to perform well because its ability to balance exploration and exploitation effectively. By consistently favouring stronger offspring, it drives the population towards better solutions over time while still maintaining enough diversity to explore the solution space.

Combined Replacement is the combination of the principles of Both Parent Replacement and Binary Tournament Replacement. This approach leverages the strengths of both strategies. In this approach, both parents are replaced by the offspring and then comes to the weak parent principle which compare the parents with the offsprings and replacing occur if parent are less fit. This is to ensure there is no valuable gene materials losing because of the weakness of both parent replacement. This dual mechanism allows for a broader exploration of solution space while still focusing on retaining high-quality individuals in the population. These balance of combined replacement and weak parent replacement is likely why they performed well in terms of average fitness value and average time.

Both parent replacement and binary replacement did not perform well in the results. Both parent replacement simply replaces both parents with offsprings without considering the fitness value. This may lead to loss of valuable genetic diversity especially when the offsprings have weaker fitness value than their parents. Binary replacement is a simple comparison mechanism which may lack of the sophistication to maintain the diversity. These drawbacks are likely why they not performing well in the results.

CHAPTER 5

Conclusion and Recommendation

Written By: Brandon Ting En Junn 2101751

This chapter summarises our findings of this research and recommendations on future works.

5.1 Conclusion

Written By: Loh Chia Heung 2301684

In conclusion, the experiment demonstrated that the best GA model, GA49, which utilized a combination of Dynamic Tournament Selection, Uniform Crossover, Hybrid Comparing Mutation, and Combined Replacement, achieved the lowest fitness values on three benchmark functions out of 256 tested GA models. However, the PSO model outperformed GA49 by producing the lowest average fitness values on seven out of ten benchmark functions. Additionally, in terms of average computation time, the GA model proved to be more efficient, outperforming PSO across all 10 benchmark functions.

The result aligns with the project objectives by showing that PSO generally offers better performance in minimizing the fitness values compared to GA. Specifically, PSO showed a superior ability to converge on optimal solutions, particularly in continuous search spaces.

This project contributes valuable insights into comparative performance for GA and PSO toward benchmark functions, offering guidance for future research and real-world applications. The findings indicate that PSO is particularly well-suited for continuous problems, while GA may still be preferred for scenarios where computational efficiency is a priority or for discrete search space. This insight is crucial for selecting the most appropriate optimization algorithm based on specific problem requirements. Researchers should consider the nature of their optimization problems to identify whether they are continuous or discrete, and the computational constraints when choosing between GA and PSO. This understanding will help in applying the most suitable algorithm to achieve optimal solutions efficiently.

5.2 Recommendation

Written By: Yeap Chun Hong 2206352

To further enhance the performance of GA models in future research, it is recommended to attempt the operation techniques that were not applied in this project as they might obtain better outcome. For example, applying Stochastic Universal Sampling Selection, Two Point Crossover, Flipping Mutation and Random Replacement to compare the same dataset and same parameter in this study.

Written By: Loh Chia Heung 2301684

For future research, it is recommended to further refine and optimize the Combined Crossover technique, especially considering that its performance is comparable to Uniform Crossover. Exploring various strategies for adjusting the frequency and sequence in which each crossover method is applied could potentially boost its effectiveness. Additionally, evaluating the Combined Crossover across real-world problems may also offer deeper insights into its robustness and adaptability. Such exploration could uncover new opportunities to fine-tune the approach, ultimately making it a more powerful tool for solving complex optimisation problem.

Written By: Brandon Ting En Junn 2101751

The Hybrid Comparing Mutation showed promising results but had the limitation in the uncontrollable and unknown ratio of the operator's exploration to exploitation compared to the PSO. Future works on improving the Hybrid Comparing Mutation to include additional parameters for better control of the operator's exploration to exploitation ratios. In addition to that, fine-tuning approaches can be made as modifications to improve the operator's performance dynamically. The inclusion and addition of the parameters and fine-tuning approach may improve the performance of the mutation operation for improving the GA model's performance.

Written By: Ling Ji Xiang 2104584

To further enhance the performance, it is important to continue testing and validating different replacement strategies across wide range of benchmark functions. This approach will help to identify which strategies perform best. By evaluating the effectiveness of weak parent replacement, combined replacement and both parent replacement on different optimization tasks, then we can refine and optimize the replacement strategies. This process will improve the algorithm’s adaptability to different problem and ensure the increase in performance.

REFERENCES

[1] G. Papazoglou and P. Biskas, “Review and Comparison of Genetic Algorithm and Particle Swarm Optimization in the Optimal Power Flow Problem,” Energies, vol. 16, no. 3, p. 1152, Jan. 2023, doi: https://doi.org/10.3390/en16031152.

[2] Sapna Katiyar “A Comparative Study of Genetic Algorithm and the Particle Swarm Optimization” *AKGEC International Journal of Technology*, Vol. 2, No. 2, Page No: 21-24

[3] S. Shabir and R. Singla, “A Comparative Study of Genetic Algorithm and the Particle Swarm Optimization,” International Journal of Electrical Engineering, vol. 9, no. 2, pp. 215–223, 2016, Accessed: Jul. 22, 2024. [Online]. Available: http://www.irphouse.com/ijee16/ijeev9n2_06.pdf

[4] M. Shehab, A. T. Khader, and M. A. Al-Betar, “A survey on applications and variants of the cuckoo search algorithm,” *Applied Soft Computing*, vol. 61, pp. 1041–1059, Dec. 2017, doi: https://doi.org/10.1016/j.asoc.2017.02.034.

[5] X.-S. Yang, X.-S. He, and Q.-W. Fan, “Mathematical framework for algorithm analysis,” *Elsevier eBooks*, pp. 89–108, Jan. 2020, doi: https://doi.org/10.1016/b978-0-12-819714-1.00017-8.

[6] S. Katoch, S. S. Chauhan, and V. Kumar, “A review on genetic algorithm: past, present, and future,” *Multimedia Tools and Applications*, vol. 80, no. 5, Oct. 2020, doi: https://doi.org/10.1007/s11042-020-10139-6.

[7] Saneh Lata Yadav and A. Sohal, “Comparative study of different selection techniques in genetic algorithm,” *International journal of engineering, science and mathematics*, vol. 6, no. 3, pp. 174–180, Jan. 2017.

[8] Khalid Jebari, “Selection Methods for Genetic Algorithms,” *ResearchGate*, Dec. 2013.
https://www.researchgate.net/publication/259461147_Selection_Methods_for_Genetic_Algorithms

[9] Y. Liu, D. Ćetenović, H. Li, E. Gryazina, and V. Terzija, “An optimized multi-objective reactive power dispatch strategy based on improved genetic algorithm for wind power integrated systems,” *International Journal of Electrical Power & Energy*

Systems, vol. 136, p. 107764, Mar. 2022, doi: https://doi.org/10.1016/j.ijepes.2021.107764.

[10] Y. Fang and J. Li, "A Review of Tournament Selection in Genetic Programming," *Advances in Computation and Intelligence*, pp. 181–192, 2010, doi: https://doi.org/10.1007/978-3-642-16493-4_19.

[11] Y. Lavinas, Claus Aranha, T. Sakurai, and M. Ladeira, "Experimental Analysis of the Tournament Size on Genetic Algorithms," *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 3647–3653, Oct. 2018, doi: https://doi.org/10.1109/smc.2018.00617.

[12] B. Miller and D. Goldberg, "Genetic Algorithms, Tournament Selection, and the Effects of Noise," *Complex System*, vol. 9, pp. 193–212, 1995, Available: https://content.wolfram.com/sites/13/2018/02/09-3-2.pdf

[13] S. A. Hamad and F. A. Omara, "Genetic-Based Task Scheduling Algorithm in Cloud Computing Environment," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 4, 2016, doi: https://doi.org/10.14569/ijacsa.2016.070471.

[14] A. Shukla, H. M. Pandey, and D. Mehrotra, "Comparative review of selection techniques in genetic algorithm," *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, pp. 515–519, Feb. 2015, doi: https://doi.org/10.1109/ablaze.2015.7154916.

[15] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, Mass. ; London: Mit, 1998, p. 127. Accessed: Aug. 29, 2024. [Online]. Available: https://www.boente.eti.br/fuzzy/ebook-fuzzy-mitchell.pdf

[16] H. M. Pandey, "Performance Evaluation of Selection Methods of Genetic Algorithm and Network Security Concerns," *Procedia Computer Science*, vol. 78, pp. 13–18, 2016, doi: https://doi.org/10.1016/j.procs.2016.02.004.

[17] N. Fang, J. Zhou, R. Zhang, Y. Liu, and Y. Zhang, "A hybrid of real coded genetic algorithm and artificial fish swarm algorithm for short-term optimal hydrothermal scheduling," *International Journal of Electrical Power and Energy Systems*, vol. 62, pp. 617–629, Nov. 2014, doi: https://doi.org/10.1016/j.ijepes.2014.05.017.

[18] S. Kumar, V. Kumar Sharma, and R. Kumari, “A Novel Hybrid Crossover based Artificial Bee Colony Algorithm for Optimization Problem,” *International Journal of Computer Applications*, vol. 82, no. 8, pp. 18–25, Nov. 2013, doi: https://doi.org/10.5120/14136-2266.

[19] S. M. Lim, A. B. Md. Sultan, Md. N. Sulaiman, A. Mustapha, and K. Y. Leong, “Crossover and Mutation Operators of Genetic Algorithms,” *International Journal of Machine Learning and Computing*, vol. 7, no. 1, pp. 9–12, Feb. 2017, doi: https://doi.org/10.18178/ijmlc.2017.7.1.611.

[20] P. Kora and P. Yadlapalli, “Crossover Operators in Genetic Algorithms: A Review,” *International Journal of Computer Applications*, vol. 162, no. 10, pp. 34–36, Mar. 2017, doi: https://doi.org/10.5120/ijca2017913370.

[21] F. G. Lobo and D. E. Goldberg, “The parameter-less genetic algorithm in practice,” *Information Sciences*, vol. 167, no. 1, pp. 217-232, 2004.

[22] G. Syswerda, "Uniform crossover in genetic algorithms," in Proc. of the 3rd International Conference on Genetic Algorithms, J. D. Schaffer, Ed., Morgan Kaufmann, 1989, pp. 2-9.

[23] I. Ono, H. Kita, and S. Kobayashi, “A robust real-coded genetic algorithm using Unimodal Normal Distribution Crossover augmented by Uniform Crossover: effects of self-adaptation of crossover probabilities,” *Genetic and Evolutionary Computation Conference*, pp. 496–503, Jul. 1999.

[24] R. A. Caruana, L. J. Eshelman, and J. David Schaffer, “Representation and hidden bias II: eliminating defining length bias in genetic search via shuffle crossover,” pp. 750–755, Aug. 1989.

[25] F. J. Burkowski, “Shuffle crossover and mutual information,” Jan. 2003, doi: https://doi.org/10.1109/cec.1999.782671.

[26] T. D. Giddens, “The determination of invariant parameters and operators of the traditional genetic algorithm using an adaptive genetic algorithm generator,” *Tdl.org*, Aug. 1994. https://ttu-ir.tdl.org/items/2106becb-8f85-431c-a979-edba020226a5 (accessed Sep. 02, 2024).

[27] G. S. Ladkany and M. B. Trabia, “A Genetic Algorithm with Weighted Average Normally-Distributed Arithmetic Crossover and Twinkling,” Applied Mathematics, vol. 03, no. 10, pp. 1220–1235, October. 2012, doi: https://doi.org/10.4236/am.2012.330178.

[28] S. Picek, Domagoj Jakobovic, and M. Golub, “On the recombination operator in the real-coded genetic algorithms,” IEEE Congress on Evolutionary Computation, June. 2013 doi: https://doi.org/10.1109/cec.2013.6557948.

[29] M. Furqan, Hartono, E. Ongko, and M. Ikhsan, “Performance of Arithmetic Crossover and Heuristic Crossover in Genetic Algorithm Based on Alpha Parameter,” IOSR Journal of Computer Engineering (IOSR-JCE), vol. 19, no. 5, pp. 31–36, Oct. 2017, doi: https://doi.org/10.9790/0661-1905013136.

[30] Anderson King Junior, None Suharjito, Novan Zulkarnain, Devriady Pratama, E. Gunawan, and Ditdit Nugeraha Utama, “The Effect Analysis of Crossover and Selection Methods on the Performance of GenClust++ Algorithm,” International Conference on ICT for Smart Society (ICISS) Nov. 2019, doi: https://doi.org/10.1109/iciss48059.2019.8969832.

[31] D. Gupta and S. Ghafir, “An Overview of methods maintaining Diversity in Genetic Algorithms,” International Journal of Emerging Technology and Advanced Engineering Website: www.ijetae.com, vol. 2, no. 5, p. 56, May 2012, Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=4811bc9bcd24c9a363afe454e6737b48c24f8837 (Accessed: Aug. 20, 2024.)

[32] O. Köksoy and T. Yalcinoz, “Robust Design using Pareto type optimization: A genetic algorithm with arithmetic crossover,” Computers & Industrial Engineering, vol. 55, no. 1, pp. 208–218, Aug. 2008, doi: https://doi.org/10.1016/j.cie.2007.11.019.

[33] Onur Köksoy and Tankut Yalcinoz, “A Hopfield Neural Network Approach to the Dual Response Problem,” Quality and Reliability Engineering International, vol. 21, no. 6, pp. 595–603, Mar. 2005, doi: https://doi.org/10.1002/qre.675.

[34] Dennis and W. Tu, “Dual Response Surface Optimization,” Journal of Quality Technology, vol. 27, no. 1, pp. 34–39, Jan. 1995, doi: https://doi.org/10.1080/00224065.1995.11979556.

[35] O. Köksoy and N. Doganaksoy, "Joint Optimization of Mean and Standard Deviation Using Response Surface Methods," Journal of Quality Technology, vol. 35, no. 3, pp. 239–252, Jul. 2003, doi: https://doi.org/10.1080/00224065.2003.11980218.

[36] S. P. Lim and H. Haron, "Performance of Different Techniques Applied in Genetic Algorithm towards Benchmark Functions," in ACIIDS 2013, LNAI 7802, Part I, 2013, pp. 255–264.

[37] C. H. Ong, "Performance of Different Operation Techniques Applied in Genetic Algorithm Towards Benchmark Functions," Bachelor Project Report, Dept. Com. Sc, UTAR, 2018.

[38] A. Hassanat, K. Almohammadi, E. Alkafaween, E. Abunawas, A. Hammouri, and V. B. S. Prasath, "Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach," Information, vol. 10, no. 12, p. 390, Dec. 2019, doi: https://doi.org/10.3390/info10120390.

[39] K. Deep and M. Thakur, "A new mutation operator for real coded genetic algorithms," Applied Mathematics and Computation, vol. 193, no. 1, pp. 211–230, Oct. 2007, doi: https://doi.org/10.1016/j.amc.2007.03.046.

[40] H. H. Chieng and N. Wahid, "A Performance Comparison of Genetic Algorithm's Mutation Operators in n-Cities Open Loop Travelling Salesman Problem," Advances in Intelligent Systems and Computing, pp. 89–97, 2014, doi: https://doi.org/10.1007/978-3-319-07692-8_9.

[41] B. Huang, L. Yao, and K. Raguraman, "Bi-level GA and GIS for Multi-objective TSP Route Planning," Transportation Planning and Technology, vol. 29, no. 2, pp. 105–124, Apr. 2006, doi: https://doi.org/10.1080/03081060600753404.

[42] M. Lozano, F. Herrera, and J. Cano, "Replacement strategies to preserve useful diversity in steady-state genetic algorithms," Information Sciences, vol. 178, no. 23, pp. 4421–4433, Dec. 2008, doi: 10.1016/j.ins.2008.07.031. Available: https://www.sciencedirect.com/science/article/abs/pii/S0020025508002867

[43] M. Lozano, F. Herrera, and J. R. Cano, "Replacement strategies to maintain useful diversity in Steady-State genetic algorithms," in Springer eBooks, 2006, pp. 85–96. doi: https://doi.org/10.1007/3-540-32400-3_7

[44] R. Naoum, S. Aziz, and F. Alabsi, “An enhancement of the replacement Steady State Genetic Algorithm for intrusion detection,” International Journal of Advanced Computer Research, Jan. 2014, [Online]. Available: http://accentsjournals.org/PaperDirectory/Journal/IJACR/2014/6/11.pdf

[45] D. Wang, D. Tan, and L. Liu, “Particle swarm optimization algorithm: an overview,” Soft Computing, vol. 22, no. 2, pp. 387–408, Jan. 2017, doi: https://doi.org/10.1007/s00500-016-2474-6.

[46] F. van den Bergh, “An Analysis of Particle Swarm Optimizers,” Ph.D. dissertation, Faculty of Natural and Agricultural Science, University of Pretoria, Pretoria, South Africa, 2001.

[47] J. C. Bansal, P. K. Singh, M. Saraswat, A. Verma, S. S. Jadon, and A. Abraham, “Inertia Weight strategies in Particle Swarm Optimization,” 2011 Third World Congress on Nature and Biologically Inspired Computing, Oct. 2011, doi: https://doi.org/10.1109/nabic.2011.6089659.

[48] M. Taherkhani and R. Safabakhsh, “A novel stability-based adaptive inertia weight for particle swarm optimization,” Applied Soft Computing, vol. 38, pp. 281–295, Jan. 2016, doi: https://doi.org/10.1016/j.asoc.2015.10.004.

[49] Musaed Alhussein and Syed Irtaza Haider, “Improved Particle Swarm Optimization Based on Velocity Clamping and Particle Penalization,” Dec. 2015, doi: https://doi.org/10.1109/aims.2015.20.

[50] S. P. Lim, H. Hoon, and W. H. Song, “A Comparative Study on Different Parameter Factors and Velocity Clamping for Particle Swarm Optimisation,” IOP Conference Series: Materials Science and Engineering, vol. 864, p. 012068, Jul. 2020, doi: https://doi.org/10.1088/1757-899x/864/1/012068.

[51] Y. Wu, J. Liu and C. Peng, "A New Replacement Strategy for Genetic Algorithm and Computational Experiments," IEEE Conference Publication | IEEE Xplore, Jun. 01, 2014.

[52] C. A. Souza Lima, C. M. F. Lapa, C. M. do N. A. Pereira, J. J. da Cunha, and A. C. M. Alvim, “Comparison of computational performance of GA and PSO optimization techniques when designing similar systems – Typical PWR core case,” Annals of

Nuclear Energy, vol. 38, no. 6, pp. 1339–1346, Jun. 2011, doi: https://doi.org/10.1016/j.anucene.2011.02.002.

[53] D. R. Ramdania, M. Irfan, F. Alfarisi, and D. Nuraiman, "Comparison of genetic algorithms and Particle Swarm Optimization (PSO) algorithms in course scheduling," Journal of Physics Conference Series, vol. 1402, no. 2, p. 022079, Dec. 2019, doi: 10.1088/1742-6596/1402/2/022079.

[54] D. Q. Zeebaree, H. Haron, A. M. Abdulazeez, and S. R. M. Zeebaree, "Combination of K-means Clustering with Genetic Algorithm: A Review," International Journal of Applied Engineering Research, vol. 12, no. 24, pp. 14238–14245, 2017.

[55] K. Zhu, "Population Diversity in Genetic Algorithm for Vehicle Routing Problem with Time Windows," 2003. Accessed: Aug. 31, 2024. [Online]. Available: https://www.cs.sjtu.edu.cn/~kzhu/papers/zhu-ecml04.pdf

[56] R. Ramani and L. Balasubramanian, "Genetic Algorithm solution for Cryptanalysis of Knapsack Cipher with Knapsack Sequence of Size 16," International Journal of Computer Applications, vol. 35, no. 11, pp. 975–8887, 2011, Accessed: Aug. 31, 2024. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=7cb1782c9b019b10e65eecef7aa9e5f971fcb805

[57] K. Höschel and V. Lakshminarayanan, "Genetic algorithms for lens design: a review," *Journal of Optics*, vol. 48, no. 1, pp. 134–144, Dec. 2018, doi: https://doi.org/10.1007/s12596-018-0497-3.

[58] K. Premalatha and A. M. Natarajan, "Hybrid PSO and GA for global optimization," Int. J. Open Problems Compt. Math., vol. 2, no. 4, pp. 598-608, 2009.

[59] Y. H. Shi and R. Eberhart, "A Modified Particle Swarm Optimizer," IEEE International Conference on Evolutionary Computation Proceedings, 1998, pp. 69-73.

[60] Dian Palupi Rini and Siti Mariyam Shamsuddin, "Particle Swarm Optimization: Technique, System and Challenges," International Journal of Applied Information Systems, vol. 1, no. 1, pp. 33–45, Sep. 2011, doi: https://doi.org/10.5120/ijais-3651.

[61] Y. Shi and R. Eberhart, “A modified particle swarm optimizer,” 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360), 2019, doi: https://doi.org/10.1109/icec.1998.699146.

[62] E. T. Oldewage, A. P. Engelbrecht, and C. W. Cleghorn, “The merits of velocity clamping particle swarm optimisation in high dimensional spaces,” Nov. 2017, doi: https://doi.org/10.1109/ssci.2017.8280887.

APPENDIX

256 GA Model Combinations

Model	Selection	Crossover	Mutation	Replacement
GA001	Dynamic Tournament	Combined	Hybrid Comparing	Combined
GA002	Dynamic Tournament	Combined	Hybrid Comparing	Binary Tournament
GA003	Dynamic Tournament	Combined	Hybrid Comparing	Both Parent
GA004	Dynamic Tournament	Combined	Hybrid Comparing	Weak Parent
GA005	Dynamic Tournament	Combined	Simple Inversion	Combined
GA006	Dynamic Tournament	Combined	Simple Inversion	Binary Tournament
GA007	Dynamic Tournament	Combined	Simple Inversion	Both Parent
GA008	Dynamic Tournament	Combined	Simple Inversion	Weak Parent
GA009	Dynamic Tournament	Combined	Random	Combined
GA010	Dynamic Tournament	Combined	Random	Binary Tournament
GA011	Dynamic Tournament	Combined	Random	Both Parent
GA012	Dynamic Tournament	Combined	Random	Weak Parent

GA013	Dynamic Tournament	Combined	Reversing	Combined
GA014	Dynamic Tournament	Combined	Reversing	Binary Tournament
GA015	Dynamic Tournament	Combined	Reversing	Both Parent
GA016	Dynamic Tournament	Combined	Reversing	Weak Parent
GA017	Dynamic Tournament	Arithmetic	Hybrid Comparing	Combined
GA018	Dynamic Tournament	Arithmetic	Hybrid Comparing	Binary Tournament
GA019	Dynamic Tournament	Arithmetic	Hybrid Comparing	Both Parent
GA020	Dynamic Tournament	Arithmetic	Hybrid Comparing	Weak Parent
GA021	Dynamic Tournament	Arithmetic	Simple Inversion	Combined
GA022	Dynamic Tournament	Arithmetic	Simple Inversion	Binary Tournament
GA023	Dynamic Tournament	Arithmetic	Simple Inversion	Both Parent
GA024	Dynamic Tournament	Arithmetic	Simple Inversion	Weak Parent
GA025	Dynamic Tournament	Arithmetic	Random	Combined
GA026	Dynamic Tournament	Arithmetic	Random	Binary Tournament

GA027	Dynamic Tournament	Arithmetic	Random	Both Parent
GA028	Dynamic Tournament	Arithmetic	Random	Weak Parent
GA029	Dynamic Tournament	Arithmetic	Reversing	Combined
GA030	Dynamic Tournament	Arithmetic	Reversing	Binary Tournament
GA031	Dynamic Tournament	Arithmetic	Reversing	Both Parent
GA032	Dynamic Tournament	Arithmetic	Reversing	Weak Parent
GA033	Dynamic Tournament	Shuffle	Hybrid Comparing	Combined
GA034	Dynamic Tournament	Shuffle	Hybrid Comparing	Binary Tournament
GA035	Dynamic Tournament	Shuffle	Hybrid Comparing	Both Parent
GA036	Dynamic Tournament	Shuffle	Hybrid Comparing	Weak Parent
GA037	Dynamic Tournament	Shuffle	Simple Inversion	Combined
GA038	Dynamic Tournament	Shuffle	Simple Inversion	Binary Tournament
GA039	Dynamic Tournament	Shuffle	Simple Inversion	Both Parent
GA040	Dynamic Tournament	Shuffle	Simple Inversion	Weak Parent

GA041	Dynamic Tournament	Shuffle	Random	Combined
GA042	Dynamic Tournament	Shuffle	Random	Binary Tournament
GA043	Dynamic Tournament	Shuffle	Random	Both Parent
GA044	Dynamic Tournament	Shuffle	Random	Weak Parent
GA045	Dynamic Tournament	Shuffle	Reversing	Combined
GA046	Dynamic Tournament	Shuffle	Reversing	Binary Tournament
GA047	Dynamic Tournament	Shuffle	Reversing	Both Parent
GA048	Dynamic Tournament	Shuffle	Reversing	Weak Parent
GA049	Dynamic Tournament	Uniform	Hybrid Comparing	Combined
GA050	Dynamic Tournament	Uniform	Hybrid Comparing	Binary Tournament
GA051	Dynamic Tournament	Uniform	Hybrid Comparing	Both Parent
GA052	Dynamic Tournament	Uniform	Hybrid Comparing	Weak Parent
GA053	Dynamic Tournament	Uniform	Simple Inversion	Combined
GA054	Dynamic Tournament	Uniform	Simple Inversion	Binary Tournament

GA055	Dynamic Tournament	Uniform	Simple Inversion	Both Parent
GA056	Dynamic Tournament	Uniform	Simple Inversion	Weak Parent
GA057	Dynamic Tournament	Uniform	Random	Combined
GA058	Dynamic Tournament	Uniform	Random	Binary Tournament
GA059	Dynamic Tournament	Uniform	Random	Both Parent
GA060	Dynamic Tournament	Uniform	Random	Weak Parent
GA061	Dynamic Tournament	Uniform	Reversing	Combined
GA062	Dynamic Tournament	Uniform	Reversing	Binary Tournament
GA063	Dynamic Tournament	Uniform	Reversing	Both Parent
GA064	Dynamic Tournament	Uniform	Reversing	Weak Parent
GA065	Linear Ranking	Combined	Hybrid Comparing	Combined
GA066	Linear Ranking	Combined	Hybrid Comparing	Binary Tournament
GA067	Linear Ranking	Combined	Hybrid Comparing	Both Parent
GA068	Linear Ranking	Combined	Hybrid Comparing	Weak Parent

GA069	Linear Ranking	Combined	Simple Inversion	Combined
GA070	Linear Ranking	Combined	Simple Inversion	Binary Tournament
GA071	Linear Ranking	Combined	Simple Inversion	Both Parent
GA072	Linear Ranking	Combined	Simple Inversion	Weak Parent
GA073	Linear Ranking	Combined	Random	Combined
GA074	Linear Ranking	Combined	Random	Binary Tournament
GA075	Linear Ranking	Combined	Random	Both Parent
GA076	Linear Ranking	Combined	Random	Weak Parent
GA077	Linear Ranking	Combined	Reversing	Combined
GA078	Linear Ranking	Combined	Reversing	Binary Tournament
GA079	Linear Ranking	Combined	Reversing	Both Parent
GA080	Linear Ranking	Combined	Reversing	Weak Parent
GA081	Linear Ranking	Arithmetic	Hybrid Comparing	Combined
GA082	Linear Ranking	Arithmetic	Hybrid Comparing	Binary Tournament
GA083	Linear Ranking	Arithmetic	Hybrid Comparing	Both Parent
GA084	Linear Ranking	Arithmetic	Hybrid Comparing	Weak Parent
GA085	Linear Ranking	Arithmetic	Simple Inversion	Combined

GA086	Linear Ranking	Arithmetic	Simple Inversion	Binary Tournament
GA087	Linear Ranking	Arithmetic	Simple Inversion	Both Parent
GA088	Linear Ranking	Arithmetic	Simple Inversion	Weak Parent
GA089	Linear Ranking	Arithmetic	Random	Combined
GA090	Linear Ranking	Arithmetic	Random	Binary Tournament
GA091	Linear Ranking	Arithmetic	Random	Both Parent
GA092	Linear Ranking	Arithmetic	Random	Weak Parent
GA093	Linear Ranking	Arithmetic	Reversing	Combined
GA094	Linear Ranking	Arithmetic	Reversing	Binary Tournament
GA095	Linear Ranking	Arithmetic	Reversing	Both Parent
GA096	Linear Ranking	Arithmetic	Reversing	Weak Parent
GA097	Linear Ranking	Shuffle	Hybrid Comparing	Combined
GA098	Linear Ranking	Shuffle	Hybrid Comparing	Binary Tournament
GA099	Linear Ranking	Shuffle	Hybrid Comparing	Both Parent
GA100	Linear Ranking	Shuffle	Hybrid Comparing	Weak Parent
GA101	Linear Ranking	Shuffle	Simple Inversion	Combined
GA102	Linear Ranking	Shuffle	Simple Inversion	Binary Tournament

GA103	Linear Ranking	Shuffle	Simple Inversion	Both Parent
GA104	Linear Ranking	Shuffle	Simple Inversion	Weak Parent
GA105	Linear Ranking	Shuffle	Random	Combined
GA106	Linear Ranking	Shuffle	Random	Binary Tournament
GA107	Linear Ranking	Shuffle	Random	Both Parent
GA108	Linear Ranking	Shuffle	Random	Weak Parent
GA109	Linear Ranking	Shuffle	Reversing	Combined
GA110	Linear Ranking	Shuffle	Reversing	Binary Tournament
GA111	Linear Ranking	Shuffle	Reversing	Both Parent
GA112	Linear Ranking	Shuffle	Reversing	Weak Parent
GA113	Linear Ranking	Uniform	Hybrid Comparing	Combined
GA114	Linear Ranking	Uniform	Hybrid Comparing	Binary Tournament
GA115	Linear Ranking	Uniform	Hybrid Comparing	Both Parent
GA116	Linear Ranking	Uniform	Hybrid Comparing	Weak Parent
GA117	Linear Ranking	Uniform	Simple Inversion	Combined
GA118	Linear Ranking	Uniform	Simple Inversion	Binary Tournament
GA119	Linear Ranking	Uniform	Simple Inversion	Both Parent

GA120	Linear Ranking	Uniform	Simple Inversion	Weak Parent
GA121	Linear Ranking	Uniform	Random	Combined
GA122	Linear Ranking	Uniform	Random	Binary Tournament
GA123	Linear Ranking	Uniform	Random	Both Parent
GA124	Linear Ranking	Uniform	Random	Weak Parent
GA125	Linear Ranking	Uniform	Reversing	Combined
GA126	Linear Ranking	Uniform	Reversing	Binary Tournament
GA127	Linear Ranking	Uniform	Reversing	Both Parent
GA128	Linear Ranking	Uniform	Reversing	Weak Parent
GA129	Tournament	Combined	Hybrid Comparing	Combined
GA130	Tournament	Combined	Hybrid Comparing	Binary Tournament
GA131	Tournament	Combined	Hybrid Comparing	Both Parent
GA132	Tournament	Combined	Hybrid Comparing	Weak Parent
GA133	Tournament	Combined	Simple Inversion	Combined
GA134	Tournament	Combined	Simple Inversion	Binary Tournament
GA135	Tournament	Combined	Simple Inversion	Both Parent
GA136	Tournament	Combined	Simple Inversion	Weak Parent

GA137	Tournament	Combined	Random	Combined
GA138	Tournament	Combined	Random	Binary Tournament
GA139	Tournament	Combined	Random	Both Parent
GA140	Tournament	Combined	Random	Weak Parent
GA141	Tournament	Combined	Reversing	Combined
GA142	Tournament	Combined	Reversing	Binary Tournament
GA143	Tournament	Combined	Reversing	Both Parent
GA144	Tournament	Combined	Reversing	Weak Parent
GA145	Tournament	Arithmetic	Hybrid Comparing	Combined
GA146	Tournament	Arithmetic	Hybrid Comparing	Binary Tournament
GA147	Tournament	Arithmetic	Hybrid Comparing	Both Parent
GA148	Tournament	Arithmetic	Hybrid Comparing	Weak Parent
GA149	Tournament	Arithmetic	Simple Inversion	Combined
GA150	Tournament	Arithmetic	Simple Inversion	Binary Tournament
GA151	Tournament	Arithmetic	Simple Inversion	Both Parent
GA152	Tournament	Arithmetic	Simple Inversion	Weak Parent
GA153	Tournament	Arithmetic	Random	Combined

GA154	Tournament	Arithmetic	Random	Binary Tournament
GA155	Tournament	Arithmetic	Random	Both Parent
GA156	Tournament	Arithmetic	Random	Weak Parent
GA157	Tournament	Arithmetic	Reversing	Combined
GA158	Tournament	Arithmetic	Reversing	Binary Tournament
GA159	Tournament	Arithmetic	Reversing	Both Parent
GA160	Tournament	Arithmetic	Reversing	Weak Parent
GA161	Tournament	Shuffle	Hybrid Comparing	Combined
GA162	Tournament	Shuffle	Hybrid Comparing	Binary Tournament
GA163	Tournament	Shuffle	Hybrid Comparing	Both Parent
GA164	Tournament	Shuffle	Hybrid Comparing	Weak Parent
GA165	Tournament	Shuffle	Simple Inversion	Combined
GA166	Tournament	Shuffle	Simple Inversion	Binary Tournament
GA167	Tournament	Shuffle	Simple Inversion	Both Parent
GA168	Tournament	Shuffle	Simple Inversion	Weak Parent
GA169	Tournament	Shuffle	Random	Combined
GA170	Tournament	Shuffle	Random	Binary Tournament

GA171	Tournament	Shuffle	Random	Both Parent
GA172	Tournament	Shuffle	Random	Weak Parent
GA173	Tournament	Shuffle	Reversing	Combined
GA174	Tournament	Shuffle	Reversing	Binary Tournament
GA175	Tournament	Shuffle	Reversing	Both Parent
GA176	Tournament	Shuffle	Reversing	Weak Parent
GA177	Tournament	Uniform	Hybrid Comparing	Combined
GA178	Tournament	Uniform	Hybrid Comparing	Binary Tournament
GA179	Tournament	Uniform	Hybrid Comparing	Both Parent
GA180	Tournament	Uniform	Hybrid Comparing	Weak Parent
GA181	Tournament	Uniform	Simple Inversion	Combined
GA182	Tournament	Uniform	Simple Inversion	Binary Tournament
GA183	Tournament	Uniform	Simple Inversion	Both Parent
GA184	Tournament	Uniform	Simple Inversion	Weak Parent
GA185	Tournament	Uniform	Random	Combined
GA186	Tournament	Uniform	Random	Binary Tournament
GA187	Tournament	Uniform	Random	Both Parent
GA188	Tournament	Uniform	Random	Weak Parent

GA189	Tournament	Uniform	Reversing	Combined
GA190	Tournament	Uniform	Reversing	Binary Tournament
GA191	Tournament	Uniform	Reversing	Both Parent
GA192	Tournament	Uniform	Reversing	Weak Parent
GA193	Roulette Wheel	Combined	Hybrid Comparing	Combined
GA194	Roulette Wheel	Combined	Hybrid Comparing	Binary Tournament
GA195	Roulette Wheel	Combined	Hybrid Comparing	Both Parent
GA196	Roulette Wheel	Combined	Hybrid Comparing	Weak Parent
GA197	Roulette Wheel	Combined	Simple Inversion	Combined
GA198	Roulette Wheel	Combined	Simple Inversion	Binary Tournament
GA199	Roulette Wheel	Combined	Simple Inversion	Both Parent
GA200	Roulette Wheel	Combined	Simple Inversion	Weak Parent
GA201	Roulette Wheel	Combined	Random	Combined
GA202	Roulette Wheel	Combined	Random	Binary Tournament
GA203	Roulette Wheel	Combined	Random	Both Parent
GA204	Roulette Wheel	Combined	Random	Weak Parent
GA205	Roulette Wheel	Combined	Reversing	Combined

GA206	Roulette Wheel	Combined	Reversing	Binary Tournament
GA207	Roulette Wheel	Combined	Reversing	Both Parent
GA208	Roulette Wheel	Combined	Reversing	Weak Parent
GA209	Roulette Wheel	Arithmetic	Hybrid Comparing	Combined
GA210	Roulette Wheel	Arithmetic	Hybrid Comparing	Binary Tournament
GA211	Roulette Wheel	Arithmetic	Hybrid Comparing	Both Parent
GA212	Roulette Wheel	Arithmetic	Hybrid Comparing	Weak Parent
GA213	Roulette Wheel	Arithmetic	Simple Inversion	Combined
GA214	Roulette Wheel	Arithmetic	Simple Inversion	Binary Tournament
GA215	Roulette Wheel	Arithmetic	Simple Inversion	Both Parent
GA216	Roulette Wheel	Arithmetic	Simple Inversion	Weak Parent
GA217	Roulette Wheel	Arithmetic	Random	Combined
GA218	Roulette Wheel	Arithmetic	Random	Binary Tournament
GA219	Roulette Wheel	Arithmetic	Random	Both Parent
GA220	Roulette Wheel	Arithmetic	Random	Weak Parent
GA221	Roulette Wheel	Arithmetic	Reversing	Combined
GA222	Roulette Wheel	Arithmetic	Reversing	Binary Tournament

GA223	Roulette Wheel	Arithmetic	Reversing	Both Parent
GA224	Roulette Wheel	Arithmetic	Reversing	Weak Parent
GA225	Roulette Wheel	Shuffle	Hybrid Comparing	Combined
GA226	Roulette Wheel	Shuffle	Hybrid Comparing	Binary Tournament
GA227	Roulette Wheel	Shuffle	Hybrid Comparing	Both Parent
GA228	Roulette Wheel	Shuffle	Hybrid Comparing	Weak Parent
GA229	Roulette Wheel	Shuffle	Simple Inversion	Combined
GA230	Roulette Wheel	Shuffle	Simple Inversion	Binary Tournament
GA231	Roulette Wheel	Shuffle	Simple Inversion	Both Parent
GA232	Roulette Wheel	Shuffle	Simple Inversion	Weak Parent
GA233	Roulette Wheel	Shuffle	Random	Combined
GA234	Roulette Wheel	Shuffle	Random	Binary Tournament
GA235	Roulette Wheel	Shuffle	Random	Both Parent
GA236	Roulette Wheel	Shuffle	Random	Weak Parent
GA237	Roulette Wheel	Shuffle	Reversing	Combined
GA238	Roulette Wheel	Shuffle	Reversing	Binary Tournament
GA239	Roulette Wheel	Shuffle	Reversing	Both Parent
GA240	Roulette Wheel	Shuffle	Reversing	Weak Parent

GA241	Roulette Wheel	Uniform	Hybrid Comparing	Combined
GA242	Roulette Wheel	Uniform	Hybrid Comparing	Binary Tournament
GA243	Roulette Wheel	Uniform	Hybrid Comparing	Both Parent
GA244	Roulette Wheel	Uniform	Hybrid Comparing	Weak Parent
GA245	Roulette Wheel	Uniform	Simple Inversion	Combined
GA246	Roulette Wheel	Uniform	Simple Inversion	Binary Tournament
GA247	Roulette Wheel	Uniform	Simple Inversion	Both Parent
GA248	Roulette Wheel	Uniform	Simple Inversion	Weak Parent
GA249	Roulette Wheel	Uniform	Random	Combined
GA250	Roulette Wheel	Uniform	Random	Binary Tournament
GA251	Roulette Wheel	Uniform	Random	Both Parent
GA252	Roulette Wheel	Uniform	Random	Weak Parent
GA253	Roulette Wheel	Uniform	Reversing	Combined
GA254	Roulette Wheel	Uniform	Reversing	Binary Tournament
GA255	Roulette Wheel	Uniform	Reversing	Both Parent
GA256	Roulette Wheel	Uniform	Reversing	Weak Parent

GA Results on Average Fitness

GA Model	Average Fitness										Performance
	Sphere	Ackley	Rastrigin	Zakharov	Axis-Parallel	Griewank	Sum of Different Powers	Rotated Hyper-Ellipsoid	Schwefel 2.22	Exponential	
GA001	5.171826E-02	2.731250E+00	4.357121E+01	4.424637E+04	4.728743E-01	1.206560E+00	3.009819E-07	2.159622E+02	1.196187E+00	-9.979053E-01	0/10
GA002	6.213081E-02	1.662573E+00	1.792644E+01	1.094069E+02	2.482101E+00	1.432483E+00	7.535564E-06	6.138573E+02	6.671384E-01	-9.993071E-01	0/10
GA003	1.376655E+00	5.831965E+00	1.530576E+02	9.539615E+08	1.428665E+01	4.602608E+00	4.631712E-05	1.920763E+03	3.743330E+00	-9.834357E-01	0/10
GA004	3.091899E-02	1.653336E+00	2.426802E+01	3.089369E+01	4.315668E-01	1.087193E+00	2.686469E-06	1.830337E+02	5.890799E-01	-9.991267E-01	1/10
GA005	1.335892E-01	1.681182E+00	5.471498E+00	1.463280E+08	5.060805E-01	1.219032E+00	3.657521E-08	1.969933E+02	4.511763E-01	-9.995562E-01	0/10
GA006	9.350947E-01	3.876719E+00	1.367764E+01	9.547055E+01	2.343014E+01	2.635915E+00	2.327499E-05	1.614100E+03	1.384843E+00	-9.905849E-01	0/10
GA007	5.985657E-01	4.394615E+00	5.389029E+01	6.090281E+08	6.269545E+00	2.021272E+00	5.683116E-07	1.531509E+03	1.411785E+00	-9.918462E-01	0/10
GA008	1.308319E-01	2.219906E+00	1.555026E+01	5.402530E+01	5.885964E+00	1.878673E+00	5.400799E-09	4.009384E+02	7.538916E-01	-9.965834E-01	1/10
GA009	2.337016E+00	7.904851E+00	1.395359E+02	4.271774E+01	3.836668E+01	1.014789E+01	1.586809E-06	5.592524E+03	5.221723E+00	-9.578843E-01	0/10
GA010	1.934789E+00	5.203863E+00	3.102797E+01	1.236523E+02	3.690288E+01	5.181211E+00	5.110956E-05	2.251639E+03	3.045261E+00	-9.741900E-01	0/10
GA011	4.981432E+00	9.246731E+00	1.809426E+02	8.119826E+08	7.451557E+01	1.885176E+01	4.276630E-05	1.189566E+04	7.810742E+00	-9.373990E-01	0/10
GA012	7.489558E-01	5.133802E+00	3.892240E+01	1.101140E+02	6.942795E+00	3.626440E+00	1.034554E-06	9.272802E+02	2.767533E+00	-9.902977E-01	0/10
GA013	1.062842E-01	2.764732E+00	7.469754E+01	1.646164E+07	1.399305E+00	1.299157E+00	2.236888E-08	3.662659E+02	1.984378E+00	-9.973072E-01	0/10
GA014	5.125166E-01	3.794348E+00	2.224619E+01	1.026576E+02	1.780085E+01	3.229244E+00	3.761319E-04	2.037727E+03	1.464932E+00	-9.981767E-01	0/10
GA015	1.705723E+00	6.369295E+00	1.666655E+02	4.972066E+08	2.291331E+01	6.437832E+00	2.396611E-06	3.225576E+03	5.415266E+00	-9.766475E-01	0/10
GA016	1.731689E-01	3.145344E+00	2.331195E+01	4.371970E+01	2.667771E+00	1.390974E+00	2.314432E-06	8.194545E+02	1.155789E+00	-9.982870E-01	0/10
GA017	3.989883E-01	3.777687E+00	8.695722E+01	4.117177E+01	4.457331E+00	1.883165E+00	5.125390E-07	8.001101E+02	3.320787E+00	-9.951548E-01	0/10
GA018	5.702770E-01	3.750724E+00	3.004331E+01	1.672217E+02	7.821612E+00	2.820164E+00	3.506201E-05	1.952660E+03	2.665692E+00	-9.932429E-01	0/10
GA019	1.128971E+00	5.935223E+00	1.355580E+02	5.960455E+08	1.506753E+01	4.707740E+00	1.326478E-05	2.619482E+03	4.213677E+00	-9.830107E-01	0/10
GA020	2.546361E-01	3.878833E+00	4.023335E+01	2.832643E+02	3.176788E+00	1.929804E+00	1.097228E-06	7.427240E+02	2.243756E+00	-9.936623E-01	0/10
GA021	1.866483E+00	5.379456E+00	1.131258E+02	5.349473E+08	1.495190E+01	5.848914E+00	1.016638E-05	3.882320E+03	2.831330E+00	-9.641461E-01	0/10
GA022	3.270942E+00	7.301767E+00	9.288095E+01	6.876336E+05	4.152998E+01	1.645497E+01	8.917497E-06	4.628864E+03	5.331978E+00	-9.803117E-01	0/10
GA023	7.942850E-01	3.129570E+00	4.290046E+01	5.258468E+08	1.092453E+01	1.666948E+00	2.445304E-06	5.210512E+02	1.917364E+00	-9.887533E-01	0/10
GA024	2.515557E+00	6.251246E+00	1.073123E+02	1.470510E+05	5.620732E+01	8.677622E+00	2.502553E-06	3.802472E+03	4.636622E+00	-9.733008E-01	0/10
GA025	3.549180E+00	8.996783E+00	1.487921E+02	8.733753E+01	6.341190E+01	1.162640E+01	9.779130E-06	6.168595E+03	6.030058E+00	-9.425881E-01	0/10
GA026	2.804247E+00	6.892659E+00	3.606433E+01	2.281054E+02	4.856417E+01	6.153137E+00	8.875050E-05	2.236175E+03	4.145503E+00	-9.684112E-01	0/10
GA027	6.867880E+00	1.046620E+01	1.941591E+02	1.326791E+09	9.679329E+01	2.133244E+01	6.747074E-05	1.602380E+04	8.507453E+00	-9.179438E-01	0/10
GA028	1.837634E+00	5.814022E+00	3.170643E+01	2.178796E+02	2.012035E+01	3.481371E+00	1.562136E-05	1.012554E+03	3.429405E+00	-9.694371E-01	0/10
GA029	2.255423E+00	6.896261E+00	1.574602E+02	7.317756E+08	3.482807E+01	8.314873E+00	2.174329E-05	3.507945E+03	6.151920E+00	-9.829945E-01	0/10
GA030	7.389042E+00	9.433845E+00	7.374587E+01	1.665897E+08	5.787522E+01	3.521910E+01	1.752645E-03	1.438740E+04	8.728745E+00	-9.583740E-01	0/10
GA031	1.601456E+00	6.343985E+00	1.770870E+02	1.011342E+09	2.171866E+01	6.915448E+00	1.035097E-05	4.939651E+03	4.414136E+00	-9.701888E-01	0/10
GA032	1.527533E+00	7.300680E+00	7.252421E+01	1.840550E+08	2.961351E+01	1.222700E+01	1.794949E-05	2.655595E+03	7.237293E+00	-9.562146E-01	0/10
GA033	4.664650E-02	1.225805E+00	1.801247E+00	1.451088E+02	6.129491E-01	1.030491E+00	1.175293E-05	3.521229E+02	8.372756E-01	-9.992982E-01	0/10
GA034	1.067761E-01	2.514646E+00	4.978944E+00	3.629080E+02	2.291247E+00	1.681050E+00	5.504237E-04	1.435145E+03	9.041421E-01	-9.979690E-01	0/10
GA035	4.360151E+01	1.690135E+01	2.295137E+02	2.069539E+02	6.474021E+02	2.018439E+02	2.392153E-02	1.089408E+05	3.011608E+01	-3.988442E-01	0/10
GA036	1.078948E-01	2.508377E+00	6.738598E+00	3.586943E+02	3.061123E+00	1.068298E+00	2.876324E-05	7.873809E+02	6.879788E-01	-9.979059E-01	0/10
GA037	2.561252E+00	7.121330E+00	1.602942E+01	2.101366E+02	2.288871E+01	7.936088E+00	3.033666E-03	2.504887E+03	1.261543E+00	-9.792482E-01	0/10
GA038	4.535947E+00	8.978973E+00	3.267570E+01	4.307307E+02	1.159046E+02	1.820430E+01	8.739194E-03	8.423216E+03	1.999390E+00	-9.518477E-01	0/10
GA039	7.171433E+01	1.923953E+01	2.758521E+02	5.100190E+02	1.134126E+03	2.888691E+02	6.222551E-02	2.002364E+05	4.319403E+01	-1.756687E-01	0/10
GA040	3.217048E+00	5.736300E+00	1.851428E+01	3.008284E+02	7.516930E+01	1.228065E+01	1.778867E-03	7.392211E+03	1.334148E+00	-9.371692E-01	0/10
GA041	8.961632E+00	1.235039E+01	8.061154E+01	3.249868E+02	1.103371E+02	4.241275E+01	3.027083E-04	1.906236E+04	7.865260E+00	-8.074859E-01	0/10
GA042	6.815194E+00	8.435733E+00	3.695109E+01	2.865500E+02	7.597415E+01	1.736715E+01	9.690339E-04	4.312587E+03	5.374006E+00	-9.077557E-01	0/10
GA043	1.054407E+02	1.976100E+01	3.592010E+02	1.097386E+03	1.504125E+03	3.788644E+02	1.422372E-01	2.382684E+05	4.766443E+01	-1.081047E-01	0/10
GA044	2.571158E+00	7.835463E+00	3.999445E+01	3.262750E+02	4.753439E+01	1.187303E+01	4.521587E-03	2.718488E+03	4.736350E+00	-9.364559E-01	0/10
GA045	3.604465E-01	3.920626E+00	9.051959E+00	3.834311E+02	3.645337E+01	4.136720E+00	8.707851E-04	1.883476E+03	1.075776E+00	-9.871897E-01	0/10
GA046	9.863805E+00	1.014968E+01	6.998248E+01	4.058748E+02	8.033393E+01	2.097307E+01	1.016524E-03	9.033418E+03	4.224832E+00	-9.219690E-01	0/10
GA047	9.124900E+01	1.957825E+01	3.111049E+02	3.570380E+06	1.137147E+03	3.496882E+02	1.283520E-01	2.275068E+05	4.092717E+01	-1.851729E-01	0/10
GA048	6.125336E+00	1.023929E+01	5.470302E+01	4.233138E+02	2.117890E+01	1.777241E+01	1.409619E-03	5.063016E+03	2.346752E+00	-8.933394E-01	0/10
GA049	1.824120E-02	5.143914E-01	1.250391E+00	1.817077E+02	4.702494E-01	1.042101E+00	1.277046E-05	1.160845E+03	3.222145E-01	-9.994279E-01	3/10
GA050	1.224036E-01	2.559595E+00	9.047510E+00	3.321052E+02	1.434850E+00	1.355023E+00	5.024684E-04	1.979549E+03	9.163186E-01	-9.988825E-01	0/10
GA051	3.264630E+01	1.687727E+01	2.340144E+02	1.915710E+02	5.151111E+02	1.163301E+02	7.291971E-03	8.407070E+04	2.822812E+01	-3.736145E-01	0/10
GA052	6.698909E-02	2.896684E+00	7.782264E+00	3.337848E+02	8.063289E-01	1.023759E+00	1.325409E-05	1.142134E+03	6.259041E-01	-9.983378E-01	0/10
GA053	1.412396E+00	5.914663E+00	1.719703E+01	2.783096E+02	3.265303E+01	6.105855E+00	2.036191E-03	2.686624E+03	8.532977E-01	-9.730179E-01	0/10
GA054	7.666738E+00	7.880339E+00	2.921069E+01	4.217993E+02	1.416084E+02	1.171680E+01	6.363074E-03	1.137412E+04	1.031220E+00	-9.375123E-01	0/10
GA055	8.291785E+01	1.933483E+01	2.809637E+02	3.418610E+02	1.112223E+03	2.787939E+02	5.590867E-02	2.057409E+05	3.904024E+01	-1.668121E-01	0/10
GA056	9.838243E-01	8.261100E+00	2.585692E+01	2.713694E+02	5.735906E+01	1.370706E+01	6.164447E-04	3.908000E+03	1.950041E+00	-9.545105E-01	0/10
GA057	6.734779E+00	1.119315E+01	5.977488E+01	3.424414E+02	8.211935E+01	3.158660E+01	4.010035E-03	1.557632E+04	7.275155E+00	-8.734000E-01	0/10
GA058	3.619582E+00	7.163214E+00	3.272553E+01	3.410302E+02	9.597286E+01	1.050044E+01	6.004706E-04	3.798079E+03	5.038697E+00	-9.419592E-01	0/10
GA059	1.037466E+02	1.989077E+01	3.279801E+02	4.647937E+02	1.436438E+03	3.518360E+02	1.379184E-01	2.180969E+05	4.358798E+01	-1.281737E-01	0/10
GA060	2.430508E+00	7.063437E+00	3.503569E+01	3.315851E+02	4.906566E+01	1.030027E+01	1.329165E-04	2.993530E+03	5.079257E+00	-9.038763E-01	0/10
GA061	2.481035E-01	5.206089E+00	1.551520E+01	3.121034E+02	2.324466E+01	4.144553E+00	4.221916E-04	3.044494E+03	7.907356E-01	-9.970869E-01	0/10
GA062	9.423679E+00	1.065703E+01	8.063336E+01	4.851045E+02	8.105432E+01	3.099870E+01	7.726591E-03	1.232574E+04	3.378888E+00	-8.641660E-01	0/10
GA063	8.743549E+01	1.934375E+01	2.922137E+02	3.434918E+02	1.080980E+03	2.715509E+02	5.112512E-02	1.909669E+05	4.186264E+01	-1.672130E-01	0/10
GA064	1.167312E+01	1.183179E+01	7.035772E+01	4.556786E+02	8.540796E+01	5.478925E+00	1.579493E-03	7.555500E+03	2.412731E+00	-9.445095E-01	0/10
GA065	1.999609E-01	3.430619E+00	9.731373E+01	1.450917E+07	3.171525E+00	1.623026E+00	2.968850E-06	3.788360E+02	1.767991E+00	-9.969964E-01	0/10
GA066	8.196896E-01	5.561894E+00	9.231776E+01	9.031964E+01	1.408609E+01	5.653802E+00	4.538549E-05	2.338858E+03	3.307617E+00	-9.793156E-01	0/10
GA067	1.072243E+00	5.611064E+00	1.615002E+02	8.990035E+08	1.396624E+01	4.727984E+00	1.513999E-05	2.979843E+03	4.229540E+00	-9.816321E-01	0/10
GA068	2.549375E+00	8.680808E+00	1.716411E+02	9.114214E+05	4.206566E+01	1.152745E+01	8.405063E-05	7.638538E+03	7.094941E+00	-9.481732E-01	0/10
GA069	1.299268E-01	1.947156E+00	1.626454E+01	4.677466E+08	4.776233E+00	1.095878E+00	1.428351E-06	2.227084E+02	1.060385E+00	-9.984314E-01	0/10
GA070	2.896696E-01	2.994248E+00	8.993540E+01	6.292250E+01	5.637666E+00	2.398102E+00	4.225362E-07	1.111203E+03	1.680293E+00	-9.900109E-01	0/10
GA071	1.722750E-01	4.063293E+00	5.062910E+01	7.849743E+08	7.874382E+00	1.948468E+00	2.124110E-06	2.828358E+02	2.478791E+00	-9.905827E-01	0/10
GA072	5.978374E-01	4.746620E+00	1.600510E+02	2.661915E+02	9.810984E+00	2.460789E+00	1.855150E-05	9.198620E+02	2.308544E+00	-9.902152E-01	0/10
GA073	1.956412E+00	6.820668E+00	1.541979E+02	3.175058E+01	2.443075E+01	7.736630E+00	1.787695E-05	3.988705E+03	5.349177E+00	-9.644838E-01	0/10
GA074	3.556276E+00	8.559277E+00	1.055050E+02	9.269516E+01	4.495682E+01	1.511499E+01	2.345801E-06	7.670056E+03	6.567312E+00	-9.316091E-01	0/10
GA075	2.813243E+00	8.371828E+00	1.732785E+02	5.641767E+08	4.824619E+01	1.095067E+01	3.227824E-05	6.731448E+03	6.896377E+00	-9.437809E-01	0/10
GA076	5.998916E+00	1.042007E+01	1.877499E+02	1.955833E+02	8.850421E+01	2.530360E+01	9.040301E-05	1.578042E+04	1.038371E+01	-8.924617E-01	0/10
GA077	2.582122E-01	3.535113E+00	7.688239E+01	7.323206E+08	4.492692E+00	1.812574E+00	1.081874E-06	5.544586E+02	2.134881E+00	-9.948405E-01	0/10
GA078	1.150857E+00	6.837049E+00	8.772754E+01	8.754536E+01	2.288454E+01	5.097716E+00	8.460327E-06	3.028310E+03	4.459467E+00	-9.793356E-01	0/10
GA079	1.180766E+00	6.661387E+00	1.763147E+02	1.347806E+09	1.530786E+01	7.211025E+00	1.565438E-04	3.630522E+03	5.071264E+00	-9.663149E-01	0/10
GA080	3.033292E+00	8.149730E+00	1.603776E+02	4.738327E+05	5.410942E+01	1.180169E+01	2.371592E-05	6.734990E+03	7.426725E+00	-9.310254E-01	0/10
GA081	7.244361E-01	5.258131E+00	1.284652E+02	7.148170E+02	8.796418E+00	2.689966E+00	6.094911E-06	1.466666E+03	3.323752E+00	-9.889672E-01	0/10
GA082	1.731800E+00	6.789076E+00	1.151075E+02	2.134883E+02	2.909913E+01	6.950639E+00	4.276082E-06	4.949305E+03	5.092610E+00	-9.675507E-01	0/10
GA083	8.731855E-01	5.629431E+00	1.752590E+02	5.746919E+08	2.107962E+01	5.560622E+00	3.221593E-05	2.882134E+03	4.366954E+00	-9.786080E-01	0/10
GA084	2.634980E+00	7.751510E+00	1.774096E+02	6.350123E+07	3.499497E+01	1.126148E+01	1.651356E-05	8.232801E+03	6.664333E+00	-9.479978E-01	0/10
GA085	1.100743E+00	5.437887E+00	8.844624E+01	8.174867E+08	2.170854E+01	3.998663E+00	3.752033E-06	9.520244E+02	2.463334E+00	-9.802141E-01	0/10
GA086	1.216884E+00	5.534249E+00	1.303722E+02	3.212388E+05	1.209388E+01	4.688221E+00	7.336030E-06	2.704399E+03	5.406042E+00	-9.802910E-01	0/10
GA087	3.152138E-01	4.008136E+00	4.499920E+01	2.289257E+09	1.218027E+01	2.459033E+00	5.572022E-06	5.209946E+02	1.346847E+00	-9.952282E-01	0/10
GA088	6.411523E-01	4.581363E+00	1.797126E+02	1.694664E+08	1.036004E+01	3.016483E+00	1.309990E-05	1.523622E+03	4.491464E+00	-9.869504E-01	0/10
GA089	2.303475E+00	8.351707E+00	1.546238E+02	6.047071E+01	2.557786E+01	9.569012E+00	7.708379E-06	5.051064E+03	5.597251E+00	-9.623972E-01	0/10
GA090	6.249445E+00	9.000737E+00	9.876157E+01	2.452194E+02	6.342841E+01	1.432541E+01	3.342507E-05	9.317210E+03	7.513316E+00	-8.927767E-01	0/10
GA091	3.612808E+00	8.969309E+00	1.897180E+02	1.521845E+09	6.240834E+01	1.407862E+01	2.560300E-05	8.704415E+03	8.726509E+00	-9.285753E-01	0/10
GA092	5.393659E+00	1.045050E+01	1.928281E+02	1.335130E+03	8.998588E+01	2.239958E+01	5.116754E-05	1.204542E+04	9.308367E+00	-8.837653E-01	0/10
GA093	2.475116E+00	7.002840E+00	1.702165E+02	1.511190E+09	3.438610E+01	8.145059E+00	5.009588E-05	3.876159E+03	6.396411E+00	-9.667853E-01	0/10
GA094	3.543912E+00	7.867626E+00	1.136483E+02	3.571795E+08	5.044301E+01	1.152426E+01	3.492005E-05	6.896472E+03	8.897608E+00	-9.434929E-01	0/10
GA095	1.511776E+00	6.880922E+00	1.496858E+02	7.244343E+08	1.899488E+01	6.121189E+00	8.758207E-05	2.839984E+03	3.332082E+00	-9.725304E-01	0/10

GA096	3.106981E+00	7.905755E+00	1.777108E+02	1.763905E+09	3.292864E+01	1.526827E+01	8.234632E-05	6.438347E+03	6.658466E+00	-9.366357E-01	0/10
GA097	3.959938E-01	3.650836E+00	8.759579E+00	2.503704E+02	3.416204E+00	2.387630E+00	1.321933E-04	1.015501E+03	1.058321E+00	-9.930943E-01	0/10
GA098	4.358289E+00	9.376761E+00	5.713178E+01	2.504032E+02	7.520684E+01	2.007982E+01	8.437103E-04	8.749751E+03	5.139836E+00	-9.278866E-01	0/10
GA099	9.057872E+01	1.903903E+01	3.392387E+02	1.635701E+05	1.099729E+03	2.895560E+02	1.125632E-01	1.967182E+05	3.380112E+01	-2.645503E-01	0/10
GA100	2.931346E+01	1.660812E+01	1.805153E+02	3.322030E+02	3.773253E+02	1.191333E+02	6.356777E-03	7.282450E+04	2.024610E+01	-5.404904E-01	0/10
GA101	1.000507E+00	4.560009E+00	6.686723E+00	2.741136E+02	3.290484E+01	2.925406E+00	5.800591E-04	1.841462E+03	1.371441E+00	-9.900863E-01	0/10
GA102	3.123803E+00	7.203001E+00	3.005121E+01	3.007938E+02	7.334723E+01	7.590591E+00	4.036680E-04	8.307824E+03	2.349804E+00	-9.726572E-01	0/10
GA103	1.175069E+02	2.016289E+01	3.563667E+02	4.315423E+06	1.599708E+03	4.124945E+02	2.727929E-01	2.636775E+05	4.925205E+01	-7.411238E-02	0/10
GA104	3.542456E+01	1.577893E+01	1.581775E+02	3.315448E+02	4.305282E+02	1.407632E+02	2.031578E-02	8.545450E+04	1.713489E+01	-5.743125E-01	0/10
GA105	8.954782E+00	1.148087E+01	6.887908E+01	2.968509E+02	1.011837E+02	3.433771E+01	5.583802E-04	1.783511E+04	8.954952E+00	-8.419217E-01	0/10
GA106	1.267982E+01	1.294082E+01	9.486477E+01	3.390574E+02	1.641395E+02	4.216465E+01	1.459156E-03	2.868396E+04	1.089181E+01	-8.200142E-01	0/10
GA107	1.306932E+02	2.026095E+01	3.806148E+02	2.290205E+05	1.738296E+03	4.328631E+02	1.620993E-01	2.850484E+05	4.874793E+01	-8.144415E-02	0/10
GA108	4.767353E+01	1.760115E+01	2.103307E+02	3.713778E+02	5.040482E+02	1.468845E+02	1.132501E-02	9.250476E+04	2.252478E+01	-4.094824E-01	0/10
GA109	7.402456E-01	5.420387E+00	1.804123E+01	2.887932E+02	2.139053E+01	3.347959E+00	3.963459E-03	1.900510E+03	1.225806E+00	-9.839601E-01	0/10
GA110	6.749209E+00	1.006365E+01	6.546392E+01	3.066651E+02	1.313908E+02	3.452314E+01	1.834330E-03	2.193593E+04	7.508291E+00	-8.778954E-01	0/10
GA111	1.303606E+02	2.018223E+01	3.674564E+02	5.031826E+04	1.842418E+03	4.498898E+02	3.329094E-01	3.061400E+05	5.111829E+01	-1.075126E-01	0/10
GA112	3.789643E+01	1.731552E+01	1.828654E+02	3.692984E+02	5.088363E+02	1.196213E+02	2.066148E-02	8.698321E+04	2.219120E+01	-5.428234E-01	0/10
GA113	1.363956E-01	2.968900E+00	5.291752E+00	1.930039E+02	3.180994E+00	1.353899E+00	5.210802E-05	7.871413E+02	5.461282E-01	-9.982007E-01	0/10
GA114	3.755379E+00	8.944487E+00	5.800903E+01	2.723927E+02	3.980857E+01	1.345994E+01	1.291056E-03	7.640723E+03	4.033815E+00	-9.263520E-01	0/10
GA115	6.976823E+01	1.879691E+01	3.066796E+02	2.790676E+02	9.898589E+02	2.580109E+02	2.751517E-02	1.589065E+05	3.404152E+01	-2.468667E-01	0/10
GA116	3.848180E+01	1.704731E+01	2.057350E+02	3.382476E+02	5.938863E+02	1.494544E+02	1.885353E-02	9.445306E+04	2.422342E+01	-4.709091E-01	0/10
GA117	5.329590E-01	3.944416E+00	4.466740E+00	2.842087E+02	1.762108E+01	5.053053E+00	2.742013E-04	4.182882E+02	7.167267E-01	-9.849442E-01	0/10
GA118	2.043969E+00	6.191573E+00	1.735760E+01	2.713526E+02	4.352828E+01	7.438801E+00	6.112153E-04	7.621214E+03	2.066927E+00	-9.688367E-01	0/10
GA119	1.331968E+02	2.011581E+01	3.711733E+02	2.081209E+03	1.605646E+03	4.402649E+02	1.807059E-01	2.590975E+05	4.899976E+01	-1.057716E-01	0/10
GA120	4.095349E+01	1.720309E+01	1.981650E+02	3.899406E+02	6.655029E+02	1.620070E+02	2.902030E-02	1.013837E+05	2.283882E+01	-4.127049E-01	0/10
GA121	6.409487E+00	1.073192E+01	6.186171E+01	3.234777E+02	8.897853E+01	2.281794E+01	3.048689E-04	1.358133E+04	6.700734E+00	-8.711375E-01	0/10
GA122	1.280195E+01	1.259764E+01	9.859565E+01	3.150271E+02	1.588811E+02	4.446766E+01	2.494286E-03	2.388785E+04	1.067529E+01	-8.010707E-01	0/10
GA123	1.205347E+02	2.026085E+01	3.635332E+02	2.147663E+04	1.630123E+03	4.433896E+02	2.144729E-01	2.994334E+05	5.043615E+01	-1.156726E-01	0/10
GA124	5.177637E+01	1.796388E+01	2.241928E+02	3.671153E+02	6.887773E+02	1.673432E+02	2.933828E-02	1.151137E+05	2.807852E+01	-3.656273E-01	0/10
GA125	3.605192E-01	3.959388E+00	1.779328E+01	3.051433E+02	1.187055E+01	3.671708E+00	5.379168E-04	1.295041E+03	6.420484E-01	-9.889920E-01	0/10
GA126	5.197133E+00	9.992219E+00	4.885702E+01	3.139957E+02	8.276327E+01	1.932691E+01	1.600772E-03	1.022401E+04	6.407688E+00	-8.491748E-01	0/10
GA127	1.255672E+02	2.028167E+01	3.654145E+02	6.169268E+03	1.655154E+03	4.123673E+02	1.561348E-01	2.861480E+05	4.934351E+01	-8.961246E-02	0/10
GA128	4.485701E+01	1.813337E+01	2.166749E+02	3.948130E+02	6.835935E+02	1.790130E+02	3.806777E-02	1.185549E+05	2.520515E+01	-3.552137E-01	0/10
GA129	4.824474E-02	2.849575E+00	7.064087E+01	4.062054E+04	8.073761E-01	1.213191E+00	5.602996E-08	1.851723E+02	8.333932E-01	-9.990882E-01	0/10
GA130	4.171431E-02	2.372568E+00	1.198589E+01	9.693984E+01	5.197019E-01	1.170983E+00	4.385081E-07	3.732124E+02	5.106022E-01	-9.990653E-01	0/10
GA131	1.463260E+00	5.815202E+00	1.407823E+02	6.036874E+08	1.592253E+01	4.971835E+00	8.215322E-06	2.396124E+03	4.888175E+00	-9.791624E-01	0/10
GA132	3.455458E-02	2.313881E+00	2.904014E+01	2.969400E+01	1.082302E+00	1.205262E+00	1.953732E-08	2.847147E+02	4.305375E-01	-9.985225E-01	1/10
GA133	9.292286E-02	1.724398E+00	1.307816E+01	1.494780E+07	7.012828E-01	8.608545E-01	6.131270E-08	1.399544E+02	2.731055E-01	-9.995527E-01	1/10
GA134	8.485829E-01	4.188398E+00	9.538884E+00	1.768002E+02	2.659228E+01	3.837596E+00	3.925941E-06	9.817412E+02	1.420042E+00	-9.762947E-01	0/10
GA135	3.330907E-01	4.052190E+00	5.425926E+01	5.092262E+08	3.973379E+00	1.737814E+00	8.731129E-07	8.672316E+02	1.656627E+00	-9.891069E-01	0/10
GA136	3.009742E-01	2.936324E+00	1.246383E+01	4.238363E+01	4.980393E+00	1.145039E+00	2.768294E-08	4.114229E+02	7.821215E-01	-9.971301E-01	0/10
GA137	2.292686E+00	7.476605E+00	1.439137E+02	3.987140E+01	4.806719E+01	9.194996E+00	3.599274E-06	5.306775E+03	6.103113E+00	-9.573015E-01	0/10
GA138	1.479475E+00	6.117858E+00	3.170740E+01	1.684468E+02	2.775336E+01	6.477346E+00	9.573953E-05	1.054901E+03	2.108386E+00	-9.813439E-01	0/10
GA139	4.662233E+00	9.247376E+00	1.852545E+02	3.542680E+08	7.211073E+01	1.514019E+01	2.255768E-05	9.842806E+03	8.475224E+00	-9.255622E-01	0/10
GA140	7.827568E-01	4.355849E+00	3.785689E+01	1.414711E+02	1.003035E+01	3.509851E+00	2.267139E-06	1.232881E+03	2.678912E+00	-9.901504E-01	0/10
GA141	6.246182E-02	3.564470E+00	8.823740E+01	1.196274E+07	1.666905E+00	1.293006E+00	5.151415E-08	3.421972E+02	1.088806E+00	-9.978630E-01	0/10
GA142	2.118885E-01	3.577681E+00	2.037531E+01	1.023049E+02	1.138632E+01	2.512983E+00	1.628078E-04	1.601366E+03	1.219428E+00	-9.943942E-01	0/10
GA143	9.598864E-01	6.454354E+00	1.570332E+02	1.002684E+09	1.855972E+01	4.695450E+00	4.315343E-05	3.184722E+03	5.327368E+00	-9.627276E-01	0/10
GA144	9.558705E-02	2.425880E+00	3.203443E+01	3.705440E+01	3.702559E+00	1.415941E+00	2.322538E-06	4.007101E+02	7.529652E-01	-9.984473E-01	0/10
GA145	3.363142E-01	4.268962E+00	1.263778E+02	4.431442E+01	3.308101E+00	1.858853E+00	1.162732E-06	6.544743E+02	2.333318E+00	-9.954008E-01	0/10
GA146	3.059927E-01	4.239401E+00	4.656414E+01	2.591102E+02	4.183607E+00	1.880629E+00	8.100418E-06	1.094970E+03	2.058450E+00	-9.934447E-01	0/10
GA147	1.210590E+00	5.890290E+00	1.409234E+02	1.095489E+09	1.596122E+01	5.055827E+00	1.653805E-05	3.410114E+03	4.968484E+00	-9.819392E-01	0/10
GA148	3.921551E-01	3.698647E+00	4.910108E+01	2.148189E+02	4.267769E+00	2.047973E+00	2.131408E-06	7.991544E+02	1.959941E+00	-9.921419E-01	0/10
GA149	1.286007E+00	5.831955E+00	6.016678E+01	4.583432E+08	1.599919E+01	6.618598E+00	7.466753E-06	1.804030E+03	2.930784E+00	-9.834615E-01	0/10
GA150	2.396743E+00	6.767940E+00	1.019339E+02	1.678748E+07	3.787960E+01	9.280239E+00	5.330239E-07	3.452386E+03	6.783831E+00	-9.629837E-01	0/10
GA151	2.691684E-01	4.443778E+00	5.077744E+01	1.512593E+09	1.621612E+01	2.172457E+00	1.630534E-06	1.049910E+03	3.500227E+00	-9.877414E-01	0/10
GA152	2.931943E+00	6.266657E+00	1.110767E+02	2.598315E+02	3.211019E+01	6.789237E+00	4.554160E-07	2.520377E+03	5.208038E+00	-9.735323E-01	0/10
GA153	2.937499E+00	9.029820E+00	1.443001E+02	7.507206E+01	4.872203E+01	1.257297E+01	2.125211E-05	5.415677E+03	6.820152E+00	-9.493960E-01	0/10
GA154	1.778397E+00	5.786794E+00	3.593580E+01	2.321858E+02	2.996883E+01	4.695601E+00	1.035545E-04	1.972153E+03	3.094320E+00	-9.528196E-01	0/10
GA155	5.659671E+00	1.025858E+01	1.862771E+02	2.160828E+08	8.532359E+01	2.073894E+01	7.429075E-05	1.257675E+04	9.671793E+00	-9.026248E-01	0/10
GA156	1.203609E+00	6.438809E+00	4.248866E+01	2.556923E+02	2.542096E+01	5.503643E+00	4.276328E-05	1.353020E+03	2.597827E+00	-9.807217E-01	0/10
GA157	2.225646E+00	7.728480E+00	1.625082E+02	2.257575E+09	3.476372E+01	1.118212E+01	1.316093E-04	3.545819E+03	6.429077E+00	-9.697404E-01	0/10
GA158	6.211352E+00	8.577522E+00	7.357036E+01	1.633695E+08	1.011335E+02	1.204064E+01	2.510168E-04	9.693822E+03	9.147218E+00	-8.803960E-01	0/10
GA159	1.367096E+00	6.853632E+00	1.810382E+02	3.305935E+09	2.676542E+01	6.753888E+00	1.330173E-05	3.451597E+03	5.236330E+00	-9.598490E-01	0/10
GA160	2.153927E+00	8.178688E+00	7.460169E+01	3.586608E+08	3.295104E+01	8.262822E+00	1.201120E-04	4.779111E+03	6.553650E+00	-9.492329E-01	0/10
GA161	7.453974E-02	1.676808E+00	6.627191E+00	1.927092E+02	7.784324E-01	1.168130E+00	1.904844E-03	1.322373E+03	3.907284E-01	-9.995779E-01	0/10
GA162	6.977387E-02	2.241351E+00	5.929422E+00	4.703356E+02	2.573339E+00	1.129619E+00	6.598212E-04	1.064111E+03	8.144158E-01	-9.981613E-01	0/10
GA163	4.427304E+01	1.735043E+01	2.807934E+02	2.208230E+02	7.768943E+02	1.454340E+02	2.547668E-02	1.163577E+05	2.317793E+01	-3.395793E-01	0/10
GA164	9.166498E-02	2.447398E+00	9.521689E+00	3.330169E+02	1.104741E+00	1.085509E+00	1.310580E-04	7.546334E+02	8.761242E-01	-9.968916E-01	0/10
GA165	1.822823E+00	5.483439E+00	2.379016E+01	2.233695E+02	6.616077E+01	7.719792E+00	1.127627E-03	2.860019E+03	1.308328E+00	-9.731422E-01	0/10
GA166	3.182445E+00	5.614899E+00	2.237617E+01	4.552312E+02	8.486314E+01	7.942444E+00	8.170815E-04	5.208071E+03	2.134068E+00	-9.654685E-01	0/10
GA167	7.435274E+01	1.931431E+01	2.888677E+02	3.323666E+02	1.272459E+03	2.970726E+02	1.057470E-01	1.879011E+05	3.613063E+01	-1.914417E-01	0/10
GA168	2.792129E+00	6.381447E+00	2.624417E+01	3.738834E+02	6.947385E+01	4.808534E+00	4.478284E-04	6.097573E+03	1.441024E+00	-9.691687E-01	0/10
GA169	1.032048E+01	1.239107E+01	7.345834E+01	3.066658E+02	1.246028E+02	3.315650E+01	4.661315E-04	1.581285E+04	8.276071E+00	-8.393130E-01	0/10
GA170	4.147974E+00	8.334478E+00	3.839604E+01	3.524642E+02	5.568345E+01	1.861347E+01	4.239657E-04	2.793584E+03	4.129428E+00	-8.976608E-01	0/10
GA171	1.196584E+02	2.013247E+01	3.500507E+02	3.012531E+03	1.585834E+03	4.101461E+02	1.654811E-01	2.748543E+05	4.793718E+01	-1.160969E-01	0/10
GA172	2.827546E+00	6.899099E+00	4.460163E+01	3.065477E+02	6.683438E+01	7.240901E+00	2.230466E-04	2.751920E+03	3.904781E+00	-9.634900E-01	0/10
GA173	8.941518E-01	4.252714E+00	1.709797E+01	3.712538E+02	2.279452E+01	4.487881E+00	1.749796E-03	3.068299E+03	9.761048E-01	-9.915065E-01	0/10
GA174	6.445589E+00	9.911106E+00	8.684003E+01	4.324620E+02	6.826745E+01	2.063098E+01	1.716767E-03	7.722306E+03	4.670370E+00	-9.202331E-01	0/10
GA175	9.553385E+01	1.958794E+01	3.262246E+02	4.631915E+02	1.263907E+03	3.416925E+02	9.781092E-02	1.944476E+05	4.287809E+01	-2.087560E-01	0/10
GA176	3.957350E+00	1.068314E+01	6.449929E+01	3.977782E+02	5.868215E+01	1.106461E+01	1.847050E-03	4.638743E+03	5.598864E+00	-8.738393E-01	0/10
GA177	2.825375E-02	1.630749E+00	1.398673E+00	2.324674E+02	1.096573E+00	1.046272E+00	6.005493E-05	7.718954E+02	3.469343E-01	-9.996048E-01	1/10
GA178	8.681851E-02	2.629206E+00	6.718927E+00	4.303074E+02	1.413398E+00	1.128104E+00	1.044414E-05	7.813586E+02	6.571849E-01	-9.991878E-01	0/10
GA179	4.115768E+01	1.736016E+01	2.470025E+02	1.938784E+02	5.780323E+02	1.439806E+02	1.340386E-02	8.861022E+04	2.469480E+01	-4.540161E-01	0/10
GA180	1.247028E-01	2.822409E+00	5.953340E+00	3.679793E+02	8.902562E-01	1.079291E+00	3.734110E-05	1.274588E+03	6.572469E-01	-9.984249E-01	0/10
GA181	1.789114E+00	5.335677E+00	9.383056E+00	1.772612E+02	1.119313E+01	5.009963E+00	3.036868E-04	1.938477E+03	1.416304E+00	-9.804938E-01	0/10
GA182	2.229316E+00	5.133789E+00	2.870111E+01	4.611725E+02	8.508745E+01	1.864017E+01	3.566375E-03	1.040864E+04	1.924711E+00	-9.643430E-01	0/10
GA183	8.360379E+01	1.960587E+01	3.013368E+02	3.408451E+02	1.260412E+03	3.006192E+02	7.560674E-02	2.041058E+05	3.573272E+01	-1.909985E-01	0/10
GA184	1.710743E+00	8.720015E+00	2.495781E+01	4.694345E+02	7.175501E+01	6.178813E+00	8.094269E-04	2.241772E+03	1.154758E+00	-9.467705E-01	0/10
GA185	6.660222E+00	1.087301E+01	5.917103E+01	3.065267E+02	8.707771E+01	2.658492E+01	5.899897E-04	1.480499E+04	7.379133E+00	-8.646197E-01	0/10
GA186	4.090118E+00	8.719858E+00	4.424480E+01	3.383078E+02	1.126139E+02	1.220152E+01	1.311489E-04	3.041311E+03	4.557665E+00	-9.050766E-01	0/10
GA187	1.067431E+02	1.998345E+01	3.268651E+02	4.378352E+02	1.449250E+03	3.600194E+02	1.087631E-01	2.372815E+05	4.373888E+01	-1.364697E-01	0/10
GA188	2.435103E+00	7.989315E+00	4.218667E+01	3.372551E+02	4.860261E+01	1.123600E+01	4.106725E-03	3.193777E+03	4.338786E+00	-9.521678E-01	0/10
GA189	4.993085E-01	3.958488E+00	1.653910E+01	2.788836E+02	1.767878E+01	1.721541E+00	2.916165E-04	2.642958E+03	4.237802E-01	-9.920795E-01	0/10
GA190	4.362296E+00	9.574924E+00	5.218049E+01	4.275310E+02	6.392400E+01	6.830739E+00	6.851264E-04	7.168339E+03	4.959224E+00	-9.343118E-01	0/10
GA191	8.608140E+01	1.948341E+01	3.002368E+02	3.603668E+02	1.168781E+03	3.003984E+02	6.906208E-02	2.031919E+05	3.801638E+01	-2.054451E-01	0/10
GA192	3.227291E+00	9.202958E+00	6.912940E+01	3.956645E+02	3.362175E+01	9.111892E+00	9.736315E-04	3.526592E+03	4.462754E+00	-9.035132E-01	0/10
GA193	9.754292E-02	3.868140E+00	1.032855E+02	4.793231E+05	2.163285E+00	1.536290E+00	6.477458E-07	1.747630E+02	6.611043E-01	-9.962038E-01	0/10
GA194	5.518539E-01	4.855280E+00	5.040222E+01	1.490659E+02	4.821392E+00	3.067999E+00	9.418934E-06	1.095770E+03	4.339125E+00	-4.421160E-01	0/10
GA195	8.865185E-01	5.845484E+00	1.459024E+02	3.444296E+08	1.847657E+01	4.619001E+00	7.151817E-06	2.466504E+03	4.006098E+00	-9.778662E-01	0/10
GA196	1.202354E-01	2.771613E+00	3.979993E+01	2.644491E+02	1.872321E+00	1.463326E+00	6.164227E-06	2.973940E+02	3.246087E-01	-9.968608E-01	0/10
GA197	1.480132E-01	2.462673E+00	1.615301E+01	6.314044E+06	3.578865E+00	1.077924E+00	3.125948E-06	1.174746E+02	3.209202E+00	-9.960441E-01	1/10
GA198	6.819452E-01	3.123644E+00	5.210053E+01	2.408830E+02	7.923029E+00	2.649097E+00	7.995321E-05	6.898166E+02	4.479334E+00	-4.626798E-01	0/10
GA199	5.203820E-01	4.291042E+00	5.094550E+01	4.703642E+08	5.884029E+00	1.691955E+00	9.012504E-07	3.709097E+02	5.178633E+00	-9.900119E-01	0/10
GA200	2.333265E-01	2.168415E+00	1.785102E+01	2.923191E+02	2.209900E+00	1.609645E+00	2.324587E-06	2.215627E+02	3.181747E+00	-9.958468E-01	0/10

GA201	1.865610E+00	6.782334E+00	1.463284E+02	1.247394E+02	2.293147E+01	6.789549E+00	1.477298E-05	3.888477E+03	1.405475E+01	-9.674444E-01	0/10
GA202	2.650834E+00	7.911812E+00	6.216866E+01	3.046312E+02	4.306648E+01	1.087868E+01	1.413466E-04	4.727909E+03	1.008818E+01	-4.449389E-01	0/10
GA203	3.587086E+00	8.427190E+00	1.744491E+02	2.765201E+07	5.374446E+01	1.239160E+01	2.841566E-05	6.397878E+03	1.788221E+01	-9.433910E-01	0/10
GA204	1.054512E+00	5.632889E+00	6.567161E+01	3.067110E+02	1.778493E+01	4.502099E+00	3.358832E-05	1.908121E+03	2.319663E+00	-9.817680E-01	0/10
GA205	1.926127E-01	3.690739E+00	1.140216E+02	3.760595E+07	8.118649E-01	1.543219E+00	4.838166E-06	3.362816E+02	2.368796E+00	-9.954931E-01	0/10
GA206	9.765531E-01	5.482953E+00	5.065271E+01	2.419100E+02	1.115783E+01	3.528103E+00	1.142283E-04	3.134407E+03	9.038561E+00	-4.937101E-01	0/10
GA207	1.130580E+00	6.858906E+00	1.525904E+02	1.426130E+08	1.860743E+01	5.378537E+00	1.479806E-05	3.521951E+03	8.450926E+00	-9.715251E-01	0/10
GA208	2.161016E-01	3.382879E+00	4.817871E+01	3.044332E+02	4.870500E+00	1.512935E+00	2.209846E-05	8.754433E+02	1.068300E+00	-9.975294E-01	0/10
GA209	7.222290E-01	5.290053E+00	1.320668E+02	7.257107E+01	8.177801E+00	3.584218E+00	2.525922E-05	1.101081E+03	1.516171E+00	-9.866130E-01	0/10
GA210	1.693292E+00	6.657935E+00	9.022282E+01	3.236444E+02	1.808784E+01	5.101167E+00	2.722815E-05	2.994075E+03	8.513092E+00	-5.465800E-01	0/10
GA211	1.000692E+00	5.955574E+00	1.528316E+02	1.223927E+09	1.890224E+01	4.893682E+00	9.442063E-06	2.981240E+03	4.368107E+00	-9.776413E-01	0/10
GA212	6.348293E-01	5.598227E+00	8.666423E+01	3.812645E+02	1.096670E+01	3.833442E+00	1.816183E-06	2.215204E+03	2.293204E+00	-9.861287E-01	0/10
GA213	1.142069E+00	4.683596E+00	5.226694E+01	7.532451E+08	1.530330E+01	9.278441E+00	4.916793E-04	8.519029E+02	8.465503E+00	-9.863761E-01	0/10
GA214	1.442190E+00	5.897314E+00	1.160742E+02	1.129796E+08	1.781889E+01	5.199149E+00	4.896709E-06	4.085658E+03	1.347944E+01	-5.386073E-01	0/10
GA215	4.084801E-01	3.749967E+00	9.408363E+01	1.961979E+09	3.588410E+00	3.134028E+00	6.827470E-06	8.308381E+02	6.884284E+00	-9.944543E-01	0/10
GA216	1.304065E+00	4.438817E+00	8.432038E+01	7.179305E+07	1.451904E+01	4.630428E+00	4.674515E-06	2.082357E+03	1.454074E+01	-9.893720E-01	0/10
GA217	2.263963E+00	8.156971E+00	1.523509E+02	1.894311E+02	3.911314E+01	1.237328E+01	1.098683E-05	4.603439E+03	1.429390E+01	-9.492501E-01	0/10
GA218	3.166835E+00	8.771824E+00	8.689252E+01	3.481049E+02	5.501501E+01	1.393740E+01	2.201576E-04	6.540917E+03	1.015365E+01	-5.275461E-01	0/10
GA219	3.812113E+00	9.024548E+00	1.728444E+02	8.829816E+08	5.937070E+01	1.509383E+01	7.720917E-05	8.957544E+03	2.177985E+01	-9.270798E-01	0/10
GA220	2.601738E+00	7.578479E+00	7.853504E+01	3.314637E+02	3.005195E+01	9.654648E+00	8.308917E-05	4.546894E+03	2.633431E+00	-9.631663E-01	0/10
GA221	2.252498E+00	7.411840E+00	1.471682E+02	1.034631E+08	2.210403E+01	8.768563E+00	2.381761E-04	5.804235E+03	9.116934E+00	-9.777133E-01	0/10
GA222	4.298320E+00	8.684893E+00	1.053670E+02	1.946729E+08	6.961223E+01	1.443949E+01	1.351535E-03	9.666000E+03	2.150997E+01	-5.146584E-01	0/10
GA223	1.460702E+00	6.466097E+00	1.772739E+02	1.324428E+09	1.828290E+01	5.010847E+00	3.497046E-05	2.966844E+03	1.014410E+01	-9.776764E-01	0/10
GA224	2.643965E+00	7.765834E+00	8.782124E+01	7.262680E+07	3.278408E+01	8.830487E+00	2.092472E-04	5.519530E+03	2.588993E+01	-9.734364E-01	0/10
GA225	2.130677E-01	3.653304E+00	1.057539E+01	1.703417E+02	2.749833E+00	1.300294E+00	7.249035E-05	8.244680E+02	3.740045E-01	-9.934116E-01	0/10
GA226	1.801511E+00	7.085824E+00	3.455957E+01	3.231337E+02	1.304598E+01	4.136255E+00	1.439157E-02	3.776352E+03	2.918792E+00	-2.102774E-01	0/10
GA227	8.324216E+01	1.894421E+01	3.375717E+02	1.104182E+02	1.022657E+03	2.778692E+02	7.568992E-02	1.643384E+05	1.219916E+01	-2.362662E-01	0/10
GA228	3.244146E-01	3.889264E+00	1.389238E+01	4.283885E+02	4.582602E+00	2.478798E+00	8.599822E-05	2.124970E+03	1.494888E+00	-9.894529E-01	0/10
GA229	2.152023E+00	5.132064E+00	7.812840E+00	3.051396E+02	7.647329E+00	7.582252E+00	3.258079E-04	4.795229E+02	2.570359E+00	-9.942029E-01	0/10
GA230	1.676933E+00	5.552543E+00	1.948096E+01	2.915718E+02	5.453123E+01	1.379773E+01	3.799200E-03	5.132882E+03	4.260902E+00	-1.617891E-01	0/10
GA231	1.184026E+02	2.014940E+01	3.486036E+02	2.967084E+02	1.262761E+03	4.438006E+02	1.545783E-01	2.392147E+05	3.810122E+01	-7.332915E-02	0/10
GA232	1.053795E+00	2.834138E+00	1.997044E+01	5.428877E+02	3.146619E+01	6.932045E+00	1.944383E-03	1.848462E+03	2.157494E+01	-9.886020E-01	0/10
GA233	8.953751E+00	1.157317E+01	7.544765E+01	3.066712E+02	1.355781E+02	3.459900E+01	4.091366E-04	1.700649E+04	1.656922E+01	-8.402134E-01	0/10
GA234	5.415145E+00	1.267939E+01	7.026304E+01	4.055373E+02	9.938768E+01	2.162330E+01	1.161174E-04	1.099573E+04	9.493153E+00	-1.959154E-01	0/10
GA235	1.338176E+02	2.028969E+01	3.712100E+02	4.644397E+02	1.685248E+03	4.261654E+02	1.980433E-01	2.922861E+05	4.826176E+01	-6.904765E-02	0/10
GA236	3.118764E+00	8.585871E+00	4.722360E+01	2.881269E+02	4.652868E+01	1.271619E+01	1.056086E-03	5.541236E+03	3.742242E+00	-9.281686E-01	0/10
GA237	7.632912E-01	5.610506E+00	1.916542E+01	4.407833E+02	1.564116E+01	4.619114E+00	2.179690E-03	1.873183E+03	2.114964E+00	-9.837786E-01	0/10
GA238	4.101627E+00	1.005301E+01	5.435053E+01	4.313963E+02	1.003285E+02	1.399853E+01	3.491819E-03	8.159035E+03	9.297494E+00	-2.028362E-01	0/10
GA239	1.236145E+02	2.026972E+01	3.940158E+02	3.929574E+02	1.743101E+03	4.308385E+02	1.868520E-01	2.936246E+05	4.257845E+01	-5.064421E-02	0/10
GA240	1.039220E+00	5.779590E+00	2.271767E+01	4.760206E+02	2.352951E+01	5.585491E+00	3.167762E-03	4.361894E+03	4.213593E+01	-9.848688E-01	0/10
GA241	1.244025E-01	2.888853E+00	9.292877E+00	1.988681E+02	2.505319E+00	1.315975E+00	7.555796E-05	5.779504E+02	1.647999E-01	-9.974097E-01	1/10
GA242	1.088989E+00	7.378538E+00	3.561917E+01	3.726878E+02	1.647347E+01	4.487942E+00	3.862084E-05	2.564792E+03	1.835537E+00	-1.935460E-01	0/10
GA243	5.929849E+01	1.863222E+01	3.007625E+02	1.289370E+02	9.262551E+02	2.261782E+02	2.067633E-02	1.317287E+05	7.415449E+00	-2.383348E-01	0/10
GA244	2.266389E-01	3.898845E+00	1.532434E+01	5.122890E+02	4.375452E+00	2.599541E+00	4.889120E-05	1.039594E+03	1.686423E+00	-9.959155E-01	0/10
GA245	1.050570E+00	4.814291E+00	7.090141E+00	2.744406E+02	2.641966E+01	7.487044E+00	4.856830E-04	1.048405E+03	2.983033E+00	-9.823099E-01	0/10
GA246	3.069392E+00	6.568070E+00	2.151584E+01	2.934998E+02	6.577929E+01	8.107677E+00	2.108855E-03	4.038395E+03	3.108920E+00	-1.820049E-01	0/10
GA247	1.255551E+02	2.025752E+01	3.649014E+02	3.616840E+02	1.562442E+03	4.289190E+02	1.603757E-01	2.689955E+05	4.974951E+01	-6.515820E-02	0/10
GA248	9.364565E-01	3.434076E+00	1.133517E+01	5.323639E+02	2.546858E+01	8.761083E+00	2.054935E-03	1.352521E+03	9.179110E+00	-9.782027E-01	0/10
GA249	6.217227E+00	1.053780E+01	5.518020E+01	3.351467E+02	8.470821E+01	2.334704E+01	6.369885E-04	1.304393E+04	1.242516E+01	-8.748696E-01	0/10
GA250	6.372596E+00	1.111078E+01	7.434850E+01	3.787502E+02	9.604580E+01	2.118313E+01	2.235299E-04	1.021567E+04	8.583178E+00	-1.866708E-01	0/10
GA251	1.275509E+02	2.003380E+01	3.645038E+02	4.805977E+02	1.663438E+03	4.538372E+02	1.547061E-01	2.826073E+05	5.140055E+01	-8.802521E-02	0/10
GA252	2.784584E+00	9.102195E+00	4.614544E+01	3.446776E+02	5.081873E+01	1.341325E+01	9.998591E-05	4.584832E+03	4.271858E+00	-9.509025E-01	0/10
GA253	4.982698E-01	3.365272E+00	9.444996E+00	4.599576E+02	1.090161E+01	2.295410E+00	9.051936E-04	8.993676E+02	7.134620E-01	-9.952256E-01	0/10
GA254	5.416116E+00	9.782558E+00	4.206016E+01	4.722020E+02	5.146372E+01	7.165645E+00	1.224820E-02	6.975624E+03	6.758569E+00	-1.583063E-01	0/10
GA255	1.231696E+02	2.005883E+01	3.730448E+02	4.321565E+02	1.529268E+03	4.093167E+02	1.652035E-01	2.781704E+05	6.272273E+01	-5.158480E-02	0/10
GA256	1.025227E+00	5.150329E+00	1.521421E+01	6.015695E+02	2.243317E+01	5.020202E+00	2.864065E-03	3.261725E+03	4.106552E+01	-9.846822E-01	0/10
Min	1.824120E-02	5.143914E-01	1.250391E+00	2.969400E+01	4.315668E-01	8.608545E-01	5.400799E-09	1.174746E+02	1.647999E-01	-9.996048E-01	
Max	1.338176E+02	2.028969E+01	3.940158E+02	3.305935E+09	1.842418E+03	4.538372E+02	3.329094E-01	3.061400E+05	6.272273E+01	-5.064421E-02	

GA Results on Average Time

GA Model	Average Time										Performance
	Sphere	Ackley	Rastrigin	Zakharov	Axis-Parallel	Griewank	Sum of Different Powers	Rotated Hyper-Ellipsoid	Schwefel 2.22	Exponential	
GA001	0.024100	0.028700	0.028900	0.024800	0.026700	0.030500	0.023200	0.050100	0.019800	0.023500	0/10
GA002	0.025300	0.031600	0.029300	0.025600	0.024800	0.030600	0.025100	0.052800	0.023300	0.025600	0/10
GA003	0.025000	0.031600	0.031100	0.025900	0.024600	0.031600	0.025300	0.052100	0.022500	0.024400	0/10
GA004	0.025200	0.032000	0.030900	0.025700	0.028100	0.030300	0.028200	0.050900	0.023300	0.024700	0/10
GA005	0.025600	0.032400	0.030500	0.026300	0.025500	0.031100	0.025800	0.053900	0.022500	0.025400	0/10
GA006	0.024400	0.029500	0.031800	0.025300	0.024300	0.028400	0.023900	0.049700	0.022100	0.025000	0/10
GA007	0.024300	0.031300	0.030500	0.025500	0.024000	0.029700	0.024100	0.053200	0.022200	0.023700	0/10
GA008	0.024000	0.030600	0.030500	0.025600	0.024400	0.029800	0.026900	0.059900	0.023100	0.024100	0/10
GA009	0.026500	0.031600	0.029800	0.024700	0.023000	0.028600	0.023700	0.049500	0.019700	0.023100	0/10
GA010	0.022300	0.028800	0.026700	0.022700	0.022400	0.026900	0.022700	0.051300	0.019800	0.023900	0/10
GA011	0.023200	0.028400	0.029900	0.023100	0.022900	0.027400	0.023900	0.050800	0.021600	0.023200	0/10
GA012	0.021900	0.029200	0.028100	0.022900	0.022600	0.027200	0.022800	0.050400	0.020200	0.022300	0/10
GA013	0.021100	0.027500	0.028400	0.023200	0.022900	0.026500	0.021800	0.050100	0.019500	0.022900	0/10
GA014	0.021900	0.028800	0.027200	0.021500	0.021900	0.026600	0.021700	0.051300	0.020000	0.021600	0/10
GA015	0.021600	0.030200	0.028900	0.022800	0.021800	0.027200	0.022500	0.051300	0.020500	0.022700	0/10
GA016	0.021900	0.028700	0.027800	0.022800	0.022300	0.026600	0.021800	0.050300	0.020400	0.022900	0/10
GA017	0.022500	0.027900	0.027000	0.020800	0.021600	0.025300	0.021000	0.048600	0.018700	0.021200	0/10
GA018	0.020800	0.027000	0.026200	0.021100	0.020400	0.025100	0.021000	0.051100	0.018200	0.020500	0/10
GA019	0.020100	0.027800	0.027600	0.022900	0.021600	0.026200	0.022000	0.049400	0.018300	0.021100	0/10
GA020	0.022400	0.027800	0.026900	0.021400	0.021000	0.026400	0.021500	0.049600	0.018600	0.021700	0/10
GA021	0.021400	0.028500	0.028800	0.022000	0.022200	0.026500	0.021900	0.047800	0.018700	0.021100	0/10
GA022	0.021300	0.026600	0.027100	0.021100	0.021100	0.024900	0.020900	0.048100	0.018300	0.020900	0/10
GA023	0.021700	0.028300	0.027900	0.021800	0.021100	0.026000	0.022500	0.049300	0.019300	0.021900	0/10
GA024	0.021200	0.027700	0.028000	0.021900	0.020900	0.024900	0.020900	0.054600	0.017800	0.021100	0/10
GA025	0.022500	0.027000	0.027700	0.023000	0.021200	0.028000	0.020700	0.056300	0.019100	0.022700	0/10
GA026	0.020000	0.026500	0.027500	0.022100	0.021000	0.026600	0.021700	0.056100	0.018500	0.021000	0/10
GA027	0.021000	0.027400	0.029600	0.022200	0.022200	0.026800	0.021400	0.053400	0.019300	0.020800	0/10
GA028	0.019900	0.028500	0.028600	0.022700	0.020700	0.025100	0.022200	0.060200	0.018000	0.022200	0/10
GA029	0.022100	0.029200	0.027000	0.022200	0.021300	0.025600	0.022500	0.058600	0.019000	0.020800	0/10
GA030	0.021100	0.027500	0.027700	0.021500	0.020600	0.026700	0.020200	0.054500	0.018600	0.021700	0/10
GA031	0.020000	0.029100	0.027800	0.022200	0.021600	0.026500	0.021300	0.055600	0.018300	0.020000	0/10
GA032	0.020400	0.026400	0.028000	0.022500	0.021600	0.027000	0.021600	0.056800	0.017800	0.020600	0/10
GA033	0.022100	0.028800	0.029300	0.024600	0.021700	0.028100	0.022500	0.055300	0.021600	0.024600	0/10
GA034	0.022300	0.030100	0.028000	0.023100	0.022000	0.027000	0.022500	0.053800	0.019200	0.022600	0/10
GA035	0.023100	0.028100	0.029300	0.024200	0.023500	0.028800	0.023400	0.056000	0.019700	0.022300	0/10
GA036	0.021500	0.028300	0.029400	0.023400	0.024200	0.027300	0.022200	0.060700	0.020600	0.023700	0/10
GA037	0.023700	0.029400	0.029400	0.025000	0.023000	0.027700	0.023400	0.061000	0.019900	0.024400	0/10
GA038	0.022400	0.030200	0.029400	0.022500	0.021800	0.028900	0.022700	0.055800	0.019100	0.023000	0/10
GA039	0.022500	0.029200	0.030600	0.022900	0.022900	0.029500	0.023400	0.059700	0.020400	0.023700	0/10
GA040	0.021400	0.029800	0.029000	0.022600	0.021500	0.026200	0.022100	0.054600	0.020000	0.021100	0/10
GA041	0.024100	0.030200	0.029100	0.023700	0.022400	0.029800	0.024900	0.054300	0.021600	0.024000	0/10
GA042	0.022200	0.029100	0.028400	0.023200	0.023800	0.028200	0.024000	0.054300	0.021800	0.022700	0/10
GA043	0.023700	0.051100	0.030500	0.024200	0.023200	0.028000	0.024200	0.052400	0.020800	0.022600	0/10
GA044	0.022900	0.030000	0.028700	0.022700	0.022300	0.027000	0.022800	0.062900	0.021800	0.022900	0/10
GA045	0.022800	0.032900	0.030200	0.023400	0.022900	0.026900	0.022500	0.055000	0.024000	0.023200	0/10
GA046	0.022400	0.029300	0.028600	0.023100	0.022200	0.027000	0.022300	0.050200	0.020800	0.023300	0/10
GA047	0.023200	0.030000	0.028600	0.024400	0.023100	0.027400	0.023300	0.052800	0.021000	0.023200	0/10
GA048	0.023200	0.029300	0.028400	0.022500	0.022100	0.027200	0.021600	0.050300	0.020000	0.022100	0/10
GA049	0.023200	0.029800	0.027800	0.023300	0.022000	0.026800	0.023000	0.048500	0.020300	0.022200	0/10
GA050	0.021900	0.028600	0.026100	0.021500	0.021200	0.025500	0.021000	0.049200	0.019300	0.020800	0/10
GA051	0.022600	0.028900	0.029000	0.022200	0.023100	0.027800	0.021100	0.069600	0.026200	0.022000	0/10
GA052	0.022500	0.028000	0.027000	0.022100	0.021000	0.026000	0.022400	0.049500	0.020100	0.022000	0/10
GA053	0.022300	0.027900	0.029000	0.023500	0.022800	0.026900	0.022100	0.049500	0.019800	0.022100	0/10
GA054	0.021200	0.028400	0.027100	0.022600	0.022300	0.026300	0.021700	0.047000	0.019200	0.022500	0/10
GA055	0.022800	0.029200	0.028800	0.023100	0.021800	0.026900	0.022600	0.053100	0.019900	0.022500	0/10
GA056	0.021400	0.027300	0.026900	0.022500	0.021500	0.026100	0.022100	0.051500	0.019800	0.022000	0/10
GA057	0.022000	0.029700	0.029800	0.023000	0.022800	0.027400	0.023100	0.048100	0.020200	0.021400	0/10
GA058	0.021100	0.028200	0.028500	0.023800	0.021800	0.027200	0.022800	0.049700	0.020400	0.021800	0/10
GA059	0.022700	0.029700	0.028800	0.023600	0.022400	0.027700	0.022900	0.049300	0.020800	0.022800	0/10
GA060	0.022500	0.027900	0.028100	0.022200	0.022700	0.027000	0.022700	0.048400	0.020600	0.022700	0/10
GA061	0.022200	0.028300	0.027500	0.022500	0.023100	0.026700	0.022700	0.047500	0.019600	0.022900	0/10
GA062	0.022000	0.028800	0.027600	0.022100	0.021100	0.026100	0.020900	0.048900	0.019600	0.021900	0/10
GA063	0.023400	0.029400	0.028400	0.023900	0.022500	0.026900	0.021800	0.050100	0.020400	0.023400	0/10
GA064	0.022500	0.027800	0.027500	0.022700	0.021600	0.027600	0.022200	0.050000	0.019000	0.022600	0/10
GA065	0.029000	0.034500	0.035300	0.030100	0.028500	0.032400	0.029100	0.055600	0.026800	0.029700	0/10
GA066	0.029000	0.034700	0.034000	0.028600	0.028200	0.032300	0.028400	0.054700	0.026400	0.028800	0/10
GA067	0.028700	0.034500	0.034100	0.029800	0.028900	0.033500	0.028900	0.054300	0.026500	0.028300	0/10
GA068	0.027800	0.036100	0.035700	0.028100	0.027800	0.032200	0.028700	0.053800	0.026500	0.028100	0/10
GA069	0.027900	0.034800	0.033400	0.030000	0.028500	0.032500	0.029700	0.057100	0.027200	0.030800	0/10
GA070	0.028400	0.036200	0.033400	0.027700	0.026800	0.032400	0.029600	0.057200	0.027200	0.028900	0/10
GA071	0.028600	0.036200	0.034500	0.029400	0.027900	0.032400	0.029400	0.059100	0.027100	0.029500	0/10
GA072	0.029200	0.036900	0.039800	0.029700	0.030600	0.035400	0.032700	0.065600	0.026000	0.028800	0/10
GA073	0.035200	0.038500	0.040600	0.034700	0.034800	0.044700	0.035100	0.065400	0.027400	0.030800	0/10
GA074	0.029200	0.037500	0.034000	0.028800	0.028900	0.033900	0.027400	0.054300	0.026000	0.029400	0/10
GA075	0.031000	0.036400	0.039300	0.029500	0.031300	0.035300	0.030400	0.063700	0.026500	0.031300	0/10
GA076	0.030200	0.036800	0.035400	0.030700	0.029700	0.036900	0.029400	0.059000	0.027000	0.028500	0/10
GA077	0.029500	0.035400	0.036100	0.030900	0.031100	0.033500	0.029000	0.055000	0.027200	0.028700	0/10
GA078	0.029000	0.035600	0.035700	0.029000	0.028400	0.032500	0.028800	0.054500	0.026700	0.030300	0/10
GA079	0.029700	0.037300	0.034500	0.029900	0.029400	0.032900	0.030200	0.056400	0.026500	0.029700	0/10
GA080	0.029600	0.037400	0.035900	0.028800	0.028500	0.033200	0.028700	0.055500	0.027200	0.029700	0/10
GA081	0.030300	0.034700	0.033700	0.029400	0.027800	0.032800	0.027000	0.054300	0.026000	0.029100	0/10
GA082	0.026800	0.035000	0.033200	0.028200	0.027200	0.032200	0.027300	0.056400	0.026400	0.029000	0/10
GA083	0.028200	0.034100	0.032700	0.029800	0.027200	0.032600	0.027800	0.054700	0.026300	0.028600	0/10
GA084	0.028300	0.035100	0.033100	0.028300	0.028300	0.031000	0.028700	0.058100	0.024900	0.028300	0/10
GA085	0.028100	0.033600	0.033300	0.027000	0.027400	0.031300	0.028300	0.054500	0.027800	0.028700	0/10
GA086	0.029100	0.033000	0.032400	0.027700	0.026500	0.031300	0.027700	0.056900	0.026400	0.027900	0/10
GA087	0.028100	0.033800	0.033800	0.028900	0.027700	0.032300	0.029800	0.057500	0.026700	0.027900	0/10
GA088	0.026700	0.033800	0.033100	0.027500	0.028800	0.031400	0.028100	0.058100	0.023900	0.028000	0/10
GA089	0.028000	0.035200	0.033900	0.028900	0.029900	0.032200	0.029200	0.055300	0.025800	0.028200	0/10
GA090	0.027600	0.034100	0.032900	0.029400	0.026900	0.033900	0.027500	0.052500	0.025400	0.027900	0/10
GA091	0.027600	0.034000	0.036100	0.029700	0.029800	0.032400	0.027500	0.055400	0.025200	0.027600	0/10
GA092	0.027100	0.036200	0.033200	0.029700	0.028000	0.031100	0.027200	0.053500	0.025300	0.027200	0/10
GA093	0.028900	0.035300	0.034300	0.028100	0.028400	0.031900	0.028600	0.053900	0.025600	0.029000	0/10
GA094	0.027300	0.035500	0.032100	0.027800	0.027100	0.031000	0.027300	0.053900	0.023700	0.028200	0/10
GA095	0.028600	0.034400	0.034700	0.027600	0.027000	0.032300	0.027000	0.052400	0.024900	0.030000	0/10

GA096	0.027200	0.035400	0.033500	0.027500	0.027100	0.032100	0.026700	0.053700	0.025300	0.026700	0/10
GA097	0.029900	0.036500	0.036100	0.029700	0.029900	0.033700	0.028200	0.055400	0.028300	0.029900	0/10
GA098	0.030500	0.035900	0.033900	0.030000	0.029800	0.033600	0.030100	0.053600	0.027100	0.030000	0/10
GA099	0.028800	0.037600	0.035400	0.030600	0.028700	0.034200	0.028900	0.055300	0.027700	0.030000	0/10
GA100	0.028400	0.037200	0.035100	0.029100	0.028400	0.033200	0.029200	0.056600	0.027000	0.029400	0/10
GA101	0.030900	0.034500	0.034200	0.030200	0.029900	0.033200	0.029800	0.056300	0.029100	0.029700	0/10
GA102	0.030200	0.035000	0.034000	0.029500	0.028500	0.033500	0.027800	0.057200	0.027200	0.028600	0/10
GA103	0.028900	0.036000	0.035300	0.029600	0.029600	0.034100	0.030300	0.061200	0.027400	0.028900	0/10
GA104	0.028200	0.035400	0.033600	0.028600	0.029200	0.034900	0.028700	0.057600	0.027400	0.029400	0/10
GA105	0.030400	0.036200	0.034800	0.031000	0.030600	0.036300	0.029300	0.057000	0.027500	0.029700	0/10
GA106	0.029700	0.036900	0.036100	0.030200	0.028200	0.033800	0.029000	0.056700	0.027300	0.029100	0/10
GA107	0.031000	0.036400	0.037000	0.030500	0.029500	0.035000	0.029900	0.056600	0.027000	0.030400	0/10
GA108	0.031100	0.036200	0.036300	0.029700	0.029500	0.033700	0.029400	0.057200	0.027100	0.029500	0/10
GA109	0.029400	0.037300	0.035100	0.030900	0.029400	0.033800	0.030500	0.054800	0.027600	0.030200	0/10
GA110	0.029000	0.037000	0.034000	0.029800	0.030000	0.033500	0.028100	0.055300	0.026900	0.029500	0/10
GA111	0.029100	0.038200	0.035200	0.030100	0.029200	0.033800	0.028200	0.056300	0.027800	0.029300	0/10
GA112	0.029900	0.034900	0.034800	0.030000	0.028700	0.032600	0.028800	0.054200	0.028500	0.028800	0/10
GA113	0.030800	0.035100	0.033400	0.030100	0.028800	0.032700	0.028500	0.057500	0.027600	0.029500	0/10
GA114	0.027800	0.033800	0.033600	0.028100	0.027300	0.032400	0.027900	0.061100	0.026300	0.027200	0/10
GA115	0.028100	0.034000	0.034200	0.029700	0.028100	0.035000	0.028100	0.059600	0.026600	0.028500	0/10
GA116	0.027000	0.034200	0.033200	0.028100	0.028600	0.031700	0.028500	0.053600	0.025600	0.028700	0/10
GA117	0.027500	0.033700	0.034200	0.031200	0.030400	0.034800	0.029500	0.054600	0.025600	0.028700	0/10
GA118	0.028100	0.036200	0.034100	0.030500	0.028400	0.033600	0.028300	0.055000	0.025300	0.027800	0/10
GA119	0.030400	0.036800	0.035600	0.029800	0.029200	0.033400	0.027700	0.055400	0.025800	0.029500	0/10
GA120	0.030200	0.035600	0.036400	0.030300	0.027700	0.032600	0.027300	0.054400	0.025500	0.029200	0/10
GA121	0.029800	0.038800	0.034800	0.029500	0.029700	0.035600	0.029000	0.055600	0.027000	0.031600	0/10
GA122	0.028800	0.035600	0.034500	0.028900	0.027400	0.032000	0.027800	0.053600	0.025900	0.027500	0/10
GA123	0.030800	0.034900	0.036100	0.029800	0.029000	0.033400	0.028900	0.054000	0.028300	0.030000	0/10
GA124	0.028000	0.036100	0.034400	0.027800	0.027500	0.032700	0.028100	0.053500	0.025500	0.028400	0/10
GA125	0.027700	0.036200	0.036400	0.029600	0.027900	0.032800	0.028400	0.053900	0.027500	0.028500	0/10
GA126	0.030900	0.034400	0.033600	0.025600	0.027600	0.031500	0.028900	0.060400	0.027000	0.028500	0/10
GA127	0.030100	0.037200	0.034600	0.030200	0.029800	0.033700	0.028300	0.060300	0.026500	0.029500	0/10
GA128	0.026300	0.033400	0.033500	0.028500	0.027400	0.032300	0.026800	0.056200	0.026900	0.028500	0/10
GA129	0.022900	0.029200	0.028000	0.022800	0.022000	0.026800	0.022400	0.048400	0.020800	0.022200	0/10
GA130	0.022300	0.028700	0.027600	0.022400	0.020800	0.025700	0.021900	0.048600	0.020100	0.022800	0/10
GA131	0.021600	0.027800	0.028900	0.022800	0.022100	0.026700	0.022100	0.048800	0.020000	0.021500	0/10
GA132	0.022600	0.028100	0.027800	0.023100	0.022100	0.026400	0.022300	0.048200	0.019900	0.022500	0/10
GA133	0.022600	0.030000	0.027100	0.023200	0.025000	0.025300	0.022400	0.050200	0.020200	0.022000	0/10
GA134	0.021700	0.028100	0.028900	0.022800	0.022900	0.026800	0.022600	0.048100	0.019600	0.022300	0/10
GA135	0.022100	0.028000	0.027600	0.023500	0.022400	0.027000	0.022300	0.048900	0.019900	0.022700	0/10
GA136	0.022000	0.027100	0.026900	0.022800	0.022200	0.025700	0.022600	0.049200	0.019800	0.021000	0/10
GA137	0.020800	0.028000	0.027500	0.023200	0.022500	0.029200	0.021800	0.052100	0.020600	0.021400	0/10
GA138	0.021100	0.028200	0.026500	0.023400	0.021600	0.026900	0.022800	0.054300	0.019700	0.020800	0/10
GA139	0.022400	0.028200	0.028400	0.022000	0.021500	0.026000	0.022500	0.052800	0.022000	0.022700	0/10
GA140	0.020900	0.027300	0.027000	0.022000	0.021900	0.026000	0.022100	0.054000	0.019500	0.021600	0/10
GA141	0.022000	0.027900	0.028000	0.022100	0.021300	0.026200	0.021800	0.049700	0.019700	0.022000	0/10
GA142	0.021700	0.029300	0.027200	0.021700	0.021700	0.026100	0.021700	0.047400	0.019600	0.022400	0/10
GA143	0.021500	0.028900	0.028100	0.022700	0.022000	0.026200	0.021600	0.049000	0.020000	0.021700	0/10
GA144	0.023200	0.027100	0.027000	0.022400	0.021800	0.026800	0.021700	0.047900	0.020500	0.022100	0/10
GA145	0.021200	0.028300	0.026800	0.021500	0.021000	0.025000	0.020500	0.048300	0.019300	0.020900	0/10
GA146	0.020200	0.027300	0.026500	0.021400	0.020400	0.026600	0.020000	0.048200	0.018600	0.021500	0/10
GA147	0.020100	0.026900	0.026200	0.020800	0.021300	0.025400	0.019800	0.051600	0.018000	0.019900	1/10
GA148	0.020700	0.026900	0.025100	0.020400	0.020200	0.024400	0.020800	0.047000	0.018000	0.020300	3/10
GA149	0.019900	0.027000	0.025900	0.021000	0.019800	0.025200	0.021000	0.048900	0.018900	0.020500	1/10
GA150	0.020500	0.026200	0.026100	0.021000	0.020200	0.024400	0.020600	0.046100	0.018300	0.021100	3/10
GA151	0.020600	0.028400	0.025800	0.021000	0.020300	0.025000	0.020500	0.047700	0.018000	0.020900	0/10
GA152	0.021400	0.028200	0.026900	0.022000	0.020800	0.025500	0.020200	0.055200	0.017600	0.020100	1/10
GA153	0.019700	0.027000	0.028300	0.022700	0.021600	0.025300	0.021400	0.049300	0.018700	0.020600	1/10
GA154	0.020200	0.027100	0.026100	0.022100	0.021100	0.025300	0.021500	0.050600	0.019100	0.021800	0/10
GA155	0.021500	0.028000	0.030400	0.023000	0.023000	0.027400	0.021400	0.056800	0.018800	0.021000	0/10
GA156	0.020100	0.027700	0.026300	0.022600	0.021000	0.026700	0.021100	0.055500	0.018700	0.021800	0/10
GA157	0.020200	0.027600	0.027100	0.022300	0.020900	0.025300	0.021500	0.050600	0.018900	0.020500	0/10
GA158	0.020200	0.027400	0.027100	0.021600	0.020800	0.024800	0.020900	0.051600	0.018400	0.020800	0/10
GA159	0.020600	0.028200	0.027700	0.021000	0.021700	0.024900	0.020100	0.049400	0.019100	0.020300	0/10
GA160	0.020400	0.028000	0.026600	0.023800	0.020900	0.025800	0.020200	0.046400	0.018900	0.019800	1/10
GA161	0.022600	0.028800	0.029800	0.022900	0.022900	0.028200	0.022200	0.050700	0.020300	0.022100	0/10
GA162	0.022500	0.027900	0.028000	0.024000	0.022600	0.027600	0.022400	0.052300	0.022100	0.022600	0/10
GA163	0.022200	0.029500	0.029400	0.024000	0.022400	0.027700	0.023300	0.050000	0.020200	0.022300	0/10
GA164	0.022100	0.027800	0.027500	0.023300	0.023600	0.027200	0.022800	0.051700	0.020400	0.022000	0/10
GA165	0.022100	0.028400	0.028100	0.023500	0.022700	0.028100	0.022600	0.051800	0.020200	0.022600	0/10
GA166	0.022000	0.028500	0.027700	0.023300	0.022100	0.027500	0.022500	0.052500	0.020400	0.022000	0/10
GA167	0.022900	0.029000	0.028800	0.023200	0.022500	0.026500	0.023500	0.052600	0.020800	0.021600	0/10
GA168	0.022700	0.027300	0.027000	0.022900	0.022300	0.026500	0.022700	0.052100	0.020400	0.023500	0/10
GA169	0.022600	0.028800	0.027800	0.023000	0.022700	0.027100	0.022900	0.055800	0.020600	0.023800	0/10
GA170	0.022300	0.029100	0.028000	0.023200	0.022000	0.027500	0.022500	0.051800	0.020000	0.021900	0/10
GA171	0.024200	0.030500	0.029000	0.022700	0.023000	0.027600	0.022500	0.050000	0.021200	0.023500	0/10
GA172	0.023500	0.030200	0.028500	0.022700	0.022500	0.026600	0.022200	0.050500	0.020900	0.023400	0/10
GA173	0.022400	0.030100	0.028800	0.024200	0.022100	0.027000	0.021600	0.048100	0.019900	0.021600	0/10
GA174	0.023100	0.030000	0.029200	0.023300	0.022800	0.026800	0.021500	0.048300	0.020100	0.022100	0/10
GA175	0.022800	0.030600	0.027800	0.024500	0.022700	0.028200	0.022500	0.053600	0.019900	0.022000	0/10
GA176	0.021700	0.029800	0.028200	0.022400	0.023300	0.026200	0.022500	0.048000	0.019800	0.022100	0/10
GA177	0.022200	0.027900	0.027700	0.022600	0.021400	0.027100	0.022700	0.049000	0.019700	0.021700	0/10
GA178	0.021600	0.027000	0.026200	0.022000	0.021800	0.025300	0.022500	0.048600	0.019200	0.021500	0/10
GA179	0.021500	0.028400	0.028400	0.022900	0.022400	0.027300	0.021600	0.052200	0.019800	0.021400	0/10
GA180	0.021100	0.027500	0.026700	0.021500	0.020600	0.026700	0.021800	0.050400	0.018400	0.021200	0/10
GA181	0.021800	0.027000	0.027000	0.022700	0.021800	0.025800	0.022100	0.051800	0.019600	0.022100	0/10
GA182	0.021100	0.026800	0.026200	0.022200	0.021400	0.025100	0.021300	0.051100	0.019600	0.022600	0/10
GA183	0.022800	0.028800	0.028100	0.022500	0.021100	0.026700	0.021600	0.048800	0.019600	0.021700	0/10
GA184	0.021100	0.028200	0.027000	0.021800	0.020800	0.025100	0.021500	0.048900	0.018700	0.021300	0/10
GA185	0.023000	0.029300	0.028400	0.023000	0.021500	0.026100	0.021600	0.049600	0.019600	0.021900	0/10
GA186	0.020900	0.029600	0.026700	0.022600	0.021100	0.026200	0.021700	0.047500	0.019700	0.021400	0/10
GA187	0.022400	0.028400	0.028800	0.022900	0.022700	0.027500	0.022000	0.050200	0.019600	0.022400	0/10
GA188	0.021700	0.027300	0.027600	0.023600	0.021400	0.027700	0.022100	0.047300	0.019300	0.022000	0/10
GA189	0.022100	0.027800	0.026700	0.022000	0.022600	0.026300	0.020500	0.050400	0.019100	0.021500	0/10
GA190	0.020900	0.027700	0.026500	0.022600	0.020300	0.026100	0.022200	0.052900	0.019000	0.020800	0/10
GA191	0.021700	0.027700	0.028000	0.021800	0.021600	0.026000	0.022100	0.056600	0.020200	0.022400	0/10
GA192	0.020800	0.028000	0.026800	0.022400	0.021500	0.026500	0.022800	0.059100	0.019400	0.022500	0/10
GA193	0.024100	0.030800	0.030200	0.023600	0.022700	0.027200	0.022400	0.053500	0.021200	0.024000	0/10
GA194	0.022600	0.029800	0.028300	0.022500	0.021600	0.027000	0.022700	0.055000	0.020700	0.022900	0/10
GA195	0.022400	0.028500	0.027900	0.023200	0.022800	0.027100	0.023100	0.050200	0.021500	0.023100	0/10
GA196	0.023000	0.029000	0.028400	0.022200	0.022700	0.027600	0.022700	0.049800	0.021000	0.024500	0/10
GA197	0.023600	0.030700	0.028000	0.023800	0.022800	0.028500	0.022700	0.049300	0.020400	0.023700	0/10
GA198	0.022900	0.030100	0.029300	0.023900	0.022400	0.027100	0.023300	0.050400	0.020900	0.023200	0/10
GA199	0.023300	0.029500	0.029900	0.023700	0.022100	0.027500	0.022900	0.050000	0.020400	0.022600	0/10
GA200	0.024000	0.029500	0.029600	0.024500	0.022900	0.027300	0.023000	0.050000	0.020900	0.023800	0/10

GA201	0.023700	0.030500	0.030400	0.025300	0.023300	0.027500	0.022800	0.050600	0.021000	0.022800	0/10
GA202	0.022600	0.031300	0.029700	0.023800	0.023700	0.027900	0.023000	0.050000	0.020800	0.022700	0/10
GA203	0.023600	0.030200	0.030500	0.024400	0.024600	0.028400	0.022900	0.051100	0.020600	0.023100	0/10
GA204	0.022300	0.029400	0.030300	0.023600	0.023400	0.028400	0.023000	0.050000	0.020700	0.022800	0/10
GA205	0.023000	0.029700	0.029500	0.024500	0.023200	0.029100	0.022600	0.050300	0.020900	0.023100	0/10
GA206	0.022300	0.029800	0.029300	0.024800	0.022200	0.028500	0.023200	0.049900	0.020500	0.022700	0/10
GA207	0.022800	0.028200	0.029700	0.024000	0.023400	0.027700	0.023800	0.050200	0.021200	0.023200	0/10
GA208	0.022500	0.029500	0.028100	0.023000	0.023200	0.026900	0.022600	0.052300	0.020100	0.023900	0/10
GA209	0.021700	0.028300	0.027900	0.022500	0.022400	0.028300	0.022200	0.055200	0.019800	0.022600	0/10
GA210	0.021300	0.027900	0.027300	0.022600	0.021800	0.026300	0.023100	0.052300	0.019600	0.022700	0/10
GA211	0.021100	0.027900	0.027500	0.022600	0.021200	0.026100	0.022200	0.052400	0.020900	0.022600	0/10
GA212	0.021900	0.028100	0.028100	0.023200	0.021800	0.026800	0.022200	0.053200	0.019500	0.022300	0/10
GA213	0.022200	0.027300	0.027800	0.022900	0.021800	0.025800	0.022000	0.050800	0.019500	0.021900	0/10
GA214	0.022900	0.028600	0.027500	0.022500	0.021900	0.025900	0.021200	0.052600	0.019400	0.022200	0/10
GA215	0.021700	0.029900	0.027700	0.023100	0.021600	0.025900	0.021300	0.051900	0.019400	0.022300	0/10
GA216	0.021900	0.028300	0.028600	0.022300	0.022400	0.025900	0.021700	0.049200	0.019900	0.022100	0/10
GA217	0.021500	0.029100	0.027100	0.022900	0.022300	0.027200	0.021500	0.048900	0.020200	0.022000	0/10
GA218	0.021300	0.027800	0.027400	0.022200	0.022400	0.026900	0.022200	0.052100	0.019800	0.022100	0/10
GA219	0.021900	0.028300	0.029800	0.023600	0.023000	0.027500	0.021700	0.053300	0.019500	0.022100	0/10
GA220	0.021100	0.028700	0.027500	0.022300	0.021900	0.027300	0.022400	0.052000	0.020000	0.021400	0/10
GA221	0.021000	0.027700	0.027900	0.022100	0.022000	0.025800	0.022400	0.051100	0.019600	0.021500	0/10
GA222	0.021300	0.028800	0.026400	0.022400	0.021200	0.026200	0.021100	0.051300	0.019400	0.021600	0/10
GA223	0.021200	0.027700	0.026700	0.022800	0.021500	0.025800	0.022400	0.046600	0.020100	0.021400	0/10
GA224	0.021900	0.029000	0.027200	0.021800	0.020800	0.026100	0.021400	0.046800	0.019200	0.021600	0/10
GA225	0.023100	0.030500	0.030200	0.024500	0.022900	0.027600	0.023000	0.049900	0.021000	0.023200	0/10
GA226	0.023600	0.030700	0.029400	0.023600	0.022800	0.027400	0.022800	0.049500	0.020800	0.022800	0/10
GA227	0.023200	0.030300	0.030700	0.024700	0.023900	0.028400	0.022900	0.050100	0.020900	0.022700	0/10
GA228	0.022900	0.028900	0.028900	0.024700	0.023000	0.029100	0.022100	0.050100	0.020500	0.022600	0/10
GA229	0.022200	0.030800	0.029500	0.024800	0.023800	0.028400	0.023900	0.051300	0.020800	0.024500	0/10
GA230	0.022800	0.029800	0.028500	0.024300	0.023800	0.027500	0.023800	0.050200	0.021000	0.023200	0/10
GA231	0.024400	0.030600	0.029300	0.024900	0.024100	0.029300	0.023500	0.052200	0.021700	0.023900	0/10
GA232	0.022900	0.028400	0.028100	0.023800	0.023000	0.029000	0.023000	0.057500	0.020600	0.023000	0/10
GA233	0.023000	0.029600	0.030400	0.024000	0.023400	0.029900	0.023500	0.053600	0.021000	0.024000	0/10
GA234	0.023000	0.030500	0.029900	0.023200	0.023400	0.028300	0.023400	0.053300	0.021600	0.024100	0/10
GA235	0.023500	0.030600	0.029700	0.023500	0.023000	0.028500	0.023100	0.049400	0.021200	0.023700	0/10
GA236	0.022900	0.029700	0.028700	0.024600	0.024000	0.027400	0.023800	0.052500	0.020400	0.021800	0/10
GA237	0.024100	0.029900	0.031500	0.023900	0.023200	0.030100	0.027200	0.056200	0.020000	0.023100	0/10
GA238	0.023100	0.029100	0.028000	0.023000	0.022900	0.027700	0.023500	0.051400	0.020100	0.022700	0/10
GA239	0.023000	0.028200	0.028500	0.023800	0.023500	0.027300	0.023700	0.053800	0.021600	0.022400	0/10
GA240	0.022100	0.030100	0.027700	0.022900	0.022600	0.027800	0.023600	0.059000	0.020700	0.022100	0/10
GA241	0.021700	0.027700	0.027000	0.021500	0.020700	0.026100	0.021700	0.052400	0.019700	0.025200	0/10
GA242	0.021900	0.028200	0.027400	0.022100	0.021900	0.026000	0.021900	0.051200	0.019500	0.021600	0/10
GA243	0.022600	0.029900	0.027600	0.022700	0.023000	0.027300	0.022700	0.051000	0.020300	0.021900	0/10
GA244	0.022000	0.028500	0.027400	0.022700	0.023100	0.026200	0.021400	0.049600	0.018300	0.022000	0/10
GA245	0.022500	0.028200	0.029500	0.021600	0.021800	0.026700	0.022200	0.049700	0.019800	0.022200	0/10
GA246	0.023500	0.028200	0.027500	0.023700	0.022500	0.027500	0.021800	0.049500	0.020300	0.022700	0/10
GA247	0.022500	0.029400	0.027300	0.022700	0.022800	0.026800	0.023100	0.051900	0.020800	0.022700	0/10
GA248	0.022700	0.029100	0.026800	0.022300	0.022200	0.028000	0.021300	0.053100	0.020000	0.021300	0/10
GA249	0.021900	0.029800	0.027900	0.023400	0.023500	0.028200	0.022800	0.053600	0.020400	0.022400	0/10
GA250	0.022000	0.028800	0.028700	0.023000	0.022300	0.026700	0.023100	0.053500	0.020100	0.021700	0/10
GA251	0.023000	0.028800	0.027100	0.022900	0.022100	0.027000	0.022100	0.053100	0.021100	0.022200	0/10
GA252	0.021500	0.028700	0.029000	0.022400	0.021300	0.027000	0.022800	0.054100	0.019700	0.023800	0/10
GA253	0.021300	0.028600	0.027600	0.022700	0.021700	0.026300	0.022100	0.050900	0.019800	0.022800	0/10
GA254	0.022800	0.028800	0.027200	0.023400	0.022200	0.026400	0.021500	0.049100	0.019900	0.021700	0/10
GA255	0.023100	0.030700	0.029100	0.023200	0.022800	0.026900	0.022300	0.049900	0.020800	0.022700	0/10
GA256	0.021800	0.030200	0.028100	0.022200	0.022300	0.027000	0.022600	0.051800	0.020200	0.024500	0/10
Min	0.019700	0.026200	0.025100	0.020400	0.019800	0.024400	0.019800	0.046100	0.017600	0.019800	
Max	0.035200	0.051100	0.040600	0.034700	0.034800	0.044700	0.035100	0.069600	0.029100	0.031600	

GA Source Code

```
#define _USE_MATH_DEFINES

#include <iostream>

#include <conio.h>

#include <fstream>

#include <cstdlib>

#include <stdio.h>

#include <time.h>

#include <string>

#include <iomanip>

#include <algorithm>

#include <windows.h>

#include <cmath>

#include <limits>

using namespace std;


/*******************************************************   CODE   GUIDELINE
******************************************************

* Parameter Settings


- Benchmark Function

- External Function

* Selection Function

* Crossover Function

* Mutation Function

* Replacement Function


- GA Automation

      -> Benchmark Automation
```

```
             -> Experiment Automation

                   -> Generate Population

                   -> Fitness Evaluation 1

                   -> Termination Criteria (Maximum Generation) [Can modify
starting from here (GA, DE, PSO)]

                         *> Selection Operation

                         *> Crossover Operation

                         *> Mutation Operation

                         -> Fitness Evaluation 2

                         *> Replacement Operation

*******************************************************   CODE   GUIDELINE
*******************************************************/


//--------------------------------------------------------------------------
-----------------------------------------------------

// Parameter Settings

//--------------------------------------------------------------------------
-----------------------------------------------------

#define MINI_PROJECT 1                                             // Project
-> 1 | Assignment -> 0 | Demo -> -1 | Manual -> -2


const double dcp = 0.7, dmp = 0.01;                         //      Crossover
Probability, Mutation Probability

const int tournamentSize = 5;                               //     Tournament
Selection Size

const double highDiverseThreshold = 0.04;           //  Threshold  for  High
Diversity

const double lowDiverseThreshold = 0.01;            //  Threshold  for  Low
Diversity

//--------------------------------------------------------------------------
-----------------------------------------------------


#if MINI_PROJECT == 1
```

```
	#define TECHNIQUE 4                                     // No. Of
Techniques

	string GA = "0000000000000000";                     //   GA   Binary
Combinations

#elif MINI_PROJECT == 0 || MINI_PROJECT == -1 || MINI_PROJECT == -2

	#define TECHNIQUE 2

	string GA = "00000000";

#else

	#define EXIT

	#define TECHNIQUE

	string GA = "";

#endif


#define getrandom(min, max) (((double)rand() / RAND_MAX) * ((max) - (min)) +
(min))

#define gen 2000                                                // number
of iterations (number of generations)

#define pSize 40                                                // number
of chromosomes (population size)

#define dimension 30                                            // number
of bits (dimension size)


int GA_COMBINATION[4][TECHNIQUE] = {0};

int BENCHMARK = 1;

string benchmarkFunction = "";

int rangeMin = 0, rangeMax = 0, rangeDiv = 1000;


double chromosome[pSize][dimension] = {0};          // chromosome

double paroff[4][dimension] = {0};                       //  parent  and
offspring

double fit[pSize] = {0};                                 // fitness value
for each chromosome

double r = 0, gcp = 0, gmp = 0;
```

```
int crb = 0, mb1 = 0, mb2 = 0;

int rp1 = 0, rp2 = 0;

double mb1v = 0, mb2v = 0;

double fv = 0, sumFit = 0;

double fit1 = 0, fit2 = 0;

double tfit[4] = {0};


int lFvIndex = 0;

double lFv = 0;

double lowestGene[dimension] = {0};

double lowestGeneFV = 0;


int dynamicTournamentSize = tournamentSize;


//---------------------------------------------------------------------------------------------------------------------

// Benchmark Function

//---------------------------------------------------------------------------------------------------------------------

// No.1 - Sphere Function +-5.12

double Sphere(double a[])

{

	sumFit = 0;


	for (int j = 0; j < dimension; j++)

	{

		fv = pow(a[j], 2);

		sumFit = sumFit + fv;

	}
	return sumFit;
}


// No.2 - Ackley Function +-32.768
// Done By: Yeap Chun Hong 2206352
double Ackley(double a[])
{
	sumFit = 0;
	double sum1 = 0, sum2 = 0;

	for (int j = 0; j < dimension; j++)
	{
		sum1 += pow(a[j], 2);
		sum2 += cos(2 * M_PI * a[j]);
	}

	double term1 = -20 * exp(-0.2 * sqrt(sum1 / dimension));
	double term2 = exp(sum2 / dimension);

	sumFit = term1 - term2 + 20 + exp(1);

	return sumFit;
}


// No.3 - Rastrigin Function +-5.12
// Done By: Yeap Chun Hong 2206352
double Rastrigin(double a[])
{
	sumFit = 0;
	for (int j = 0; j < dimension; j++)
	{
		fv = (pow(a[j], 2)) - (10 * cos(2 * M_PI * a[j]));
		sumFit = sumFit + fv;
	}

	sumFit += 10 * dimension;

	return sumFit;
}

// No.4 - Zakharov Function -5, +10
// Done By: Brandon Ting En Junn 2101751
double Zakharov(double a[])
{
	sumFit = 0;
	double sumFit1 = 0, sumFit2 = 0, sumFit3 = 0;

	// sumFit1
	for (int j = 0; j < dimension; j++)
	{
		fv = pow(a[j], 2);
		sumFit1 = sumFit1 + fv;
	}

	// sumFit2
	for (int j = 0; j < dimension; j++)
	{
		fv = 0.5 * (j + 1) * a[j];

		sumFit2 = sumFit2 + fv;

	}

	sumFit2 = pow(sumFit2, 2);


	// sumFit3

	for (int j = 0; j < dimension; j++)

	{

		fv = 0.5 * (j + 1) * a[j];

		sumFit3 = sumFit3 + fv;

	}

	sumFit3 = pow(sumFit3, 4);


	sumFit = sumFit1 + sumFit2 + sumFit3;


	return sumFit;

}


// No.5 - Axis Parallel Hyper-Ellipsoid Function +-5.12

// Done By: Loh Chia Heung 2301684

double AxisParallel(double a[])

{

	sumFit = 0;


	for (int j = 0; j < dimension; j++)

	{

		fv = (j + 1) * pow(a[j], 2);

		sumFit = sumFit + fv;

	}
```

```
        return sumFit;

}


// No.6 - Griewank Function +-600

// Done By: Loh Chia Heung 2301684

double Griewank(double a[])

{

        sumFit = 0;

        double product = 1.0;


        for (int j = 0; j < dimension; j++)

        {

                sumFit += (a[j] * a[j]) / 4000.0;

                product *= cos(a[j] / sqrt(j + 1));

        }


        sumFit = sumFit - product + 1;


        return sumFit;

}


// No.7 - Sum of Different Powers function +-1.00

// Done By: Yeap Chun Hong 2206352

double SumOfDifferentPowers(double a[])

{

        sumFit = 0;


        for (int j = 0; j < dimension; j++)
```

```
	{
		fv = pow(fabs(a[j]), j + 2);
		sumFit = sumFit + fv;
	}

	return sumFit;
}

// No.8 - Rotated Hyper-Ellipsoid Function +-65.536
// Done By: Brandon Ting En Junn 2101751
double Rotated(double a[])
{
	sumFit = 0;

	for (int i = 0; i < dimension; i++)
	{
		fv = 0;

		for (int j = 0; j <= i; j++)
		{
			fv += pow(a[j], 2);
		}

		sumFit = sumFit + fv;
	}

	return sumFit;
}
```

```
// No.9 - Schwefel 2.22 Function +-5 [Updated]

// Done By: Ling Ji Xiang 2104584

double Schwefel(double a[])

{

    sumFit = 0;

    double absolute = 0, sum = 0, product = 1.0;


    for (int i = 0; i < dimension; i++)

    {

        absolute = fabs(a[i]);

        sum += absolute;


        product *= absolute;

    }


    sumFit = sum + product;


    return sumFit;

}


// No.10 - Exponential function Function +-1.00 [f(x) = -1]

// Done By: Ling Ji Xiang 2104584

double Exponential(double a[])

{

    sumFit = 0;


    // calculate the sum of squares

    for (int j = 0; j < dimension; j++)

    {
```

```
			fv = pow(a[j], 2);

			sumFit = sumFit + fv;

		}


		// Calculate the exponential of sum of squares

		sumFit = -exp(-0.5 * sumFit);


		return sumFit;

}


/* Benchmark_Range */

void initialiseBenchmark_Range()

{

		switch (BENCHMARK)

		{

		case 1: // No.1 - Sphere Function +-5.12

			benchmarkFunction = "-Sphere";

			rangeMin = rangeMax = 5120;

			break;


		case 2: // No.2 - Ackley Function +-32.768

			benchmarkFunction = "-Ackley";

			rangeMin = rangeMax = 32768;

			break;


		case 3: // No.3 - Rastrigin Function +-5.12

			benchmarkFunction = "-Rastrigin";

			rangeMin = rangeMax = 5120;

			break;
```

```
case 4: // No.4 - Zakharov Function -5, +10

       benchmarkFunction = "-Zakharov";

       rangeMin = 5000;

       rangeMax = 10000;

       break;


case 5: // No.5 - Axis Parallel Hyper-Ellipsoid Function +-5.12

       benchmarkFunction = "-AxisParallel";

       rangeMin = rangeMax = 5120;

       break;


case 6: // No.6 - Griewank Function +-600

       benchmarkFunction = "-Griewank";

       rangeMin = rangeMax = 600000;

       break;


case 7: // No.7 - Sum of Different Powers function +-1.00

       benchmarkFunction = "-SumOfDifferentPowers";

       rangeMin = rangeMax = 1000;

       break;


case 8: // No.8 - Rotated Hyper-Ellipsoid Function +-65.536

       benchmarkFunction = "-Rotated";

       rangeMin = rangeMax = 65536;

       break;


case 9: // No.9 - Schwefel 2.22 Function +-5 [Updated]

       benchmarkFunction = "-Schwefel";
```

```
			rangeMin = rangeMax = 5000;

			break;


		case 10: // No.10 - Exponential function Function +-1.00 [f(x) = -1]

			benchmarkFunction = "-Exponential";

			rangeMin = rangeMax = 1000;

			break;


		default: // ERROR

			cout << "Error BENCHMARK: Invalid Benchmark Function\n\nPress
Any Key to Exit..." << endl;

			getch();

			exit(0);

		}

}


/* Fitness Function */

double Fitness(double a[])

{

		switch (BENCHMARK)

		{

		case 1: // No.1 - Sphere Function +-5.12

			return Sphere(a);


		case 2: // No.2 - Ackley Function +-32.768

			return Ackley(a);


		case 3: // No.3 - Rastrigin Function +-5.12

			return Rastrigin(a);
```

```
        case 4: // No.4 - Zakharov Function -5, +10

                return Zakharov(a);


        case 5: // No.5 - Axis Parallel Hyper-Ellipsoid Function +-5.12

                return AxisParallel(a);


        case 6: // No.6 - Griewank Function +-600

                return Griewank(a);


        case 7: // No.7 - Sum of Different Powers function +-1.00

                return SumOfDifferentPowers(a);


        case 8: // No.8 - Rotated Hyper-Ellipsoid Function +-65.536

                return Rotated(a);


        case 9: // No.9 - Schwefel 2.22 Function +-5 [Updated]

                return Schwefel(a);


        case 10: // No.10 - Exponential function Function +-1.00 [f(x) = -1]

                return Exponential(a);


        default: // ERROR

                cout << "Error BENCHMARK: Invalid Benchmark Function\n\nPress
Any Key to Exit..." << endl;

                getch();

                exit(0);

        }

}

//-------------------------------------------------------------------------
-----------------------------------------------------
```

```
//----------------------------------------------------------------------------
------------------------------------------------------

// External Function

//----------------------------------------------------------------------------
------------------------------------------------------

int charToInt(char c)

{

	if (c >= '0' && c <= '9')

	{

		return c - '0';

	}

	else

	{

		// ERROR

		cout << "Error charToInt(): Invalid Conversion\n\nPress Any Key
to Exit..." << endl;

		getch();

		exit(0);

	}

}


string addBinary(string a, string b)

{

	string result = "";        // Initialize the result as an empty string

	int s = 0;                        // Initialize the carry


	// Ensure both strings are of the same length by padding with leading
zeros

	int n = max(a.size(), b.size());

	while (a.size() < n)
```

```
            a.insert(a.begin(), '0');

        while (b.size() < n)

            b.insert(b.begin(), '0');


        // Traverse both strings from right to left

        for (int i = n - 1; i >= 0; i--)

        {

            int sum = (a[i] - '0') + (b[i] - '0') + s;          //
Calculate the sum of the current digits and carry

            result.insert(result.begin(), (sum % 2) + '0'); // Insert the
current bit to the result

            s = sum / 2;
        // Calculate the new carry

        }


        // If there's a carry left, add it to the result

        if (s != 0)

        {

            result.insert(result.begin(), '1');

        }


        return result;

}


void GA_TO_GA_COMBINATION()

{

        int index = 0;


        for (int i = 0; i < 4; i++)

        {

            for (int j = 0; j < TECHNIQUE; j++)
            {
                  GA_COMBINATION[i][j] = charToInt(GA[index++]);
            }
      }
}

bool isValidCombination(int combination[4][TECHNIQUE])
{
      int ones = 0;

      // Check Each Operation Technique
      for (int i = 0; i < 4; i++)
      {

            ones = 0;

            // Check Each Technique
            for (int j = 0; j < TECHNIQUE; j++)
            {
                  if (combination[i][j] == 1)
                  {
                        ones++;
                  }
            }

            if (ones != 1)
            {
                  return false;
            }
```

```
	}

	return true;
}

void resetExperiment()
{
	for (int i = 0; i < pSize; i++)
	{
		for (int j = 0; j < dimension; j++)
		{
			chromosome[i][j] = 0;

			if (i < 4)
			{
				paroff[i][j] = 0;
				tfit[i] = 0;
			}

			if (i == 0)
			{
				lowestGene[j] = 0;
			}
		}

		fit[i] = 0;
	}

	r = gcp = gmp = 0;
	crb = mb1 = mb2 = 0;

	rp1 = rp2 = 0;

	mb1v = mb2v = 0;

	fv = sumFit = 0;

	fit1 = fit2 = 0;


	lFvIndex = 0;

	lFv = 0;

	lowestGeneFV = 0;


	dynamicTournamentSize = tournamentSize;

}

//----------------------------------------------------------------------
-----------------------------------------------------


//----------------------------------------------------------------------
-----------------------------------------------------

// Selection Function

// Done By: Yeap Chun Hong 2206352

//----------------------------------------------------------------------
-----------------------------------------------------

int RouletteWheelSelection2(double fitness[])

{


	double inverseFit[pSize];

	double totalFit = 0;


	for (int i = 0; i < pSize; i++)

	{

		inverseFit[i] = 1 / fitness[i]; //inverse so that smaller
fitness value will have a higher portion
```

```
		totalFit += inverseFit[i];

		//cout <<fitness[i] <<"\t" <<inverseFit[i] <<endl;

	}

	//cout <<totalFit <<endl;


	//calculate the cumulative probability and normalise it to [0,1]

	double cumulativeProbability[pSize];

	cumulativeProbability[0] = inverseFit[0] / totalFit;

	//cout <<"cumulativeProbability" << cumulativeProbability[0] <<endl;

	for (int i = 1; i < pSize; i++)

	{

		cumulativeProbability[i]  =  cumulativeProbability[i  -  1]  +
(inverseFit[i] / totalFit);

		//cout <<i << "\t"<<cumulativeProbability[i] <<endl;

	}


	// generate number from 0 to 1

	double       spin       =       static_cast<double>(rand())       /
static_cast<double>(RAND_MAX);   //cout <<"spin: " <<spin <<endl;

	//cout << spin << endl;


	for (int i = 0; i < pSize; i++)

	{

		if (spin <= cumulativeProbability[i])

		{

			//cout << "chosen: " << i << endl;

			return i;

		}

	}
	// In case of rounding errors, return the last individual
	return pSize;
}

void RouletteWheelSelection(double fitness[], int &parent1, int &parent2)
{
	parent1 = RouletteWheelSelection2(fitness);
	parent2 = RouletteWheelSelection2(fitness);

	// cout <<"Parent1: "<<parent1<<"parent2 "<<parent2<<endl;
	// getch();
}

int TournamentSelection2(double fitness[], int tournamentSize)
{
	int best = -1;
	double bestFitness = numeric_limits<double>::max();
	bool selectedIndices[pSize] = { false };

	//Randomly select unique individuals and perform tournament
	for (int i = 0; i < tournamentSize; i++)
	{
		int index;
		do
		{
			index = rand() % pSize;
		} while (selectedIndices[index]);
		//cout <<"index "<< index <<endl;
		selectedIndices[index] = true;


		//cout <<"fitness[index] "<< fitness[index]<<" bestFitness "<< 
bestFitness <<endl;

		if (fitness[index] < bestFitness)

		{

			best = index;

			bestFitness = fitness[index];

			//cout <<"best "<< best<<" bestFitness "<< bestFitness 
<<endl;

		}

	}

	//getch();

	return best;

}


void TournamentSelection(double fitness[], int tournamentSize, int &parent1, 
int &parent2)

{

	// Select first parent

	parent1 = TournamentSelection2(fitness, tournamentSize);


	// Select second parent

	parent2 = TournamentSelection2(fitness, tournamentSize);


	// cout <<"Parent1: "<<parent1<<"parent2 "<<parent2<<endl;

	// getch();

}


int LinearRankingSelection2(double fitness[])

{
	//selection pressure

	double max = 1.1;

	double min = 2 - max;




	//sort fitness value in descending order

	sort(fitness, fitness + pSize, greater<double>());

	double probability[pSize];

	double totalFit = 0;

	for (int i = 0; i < pSize; i++)

	{

		probability[i] = (min + (max - min) * i / (pSize - 1)) / pSize;

		totalFit += probability[i];


	}



	//calculate the cumulative probability and normalise it to [0,1]

	double cumulativeProbability[pSize];

	cumulativeProbability[0] = probability[0] / totalFit;

	//cout  <<"i"  <<"\t"<<  "Fitness"<<"\t"<<  "Probability"<<  "\t"  <<
"Cumulative Prob" << endl;

	//cout  <<"0"  <<"\t"<<  fitness[0]<<"\t"<<  probability[0]<<  "\t"
<<cumulativeProbability[0] << endl;


	//cout <<"cumulativeProbability" << cumulativeProbability[0] <<endl;

	for (int i = 1; i < pSize; i++)

	{

		cumulativeProbability[i]  =  cumulativeProbability[i  -  1]  +
(probability[i] / totalFit);

		//cout <<i <<"\t"<< fitness[i]<<"\t"<< probability[i]<< "\t"
<<cumulativeProbability[i] << endl;
```

```
            //cout <<i << "\t"<<cumulativeProbability[i] <<endl;
        }


        // generate number from 0 to 1
        double        spin        =        static_cast<double>(rand())        /
static_cast<double>(RAND_MAX);   //cout <<"spin: " <<spin <<endl;
        //cout << spin <<endl;


        for (int i = 0; i < pSize; i++)
        {
            if (spin <= cumulativeProbability[i])
            {
                //cout << "chosen: "<<i<<endl;
                //getch();
                return i;
            }
        }


        // In case of rounding errors, return the last individual
        return pSize;
}


void LinearRankingSelection(double fitness[], int& parent1, int& parent2)
{
        // Select first parent
        parent1 = LinearRankingSelection2(fitness);


        // Select second parent
        parent2 = LinearRankingSelection2(fitness);
```

```
      //getch();
}


double CalculateDiversity(double fitness[])
{


      double mean = 0.0;
      double sumSquare = 0.0;
      double sumFitness =0.0;
      for (int i =0; i < pSize; i++){
            sumFitness += fitness[i];
      }
      //calculate mean of fitness
      for (int i =0; i < pSize; i++){
            mean += fitness[i]/sumFitness;
      }

      mean = mean/pSize;

      //calculate sum of square
      for (int i=0; i < pSize; i++){
            sumSquare    +=    (fitness[i]/sumFitness    -    mean)    *
(fitness[i]/sumFitness - mean);
      }

      double stdDev = sqrt(sumSquare/pSize);
      return stdDev;
}
```

```
int AdjustTournamentSize(double diversity, int currentSize, int minSize, int maxSize) {

	//cout    <<"Diversity:    "<<    diversity    <<"\t"    <<"Size:    " <<currentSize<<endl;

	//getch();


    if (diversity > highDiverseThreshold) {  // if diversity > 0.04, lower tournamentsize

        return max(minSize, currentSize - 1);

    } else if (diversity < lowDiverseThreshold) {  // if diversity < 0.01, lower tournamentsize

        return min(maxSize, currentSize + 1);

    }


    //remain the same if 0.01 < diversity < 0.04

    return currentSize;

}


void        DynamicTournamentSelection(double        fitness[],        int& dynamicTournamentSize, int& parent1, int& parent2)

{

	double diversity = CalculateDiversity(fitness);

	dynamicTournamentSize                                                = AdjustTournamentSize(diversity,dynamicTournamentSize,3,7);

	// Select first parent

    parent1 = TournamentSelection2(fitness, dynamicTournamentSize);


    // Select second parent

    parent2 = TournamentSelection2(fitness, dynamicTournamentSize);


    //getch();

}
```

```
//-----------------------------------------------------------------------------------------------------------------------


//-----------------------------------------------------------------------------------------------------------------------

// Crossover Function

// Done By: Loh Chia Heung 2301684

//-----------------------------------------------------------------------------------------------------------------------

// No.1 Crossover Technique

void      UniformCrossover(double      chromosome[][dimension],      double
paroff[][dimension], double dcp, int p1, int p2)

{

	double gcp = (rand() % 1000);

	gcp = gcp / 1000;


	if (gcp <= dcp)

	{

		// Perform Uniform Crossover

		for (int j = 0; j < dimension; j++)

		{

			if (rand() % 2 == 0)

			{

				// Offspring 1 --> Parent 1

				paroff[2][j] = chromosome[p1][j];

				      // Offspring 2 --> Parent 2

				paroff[3][j] = chromosome[p2][j];

			}

			else

			{

				// Offspring 1 --> Parent 2
```

```
                        paroff[2][j] = chromosome[p2][j];

                              // Offspring 2 --> Parent 1

                        paroff[3][j] = chromosome[p1][j];

                  }

            }

      }

      else

      {

            // No crossover, directly copy parents to offspring

            for (int j = 0; j < dimension; j++)

            {

                  paroff[2][j]  =  chromosome[p1][j];  //  Offspring1  -->
Parent 1

                  paroff[3][j]  =  chromosome[p2][j];  //  Offspring2  -->
Parent 2

            }

      }

}


// No.2 Crossover Technique

void      ShuffleCrossover(double      chromosome[][dimension],      double
paroff[][dimension], double dcp, int p1, int p2)

{


      double gcp = (rand() % 1000);

      gcp = gcp / 1000;


      if (gcp <= dcp)

      {

            double temp[dimension];
            // Shuffle chromosomes between parents
            for (int j = 0; j < dimension; j++)
            {
                  temp[j]  =  (rand()  %  2  ==  0)  ?  chromosome[p1][j]  :
chromosome[p2][j];
            }

            // Assign shuffled chromosomes to offspring
            for (int j = 0; j < dimension; j++)
            {
                  paroff[2][j] = temp[j];
                  paroff[3][j] = (rand() % 2 == 0) ? chromosome[p1][j] :
chromosome[p2][j];
            }
      }
      else
      {
            // No crossover --> offspring are exact copies of parents
            for (int j = 0; j < dimension; j++)
            {
                  paroff[2][j]  =  chromosome[p1][j];  //  Offspring1  -->
Parent 1
                  paroff[3][j]  =  chromosome[p2][j];  //  Offspring2  -->
Parent 2
            }
      }
}

// No.3 Crossover Technique
void     ArithmeticCrossover(double     chromosome[][dimension],     double
paroff[][dimension], double dcp, int p1, int p2) {
    double gcp = (rand() % 1000);

    gcp = gcp / 1000;



    if (gcp <= dcp) {

        // Generate a random alpha value between 0 and 1

        double alpha = static_cast<double>(rand()) / RAND_MAX;


        for (int j = 0; j < dimension; j++) {

            // Offspring 1: alpha * Parent 1 + (1 - alpha) * Parent 2

            paroff[2][j]  =  alpha  *  chromosome[p1][j]  +  (1  -  alpha)  *
chromosome[p2][j];


            // Offspring 2:  (1 - alpha) * Parent 1 + alpha * Parent 2

            paroff[3][j]  =  (1  -  alpha)  *  chromosome[p1][j]  +  alpha  *
chromosome[p2][j];

        }

    } else {

        // No crossover --> offspring are exact copies of parents

        for (int j = 0; j < dimension; j++) {

            paroff[2][j] = chromosome[p1][j]; // Offspring 1 --> Parent 1

            paroff[3][j] = chromosome[p2][j]; // Offspring 2 --> Parent 2

        }

    }

}


// No.4 Crossover Technique

void      CombinedCrossover(double      chromosome[][dimension],      double
paroff[][dimension], double dcp, int p1, int p2) {

    // Temporary chromosomes for intermediate crossover results
```

```
double tempChrom1[dimension];

double tempChrom2[dimension];

double tempChrom3[dimension];

double tempChrom4[dimension];


// Step 1: Uniform Crossover

double gcp = static_cast<double>(rand()) / RAND_MAX;

if (gcp <= dcp) {

    for (int j = 0; j < dimension; j++) {

        if (rand() % 2 == 0) {

            tempChrom1[j] = chromosome[p1][j];

            tempChrom2[j] = chromosome[p2][j];

        } else {

            tempChrom1[j] = chromosome[p2][j];

            tempChrom2[j] = chromosome[p1][j];

        }

    }

} else {

    for (int j = 0; j < dimension; j++) {

        tempChrom1[j] = chromosome[p1][j];

        tempChrom2[j] = chromosome[p2][j];

    }

}



// Generate a random crossover point between 0 and 1

double crossoverPoint = static_cast<double>(rand()) / RAND_MAX;


// Step 2: Single Point Crossover
```

```
    int point = static_cast<int>(crossoverPoint * dimension);

    for (int j = 0; j < dimension; j++) {

        if (j < point) {

            tempChrom3[j] = tempChrom1[j];

            tempChrom4[j] = tempChrom2[j];

        } else {

            tempChrom3[j] = tempChrom2[j];

            tempChrom4[j] = tempChrom1[j];

        }

    }


    // Step 3: Arithmetic Crossover

    double alpha = static_cast<double>(rand()) / RAND_MAX;


    for (int j = 0; j < dimension; j++) {

        paroff[2][j] = alpha * tempChrom3[j] + (1 - alpha) * tempChrom4[j];

        paroff[3][j] = (1 - alpha) * tempChrom3[j] + alpha * tempChrom4[j];

    }


}


//-----------------------------------------------------------------------------------------------------------------------


//-----------------------------------------------------------------------------------------------------------------------

// Mutation Function

// Done By: Brandon Ting En Junn 2101751

//-----------------------------------------------------------------------------------------------------------------------

void ReversingMutation()
```

```
{
	for (int i = 2; i < 4; i++)
	{
		gmp = (rand() % 1000000);
		gmp = gmp / 1000000;

		// Perform Mutation
		if (gmp <= dmp) {
			mb1 = getrandom(0, dimension - 1);
			mb1v = paroff[i][mb1];

			// Swap
			if (mb1 == 0) {
				paroff[i][mb1] = paroff[i][dimension - 1];
				paroff[i][dimension - 1] = mb1v;
			} else {
				paroff[i][mb1] = paroff[i][mb1 - 1];
				paroff[i][mb1 - 1] = mb1v;
			}
		}
	}
}

void RandomMutation()
{
	for (int i = 2; i < 4; i++)
	{
		mb1 = getrandom(0, dimension - 1);
		mb2 = getrandom(0, dimension - 1);
```

```
			for (int j = 0; j < 2; j++)

			{

				gmp = (rand() % 1000000);

				gmp = gmp / 1000000;


				// Perform Mutation

				if (gmp <= dmp) {


					// Random

					r = getrandom(-rangeMin, rangeMax);

					r = r / rangeDiv;


					if (j == 0) {

						paroff[i][mb1] = r;

					} else {

						paroff[i][mb2] = r;

					}


				}

			}

		}

}


void SimpleInversionMutation()

{

		for (int i = 2; i < 4; i++)

		{

			gmp = (rand() % 1000000);
```

```
				gmp = gmp / 1000000;

				if (gmp <= dmp)
				{
					do {
						mb1 = getrandom(0, dimension - 1);
						mb2 = getrandom(0, dimension - 1);

						if (mb1 > mb2) {
							double temp = mb1;
							mb1 = mb2;
							mb2 = temp;
						}
					} while (mb1 == mb2);

					int iterations = (mb2 - mb1 + 1) / 2;
					for (int j = 0; j < iterations; j++, mb1++, mb2--)
					{
						double temp = paroff[i][mb1];
						paroff[i][mb1] = paroff[i][mb2];
						paroff[i][mb2] = temp;
					}
				}
		}
}

void HybridComparingMutation()
{
		/*
```

```
            The Hybrid Comparing Mutation is the combination of Reversing
Mutation and Random Mutation.

            Firstly, a random gene is selected for mutation (selected gene).

            Secondly, a random gene is generated (generated gene) and
compared to the selected gene. [Random Mutation]

            If the generated gene is better (closer to 0) than the selected
gene, it replaces it. [Comparing]

            Thirdly, the selected gene is reversed to its -1 position.
[Reversing Mutation]


            Special Case: If index is 0, reverse with the last index.

      */

      // Child 1 & Child 2 Loop

      for (int i = 2; i < 4; i++)

      {

            gmp = (rand() % 1000000);

            gmp = gmp / 1000000;


            // Mutation Occurs

            if (gmp <= dmp)

            {

                  // Select

                  mb1 = getrandom(0, dimension - 1);

                  mb1v = paroff[i][mb1];


                  // Generate Random

                  r = getrandom(-rangeMin, rangeMax);

                  r = r / rangeDiv;


                  // Compare and Replace

                  if (fabs(r) < fabs(mb1v)) {
                        mb1v = r;
                  }


                  // Reverse
                  if (mb1 == 0) {
                        paroff[i][mb1] = paroff[i][dimension - 1];
                        paroff[i][dimension - 1] = mb1v;
                  } else {
                        paroff[i][mb1] = paroff[i][mb1 - 1];
                        paroff[i][mb1 - 1] = mb1v;
                  }
            }
      }
}
//----------------------------------------------------------------------
---------------------------------------------------


//----------------------------------------------------------------------
---------------------------------------------------
// Replacement Function
// Done By: Ling Ji Xiang 2104584
//----------------------------------------------------------------------
---------------------------------------------------
void WeakParentReplacement(double chromosome[pSize][dimension], double
fit[pSize], double paroff[4][dimension], double tfit[4], int parent1, int
parent2)
{
      int chosenParent, notChosenParent;
      int chosenChild, notChosenChild;


      // Determine Weak and Strong Parent
```

```
if (fit[parent1] > fit[parent2])

{

      chosenParent = parent1;

      notChosenParent = parent2;

}

else

{

      chosenParent = parent2;

      notChosenParent = parent1;

}


// Determine Strong and Weak Child

if (tfit[2] < tfit[3])

{

      chosenChild = 2;

      notChosenChild = 3;

}

else

{

      chosenChild = 3;

      notChosenChild = 2;

}


if (fit[chosenParent] > tfit[chosenChild])

{

      for (int i = 0; i < dimension; i++)

      {

            chromosome[chosenParent][i] = paroff[chosenChild][i];

      }
```

```
            fit[chosenParent] = tfit[chosenChild];

        }


        if (fit[notChosenParent] > tfit[notChosenChild])

        {

            for (int i = 0; i < dimension; i++)

            {

                chromosome[notChosenParent][i] =
paroff[notChosenChild][i];

            }

            fit[notChosenParent] = tfit[notChosenChild];

        }

}


void BothParentReplacement(double chromosome[pSize][dimension], double
fit[pSize], double paroff[4][dimension], double tfit[4], int parent1, int
parent2)

{

        for (int i = 0; i < dimension; i++)

        {

            chromosome[parent1][i] = paroff[2][i]; // Replace genes of
parent1 with offspring 1

            chromosome[parent2][i] = paroff[3][i]; // Replace genes of
parent2 with offspring 2

        }

        fit[parent1] = tfit[2]; // Update fitness of parent1

        fit[parent2] = tfit[3]; // Update fitness of parent2

}


void binaryTournamentReplacement() {
	int idx1 = getrandom(0, pSize - 1);

	int idx2 = getrandom(0, pSize - 1);


	while (idx1 == idx2) {

		idx2 = getrandom(0, pSize - 1); // Ensure idx1 and idx2 are
different

	}


	int betterIdx = (fit[idx1] < fit[idx2]) ? idx1 : idx2;


	int betterOffspringIdx = (tfit[2] < tfit[3]) ? 2 : 3;


	if (tfit[betterOffspringIdx] < fit[betterIdx]) {

		for (int j = 0; j < dimension; j++) {

			chromosome[betterIdx][j]                             =
paroff[betterOffspringIdx][j];

		}

		fit[betterIdx] = tfit[betterOffspringIdx];

	}

}


void    CombinedReplacement(double    chromosome[pSize][dimension],    double
fit[pSize], double paroff[4][dimension], double tfit[4], int parent1, int
parent2)

{

	// Both Parent Replacement

	for (int i = 0; i < dimension; i++)

	{

		chromosome[parent1][i]  =  paroff[2][i];  //  Replace  genes  of
parent1 with offspring 1

		chromosome[parent2][i]  =  paroff[3][i];  //  Replace  genes  of
parent2 with offspring 2
```

```
}
fit[parent1] = tfit[2]; // Update fitness of parent1
fit[parent2] = tfit[3]; // Update fitness of parent2

// Weak Parent Replacement
int chosenParent, notChosenParent;
int chosenChild, notChosenChild;

// Determine Weak and Strong Parent
if (fit[parent1] > fit[parent2])
{
      chosenParent = parent1;
      notChosenParent = parent2;
}
else
{
      chosenParent = parent2;
      notChosenParent = parent1;
}

// Determine Strong and Weak Child
if (tfit[2] < tfit[3])
{
      chosenChild = 2;
      notChosenChild = 3;
}
else
{
      chosenChild = 3;
```

```
            notChosenChild = 2;
        }

        // Replace the weak parent with the strong child if it improves fitness
        if (fit[chosenParent] > tfit[chosenChild])
        {
            for (int i = 0; i < dimension; i++)
            {
                chromosome[chosenParent][i] = paroff[chosenChild][i];
            }
            fit[chosenParent] = tfit[chosenChild];
        }

        // Replace the non-chosen parent if the non-chosen child is better
        if (fit[notChosenParent] > tfit[notChosenChild])
        {
            for (int i = 0; i < dimension; i++)
            {
                chromosome[notChosenParent][i]                          =
paroff[notChosenChild][i];
            }
            fit[notChosenParent] = tfit[notChosenChild];
        }
}
//---------------------------------------------------------------------------
-----------------------------------------------------

int main()
{
#ifdef EXIT // ERROR
```

```
	cout << "Error MINI_PROJECT: Invalid Parameter Settings\n\nPress Any
Key to Exit..." << endl;

	getch();

	exit(0);

#endif


	HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);


	srand(time(0));


	//-------------------------------------------------------------
-------------------------------------------------------

	// GA Automation

	// Done By: Brandon Ting En Junn 2101751

	//-------------------------------------------------------------
-------------------------------------------------------

	int GACounter = 1, GAModel = 0;

	const int GALength = pow(2, GA.length());


	do

	{

		//Check GA_COMBINATION e.g. (S)00 (C)00 (M)00 (R)00

		GA_TO_GA_COMBINATION();

		if (!isValidCombination(GA_COMBINATION))

		{

			GA = addBinary(GA, "1");

			continue; //[Skip] e.g. (S)00 (C)00 (M)00 (R)00

		}


		//[Proceed] e.g. (S)01 (C)01 (M)01 (R)01

		GAModel++;
```

```
            //GA Directory e.g. GA1

            string gaDirectory = "GA" + to_string(GAModel);

            CreateDirectory(gaDirectory.c_str(), NULL);


            string gaInfo = gaDirectory + "\\\\" + gaDirectory + ".txt";
      //GA Info e.g. GA1.txt

            ofstream outfileo1Info(gaInfo.c_str(), ios::trunc);



            cout << gaDirectory << endl;

            outfileo1Info << gaDirectory << endl << endl;



#if MINI_PROJECT == 1

            outfileo1Info << GA[0] << GA[1] << GA[2] << GA[3] << endl;

            outfileo1Info << GA[4] << GA[5] << GA[6] << GA[7] << endl;

            outfileo1Info << GA[8] << GA[9] << GA[10] << GA[11] << endl;

            outfileo1Info << GA[12] << GA[13] << GA[14] << GA[15] << endl
<< endl;

#else

            outfileo1Info << GA[0] << GA[1] << endl;

            outfileo1Info << GA[2] << GA[3] << endl;

            outfileo1Info << GA[4] << GA[5] << endl;

            outfileo1Info << GA[6] << GA[7] << endl << endl;

#endif

            // getch();


            //--------------------------------------------------------------
---------------------------------------------------------------

            // Benchmark Automation

            // Done By: Brandon Ting En Junn 2101751

            //--------------------------------------------------------------
---------------------------------------------------------------
```

```
            for (; BENCHMARK < 11; BENCHMARK++)

            {

                  initialiseBenchmark_Range(); //Set benchmark mode

                  //Benchmark Directory e.g. GA1/GA1-Sphere

                  string  benchmarkDirectory  =  gaDirectory  +  "\\\\"  +
gaDirectory + benchmarkFunction;

                  CreateDirectory(benchmarkDirectory.c_str(), NULL);



                  //-------------------------------------------------------
-------------------------------------------------------------------

                  // Experiment Automation

                  // Done By: Brandon Ting En Junn 2101751

                  //-------------------------------------------------------
-------------------------------------------------------------------

                  for (int experiment = 0; experiment < 10; experiment++)

                  {

                        //  Result  File  e.g.  GA1/GA1-Sphere/GA1-Sphere-
Result1.txt

                        string  outfile1  =  benchmarkDirectory  +  "\\\\"  +
gaDirectory + benchmarkFunction + "-Result" + to_string(experiment + 1) +
".txt";

                        ofstream outfileo1(outfile1.c_str(), ios::trunc);



                        cout << "Experiment " << experiment + 1 << "..."
<< endl;



                        // CPU Time

                        clock_t start, end;

                        start = clock();



                        //-------------------------------------------------
---------------------------------------------------------------------
-
```

```
                        // Generate Population
                        //-------------------------------------------
---------------------------------------------------------------------------
-
#if MINI_PROJECT == -1
                        SetConsoleTextAttribute(hConsole,
FOREGROUND_GREEN);
                        cout << "Generate Population..." << endl;
                        SetConsoleTextAttribute(hConsole,  FOREGROUND_RED
| FOREGROUND_GREEN | FOREGROUND_BLUE);
#endif


                        for (int i = 0; i < pSize; i++)
                        {
                              for (int j = 0; j < dimension; j++)
                              {
                                    r = getrandom(-rangeMin, rangeMax);
                                    r = r / rangeDiv;
                                    chromosome[i][j] = r;
                              }


#if MINI_PROJECT == -1
                              if (i == 0 || i == pSize - 1)
                              {
                                    cout  <<  "Chromosome  "  <<  i  +  1  <<
endl;
                                    for (int j = 0; j < dimension; j++)
                                    {
                                          cout      <<      fixed      <<
setprecision(3) << chromosome[i][j] << "\t";
                                    }
                                    cout << endl
```

```
                                            << endl;
                                }
#endif


                        }
                        //--------------------------------------------
-----------------------------------------------------------------------
-


                        //--------------------------------------------
-----------------------------------------------------------------------
-
                        // Fitness Evaluation 1
                        //--------------------------------------------
-----------------------------------------------------------------------
-
                        for (int i = 0; i < pSize; i++)
                        {
                              fit[i] = Fitness(chromosome[i]);

      //cout<<fixed<<setprecision(3)<<fit[i]<<endl;
                              sumFit = 0;
                        }


                        lFv = numeric_limits<double>::max();


                        for (int i = 0; i < pSize; i++)
                        {
                              if (fit[i] < lFv)
                              {
                                    lFv = fit[i];
                                    lFvIndex = i;
                                }

                        }



                        lowestGeneFV = lFv;

                        //cout<<"The best is chromosome "<<lFvIndex+1<<"
with fitness of "<<lowestGeneFV<<endl;

                        for (int j = 0; j < dimension; j++)

                        {

                                lowestGene[j] = chromosome[lFvIndex][j];

                                //cout<<lowestGene[j]<<" ";

                        }

                        //cout<<endl;

                        //outfileo1<<"Gen\tMinimum"<<endl;

                        //--------------------------------------------
----------------------------------------------------------------------------
-



                        //--------------------------------------------
----------------------------------------------------------------------------
-

                        // Termination Criteria (Maximum Generation) [Can
modify starting from here (GA, DE, PSO)]

                        //--------------------------------------------
----------------------------------------------------------------------------
-

                        for (int i = 0; i < gen; i++)

                        {

                                //-------------------------------------
----------------------------------------------------------------------------
-----

                                // Selection Operation

                                // Done By: Yeap Chun Hong 2206352
                                //--------------------------------------
---------------------------------------------------------------------------
-----

#if MINI_PROJECT == -1

                                SetConsoleTextAttribute(hConsole,
FOREGROUND_GREEN);

                                cout << "Selection Operation..." << endl;

                                SetConsoleTextAttribute(hConsole,
FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE);

#endif


                                int parent1 = 0, parent2 = 0;



       //****************************************************************
******************************************************

#if MINI_PROJECT == -1

                                /* Best Selection */

                                if (GA_COMBINATION[0][1] == 1)

                                {

                                      RouletteWheelSelection(fit, parent1,
parent2);

                                      cout << "Roulette Wheel Selection" <<
endl;

                                      if (i == 0) { outfileo1Info << "S:
Roulette Wheel Selection" << endl; }

                                }

#elif MINI_PROJECT == -2

                                /* Manual Selection */

                                if (GA_COMBINATION[0][1] == 1)

                                {

                                      DynamicTournamentSelection(fit,
dynamicTournamentSize, parent1, parent2);
```

```
                                        if (i == 0) { outfileo1Info << "S: 
Dynamic Tournament Selection" << endl; }

                                 }

#else

                                 /* Roulette Wheel Selection */

                                 if (GA_COMBINATION[0][0] == 1)

                                 {

                                        RouletteWheelSelection(fit, parent1, 
parent2);

                                        if (i == 0) { outfileo1Info << "S: 
Roulette Wheel Selection" << endl; }

                                 }


                                 /* Tournament Selection */

                                 if (GA_COMBINATION[0][1] == 1)

                                 {

                                        TournamentSelection(fit, 
tournamentSize, parent1, parent2);

                                        if (i == 0) { outfileo1Info << "S: 
Tournament Selection" << endl; }

                                 }

#endif

#if MINI_PROJECT == 1

                                 /* Linear Ranking Selection */

                                 if (GA_COMBINATION[0][2] == 1)

                                 {

                                        LinearRankingSelection(fit, parent1, 
parent2);

                                        if (i == 0) { outfileo1Info << "S: 
Linear Ranking Selection" << endl; }

                                 }
```

```
                                /* Dynamic Tournament Selection */

                                if (GA_COMBINATION[0][3] == 1)

                                {

                                      DynamicTournamentSelection(fit,
dynamicTournamentSize, parent1, parent2);

                                      if (i == 0) { outfileo1Info << "S:
Dynamic Tournament Selection" << endl; }

                                }

#endif


      //*****************************************************************
******************************************************


                                tfit[0] = fit[parent1];

                                tfit[1] = fit[parent2];


                                for (int j = 0; j < dimension; j++)

                                {

                                      paroff[0][j]                      =
chromosome[parent1][j];

                                      paroff[1][j]                      =
chromosome[parent2][j];

                                }


#if MINI_PROJECT == -1

                                // Selected Parent 1

                                cout << "Chromosome " << parent1 + 1 <<
endl;

                                for (int j = 0; j < dimension; j++)

                                {

                                      cout << fixed << setprecision(3) <<
chromosome[parent1][j] << "\t";

                                }
                                cout << endl << endl;


                                // Selected Parent 2

                                cout  <<  "Chromosome  "  <<  parent2  +  1  <<
endl;if (i == 0) {  }

                                for (int j = 0; j < dimension; j++)

                                {

                                      cout  <<  fixed  <<  setprecision(3)  <<
chromosome[parent2][j] << "\t";

                                }

                                cout << endl << endl;

#endif

                                //---------------------------------------
--------------------------------------------------------------------------
-----


                                //---------------------------------------
--------------------------------------------------------------------------
-----

                                // Crossover Operation

                                // Done By: Loh Chia Heung 2301684

                                //---------------------------------------
--------------------------------------------------------------------------
-----

#if MINI_PROJECT == -1

                                SetConsoleTextAttribute(hConsole,
FOREGROUND_GREEN);

                                cout << "Crossover Operation..." << endl;

                                SetConsoleTextAttribute(hConsole,
FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE);

#endif
```

```
	//*****************************************************************
*****************************************************

#if MINI_PROJECT == -1

					/* Best Crossover */

					if (GA_COMBINATION[1][1] == 1)

					{

						UniformCrossover(chromosome, paroff,
dcp, parent1, parent2);

						cout << "Uniform Crossover" << endl;

						if (i == 0) { outfileo1Info << "C:
Uniform Crossover" << endl; }

					}

#elif MINI_PROJECT == -2

					/* Manual Crossover */

					if (GA_COMBINATION[1][1] == 1)

					{

					    CombinedCrossover(chromosome,   paroff,
dcp, parent1, parent2);

					    if  (i  ==  0)  {  outfileo1Info  <<  "C:
Combined Crossover" << endl; }

					}

#else

					/* Uniform Crossover */

					if (GA_COMBINATION[1][0] == 1)

					{

						UniformCrossover(chromosome, paroff,
dcp, parent1, parent2);

						if (i == 0) { outfileo1Info << "C:
Uniform Crossover" << endl; }

					}


					/* Shuffle Crossover */
```

```
                                if (GA_COMBINATION[1][1] == 1)
                                {
                                      ShuffleCrossover(chromosome, paroff,
dcp, parent1, parent2);
                                      if (i == 0) { outfileo1Info << "C:
Shuffle Crossover" << endl; }
                                }
#endif
#if MINI_PROJECT == 1
                                /* Arithmetic Crossover */
                                if (GA_COMBINATION[1][2] == 1)
                                {
                                    ArithmeticCrossover(chromosome, paroff,
dcp, parent1, parent2);
                                    if (i == 0) { outfileo1Info << "C:
Arithmetic Crossover" << endl; }
                                }


                                /* Combined Crossover */
                                if (GA_COMBINATION[1][3] == 1)
                                {
                                    CombinedCrossover(chromosome,   paroff,
dcp, parent1, parent2);
                                    if (i == 0) { outfileo1Info << "C:
Combined Crossover" << endl; }
                                }
#endif

        //*****************************************************************
******************************************************


#if MINI_PROJECT == -1
                                // Crossover Child 1
                                cout << "Child 1" << endl;

                                for (int j = 0; j < dimension; j++)

                                {

                                      cout << fixed << setprecision(3) <<
paroff[2][j] << "\t";

                                }

                                cout << endl << endl;


                                // Crossover Child 2

                                cout << "Child 2" << endl;

                                for (int j = 0; j < dimension; j++)

                                {

                                      cout  <<  fixed  <<setprecision(3)  <<
paroff[3][j] << "\t";

                                }

                                cout << endl << endl;

#endif

                                //--------------------------------------------
----------------------------------------------------------------------------
-----



                                //--------------------------------------------
----------------------------------------------------------------------------
-----

                                // Mutation Operation

                                // Done By: Brandon Ting En Junn 2101751

                                //--------------------------------------------
----------------------------------------------------------------------------
-----

#if MINI_PROJECT == -1

                                SetConsoleTextAttribute(hConsole,
FOREGROUND_GREEN);

                                cout << "Mutation Operation..." << endl;
                                SetConsoleTextAttribute(hConsole,
FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE);
#endif



      //*******************************************************************
******************************************************
#if MINI_PROJECT == -1
                                /* Best Mutation */
                                if (GA_COMBINATION[2][1] == 1)
                                {
                                      ReversingMutation();
                                      cout << "Reversing Mutation" << endl;
                                      if (i == 0) { outfileo1Info << "M:
Reversing Mutation" << endl; }
                                }
#elif MINI_PROJECT == -2
                                /* Manual Mutation */
                                if (GA_COMBINATION[2][1] == 1)
                                {
                                      HybridComparingMutation();
                                      if (i == 0) { outfileo1Info << "M:
Hybrid Comparing Mutation" << endl; }
                                }
#else
                                /* Reversing Mutation */
                                if (GA_COMBINATION[2][0] == 1)
                                {
                                      ReversingMutation();
                                      if (i == 0) { outfileo1Info << "M:
Reversing Mutation" << endl; }
                                }
                                /* Random Mutation */
                                if (GA_COMBINATION[2][1] == 1)
                                {
                                      RandomMutation();
                                      if (i == 0) { outfileo1Info << "M:
Random Mutation" << endl; }
                                }
#endif
#if MINI_PROJECT == 1
                                /* Simple Inversion Mutation */
                                if (GA_COMBINATION[2][2] == 1)
                                {
                                      SimpleInversionMutation();
                                      if (i == 0) { outfileo1Info << "M:
Simple Inversion Mutation" << endl; }
                                }

                                /* Gostan Mutation */
                                if (GA_COMBINATION[2][3] == 1)
                                {
                                      HybridComparingMutation();
                                      if (i == 0) { outfileo1Info << "M:
Hybrid Comparing Mutation" << endl; }
                                }
#endif

      //*******************************************************************
******************************************************

#if MINI_PROJECT == -1
```

```
                                // Mutation Child 1

                                cout << "Child 1" << endl;

                                for (int j = 0; j < dimension; j++)

                                {

                                      cout << fixed << setprecision(3) <<
paroff[2][j] << "\t";

                                }

                                cout << endl << endl;


                                // Mutation Child 2

                                cout << "Child 2" << endl;

                                for (int j = 0; j < dimension; j++)

                                {

                                      cout << fixed << setprecision(3) <<
paroff[3][j] << "\t";

                                }

                                cout << endl << endl;

#endif

                                //------------------------------------------
---------------------------------------------------------------------------
-----


                                //------------------------------------------
---------------------------------------------------------------------------
-----

                                // Fitness Evaluation 2

                                //------------------------------------------
---------------------------------------------------------------------------
-----

                                fit1 = Fitness(paroff[2]);

                                sumFit = 0;


                                fit2 = Fitness(paroff[3]);
                                sumFit = 0;


                                tfit[2] = fit1;

                                tfit[3] = fit2;

                                //----------------------------------------
---------------------------------------------------------------------------
-----



                                //----------------------------------------
---------------------------------------------------------------------------
-----

                                // Replacement Operation

                                // Done By: Ling Ji Xiang 2104584

                                //----------------------------------------
---------------------------------------------------------------------------
-----

#if MINI_PROJECT == -1

                                SetConsoleTextAttribute(hConsole,
FOREGROUND_GREEN);

                                cout << "Replacement Operation..." << endl;

                                SetConsoleTextAttribute(hConsole,
FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE);

#endif



      //*****************************************************************
*****************************************************

#if MINI_PROJECT == -1

                                /* Best Replacement */

                                if (GA_COMBINATION[3][1] == 1)

                                {

                                      WeakParentReplacement(chromosome,
fit, paroff, tfit, parent1, parent2);
```

```
                                                cout << "Weak Parent Replacement" <<
endl << endl << endl;

                                                if (i == 0) { outfileo1Info << "R:
Weak Parent Replacement" << endl << endl << endl; }

                                        }

#elif MINI_PROJECT == -2

                                        /* Manual Replacement */

                                        if (GA_COMBINATION[3][1] == 1)

                                        {

                                                CombinedReplacement(chromosome, fit,
paroff, tfit, parent1, parent2);

                                                if (i == 0) { outfileo1Info << "R:
Combined Replacement" << endl << endl << endl; }

                                        }

#else

                                        /* Weak Parent Replacement */

                                        if (GA_COMBINATION[3][0] == 1)

                                        {

                                                WeakParentReplacement(chromosome,
fit, paroff, tfit, parent1, parent2);

                                                if (i == 0) { outfileo1Info << "R:
Weak Parent Replacement" << endl << endl << endl; }

                                        }


                                        /* Both Parent Replacement */

                                        if (GA_COMBINATION[3][1] == 1)

                                        {

                                                BothParentReplacement(chromosome,
fit, paroff, tfit, parent1, parent2);

                                                if (i == 0) { outfileo1Info << "R:
Both Parent Replacement" << endl << endl << endl; }

                                        }

#endif
```

```
#if MINI_PROJECT == 1
                                /* Binary Tournament Replacement */
                                if (GA_COMBINATION[3][2] == 1)
                                {
                                      binaryTournamentReplacement();
                                      if (i == 0) { outfileo1Info << "R:
Binary Tournament Replacement" << endl << endl << endl; }
                                }


                                /* Combined Replacement */
                                if (GA_COMBINATION[3][3] == 1)
                                {
                                      CombinedReplacement(chromosome, fit,
paroff, tfit, parent1, parent2);
                                      if (i == 0) { outfileo1Info << "R:
Combined Replacement" << endl << endl << endl; }
                                }
#endif

      //*****************************************************************
*******************************************************

                                //------------------------------------------
---------------------------------------------------------------------------
-----


#if MINI_PROJECT == -1
                                cout << "SUCCESSFULLY COMPLETED
DEMO...\nPress Any Key to Exit..." << endl;
                                getch();
                                exit(0);
#endif


                                lFv = numeric_limits<double>::max();
                                for (int j = 0; j < pSize; j++)

                                {

                                      if (fit[j] < lFv)

                                      {

                                              lFv = fit[j];

                                              lFvIndex = j;

                                      }


                                      //cout<<lFv<<"
"<<lowestGeneFV<<endl;

                                      if (lFv < lowestGeneFV)

                                      {

                                              lowestGeneFV = lFv;


                                              for (int k = 0; k < dimension;
k++)

                                              {

                                                      lowestGene[k]         =
chromosome[lFvIndex][k];

      //cout<<lowestGene[k]<<" ";

                                              }

                                      }

                                }


                                // Negative Fitness Value Checking

                                if (lFv < 0) {

                                      if (BENCHMARK != 10 || lFv < -1) {

                                              cout   <<   "Error   ALGORITHM:
Invalid Fitness Value\n\nPress Any Key to Exit..." << endl;

                                              getch();
                                        exit(0);

                                }

                        }


                        outfileo1 << setprecision(6) << lFv << endl;

                }  //  Termination  Criteria  (Maximum  Generation)
[Can modify starting from here (GA, DE, PSO)] LOOP

                //---------------------------------------------
-----------------------------------------------------------------------


                lFv = numeric_limits<double>::max();

                for (int j = 0; j < pSize; j++)

                {

                        if (fit[j] < lFv)

                        {

                                lFv = fit[j];

                                lFvIndex = j;

                        }

                }


                outfileo1 << endl << endl;

                cout << "Result" << endl;


                // Output Final Generation Lowest

                //cout<<fixed<<setprecision(3)<<lFv<<"
"<<lFvIndex+1<<endl<<endl;

                //for(int j = 0 ; j < dimension ; j++)

                //{

                //
        cout<<fixed<<setprecision(3)<<chromosome[lFvIndex][j]<<"\t";
```

```
                        //
      outfileo1<<setprecision(6)<<chromosome[lFvIndex][j]<<"\n";
                        //}


                        // Output Lowest Gene in Experiment
                        cout << fixed << setprecision(3) << lowestGeneFV
<< endl << endl;
                        for (int j = 0; j < dimension; j++)
                        {
                              cout   <<   fixed   <<   setprecision(3)   <<
lowestGene[j] << "\t";
                              outfileo1     <<     setprecision(6)     <<
lowestGene[j] << "\n";
                        }


                        cout << endl;
                        outfileo1 << endl;


                        end = clock();
                        cout   <<   "Time   required   for   execution:   "   <<
(double)(end - start) / CLOCKS_PER_SEC << " seconds." << "\n\n";
                        cout << endl << endl;
                        outfileo1    <<    (double)(end    -    start)    /
CLOCKS_PER_SEC << "\n\n";
                        outfileo1.close();


                        resetExperiment();
                  } //Experiment Automation LOOP
                  //---------------------------------------------------
---------------------------------------------------------------


            } //Benchmark Automation LOOP
```

```
             //------------------------------------------------------------
------------------------------------------------------------


#if MINI_PROJECT == -2

             cout << "SUCCESSFULLY COMPLETED...\nPress Any Key to Exit..."
<< endl;

             getch();

             exit(0);

#endif


             outfileo1Info.close();

             BENCHMARK = 1;                    //Reset benchmark mode

             GA = addBinary(GA, "1"); //Next GA combination

      } while (++GACounter <= GALength); //GA Automation LOOP

      //--------------------------------------------------------------------
------------------------------------------------------


      cout << "SUCCESSFULLY COMPLETED...\nPress Any Key to Exit..." << endl;

      getch();


      return 0;

}
```

PSO Source Code

```
#define _USE_MATH_DEFINES

#include <iostream>

#include <conio.h>

#include <fstream>

#include <cstdlib>

#include <stdio.h>

#include <time.h>

#include <string.h>

#include <iomanip>

#include <math.h>

#include <windows.h>

using namespace std;


/********************************************************   CODE   GUIDELINE
*******************************************************

* Parameter Settings (PSO with constant w and vc)


- Benchmark Function

- External Function

* Update Velocity Function

* Update Position Function


-> Benchmark Automation

      -> Experiment Automation

            -> Generate Population

            -> Fitness Evaluation 1

            ->  Termination  Criteria  (Maximum  Generation)  [Can  modify
starting from here (GA, DE, PSO)]

                  *> Update Velocity
```

```
                    *> Update Position

                    -> Fitness Evaluation 2

********************************************************   CODE   GUIDELINE
********************************************************/



//---------------------------------------------------------------------------
-----------------------------------------------------

// Parameter Settings (PSO with constant w and vc)

//---------------------------------------------------------------------------
-----------------------------------------------------

#define DEMO 0



const double w = 0.7;
      //Inertia Weight

const double c1 = 2, c2 = 2;                                     //C
Constant

//---------------------------------------------------------------------------
-----------------------------------------------------

#define EXPERIMENT 10
      //No. of Experiments



// #define getrandom(min,max) ((rand()%(((max)+1)-(min)))+(min))

#define getrandom(min, max) (((double)rand() / RAND_MAX) * ((max) - (min)) +
(min))

#define gen 2000
      //number of iterations (number of generations)

#define pSize 40
      //number of particle (population size)

#define dimension 30
      //number of bits (dimension size)



double particlePosition[pSize][dimension] = {0};    //particle position

double particleVelocity[pSize][dimension] = {0};    //particle velocity

double particlePBest[pSize][dimension] = {0};       //particle PBest
double particlePBestFV[pSize] = {0};

double particleGBest[dimension] = {0};              //particle GBest

double particleGBestFV = 0;


double fit[pSize] = {0};                            //fitness value for each
particle generation

double fv = 0, sumFit = 0;

double r = 0;

int GBestIndex = 0;
      //GBest index of each particle generation for potential GBest


int BENCHMARK = 1;

string benchmarkFunction = "";

int rangeMin = 0, rangeMax = 0, rangeDiv = 1000;


double vcMin = 0, vcMax = 0;                                      //
Velocity Clamping (Velocity Limits)

double posMin = 0, posMax = 0;                                    //
Position Limits


//---------------------------------------------------------------------------
-------------------------------------------------

// Benchmark Function

//---------------------------------------------------------------------------
-------------------------------------------------

// No.1 - Sphere Function +-5.12

double Sphere(double a[])

{

      sumFit = 0;


      for (int j = 0; j < dimension; j++)

      {
        fv = pow(a[j], 2);

        sumFit = sumFit + fv;

    }


    return sumFit;

}


// No.2 - Ackley Function +-32.768

// Done By: Yeap Chun Hong 2206352

double Ackley(double a[])

{

    sumFit = 0;

    double sum1 = 0, sum2 = 0;


    for (int j = 0; j < dimension; j++)

    {

        sum1 += pow(a[j], 2);

        sum2 += cos(2 * M_PI * a[j]);

    }


    double term1 = -20 * exp(-0.2 * sqrt(sum1 / dimension));

    double term2 = exp(sum2 / dimension);


    sumFit = term1 - term2 + 20 + exp(1);


    return sumFit;

}


// No.3 - Rastrigin Function +-5.12
// Done By: Yeap Chun Hong 2206352

double Rastrigin(double a[])

{

        sumFit = 0;


        for (int j = 0; j < dimension; j++)

        {

                fv = (pow(a[j], 2)) - (10 * cos(2 * M_PI * a[j]));

                sumFit = sumFit + fv;

        }


        sumFit += 10 * dimension;


        return sumFit;

}


// No.4 - Zakharov Function -5, +10

// Done By: Brandon Ting En Junn 2101751

double Zakharov(double a[])

{

        sumFit = 0;

        double sumFit1 = 0, sumFit2 = 0, sumFit3 = 0;


        // sumFit1

        for (int j = 0; j < dimension; j++)

        {

                fv = pow(a[j], 2);

                sumFit1 = sumFit1 + fv;

        }
```

```
	// sumFit2
	for (int j = 0; j < dimension; j++)
	{
		fv = 0.5 * (j + 1) * a[j];
		sumFit2 = sumFit2 + fv;
	}
	sumFit2 = pow(sumFit2, 2);

	// sumFit3
	for (int j = 0; j < dimension; j++)
	{
		fv = 0.5 * (j + 1) * a[j];
		sumFit3 = sumFit3 + fv;
	}
	sumFit3 = pow(sumFit3, 4);

	sumFit = sumFit1 + sumFit2 + sumFit3;

	return sumFit;
}

// No.5 - Axis Parallel Hyper-Ellipsoid Function +-5.12
// Done By: Loh Chia Heung 2301684
double AxisParallel(double a[])
{
	sumFit = 0;

	for (int j = 0; j < dimension; j++)
```

```
        {
                fv = (j + 1) * pow(a[j], 2);
                sumFit = sumFit + fv;
        }

        return sumFit;
}

// No.6 - Griewank Function +-600
// Done By: Loh Chia Heung 2301684
double Griewank(double a[])
{
        sumFit = 0;
        double product = 1.0;

        for (int j = 0; j < dimension; j++)
        {
                sumFit += (a[j] * a[j]) / 4000.0;
                product *= cos(a[j] / sqrt(j + 1));
        }

        sumFit = sumFit - product + 1;

        return sumFit;
}

// No.7 - Sum of Different Powers function +-1.00
// Done By: Yeap Chun Hong 2206352
double SumOfDifferentPowers(double a[])
```

```
{
	sumFit = 0;

	for (int j = 0; j < dimension; j++)
	{
		fv = pow(fabs(a[j]), j + 2);
		sumFit = sumFit + fv;
	}

	return sumFit;
}

// No.8 - Rotated Hyper-Ellipsoid Function +-65.536
// Done By: Brandon Ting En Junn 2101751
double Rotated(double a[])
{
	sumFit = 0;

	for (int i = 0; i < dimension; i++)
	{
		fv = 0;

		for (int j = 0; j <= i; j++)
		{
			fv += pow(a[j], 2);
		}

		sumFit = sumFit + fv;
	}
	return sumFit;
}

// No.9 - Schwefel 2.22 Function +-5 [Updated]
// Done By: Ling Ji Xiang 2104584
double Schwefel(double a[])
{
	sumFit = 0;
	double absolute = 0, sum = 0, product = 1.0;

	for (int i = 0; i < dimension; i++)
	{
		absolute = fabs(a[i]);
		sum += absolute;

		product *= absolute;
	}

	sumFit = sum + product;

	return sumFit;
}

// No.10 - Exponential function Function +-1.00 [f(x) = -1]
// Done By: Ling Ji Xiang 2104584
double Exponential(double a[])
{
	sumFit = 0;
```

```
        // calculate the sum of squares
        for (int j = 0; j < dimension; j++)
        {
                fv = pow(a[j], 2);
                sumFit = sumFit + fv;
        }

        // Calculate the exponential of sum of squares
        sumFit = -exp(-0.5 * sumFit);

        return sumFit;
}

/* Benchmark_Range */
void initialiseBenchmark_Range()
{
        switch (BENCHMARK)
        {
        case 1: // No.1 - Sphere Function +-5.12
                benchmarkFunction = "-Sphere";
                rangeMin = rangeMax = 5120;

                vcMax = 5120 / static_cast<double>(rangeDiv) / 2;
                vcMin = -vcMax;

                posMax = 5120 / static_cast<double>(rangeDiv);
                posMin = -posMax;
                break;
```

```
    case 2: // No.2 - Ackley Function +-32.768

        benchmarkFunction = "-Ackley";

        rangeMin = rangeMax = 32768;


        vcMax = 32768 / static_cast<double>(rangeDiv) / 2;

        vcMin = -vcMax;


        posMax = 32768 / static_cast<double>(rangeDiv);

        posMin = -posMax;

        break;


    case 3: // No.3 - Rastrigin Function +-5.12

        benchmarkFunction = "-Rastrigin";

        rangeMin = rangeMax = 5120;


        vcMax = 5120 / static_cast<double>(rangeDiv) / 2;

        vcMin = -vcMax;


        posMax = 5120 / static_cast<double>(rangeDiv);

        posMin = -posMax;

        break;


    case 4: // No.4 - Zakharov Function -5, +10

        benchmarkFunction = "-Zakharov";

        rangeMin = 5000;

        rangeMax = 10000;


        vcMax = 10000 / static_cast<double>(rangeDiv) / 2;
```

```
            vcMin = -(5000 / static_cast<double>(rangeDiv) / 2);

            posMax = 10000 / static_cast<double>(rangeDiv);
            posMin = -(5000 / static_cast<double>(rangeDiv));
            break;

        case 5: // No.5 - Axis Parallel Hyper-Ellipsoid Function +-5.12
            benchmarkFunction = "-AxisParallel";
            rangeMin = rangeMax = 5120;

            vcMax = 5120 / static_cast<double>(rangeDiv) / 2;
            vcMin = -vcMax;

            posMax = 5120 / static_cast<double>(rangeDiv);
            posMin = -posMax;
            break;

        case 6: // No.6 - Griewank Function +-600
            benchmarkFunction = "-Griewank";
            rangeMin = rangeMax = 600000;

            vcMax = 600000 / static_cast<double>(rangeDiv) / 2;
            vcMin = -vcMax;

            posMax = 600000 / static_cast<double>(rangeDiv);
            posMin = -posMax;
            break;

        case 7: // No.7 - Sum of Different Powers function +-1.00
```

```
            benchmarkFunction = "-SumOfDifferentPowers";
            rangeMin = rangeMax = 1000;

            vcMax = 1000 / static_cast<double>(rangeDiv) / 2;
            vcMin = -vcMax;

            posMax = 1000 / static_cast<double>(rangeDiv);
            posMin = -posMax;
            break;

      case 8: // No.8 - Rotated Hyper-Ellipsoid Function +-65.536
            benchmarkFunction = "-Rotated";
            rangeMin = rangeMax = 65536;

            vcMax = 65536 / static_cast<double>(rangeDiv) / 2;
            vcMin = -vcMax;

            posMax = 65536 / static_cast<double>(rangeDiv);
            posMin = -posMax;
            break;

      case 9: // No.9 - Schwefel 2.22 Function +-5 [Updated]
            benchmarkFunction = "-Schwefel";
            rangeMin = rangeMax = 5000;

            vcMax = 5000 / static_cast<double>(rangeDiv) / 2;
            vcMin = -vcMax;

            posMax = 5000 / static_cast<double>(rangeDiv);
            posMin = -posMax;

            break;


      case 10: // No.10 - Exponential function Function +-1.00 [f(x) = -1]

            benchmarkFunction = "-Exponential";

            rangeMin = rangeMax = 1000;


            vcMax = 1000 / static_cast<double>(rangeDiv) / 2;

            vcMin = -vcMax;


            posMax = 1000 / static_cast<double>(rangeDiv);

            posMin = -posMax;

            break;


      default: // ERROR

            cout << "Error BENCHMARK: Invalid Benchmark Function\n\nPress
Any Key to Exit..." << endl;

            getch();

            exit(0);

      }

}


/* Fitness Function */

double Fitness(double a[])

{

      switch (BENCHMARK)

      {

      case 1: // No.1 - Sphere Function +-5.12

            return Sphere(a);
```

```
    case 2: // No.2 - Ackley Function +-32.768

          return Ackley(a);


    case 3: // No.3 - Rastrigin Function +-5.12

          return Rastrigin(a);


    case 4: // No.4 - Zakharov Function -5, +10

          return Zakharov(a);


    case 5: // No.5 - Axis Parallel Hyper-Ellipsoid Function +-5.12

          return AxisParallel(a);


    case 6: // No.6 - Griewank Function +-600

          return Griewank(a);


    case 7: // No.7 - Sum of Different Powers function +-1.00

          return SumOfDifferentPowers(a);


    case 8: // No.8 - Rotated Hyper-Ellipsoid Function +-65.536

          return Rotated(a);


    case 9: // No.9 - Schwefel 2.22 Function +-5 [Updated]

          return Schwefel(a);


    case 10: // No.10 - Exponential function Function +-1.00 [f(x) = -1]

          return Exponential(a);


    default: // ERROR

          cout << "Error BENCHMARK: Invalid Benchmark Function\n\nPress
Any Key to Exit..." << endl;
```

```
            getch();

            exit(0);

      }

}

//----------------------------------------------------------------------
---------------------------------------------------



//----------------------------------------------------------------------
---------------------------------------------------

// External Function

//----------------------------------------------------------------------
---------------------------------------------------

int charToInt(char c)

{

      if (c >= '0' && c <= '9')

      {

            return c - '0';

      }

      else

      {

            // ERROR

            cout << "Error charToInt(): Invalid Conversion\n\nPress Any Key
to Exit..." << endl;

            getch();

            exit(0);

      }

}


void resetExperiment()

{

      for (int i = 0; i < pSize; i++)
```

```
	{
		for (int j = 0; j < dimension; j++)
		{
			particlePosition[i][j] = 0;
			particleVelocity[i][j] = 0;
			particlePBest[i][j] = 0;

			if (i == 0)
			{
				particleGBest[j] = 0;
			}
		}

		particlePBestFV[pSize] = 0;

		fit[i] = 0;
	}
	particleGBestFV = 0;

	fv = sumFit = 0;
	r = 0;
	GBestIndex = 0;
}
//--------------------------------------------------------------------------
---------------------------------------------------



//--------------------------------------------------------------------------
---------------------------------------------------
// Update Velocity Function -> Refer w | vcMin, vcMax as lower and upper
bounds | getrandom(0, 1000) / 1000;
```

```
// Done By: Brandon Ting En Junn 2101751

//----------------------------------------------------------------------------
-----------------------------------------------------

double PSO_Constant_W_VC(double position, double velocity, double PBest, double GBest)

{

	// Constant W

	double v = w * velocity + c1 * static_cast<double>(getrandom(0, 1000)) / 1000 * (PBest - position) + c2 * static_cast<double>(getrandom(0, 1000)) / 1000 * (GBest - position);


	// Velocity Clamping

	if (v < vcMin) {

		return vcMin;

	} else if (v > vcMax) {

		return vcMax;

	} else {

		return v;

	}

}


void UpdateVelocity()

{

	for (int i = 0; i < pSize; i++)

	{

		for (int j = 0; j < dimension; j++)

		{

			particleVelocity[i][j] = PSO_Constant_W_VC(particlePosition[i][j], particleVelocity[i][j], particlePBest[i][j], particleGBest[j]);

		}

	}
}

//-----------------------------------------------------------------------------------------------------------------------

//-----------------------------------------------------------------------------------------------------------------------

// Update Position Function -> Refer posMin, posMax as lower and upper bounds

// Done By: Brandon Ting En Junn 2101751

//-----------------------------------------------------------------------------------------------------------------------

void UpdatePosition()

{

	for (int i = 0; i < pSize; i++)

	{

		for (int j = 0; j < dimension; j++)

		{

			double position = particlePosition[i][j] + particleVelocity[i][j];


			// Position Limits

			if (position < posMin) {

				particlePosition[i][j] = posMin;

			} else if (position > posMax) {

				particlePosition[i][j] = posMax;

			} else {

				particlePosition[i][j] = position;

			}

		}

	}

}
```

```
//-----------------------------------------------------------------------------------------------------------------------

int main()

{

	HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);


	srand(time(0));


	// Directory e.g. PSO

	string psoDirectory = "PSO";

	CreateDirectory(psoDirectory.c_str(), NULL);


	string psoInfo = psoDirectory + "\\\\" + psoDirectory + ".txt";

	ofstream outfileo1Info(psoInfo.c_str(), ios::trunc);

	cout << "PSO with constant w and vc" << endl;

	outfileo1Info << "PSO with constant w and vc" << endl << endl;

	outfileo1Info << "Inertia Weight (w): " << w << endl;

	outfileo1Info << "Velocity Clamping (vc): Yes" << endl;

	outfileo1Info << "Constant c (c1): " << c1 << endl;

	outfileo1Info << "Constant c (c2): " << c2 << endl;

	outfileo1Info.close();


	//-----------------------------------------------------------------------------------------------------------------------

	// Benchmark Automation

	// Done By: Brandon Ting En Junn 2101751

	//-----------------------------------------------------------------------------------------------------------------------

	for (; BENCHMARK < 11; BENCHMARK++)

	{
		initialiseBenchmark_Range();


		// Directory e.g. PSO/PSO-Sphere

		string benchmarkDirectory = psoDirectory + "\\\\" + psoDirectory
+ benchmarkFunction;

		CreateDirectory(benchmarkDirectory.c_str(), NULL);


		//-----------------------------------------------------------
--------------------------------------------------------------

		// Experiment Automation

		// Done By: Brandon Ting En Junn 2101751

		//-----------------------------------------------------------
--------------------------------------------------------------

		for (int i = 0; i < EXPERIMENT; i++)

		{

			// File e.g. PSO/PSO-Sphere/PSO-Sphere-Result1.txt

			string  outfile1  =  benchmarkDirectory  +  "\\\\"  +
psoDirectory + benchmarkFunction + "-Result" + to_string(i + 1) + ".txt";

			ofstream outfileo1(outfile1.c_str(),ios::trunc);


			cout << "Experiment " << i + 1 << "..."<< endl;


			//CPU Time

			clock_t start, end;

			start = clock();

	//		srand(time(0));


			//----------------------------------------------------
---------------------------------------------------------------------

			// Generate Population

			// Done By: Brandon Ting En Junn 2101751
```

```
                    //--------------------------------------------------
----------------------------------------------------------------------

#if DEMO == 1

                    SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);

                    cout << "Generate Population..." << endl;

                    SetConsoleTextAttribute(hConsole,    FOREGROUND_RED    |
FOREGROUND_GREEN | FOREGROUND_BLUE);

#endif


                    for(int i = 0 ; i < pSize ; i++)

                    {

                    for(int j = 0 ; j < dimension ; j++)

                    {

                           r = getrandom(-rangeMin,rangeMax);

                           r = r / rangeDiv;

                           particlePosition[i][j] = r;

                           particlePBest[i][j] = r;

                    }


#if DEMO == 1

                    if (i == 0 || i == pSize - 1) {

                           cout<<"Particle "<<i+1<<" Position"<<endl;

                           for(int j = 0 ; j < dimension ; j++)

                           {

       cout<<fixed<<setprecision(3)<<particlePosition[i][j]<<"\t";

                           }

                           cout<<endl<<endl;

                           }

#endif

                    }
```

```
#if DEMO == 1
                    for (int i = 0; i < pSize; i++)
                    {
                           if (i == 0 || i == pSize - 1) {
                               cout<<"Particle "<<i+1<<" Velocity"<<endl;
                                 for(int j = 0 ; j < dimension ; j++)
                                 {
cout<<fixed<<setprecision(3)<<particleVelocity[i][j]<<"\t";
                                 }
                                 cout<<endl<<endl;
                           }
                    }
#endif
                    // getch();
                    //-------------------------------------------------------
----------------------------------------------------------------------


                    //-------------------------------------------------------
----------------------------------------------------------------------
                    // Fitness Evaluation 1
                    // Done By: Brandon Ting En Junn 2101751
                    //-------------------------------------------------------
----------------------------------------------------------------------
                    for(int i = 0 ; i < pSize ; i++)
                    {
                           fit[i] = Fitness(particlePosition[i]);
                        // cout<<fixed<<setprecision(3)<<fit[i]<<endl;
                           sumFit = 0;
                    }
                    //--------------------------------------------------------
----------------------------------------------------------------------
                    // Determine PBest and GBest
                    // Done By: Brandon Ting En Junn 2101751
                    //--------------------------------------------------------
----------------------------------------------------------------------
#if DEMO == 1
                    SetConsoleTextAttribute(hConsole, FOREGROUND_GREEN);
                    cout << "Initial PBest and GBest..." << endl;
                    SetConsoleTextAttribute(hConsole,    FOREGROUND_RED    |
FOREGROUND_GREEN | FOREGROUND_BLUE);
#endif
                    //PBest
                    for(int i = 0 ; i < pSize ; i++)
                    {
                          particlePBestFV[i] = fit[i];


#if DEMO == 1
                    if (i == 0 || i == pSize - 1) {
                          cout<<"Particle          "<<i+1<<"          PBest:
"<<particlePBestFV[i]<<endl;
                          for(int j = 0 ; j < dimension ; j++)
                          {

      cout<<fixed<<setprecision(3)<<particlePBest[i][j]<<"\t";
                          }
                          cout<<endl<<endl;
                          }
#endif
                    }
```

```
			//GBest
			particleGBestFV = numeric_limits<double>::max();
			for (int i = 0; i < pSize; i++)
			{
				if (particlePBestFV[i] < particleGBestFV)
				{
					particleGBestFV = particlePBestFV[i];
					GBestIndex = i;
				}
			}

#if DEMO == 1
			cout<<"Particle	"<<GBestIndex+1<<"	GBest: "<<particleGBestFV<<endl;
#endif
			for (int j = 0; j < dimension; j++)
			{
				particleGBest[j] = particlePBest[GBestIndex][j];
#if DEMO == 1

	cout<<fixed<<setprecision(3)<<particleGBest[j]<<"\t";
#endif
			}


#if DEMO == 1
			cout << endl << endl;
#endif


			// cout<<endl;
```

```
                    // outfileo1<<"Gen\tMinimum"<<endl;

                    //---------------------------------------------------
-------------------------------------------------------------------



                    //---------------------------------------------------
-------------------------------------------------------------------

                    // Termination Criteria (Maximum Generation) [Can modify
starting from here (GA, DE, PSO)]

                    // Done By: Brandon Ting En Junn 2101751

                    //---------------------------------------------------
-------------------------------------------------------------------

                    for(int i = 0 ; i < gen ; i++)

                    {

                    //---------------------------------------------------
----------------------------------------------------------------------

                           // Update Velocity

                           // Done By: Brandon Ting En Junn 2101751

                           //---------------------------------------------
------------------------------------------------------------------------
----

#if DEMO == 1

                           SetConsoleTextAttribute(hConsole,
FOREGROUND_GREEN);

                           cout << "Update Velocity..." << endl;

                           SetConsoleTextAttribute(hConsole,  FOREGROUND_RED
| FOREGROUND_GREEN | FOREGROUND_BLUE);

#endif



                           UpdateVelocity();



#if DEMO == 1

                           for (int i = 0; i < pSize; i++)

                           {
                                if (i == 0 || i == pSize - 1) {
                                cout<<"Particle "<<i+1<<" Velocity"<<endl;
                                    for(int j = 0 ; j < dimension ; j++)
                                    {

cout<<fixed<<setprecision(3)<<particleVelocity[i][j]<<"\t";
                                    }
                                    cout<<endl<<endl;
                                }
                          }
#endif
                          //--------------------------------------------------
---------------------------------------------------------------------------
----


                          //--------------------------------------------------
---------------------------------------------------------------------------
----
                          // Update Position
                          // Done By: Brandon Ting En Junn 2101751
                          //--------------------------------------------------
---------------------------------------------------------------------------
----
#if DEMO == 1
                          SetConsoleTextAttribute(hConsole,
FOREGROUND_GREEN);
                          cout << "Update Position..." << endl;
                          SetConsoleTextAttribute(hConsole,  FOREGROUND_RED
| FOREGROUND_GREEN | FOREGROUND_BLUE);
#endif


                          UpdatePosition();
```

```
#if DEMO == 1
                            for (int i = 0; i < pSize; i++)
                            {
                                   if (i == 0 || i == pSize - 1) {
                                   cout<<"Particle "<<i+1<<" Position"<<endl;
                                       for(int j = 0 ; j < dimension ; j++)
                                       {
cout<<fixed<<setprecision(3)<<particlePosition[i][j]<<"\t";
                                       }
                                       cout<<endl<<endl;
                                   }
                            }
#endif
                            //-----------------------------------------------
--------------------------------------------------------------------------
----


                            //-----------------------------------------------
------------------------------------------------------------------------
                            // Fitness Evaluation 2
                            // Done By: Brandon Ting En Junn 2101751
                            //-----------------------------------------------
------------------------------------------------------------------------
                            for(int i = 0 ; i < pSize ; i++)
                            {
                                   fit[i] = Fitness(particlePosition[i]);
                            // cout<<fixed<<setprecision(3)<<fit[i]<<endl;
                                   sumFit = 0;
                            }
                            //-----------------------------------------------
------------------------------------------------------------------------
```

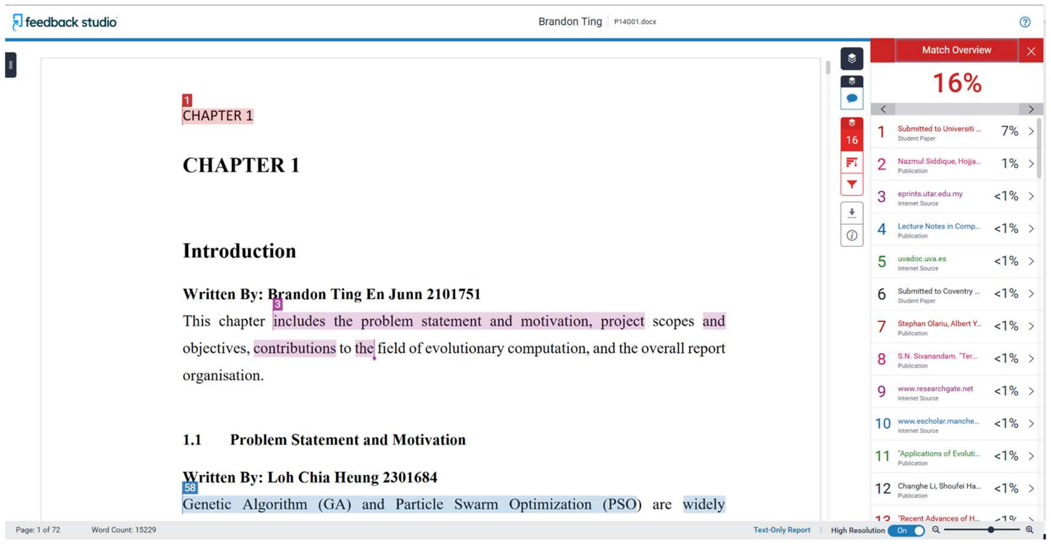
```
                    //-----------------------------------------------------
-------------------------------------------------------------------

                          // Update PBest and GBest

                          // Done By: Brandon Ting En Junn 2101751

                          //-------------------------------------------------
-------------------------------------------------------------------------

#if DEMO == 1

                          SetConsoleTextAttribute(hConsole,
FOREGROUND_GREEN);

                          cout << "Updated PBest and GBest..." << endl;

                          SetConsoleTextAttribute(hConsole,  FOREGROUND_RED
| FOREGROUND_GREEN | FOREGROUND_BLUE);

#endif

                          //PBest

                          for(int i = 0 ; i < pSize ; i++)

                          {

                                 if(fit[i] < particlePBestFV[i])

                                 {

                                        particlePBestFV[i] = fit[i];


                                        for (int j = 0; j < dimension; j++)

                                        {

                                               particlePBest[i][j]         =
particlePosition[i][j];

                                        }

                                 }

                          }


#if DEMO == 1

                          for (int i = 0; i < pSize; i++)
```

```
                        {
                              if (i == 0 || i == pSize - 1) {
                              cout<<"Particle      PBest      "<<i+1<<":
"<<particlePBestFV[i]<<endl;
                                  for(int j = 0 ; j < dimension ; j++)
                                  {

cout<<fixed<<setprecision(3)<<particlePBest[i][j]<<"\t";
                                  }
                                  cout<<endl<<endl;
                              }
                        }
#endif

                        //GBest
                        for (int i = 0; i < pSize; i++)
                        {
                              if (particlePBestFV[i] < particleGBestFV)
                              {
                                    particleGBestFV                    =
particlePBestFV[i];
                                    GBestIndex = i;
                              }
                        }
                        // cout<<"The group best particle is personal best
particle "<<GBestIndex+1<<" with fitness of "<<particleGBestFV<<endl;
                        for (int j = 0; j < dimension; j++)
                        {
                              particleGBest[j]                         =
particlePBest[GBestIndex][j];
                              // cout<<particleGBest[j]<<" ";
                        }

                        // cout<<endl;


                // fit1 = 0;

                // fit2 = 0;


#if DEMO == 1

                        cout<<"Particle      "<<GBestIndex<<"      GBest: 
"<<particleGBestFV<<endl;

                        for(int j = 0 ; j < dimension ; j++)

                        {

  cout<<fixed<<setprecision(3)<<particleGBest[j]<<"\t";

                        }

                        cout<<endl<<endl;

#endif


                outfileo1<<setprecision(6)<<particleGBestFV<<endl;

                // cout<<fixed<<setprecision(3)<<particleGBestFV<<endl;

                // getch();


#if DEMO == 1

                        cout << "SUCCESSFULLY COMPLETED DEMO...\nPress Any 
Key to Exit..." << endl;

                        getch();

                        exit(0);

#endif

              }  //Termination  Criteria  (Maximum  Generation)  [Can  modify 
starting from here (GA, DE, PSO)] LOOP

              //--------------------------------------------------------------
------------------------------------------------------------
```

```
            outfileo1<<endl<<endl;

            cout<<"Result"<<endl;


            //Output Final Generation Lowest

      //    cout<<fixed<<setprecision(3)<<lFv<<"
"<<lFvIndex+1<<endl<<endl;

      //        for(int j = 0 ; j < dimension ; j++)

      //        {

      //
cout<<fixed<<setprecision(3)<<chromosome[lFvIndex][j]<<"\t";

      //
outfileo1<<setprecision(6)<<chromosome[lFvIndex][j]<<"\n";

      //        }


                  //Output Group Best in Experiment

            cout<<fixed<<setprecision(3)<<particleGBestFV<<endl<<endl;

            for(int j = 0 ; j < dimension ; j++)

            {

                  cout<<fixed<<setprecision(3)<<particleGBest[j]<<"\t";

      outfileo1<<setprecision(6)<<particleGBest[j]<<"\n";

            }


                cout<<endl;

                outfileo1<<endl;

                end = clock();

                cout<<"Time  required  for  execution:  "<<  (double)(end-
start)/CLOCKS_PER_SEC<<" seconds."<<"\n\n";

                outfileo1<<(double)(end-start)/CLOCKS_PER_SEC<<"\n\n";

                outfileo1.close();
```

```
                    cout<<endl<<endl;


                    resetExperiment();

             } //Experiment Automation LOOP

             //------------------------------------------------------------
------------------------------------------------------------


      } //Benchmark Automation LOOP

      //---------------------------------------------------------------------
-------------------------------------------------------


      cout << "SUCCESSFULLY COMPLETED...\nPress Any Key to Exit..." << endl;

      getch();


      return 0;

}
```

PLAGIARISM CHECK RESULT



feedback studio

Brandon Ting | P14G01.docx

1
CHAPTER 1

CHAPTER 1

Introduction

Written By: Brandon Ting En Junn 2101751

This chapter includes the problem statement and motivation, project scopes and objectives, contributions to the field of evolutionary computation, and the overall report organisation.

1.1 Problem Statement and Motivation

Written By: Loh Chia Heung 2301684

Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) are widely

Match Overview

16%

1	Submitted to Universiti ... Student Paper	7%
2	Nazmul Siddique, Hojja... Publication	1%
3	eprints.utar.edu.my Internet Source	<1%
4	Lecture Notes in Comp... Publication	<1%
5	uvadoc.uva.es Internet Source	<1%
6	Submitted to Coventry ... Student Paper	<1%
7	Stephan Olariu, Albert Y... Publication	<1%
8	S.N. Sivanandam. "Ter... Publication	<1%
9	www.researchgate.net Internet Source	<1%
10	www.escholar.manche... Internet Source	<1%
11	"Applications of Evoluti... Publication	<1%
12	Changhe Li, Shoufei Ha... Publication	<1%
13	"Recent Advances of H...	<1%

Page: 1 of 72 Word Count: 15229

Text-Only Report | High Resolution On

Form Title: Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	Brandon Ting En Junn, Ling Ji Xiang, Loh Chia Heung, Yeap Chun Hong
ID Number(s)	21ACB01751, 21ACB04584, 23ACB01684, 22ACB06352
Programme / Course	Bachelor of Computer Science (Honours)
Title of Final Year Project	Performance Comparison of Genetic Algorithm with Particle Swarm Optimisation Using Benchmark Functions

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceed the limits approved by UTAR)**
Overall similarity index: <u>16</u>% **Similarity by source** Internet Sources: <~<u>18</u> % Publications: <~<u>44</u> % Student Papers: <~<u>25</u> %	
Number of individual sources listed of more than 3% similarity: <u>1</u>	
Parameters of originality required, and limits approved by UTAR are as Follows: **(i) Overall similarity index is 20% and below, and** **(ii) Matching of individual sources listed must be less than 3% each, and** **(iii) Matching texts in continuous block must not exceed 8 words** ***Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.***	

<u>Note:</u> Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

______________________________	______________________________
Signature of Supervisor	Signature of Co-Supervisor
Name: <u>Ts Dr Lim Seng Poh</u>	Name: __________________________
Date: __________________________	Date: __________________________