DYNAMIC ELECTRONIC ASSET ALLOCATION COMPARING GENETIC ALGORITHM WITH PARTICLE SWARM OPTIMIZATION

A Thesis

Submitted to the Faculty

of

Purdue University

by

Md Saiful Islam

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

December 2018

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Lauren A. Christopher, Chair

Department of Electrical and Computer Engineering

Dr. Brian S. King

Department of Electrical and Computer Engineering

Dr. Mohamed El-Sharkawy

Department of Electrical and Computer Engineering

Approved by:

Dr. Brian S. King

Head of the Graduate Program

Proclaim! (or read!) in the name of thy Lord and Cherisher, Who created. Created man, out of a clot of congealed blood. Proclaim! And thy Lord is Most Bountiful. This thesis is dedicated to my respected parents who struggled their entire life, siblings, grandparents, family members, friends, and lastly my loving wife.

ACKNOWLEDGMENTS

First of all, I would like to express my heartiest gratitude to my professor Dr. Christopher for giving me the opportunity to work with her in this project. Her in-depth knowledge, experience and invaluable capability of helping at any time have been a precious asset for accomplishing my task. She has been a great teacher and friend to me. I'd also like to thank Dr. King and Dr. El-Sharkawy for being a knowledge-worthy and true counselors as the other committee members. I appreciate support and guidance from Mrs. Sherrie Tucker and the graduate office of IUPUI. Acknowledge to previous contributors to the project: Joshua Reynolds, Jonah Crespo, and Paul Witcher. I want to thank William Boler and Calvin Wieczorek for being dedicated partners and enjoyed their experience, support, and company throughout the project. Would like to express my gratefulness to our Crane support team on this project: Dave Acton and Scot Hawkins for this support, guidance, and sponsorship. Lastly, I want to thank NSWC Crane Division for technical assistance and patronage.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

SYMBOLS

C_1	Constant for Multiplying Personal Best Component
C_2	Constant for Multiplying Neighborhood Best Component
x_i	Particle Position
v_i	Particle Velocity
p_i	Particle's Best Position
x_{pb}	Particle Personal Best Position
x_{gb}	Particle Global Best Position
x_{nb}	Particle's Neighborhood Best Position
C_r	Crossover Rate
M_r	Mutation Rate

ABBREVIATIONS

2D	Two-dimensional Space
3D	Three-dimensional Space
PSO	Particle Swarm Optimization
GA	Genetic Algorithm
GUI	Graphical User Interface
EW	Electronic Warfare
EWAAP	Electronic Warfare Asset Allocation Problem
GB	Global Best Solution
LB	Local Best
NB	Neighborhood Best
RX	Receiver
TX	Transmitter
XCVR	Transceiver

ABSTRACT

Islam, Md. Saiful M.S.E.C.E., Purdue University, December 2018. Dynamic Electronic Asset Allocation Comparing Genetic Algorithm with Particle Swarm Optimization. Major Professor: Lauren Christopher.

The contribution of this research work can be divided into two main tasks: 1) implementing this Electronic Warfare Asset Allocation Problem (EWAAP) with the Genetic Algorithm (GA); 2) Comparing performance of Genetic Algorithm to Particle Swarm Optimization (PSO) algorithm. This research problem implemented Genetic Algorithm in C++ and used QT Data Visualization for displaying three-dimensional space, pheromone, and Terrain. The Genetic algorithm implementation maintained and preserved the coding style, data structure, and visualization from the PSO implementation. Although the Genetic Algorithm has higher fitness values and better global solutions for 3 or more receivers, it increases the running time. The Genetic Algorithm is around (15-30%) more accurate for asset counts from 3 to 6 but requires (26-82%) more computational time. When the allocation problem complexity increases by adding 3D space, pheromones and complex terrains, the accuracy of GA is 3.71% better but the speed of GA is 121% slower than PSO. In summary, the Genetic Algorithm gives a better global solution in some cases but the computational time is higher for the Genetic Algorithm with than Particle Swarm Optimization.

1. INTRODUCTION

1.1 Overview of Problem Statement

Allocation of available resources (assets) is a familiar optimization problem, and electronic asset allocation has been a significant concern for defense systems nowadays. Allocating electronic assets in real-time and responding to dynamics in the battlefield is needed. Researchers tried different optimization techniques and algorithms for solving this problem. The state-of-the-art uses evolutionary computational methods: Particle Swarm Optimization (PSO), and Genetic Algorithm (GA) [1]. The Electronic Warfare research problem refers to the assignment of assets to transmitters in a multidimensional environment created by three-dimensional space, frequency, antenna azimuth, and elevation orientation. The primary goal of the research is to place radio assets in optimal places to preserve adequate coverage of transmitter targets in spatial position and bandwidth assignment. Prime factors for this assignment problem are: power received, receiver sensitivity, target priority, receiver feed-horn power limitations, spatial positioning, frequency and bandwidth coverage, and terrain constraints. A heuristic method like Particle Swarm Optimization was the initial choice for the dynamic and real-time solution of this research. Computational simplicity and quick convergence of this algorithm provides real-time solution complying with the requirement of under 1 second. Optimization algorithms can be stuck in local optimization points and the algorithm's computational efficiency varies with the nature of the problem. The research problem was to compare Particle swarm optimization with another evolutionary computational algorithm: Genetic Algorithm, testing the accuracy, ability to reach a global solution, and computational efficiency. The Genetic Algorithm implementation preserves the style of coding, data structure, and visualization of Particle Swarm Optimization.

1.2 Project Inheritance and Background

The primary goal of the research was to optimize the allocation of assets in three-dimensional space and frequency, and as an optimization algorithm, PSO was the first choice. This research was a real-time problem and needed to be solved in real time or near-real-time. The starting point of this research was initiated by Dr. Russell Eberhart for Expeditionary Electronic Warfare Division, Spectrum Warfare System Department, at Naval Surface Warfare Center (NSWC) Crane. Reynolds [2] developed the initial phase of this research, Crespo [3] designed a preliminary two dimensional environment using Qt.

Reynolds [2] integrated PSO algorithm with the problem of asset allocation in the Electronic Warfare Environment; began significant two-dimensional GUI development, developed the fitness function, and analyzed asset allocation mathematically using PSO. Jonah Crespo [3] solved the PSO for three dimensions, human-in-the-swarm integration with two dimensional user input of moving keep-away boundaries, and designed the initial combination of topological constraints. Later on, we had three more contributors: Paul Witcher, Calvin Wieczorek, and William Boler. Developing the PSO with real-time movement simulation of assets and transmitters in three dimensional space along with tracking and simulated assets to the PSO solutions using the Hungarian algorithm was Witcher's contribution [4]. Boler [5] refactored and added code to make the project more object oriented, and implemented asset's antenna direction in the three dimensional space based on radiation patterns of antennas. His contribution includes adding human-in-the-swarm construction of Pheromones with movable attraction or repelling zones or beacons in three dimensional solution space, and implemented a Meta-PSO, where PSO was used to solve for the weights of the fitness function. Wieczorek [6] implemented advanced terrain models collected from ArcGIS, modified the existing program to display three dimensional environments using Qt Data Visualization, and employed multi-threading technique. Some of the previous contributors published work is available in [7] and [8].

1.3 Project Overview

1.3.1 Particle Swarm Optimization (PSO) Algorithm

Dr. Russell Eberhart and Dr. James Kennedy in 1995 introduced Particle Swarm Optimization (PSO) replicating swarm behavior of flocking birds, fish schooling, and flying insects [9,10]. PSO is an evolutionary computational method for finding optimal solution [11, 12]. The fundamental concept of PSO is analogous to a flock of birds searching for food. Group of birds can be collectively called "swarm", or "population" and each bird is a particle of the swarm. In our research problem, we choose two hundred particles to search for a solution. They are randomly initialized in hyper-dimensional problem space consists of 3D space, frequency, antenna azimuth, and elevation orientation. Distance from the food regulates the random movement of birds, and it is called fitness or "goodness of solution". Our research goal is to place assets in an optimal location for communication or jamming in Electronic Warfare (EW) environment where each asset has transceiver antennas with corresponding bandwidth. Every time power of the asset transceiver antennas are checked to make sure that the feed-horn is not overloaded. Power of the transceiver antennas, the priority of transmitters assigned to assets, distance and spread between transceiver antennas, and the effect of pheromones constructed the fitness value. Reynolds [2] and Boler [5] designed fitness function for the research project as follows, where Wi are weights:

$$Fitness = W_1 \cdot (power) + W_2 \cdot (priority) + W_3 \cdot (spread) + W_4 \cdot (distance) + W_5 \cdot (pheromones) \tag{1.1}$$

In Equation 1.1, the coefficients Wi are the weights associated to determine a "good" solution. Figure 1.1 describes components of the fitness function and Figure 1.2 describes component's weight of the fitness function, developed by the Meta-PSO in [5]. Maximizing or minimizing the fitness value is the goal of PSO, in our case higher the fitness better the solution. Birds will determine their direction of flight

Fitness Component	Description
Power	The total power of transmitters to be assaigned each asset
Priority	The priority of transmitters to be assaigned each asset
Spread	The disperse between each assets
Distance	The centroid distance between transmitters and assets
Pheromone	Attraction or repulsion zone in space

Fig. 1.1. Description of the Components of the Fitness Function.

Fitness Component	Weight Value
Power	44.1664
Priority	99.2189
Spread	0.7956
Distance	0
Pheromone	35.5728

Fig. 1.2. Description of the Component Weight of the Fitness Function.

based on their fitness and listen to their neighbors. Birds will move closer to their final destination in every generation considering their neighborhood best and personal best. Evolving some generations fulfilling termination criteria birds will find optimize solution and converge to the food which is compared to reach the last solution in PSO.

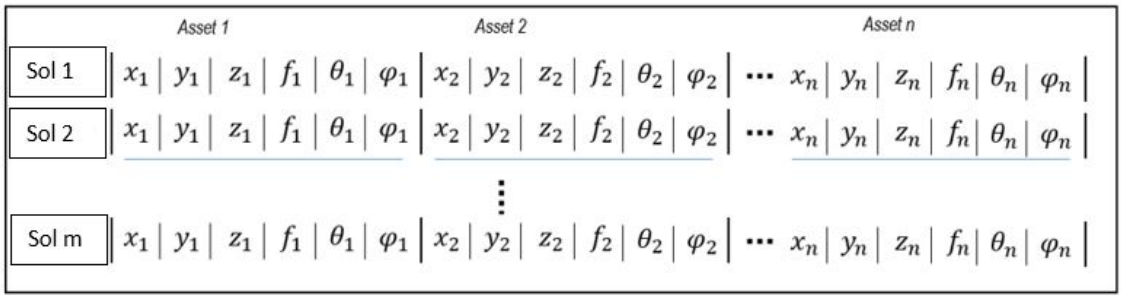

Fig. 1.3. Representation of Solution Data Structure.

As seen in Figure 1.3, each particle (row) is a possible solution. Each particle has dimensions of a 3D location, frequency, with antenna azimuth and elevation orientation. The PSO algorithm is run iteratively until it meets certain termination condition. Each iteration of the PSO calls a generation. The termination conditions are fitness slope, window size, and max generation. The window size is the number of generations PSO will run when its fitness slope remains unchanged. Fitness slope is the best minimum slope angle over window size. Max generation is a limit used in case the termination process takes too long. PSO runs by the following two Equations 1.2 and 1.3:

$$v_{i+1} = v_i I + C_1 Rand()_1(x_{pb} - x_i) + C_2 Rand()_2(x_{nb} + x_i) \tag{1.2}$$

$$x_{i+1} = x_i + v_{i+1} \tag{1.3}$$

In Equation 1.3, x_{i+1} is the next position for each particle in the solution space. It updates from the previous position, x_i and v_{i+1}, the next velocity. In Equation 1.2, I is the Inertia and v_i is the current velocity. C_1 and C_2 are constants, and $Rand()$ is a random generator. $(x_{pb} - x_i)$ is the difference between the current position and the best position found so far so that the particle will be given a larger velocity when it is further away from the current personal best. $(x_{nb} - x_i)$ is the difference between the current position, and the best position found between all the particle i's neighbors. This difference in positions will give a larger velocity when the current position is farther away from the best neighbor's position. There are several functions to follow PSO work flow: "Fly" function updates particles with velocity and position equations. "Evaluate" function finds fitness of each particle. "Update" function updates particle and neighbor particle's velocity and position. Finally, "Terminate" function ends PSO work flow based on fitness slope or window size or max generation. After each generation, every particle has a personal best x_{pb}, which is the best solution that PSO finds for a particle so far, and a neighborhood best x_{nb}, which is the best solution its neighbor has found. These particles are first initialized randomly in 3D solution

space, then they are updated using two Equations 1.2 and 1.3 from [11]. Figure 1.4 represents 2D graphical user interface view of the project. It has four sections. The top left section is the allocation plot where transmitters and assets are placed in 2D space according to their frequency, power, and priority. The top right part is the fitness plot, and it plots fitness value against some generations. Middle plot is the bandwidth plot, and it contains a plot of transmitters and assets in their power rating (dBm) against frequency (MHz). Finally, bottom plot is the text output for the project.

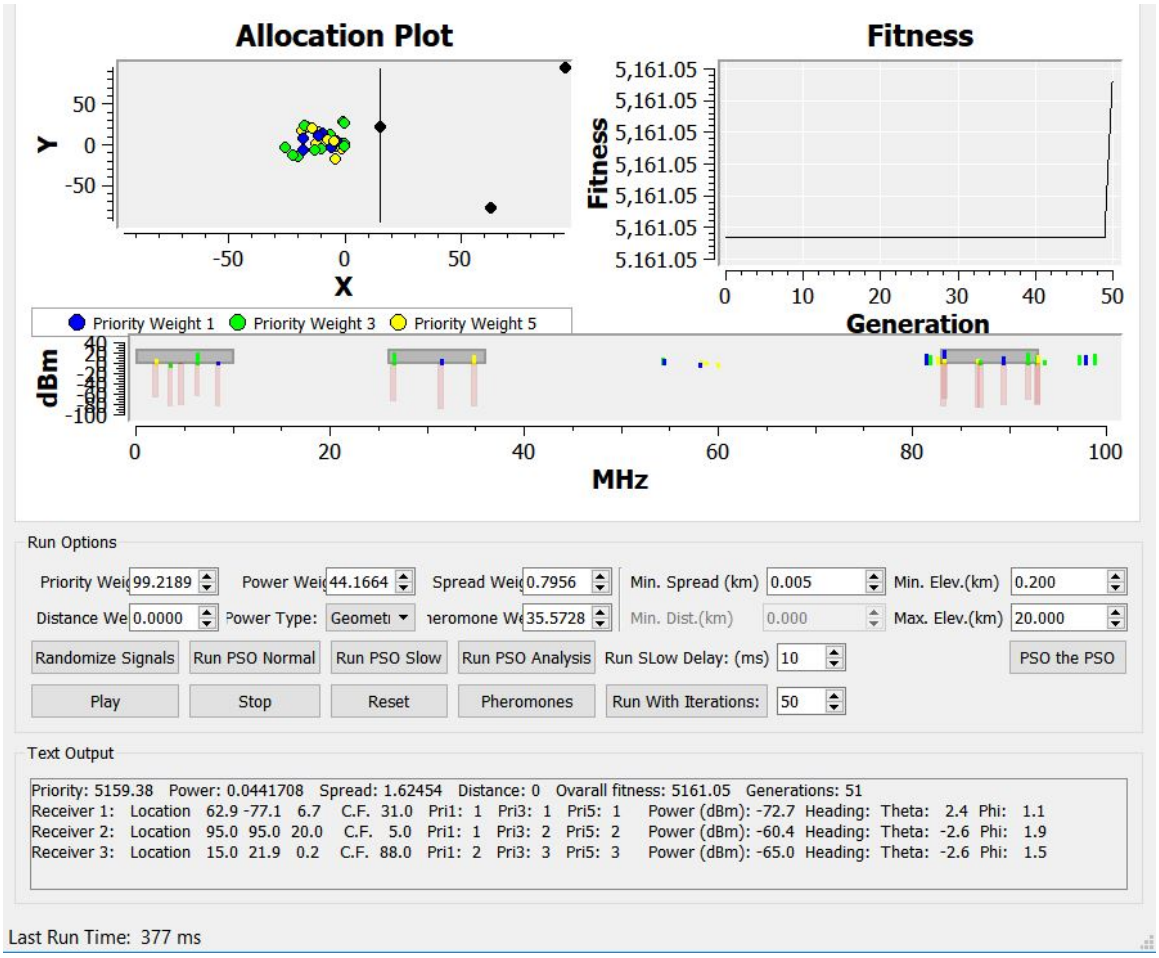

Fig. 1.4. 2D Graphical User Interface View of the Project.

1.4 Literature Review

1.4.1 PSO GA Comparison

The Genetic Algorithm [13] came into light through research done by Holland in the early 1970s. PSO was introduced by Kennedy and Eberhart in 1995 and became popular immediately after its invention. Both algorithms use the evolutionary computing. Hence, research on comparing PSO with GA is a topic of interest for researchers.

In paper [14], Hassan et al. compares PSO and GA with a set of test problems, spacecraft reliability design and telescope array configuration problem. It shows that PSO can find the same global optimal solution as GA, but with superior computational efficiency at 99% confidence level in 7 of 8 test problems. Chaturvedi et al. in [15] compares the execution time of GA, PSO, and krill herd (KH) Algorithm and the result shows that the time taken by KH Algorithm is more than PSO and GA for a hundred iterations, and PSO is the quickest. In paper [16], Shabir et al. studies implementation, features, and effectiveness of GA and PSO algorithms and concludes that hybridization of GA and PSO is a potential solution to particular limitations of these algorithms. Sharma et al. [17] states that GA performs better with large population size while PSO for small population size. Proposed hybrid algorithm was tested on five global optimization test functions (Beale, Booth, Matyas, Levy, Schaffer), and performs better than simple GA and PSO both. Interestingly, the research in [18] by Jones and Karl for the identification of model parameters shows that GA arrives its final parameter values in lesser generations, and this is the opposite of other research.

1.4.2 PSO and GA in Asset Allocation

Our research compares PSO with GA in dynamic asset allocation. There is not much research for asset allocation in real time, and comparing PSO and GA. The dynamic weapon-target assignment (DWTA) problem was solved in [19] by Chen et al.

using GA and two memetic algorithms. The authors found memetic algorithms based on greedy local search generate better DWTA decisions with less computation time than GA, especially for large-scale problems. The work done in [20] by Zeng et al. utilizes discrete particle swarm optimization (DPSO) to solve weapon-target assignment (WTA) problem. The result shows that DPSO outperforms regular GA and GA with greedy eugenics in convergence efficiency and CPU time. Integrated Yard Truck Scheduling and Storage Allocation Problem (YTS-SAP) was demonstrated by Niu et al. in [21] for computation time and solution quality, and PSO outperformed GA. In [22], according to Ohatkar and Bormane PSO show better performance than GA in respect of an improvement in the signal-to-interference ratio (SIR), reduction in interference, required computation time, and generations needed. The work done in [23] by Jiang et al. states a multi-dimensional jamming resource allocation (JRA) problem using hybrid quantum-behaved particle swarm optimization and self-adjustable genetic algorithm (HQPSOGA). They compared standard PSO, quantum-behaved PSO, and integer-value GA. Monte Carlo simulation result shows that HQPSOGA developed better interference capacity efficiently for the jammer's formation than other algorithms. The closet research to ours is in [1] and uses PSO and GA for solving the real-time resource allocation problem. They compare their performance using open source testbed SWARD (System for Weapon Allocation Research and Development). Their experiment shows that PSO can provide a high-quality solution for small-scale problems whereas GA is suitable for largest tested problem cases. Using six firing units and six targets run-time was recorded 0.104 second and with the most extensive set up: nine firing units and nine targets run-time was 1293.084 seconds. Comparing our research using three firing units or assets and thirty targets or transmitters our run time was below 1 second. Additional use of three-dimensional terrain and pheromone in our study takes a significant amount of time where the closet research [1] was not in three dimensions.

1.4.3 Individual Contribution

This research contribution is in two different areas. The first contribution implements the existing asset allocation research problem with the Genetic Algorithm optimization method. It includes maintaining the same coding style and data structure to comply with previous research contributors. The second part of the accomplishment is comparison and performance analysis of GA with PSO. The comparison is made regarding global solution accuracy and computational time. Comparison analysis between PSO and GA is an essential and integral part of this ongoing research as it provides effectiveness and insight into the utilization of the two optimization algorithms.

The following chapters will describe the research work embodied in this text. Chapter 2 discusses the implementation of the research problem in the Genetic Algorithm. It explains implementation GA in respect of coding, mathematical analysis to set up parameters of GA and their effects in comparison and analysis. Chapter 3 demonstrates the comparison of GA and PSO in global solution accuracy and computational time. It also describes the scope of the two optimization algorithms for this research problem. Chapter 4 is the summary of the research contributions and Chapter 5 provides recommendations for future work.

2. GENETIC ALGORITHM IMPLEMENTATION

2.1 Purpose

Our research compares Particle Swarm Optimization (PSO) to Genetic Algorithm (GA). We chose GA because it is an evolutionary heuristic population-based search method [24] like PSO. The GA allows us to diversify the searching behavior. Our goal was to compare the Genetic Algorithm with Particle Swarm Optimization regarding both time complexity, and global solution accuracy.

2.2 Attempts and Implementations

For implementing the Genetic Algorithm for this research project, we tried different approaches. The project already has a coding style and structure for PSO, so the careful modification of the coding was important. Various coding methods were found to be possible for the GA but some were not suitable for our coding style. In the next subsections, we will go through some attempts and their consequences.

2.3 GAlib

GAlib [25] is a C++ library of Genetic Algorithm components developed by CADlab from mechanical engineering department of Massachusetts Institute of Technology (MIT). It is a C++ library of the Genetic Algorithm objects. The library includes implementing tools to use Genetic Algorithms in C++ program using any genetic operators and representation to do optimization. GAlib has the versatility and can be used on various UNIX platforms (Linux, MacOSX, SGI, Sun, HP, DEC, IBM), MacOS and DOS/Windows systems. It can use PVM for distributed, parallel implementations as well as capable of handling the Athena or Motif widget sets, or

MFC/VC++. Although its source code is not in the public domain, but can be used at no cost for non-profit purposes. Its current versions can be found in [26]. To work with the Genetic Algorithm library, we need to work with two classes: a genome and a genetic algorithm. Every genome instance is a single solution to the problem. The Genetic Algorithm object defines the evolution process of GA. The objective function of the Genetic Algorithm determines the fitness of each genome for survival. It utilizes the genome operators and selection strategies to generate new children from parents. To solve any research problem using a Genetic Algorithm there are three things to do: defining a representation, set the genetic operators, and determine the objective function.

GAlib library is a potential source for solving optimization using Genetic Algorithm and some researchers customized it and used it for their problems. For our case the coding style and data structure didn't match and combining different structures is not efficient. Hence, we could not use GAlib and moved forward with another approach.

2.4 Genetic Algorithm Design from First Principles

In [27], Sivanandam and Deepa introduce GA's first principles, beginning with Darwin's theory of evolution: "Survival of the fittest" for Goldberg's Genetic Algorithm. The Genetic Algorithm adapts Darwin's evolution concept in a natural way to solve a problem defined by the fitness function. A single chromosome can be the solution of the Genetic Algorithm and collection of the chromosomes called as a population can also be the solution to any research problem. Moreover, a single chromosome is composed of genes where according to the problem's nature genes value can be either numerical, binary, symbols or characters. The fitness function will measure the suitability of the solution generated by GA. Crossover is the process of creating new offspring chromosomes in population from genes composition of their parents. Few of the chromosomes will also go through the process of mutation in

their genes to maintain diversity in solution. Mutation helps GA to look for a global solution while it is stuck in a local solution. Crossover rate and mutation rate is values within the range of 0 to 1 which controls the crossover and mutation process. Darwinian evolution rule controls and selects chromosome in the population for the next generation, and the fitness value of the chromosome determines it. For maximization problem chromosome with higher fitness value have a higher probability of being selected for next generation and vice versa for a minimization problem. Considering the termination criteria after several generations, the chromosome with their fitness value may converge to the best solution for the problem.

Hermawanto and Denny in [28] present a very illustrative flowchart of the Genetic Algorithm, and Figure 2.1 shows our representation of this.

2.4.1 Initialization and Evaluation

GA was implemented based on the principle of placing assets in an optimal location in the Electronic Warfare (EW) environment. The Power of the transceiver antennas, the priority of transmitters assigned to assets, distance and spread between transceiver antennas, and pheromones constructed the fitness value which is described in Chapter 1 in Equation 1.1. In Chapter 1 Figure 1.1 describes components of the fitness function and Figure 1.2 describes component's weight of the fitness function. From Figure 2.2 every row is a chromosome and dimensions inside chromosomes are genes. Figure 2.2 refers that each particle (row) is a possible solution. Each particle has dimensions of 3D location, frequency, with antenna azimuth and elevation orientation. After initialization, every chromosome fitness value is calculated according to Equation 2.1 in each generation.

$$Fitness = W_1 \cdot (power) + W_2 \cdot (priority) + W_3 \cdot (spread) + W_4 \cdot (distance) + W_5 \cdot (pheromones) \tag{2.1}$$

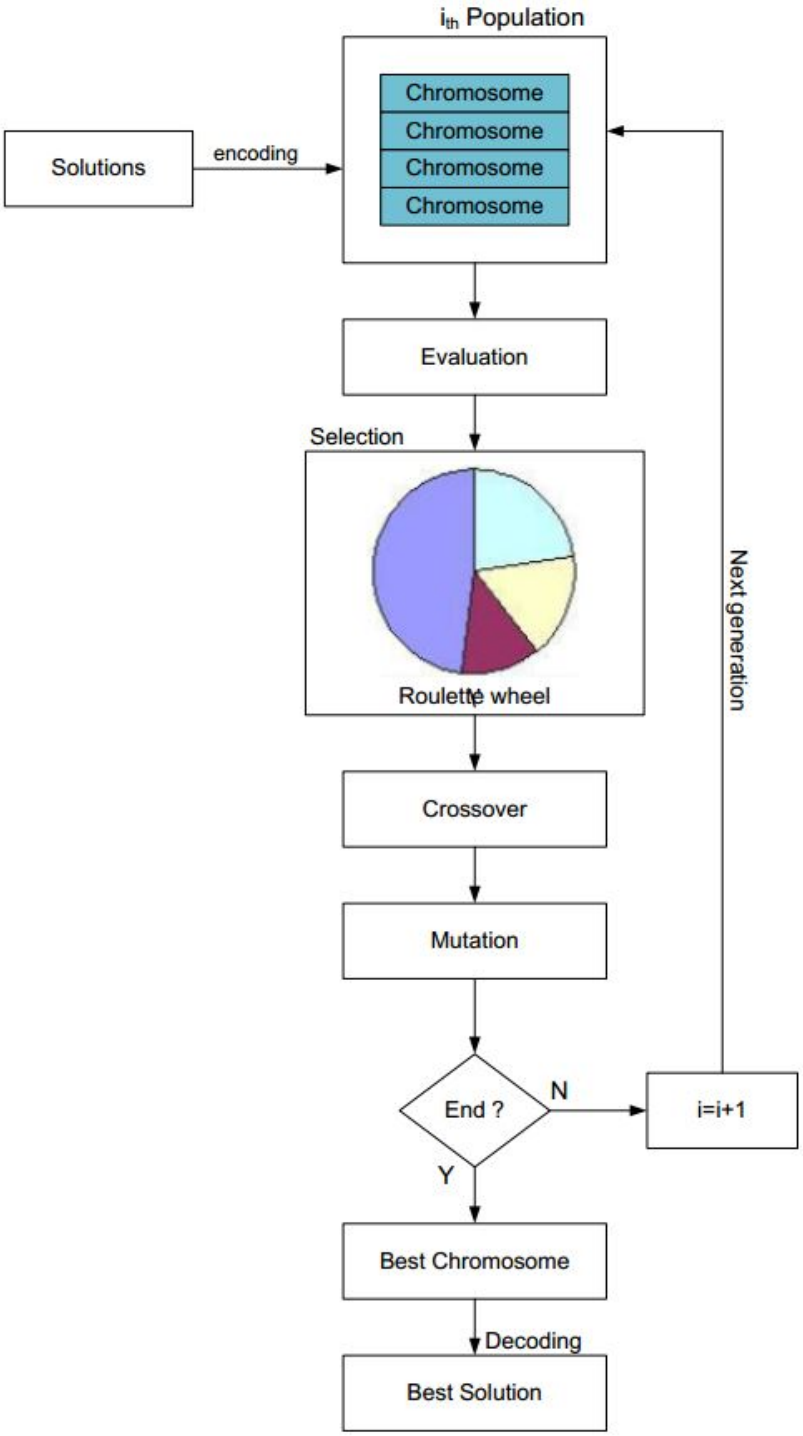

Fig. 2.1. Flow Chart of Genetic Algorithm Operation.

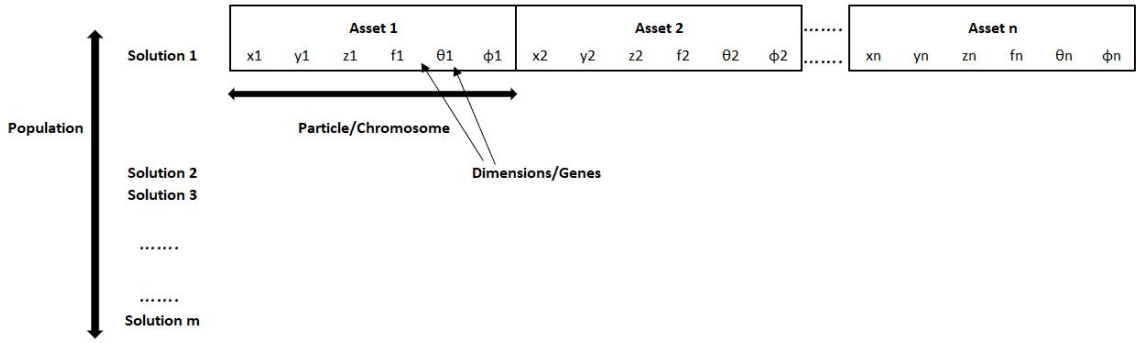

Fig. 2.2. Representation of Solutions with Chromosomes and Genes.

2.4.2 Selection Process

Selection is the next phase for GA and in our research problem, and we used a roulette wheel selection process. In our case, the chromosomes with the higher fitness value have a higher probability to be selected for the next generation. For roulette wheel, first of all the fitness value of the chromosomes are scaled to the range of 0 to 1, and then 1 is added to avoid divide-by-zero problems. The summation of fitness values of all the chromosomes are calculated, and partial fitness is evaluated. Calculating cumulative fitness value of all the chromosomes is the next phase. A random number generator is used to generate random numbers within the range of 0 to 1 for all chromosomes. Last of all, comparing the cumulative fitness value to an arbitrary number of all the chromosomes to select the best chromosomes for the next step. Elitism can also be used with roulette wheel to make the selection process more precise but, due to the computational complexity, elitism was avoided. In Figure 2.3, a basic roulette wheel selection process for the Genetic Algorithm is shown. For this case, we have a population of five chromosomes and percentage values of the chromosomes represent how much each chromosome is contributing to total fitness value. Chromosome 1 has the most highest percentage which is 40% and chromosome 1 is more likely to be selected when the wheel is rolled. Chromosome with lowest percentage has less chance to be selected for next generation.

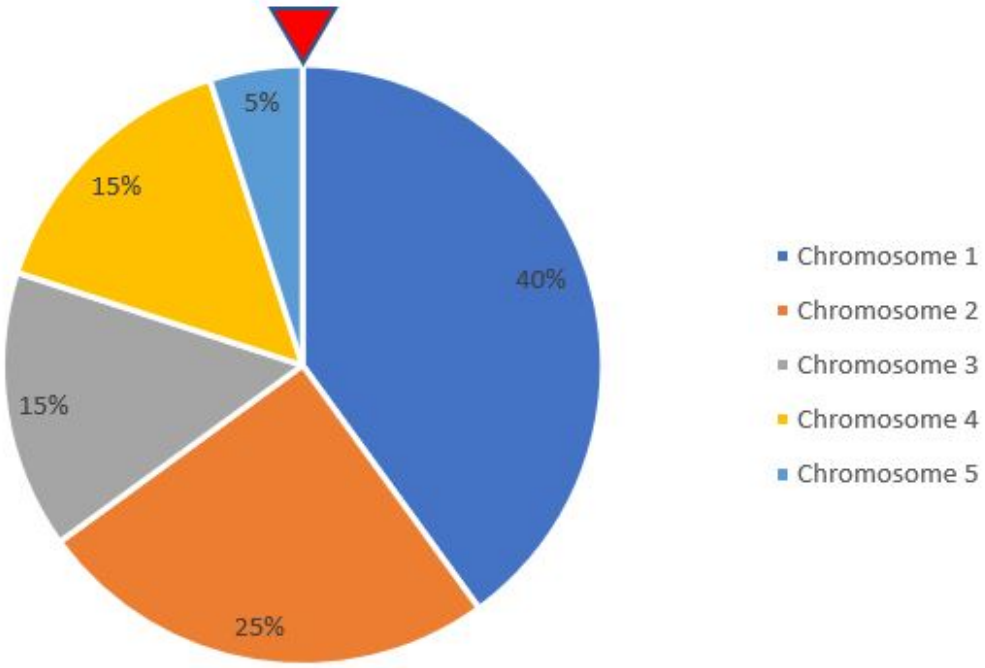

Fig. 2.3. Genetic Algorithm Roulette Wheel Selection Process.

2.4.3 Crossover Process

Crossover is the process of the crossing of the parent's chromosomes to produce new child chromosomes. There are different types of crossovers, for example: single point, double point, k point, and uniform crossover. Utilization of various crossover techniques depends on the nature of the problem. In our research, we used single point crossover which takes the randomly selected position in the parent chromosomes and then exchanges to make sub-chromosomes. A random number generator is used to generate random numbers within the range of 0 to 1 for all chromosomes. Comparing crossover rate (C_r), and random numbers parent chromosomes are randomly selected. After the selection of parent chromosome, the next process is identifying the position of the crossover point. For this purpose again random numbers are generated between 1 to (length of chromosome minus 1).

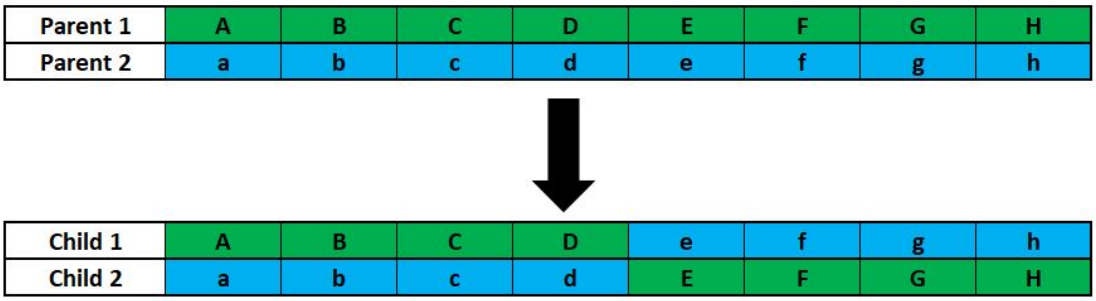

Fig. 2.4. Genetic Algorithm Single Point Crossover Process.

After determining crossover points, parents chromosomes need to cut at the crossover points and interchange their genes. Figure 2.4 demonstrates single point crossover.

2.4.4 Mutation Process

Mutation is the process of replacing genes in a chromosome at random position with a new value. The mutation rate (M_r) determines the number of genes in the population that will go through the mutation process. The total number of genes in an overall population is equal to the total number of genes in chromosome, multiplied by the population number. The number of genes to be mutated is calculated from mutation rate and a total number of genes in population. The position of mutation is also determined randomly. First of all, a random integer is generated between 1 and total number of genes, then multiplied by mutation rate. Figure 2.5 demonstrates mutation process where a random value K is added in place of value D. With the finishing of mutation process one single generation is completed. We need to evaluate the fitness function for another generation. If the fitness function value is increasing, then the GA is going towards the desired solution. So, these new chromosomes will undergo the same process as the previous generations such as: evaluation, selection, crossover, and mutation and producing a new chromosomes for the next generation. Eventually, GA will terminate based on the termination conditions.

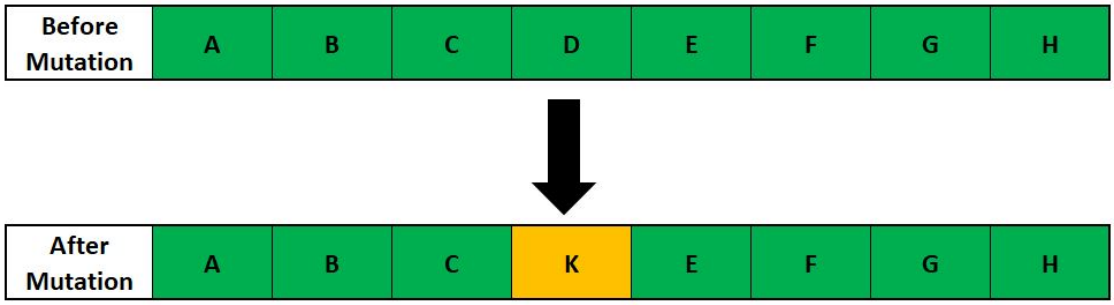

Fig. 2.5. Genetic Algorithm Mutation Process.

2.5 Setting Up GA Parameters

To analyze the performance of the Genetic Algorithm, suitable parameters must be chosen. Population size, crossover rate, and mutation rate are the parameters needed for the complete operation of Genetic Algorithm. This section describes how these parameters are selected.

2.5.1 Setting Up Population Size for Test Case Analysis

Population size is a principle parameter for Genetic Algorithm. According to Rylander et al. [29] increasing population size increases the accuracy but it leads to higher runtime. The result of experimental trails for our research was a population size of 175. Initially, a test case was developed to select the best population size for this research problem.

According to test case criteria from Table 2.1 *Max Generations* was kept 3000, *Swarm termination Window Size* was kept 1000. Moreover, *Inject Global Best* option was kept disabled and Genetic Algorithm parameters: *Crossover Rate* and *Mutation Rate* was set up as 0.60 and 0.005 respectively. According to [30] we selected values for crossover and mutation rate. Thirty transmitters are placed randomly in 3D space and then run for fifty times to record average fitness and runtime values. The population size of the chromosome was increased from 50 to 300. Increasing population size

Table 2.1. The Test Case Setting for Population Size Measurement:

Parameter Name	Normal Setting Value
Receivers	3
Transmitters	30
Tx Spread Radius	30.00
Layout	Left-Right
Frequency Step	0.10
Receiver Bandwidth	10.00
Receiver RF Front End Limit (dBm)	5.00
Receiver SF Sensitivity (dBm)	-88.00
Max Generations	***3000***
Max Run Time (ms)	0
Swarm Termination Fitness Slope	0.0100
Swarm Termination Window Size	***1000***
Initialization Seed	0
Population Size	200
Neighbors	20
Antenna Type	Aperture
Beam Width X (Degree)	60.00
Beam Width Y (Degree)	60.00
Inject Global Best	***Not Enabled***
Crossover Rate	***0.6000***
Mutation Rate	***0.0050***
Spreading Loss Factor	2.00
Absorption Factor	0.00000
Stochastic Factor	0.00000
FPF	20.00

leads to increased run time. The goal of this testing procedure is to find a balanced solution: higher fitness with the low runtime.

In Figure 2.6, population size was varied from 50 to 300 and it is seen that population size of 175 provides better average fitness value of 4889.03. In Figure 2.7 with 175 population size value, average run time was measured 9204.3. Hence, this population size provides the best result balancing higher fitness value and lower run time.

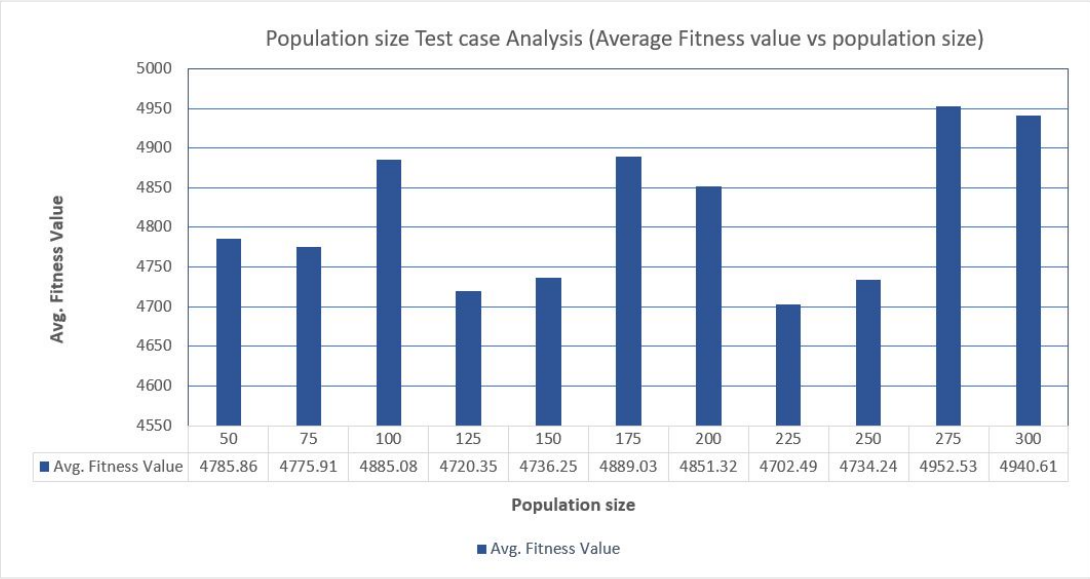

Fig. 2.6. Average Fitness Value vs Population Size for Population Size Test Case Analysis.

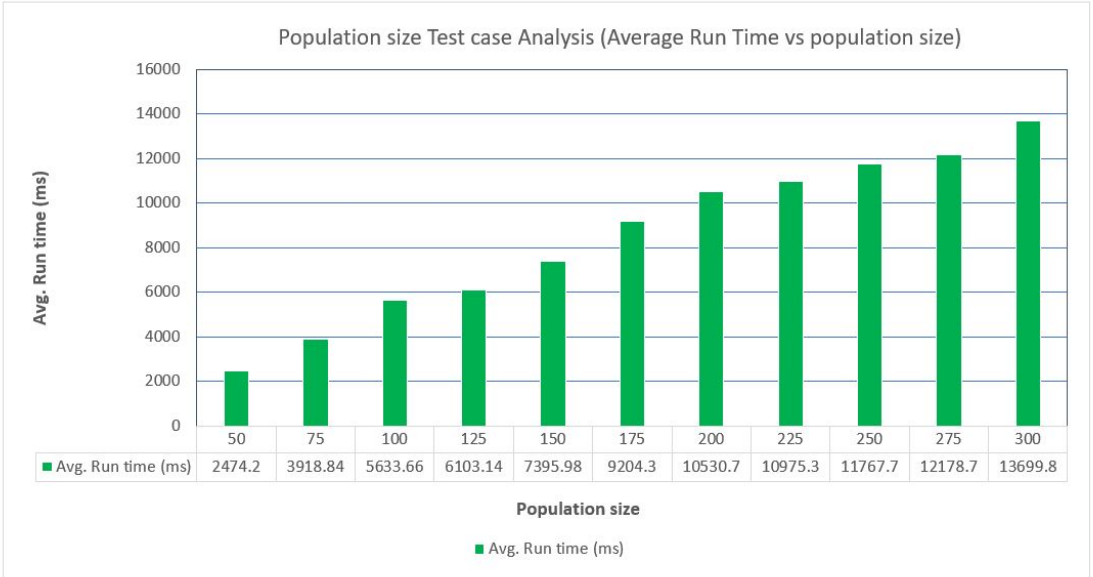

Fig. 2.7. Average Run Time vs Population Size for Population Size Test Case Analysis.

2.5.2 Test Settings for Result Analysis

Population Size=175, Crossover Rate=0.60, Mutation Rate=0.008	
Normal straight-line settings Max Generations=1000 Swarm Termination Window Size=50	**Normal Random settings** Max Generations=1000 Swarm Termination Window Size=50
Extended straight-line settings Ground truth Max Generations=5000 Swarm Termination Window Size of=2000	**Extended Random settings** Max Generations=5000 Swarm Termination Window Size of=2000

Fig. 2.8. Test Case Setup for Analysis.

In Figure 2.8, four types of test settings are shown for examining test conditions. Two of them are normal settings: straight line and random settings and two are extended settings: straight line and random settings. All of the settings have a population size of 175 with crossover rate of 0.6, and mutation rate of 0.005. Previous researchers designed normal settings so we kept same for GA to compare PSO and GA. In general, GA needs more generations to find a better result that is why extended settings are used.

2.5.3 Normal Test Settings for Test Result Analysis

In normal test settings for test result analysis max generations, swarm termination window size and population size are different from setting in Table 2.1. The normal settings differs from the extended settings by max generations, and swarm termination window size. For normal settings, max generations is set as 1000, and swarm termination window size is 50. For both normal and extended settings population size is set as 175. The normal settings has two choices: run in straight line, and run in random.

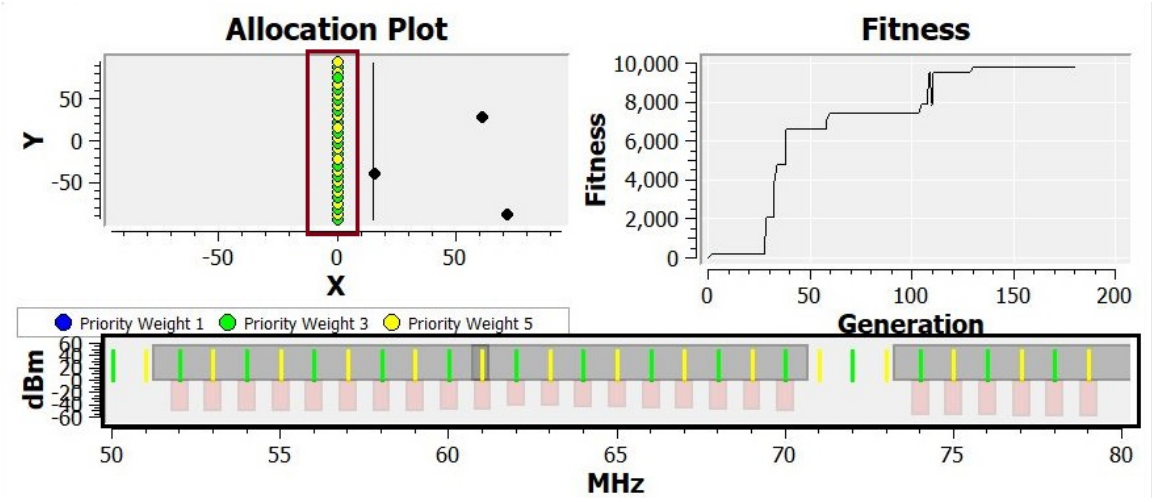

Fig. 2.9. Normal Straight Line Settings Experimental Result.

Run in straight line is a set up where a red box in allocation plot indicates all 30 transmitters placed in the straight line. Transmitters had their starting frequency from 50 Hz and spaced by 1 Hz with alternating priorities five, three, and one accordingly. The Figure 2.9 shows the allocation plot on the left side and fitness plot on the right-hand side for this experiment. From the fitness plot the fitness value is 9824.21 and run-time is 1027 ms, and the process terminates after only 180 generations for the short termination window size.

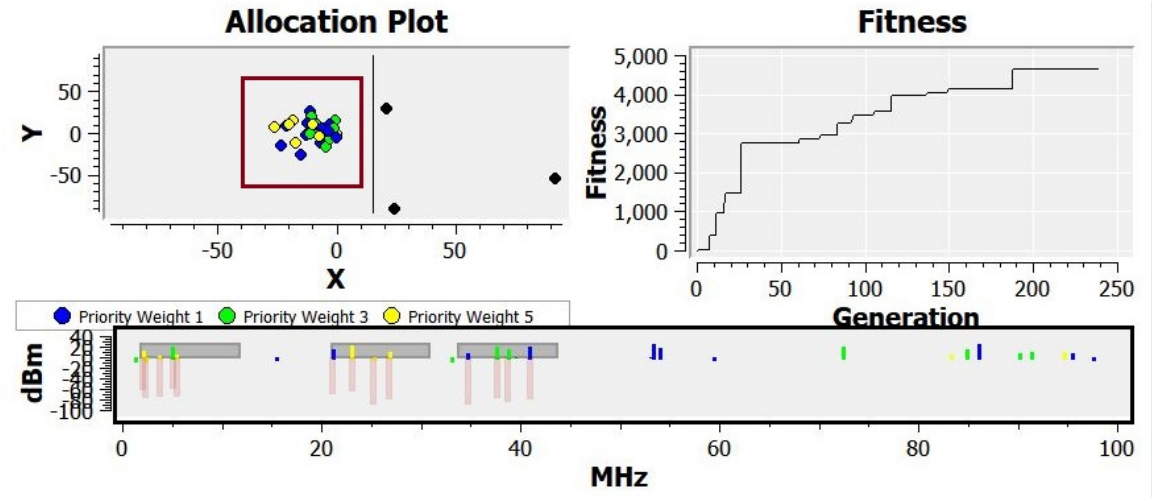

Fig. 2.10. Normal Random Settings Experimental Result.

Run in random is a set up where all 30 transmitters are placed randomly in space and also randomly assigned frequencies and priorities. A red box in the allocation plot indicates the transmitter positions. In Figure 2.10, the fitness value is 4664.83 and run-time is 934 ms, and the process terminates after only 240 generations.

From Figure 2.9 and 2.10, we observe that in the frequency plot; indicated by a black box, that the Genetic Algorithm cannot cover all the transmitters so it could not optimize correctly with these standard constraint settings.

2.5.4 Extended Test Settings for Test Result Analysis

The next experiment uses extended settings and modifies the max generations to 5000 and swarm termination window size to 2000. Like the normal settings, the extended settings also has two types of set up: run in straight line, and run in random.

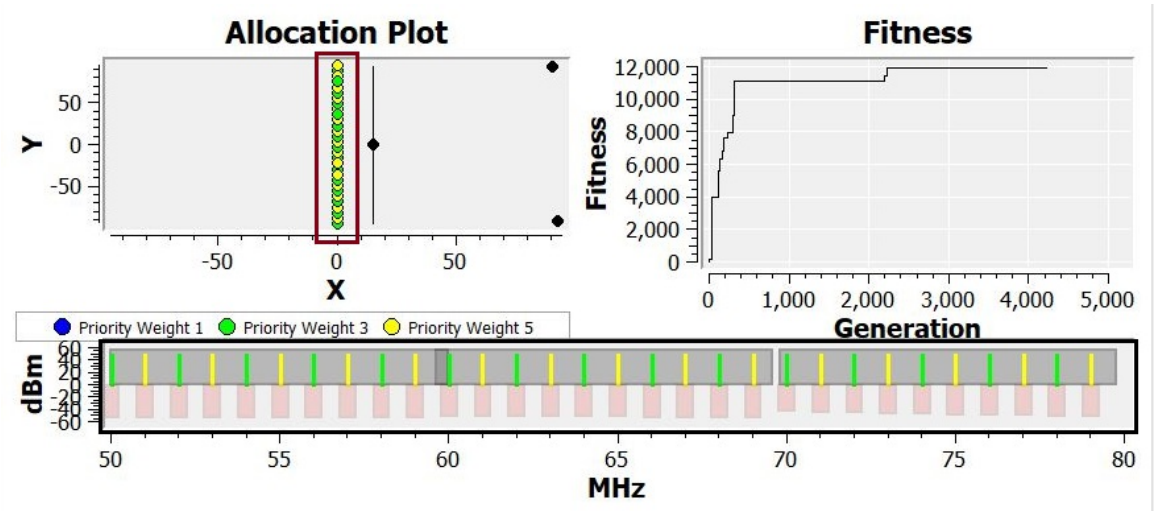

Fig. 2.11. Extended Straight Line Settings Experimental Result.

Run in straight line is a set up where where all 30 transmitters are placed in a straight line starting at frequency of 50 Hz and increasing by 1 Hz. Each have alternating priorities five, three, and one accordingly. From Figure 2.11 the fitness value is 11908 and run-time is 29174 ms. Runtime is higher, as the process lasts for 4250 generations. Although runtime is higher, the higher fitness means that the

solution is more robust than the normal settings. Run in random is a set up where all 30 transmitters are placed randomly in space and also randomly assigned frequencies and priorities. A red box in the allocation plot indicates the transmitter positions. For this experiment, the Fitness value is 5756.21 and run-time is 22460 ms. Runtime is higher as the process lasts for 4400 generations shown in Figure 2.12. The frequency plot in Figure 2.11 of the Genetic Algorithm now covers all the transmitters, hence optimizes correctly in extended constraint settings.

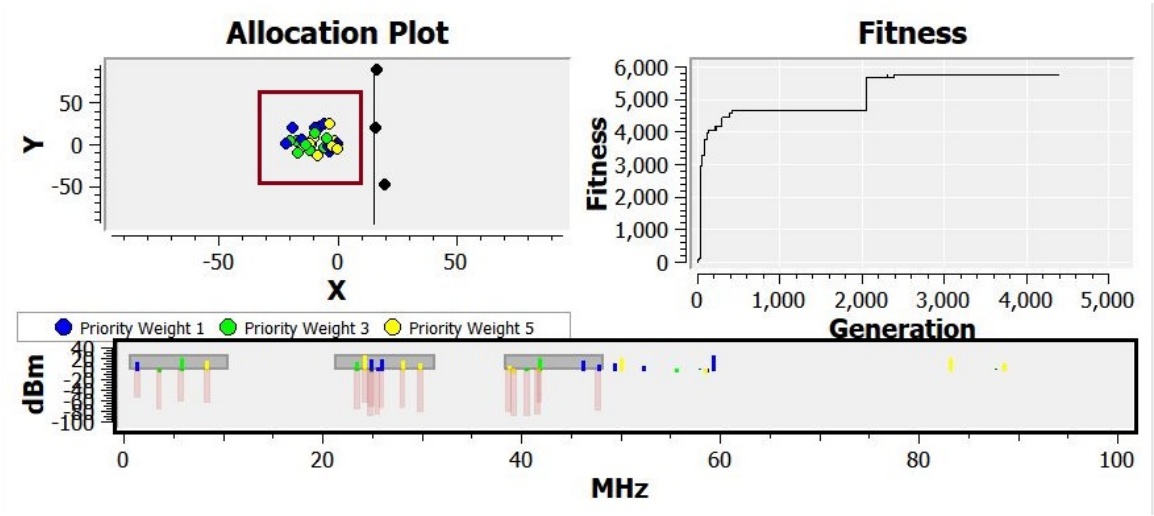

Fig. 2.12. Extended Random Settings Experimental Result.

In summary, the Genetic Algorithm needs more generations to reach its optimized solution compared to PSO. So the extended settings are required. The extended settings are used to make sure GA can converge to perfect solution compared to the PSO.

2.5.5 Setting Up Crossover and Mutation Rate for Test Case Analysis

Selection of crossover rate and mutation rate are critical to the Genetic Algorithm. The process for choosing crossover and mutation rate is similar to population size selection. A test case is set up to find the best crossover and mutation rate through analysis, and then these values are used in Genetic Algorithm with different constraint

conditions. Crossover rate and mutation rate selection process and result with these values are discussed in this section. Some papers claim that (50-100%) is the perfect crossover range where 60% is best crossover value, and (0.5-1%) is the ideal mutation range. Initially, a test case was developed to select appropriate crossover and mutation rate for this research. The test case setting for crossover rate and mutation rate is same for setting of population size. Population size is set as 175 from population size test case analysis. Thirty Transmitters are placed randomly in 3D space and run 50 times to take average fitness and runtime value.

Crossover rate was varied from 50% to 70%, and mutation rate was altered within the range of (0.5-0.8%). The goal is to select the best combination of crossover rate and mutation rate that provides an excellent solution with higher fitness and minimum runtime, i.e., minimum generations. From Figure 2.13 crossover rate of 0.6 and mutation rate of 0.008 provide the best result balancing higher fitness value and lower run time, seen in shaded row.

Test case Analysis for Crossover & Mutation rate

Crossover rate	Mutation rate	Fitness Value	Run time (ms)
0.5	0.005	4855.27	8462.28
	0.006	4851.31	8497.34
	0.007	5063.66	8419.12
	0.008	5010.06	8647.78
0.6	0.005	4670.74	9125.24
	0.006	4988.23	8245.7
	0.007	4875.12	8406.06
	0.008	4996.18	8216.32
0.7	0.005	4998.18	8507.92
	0.006	4889.04	8269.7
	0.007	4926.72	8274.58
	0.008	4964.44	8021.48

Fig. 2.13. Test Case Analysis for Crossover and Mutation Rate.

2.5.6 Normal Test Settings for Test Result Analysis

Population size, crossover rate, and mutation rate from the test case analysis will be used in this subsection with different conditions. We used two different settings for the experiment: normal settings and extended settings. Both of the settings have a population size of 175, crossover rate of 0.6, and mutation rate of 0.008.

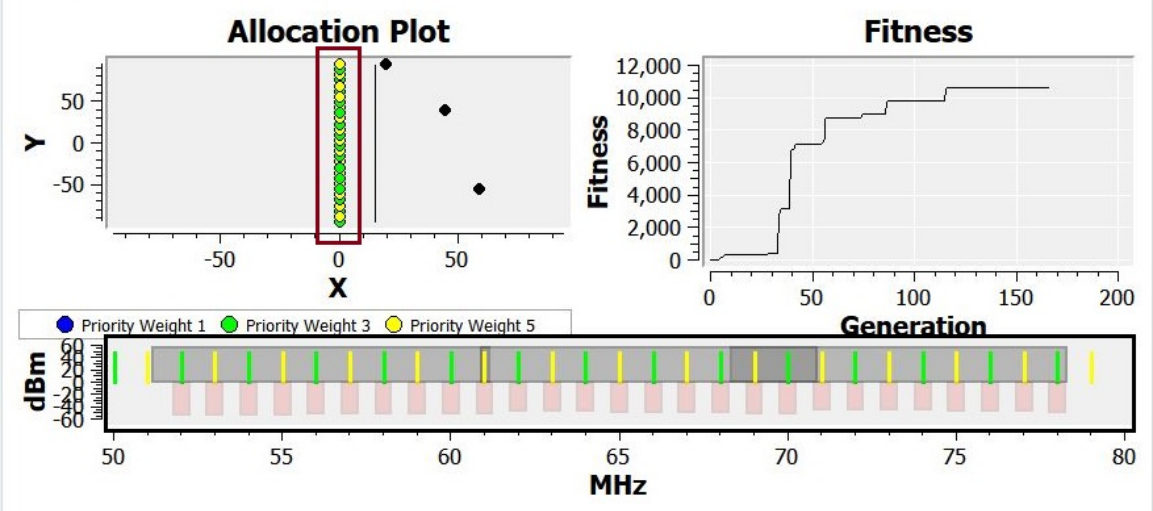

Fig. 2.14. Normal Straight Line Settings Experimental Result for Crossover and Mutation Rate.

From Figure 2.14, fitness plot shows that fitness value is 10617.9 and run-time is 1044 ms. The process terminates after 175 generations for short termination window size. From Figure 2.15, fitness value is 3970.19 and run-time is 792 ms where generations are 150. In Figure 2.14 and 2.15, the frequency plot; indicated by a black box, shows that the Genetic Algorithm cannot cover all the transmitters so it could not optimize correctly with standard constraint settings.

2.5.7 Extended Test Settings for Test Result Analysis

The extended settings for testing crossover rate and mutation rate differs from Table 2.1 by crossover rate and mutation rate. From Figure 2.16, fitness value is 11610.3 and run-time is 14730 ms in extended settings. Runtime is higher because

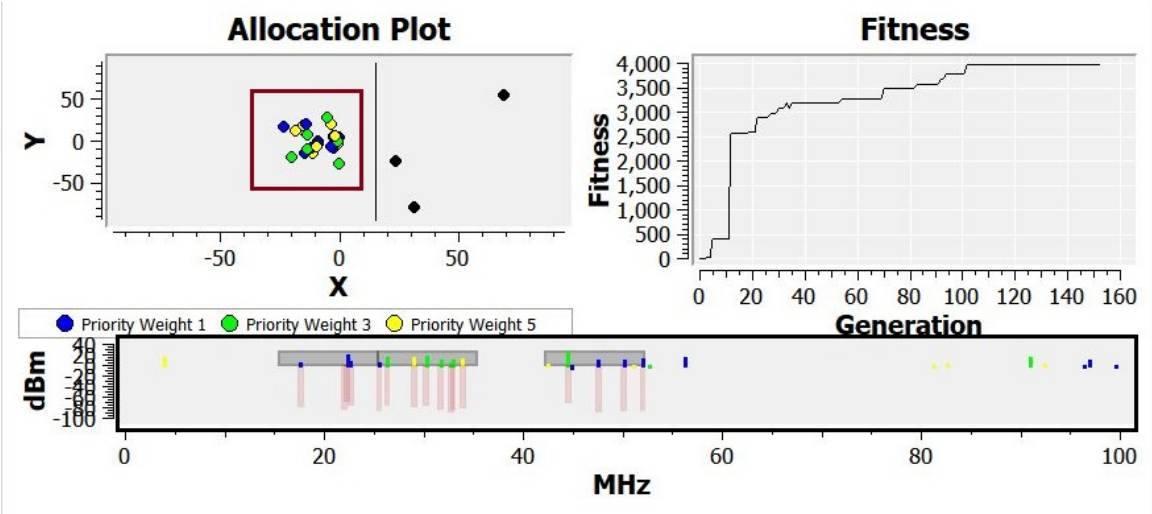

Fig. 2.15. Normal Random Settings Experimental Result for Crossover and Mutation Rate.

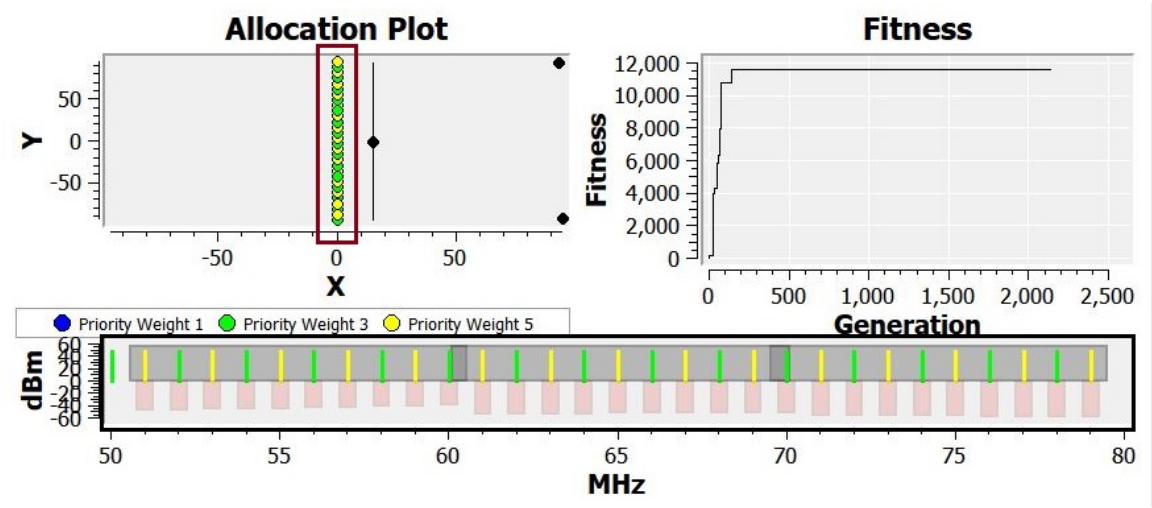

Fig. 2.16. Extended Straight Line Settings Experimental Result for Crossover and Mutation Rate.

the process lasts for 2150 generations. Although runtime is higher, higher fitness means the solution is more robust in extended settings than normal settings. In Figure 2.17, fitness value is 5756.22 and run-time is 12528 ms, and the number of generations is 2225. In frequency plot of Figure 2.11, the Genetic Algorithm covers all the transmitters hence, optimizes correctly in extended constraint settings.

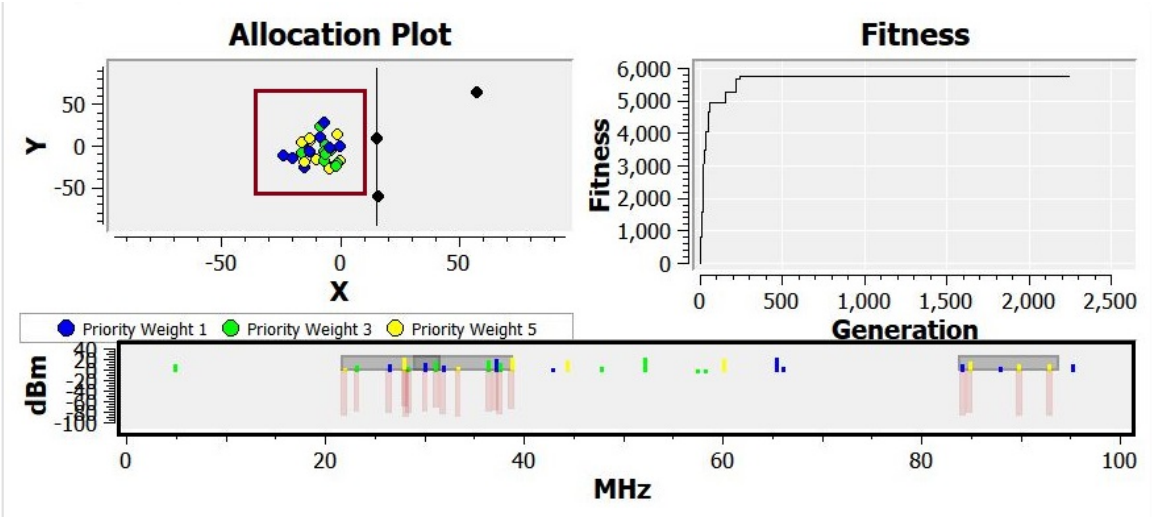

Fig. 2.17. Extended Random Settings Experimental Result for Crossover and Mutation Rate.

Typically, the Genetic Algorithm needs more generations to reach an optimized solution, so normal settings is not sufficient hence, extended settings is required. The extended settings is used to make sure the Genetic Algorithm can converge to a perfect solution like Particle Swarm Optimization. But when comparing the Genetic Algorithm with Particle Swarm Optimization, we will use normal settings.

3. COMPARISON ANALYSIS

3.1 Purpose

Initially, our asset allocation research problem was implemented with Particle Swarm Optimization (PSO). The PSO result is a fast solution that fulfills requirement of optimizing under 1 second. This chapter compares Particle Swarm Optimization Algorithm with Genetic Algorithm showing some weaknesses and strengths of the two. The *Run Time*, *Global Solution Finding* and *Accuracy of Solution* are measured for both of the algorithms.

3.2 Run Time Comparison Analysis

Runtime comparison analysis is the most critical analysis for this research project as the primary goal was to allocate assets in real time. Therefore, finding the factors both for ground truth and random position that take the most of the running time, identifying the solution ground truth were the most challenging tasks. Straight line setup is known as solution ground truth for our analysis. Combination of transmitters and receivers to have the best global solution hence, the best overall fitness value with minimum run time is the objective of this analysis. Also, run time cost analysis with increasing transmitter numbers or receiver numbers is also analyzed in this section.

3.2.1 Implementation

For run time comparison analysis, transmitters are set up in random places in 3D space. Different transmitter receiver combinations are tried to analyze best global solution or best overall fitness value with minimum run time. With normal settings, both Particle Swarm Optimization Algorithm and Genetic Algorithm are run for 50

times. The total fitness values and run-time values are calculated, and averaged. Then, the number of transmitters is increased from ten to sixty in steps of ten. Likewise, the number of receivers is increased from one to six.

3.2.2 Run Time Result Comparison

Test case Analysis for Different Transmitters-Receivers Combination

Transmittes (Tx)-Receivers (Rx)	PSO Avg. Fitness Value	GA Avg. Fitness Value	PSO Avg. Run time (ms)	GA Avg. Run time (ms)
10 Tx-1 Rx	1410.12	1056.9	86.58	101.42
20 Tx-2 Rx	2424.75	1957.64	225.96	304.92
30 Tx-3 Rx	3972.38	4075.41	444.1	889.68
40 Tx-4 Rx	5497.36	6460.45	796.92	1383.88
50 Tx-5 Rx	6863.5	8536	1105.42	2118.46
60 Tx-6 Rx	8691.02	11266.5	1694.72	5766.4

N.B. Run time averaged for running 50 times with random position of transmitters

Fig. 3.1. Run Time Comparison Analysis for Different Transmitter-Receiver Combinations.

Figure 3.1 shows run time comparison analysis for both Particle Swarm Optimization Algorithm and Genetic Algorithm for different transmitter-receiver combinations. In Figure 3.1, Particle Swarm Optimization provides better fitness value or global solution for a lower number of transmitter-receiver combinations, and the Genetic Algorithm is suitable for a higher number of transmitter-receiver combinations, typically 30 transmitters- 3 receivers combination and above. In all transmitter-receiver combinations Genetic Algorithm has a higher time cost value than Particle Swarm Optimization. Fitness function calculation dominates overall run time both for PSO and GA equally. So, the higher run time for GA is because of taking more generations to convergence than PSO. Finally from analysis 30 transmitters-3 receivers appear to be the best combination as for both algorithms.

Figure 3.2 and 3.3 shows runtime comparison analysis for both Particle Swarm Optimization Algorithm and Genetic Algorithm for increasing transmitters and in-

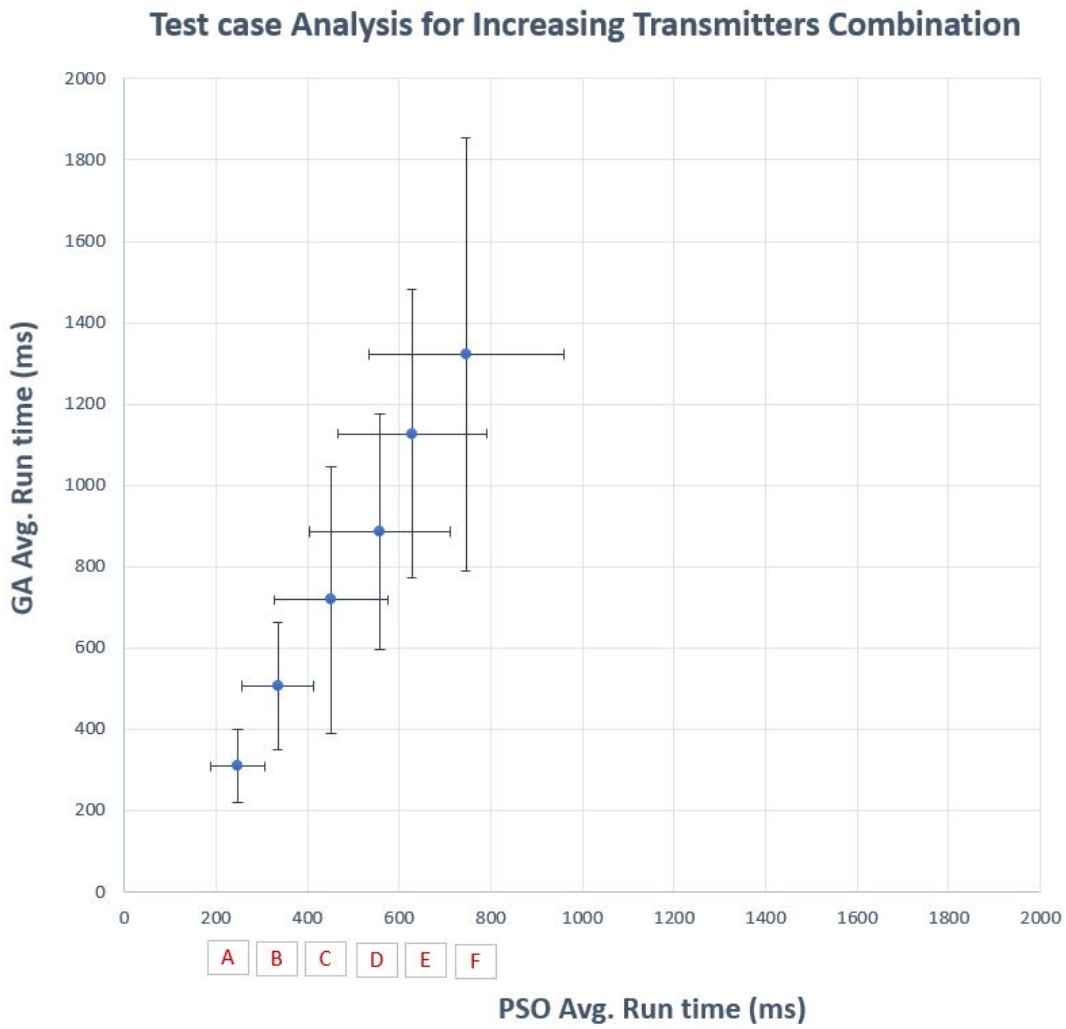

Fig. 3.2. Run Time Comparison Analysis for Increasing Transmitters.

creasing receivers respectively. In Figure 3.2 the horizontal and vertical axis represent an average run time for PSO and average run time for GA respectively. We have six data points A, B, C, D, E, and F that stand for six combinations of transmitters-receivers, where transmitters are varied from 10 to 60 and receivers are kept constant at 3. It's clear from the graph that, for most of the cases average run time of GA is higher than PSO. From two-dimensional error bars in the graph, normally GA has higher standard deviation than PSO hence, GA run time varies much in 50 runs. For Figure 3.2, we have similar six data points A, B, C, D, E, and F for six combinations

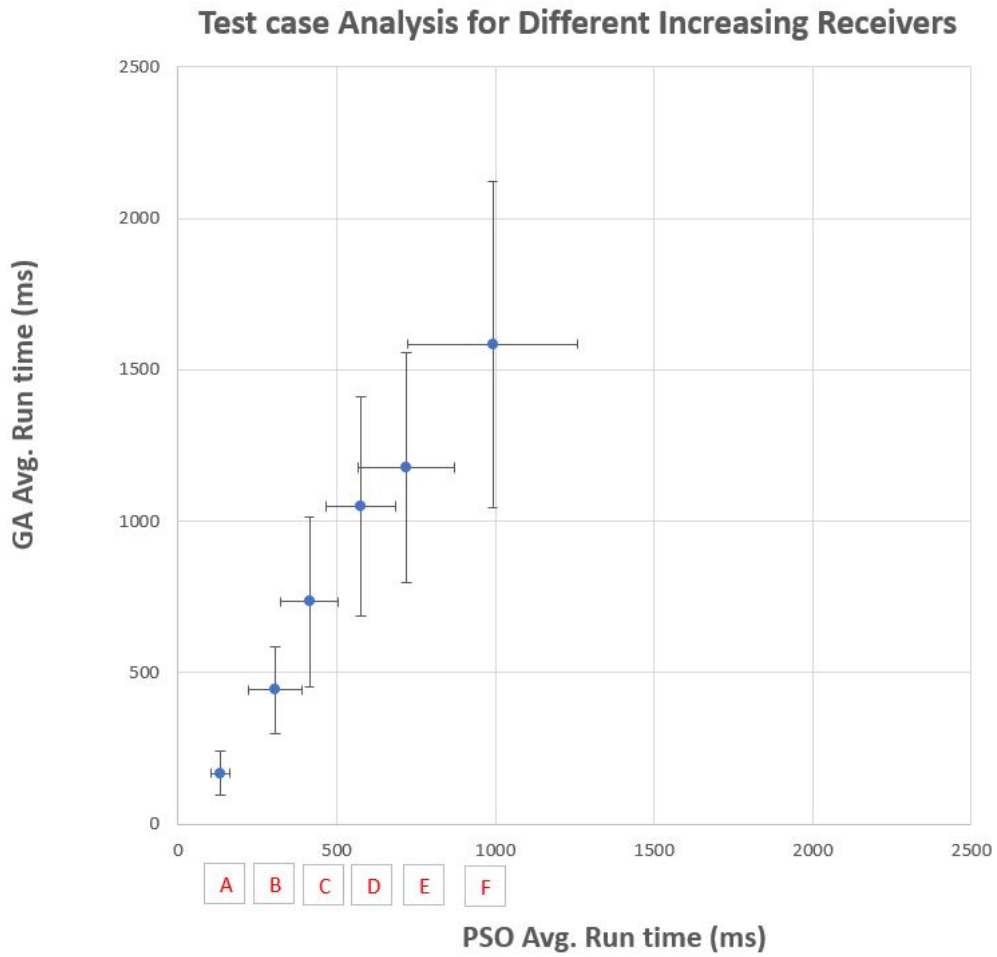

Fig. 3.3. Run Time Comparison Analysis for Increasing Receivers.

of transmitters-receivers but here receivers are varied from 1 to 6 and transmitters are kept constant at 30. Complying with the previous experiment, average run time of GA is higher than PSO in most of the cases and GA has a higher standard deviation than PSO. Comparing both experimental set-ups, increasing receivers cost a much higher run time than increasing transmitters, because the fitness function has more complex computations with increasing receivers. Normally, it is the fitness calculation that dominates overall run time in a single generation.

3.3 Global Solution, Convergence Rate, and Fitness Finding Comparison Analysis

Global solution analysis measures the ability to reach global maximal of Particle Swarm Optimization and Genetic Algorithm for this particular research problem. Sometimes in optimization problems, algorithms fail to reach a global solution and instead are stuck in a local solution. Convergence rate analysis is a test for examining the convergence speed of both algorithms and comparing their performance. The fitness finding analysis compiles the overall fitness value that both algorithms can achieve within test settings and constraints.

3.3.1 Implementation

Throughout the testing analysis, two test settings are used extensively: normal settings and extended settings with two sub settings: straight line settings and random settings. For comparing Particle Swarm Optimization with Genetic Algorithm in this subsection, normal settings and extended settings with the straight line set up are used. As straight line setting is the ground truth for this problem, so the possible global solution and fitness value are known and hence, it is easy to judge the accuracy of the algorithms. In this experiment, population size of 175, crossover rate of 0.60, and the mutation rate value of 0.008000 were used. This selection process of these parameters was analyzed and discussed in the previous chapter. For extended settings, max generations of 5000 and swarm termination window size of 2000 were chosen. Whereas, for normal settings max generations of 1000 and swarm termination window size of 50 were selected. Finally, global solution, convergence rate, and fitness finding capability of PSO and GA were compared.

3.3.2 Global Solution Result Comparison

Particle Swarm Optimization is very efficient in finding a global solution in a short time. Global solution for this research problem is the solution that covers all transmitters in three-dimensional space and frequency. Particle Swarm Optimization can cover all transmitters even in normal settings, where max generations and swarm termination window size is kept to a smaller value. "Max Generations", "Swarm Termination Window Size", and "Swarm Termination Fitness Slop" are the three termination conditions for the process.

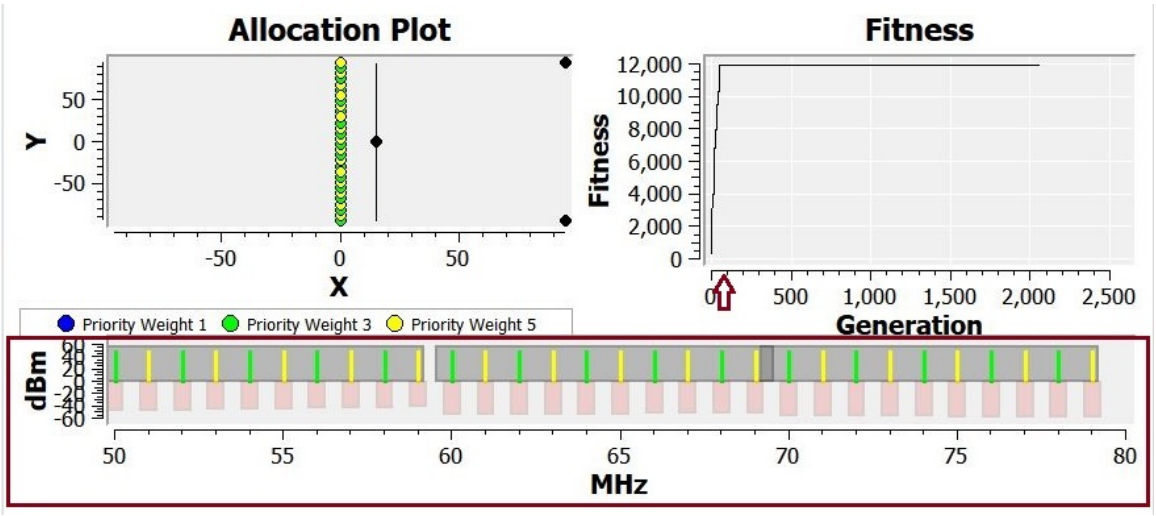

Fig. 3.4. Result of Global Solution Analysis for PSO in Extended Settings

From Figure 3.4 and 3.5, the bandwidth plot shows a red rectangle where three receivers can cover all thirty transmitters. For these settings, the overall fitness value is known to be around 12,000, and it is found 11908 for both extended settings and normal settings.

In same way, the normal settings bandwidth plot in Figure 3.6 shows that, three receivers fail to cover all thirty transmitters but in Figure 3.7 with extended settings, three receivers successfully covers all thirty transmitters. For this reason, the first figure with normal settings has overall fitness value of 10022.7 and second figure with extended settings has higher fitness value of 11908. We can conclude that, both

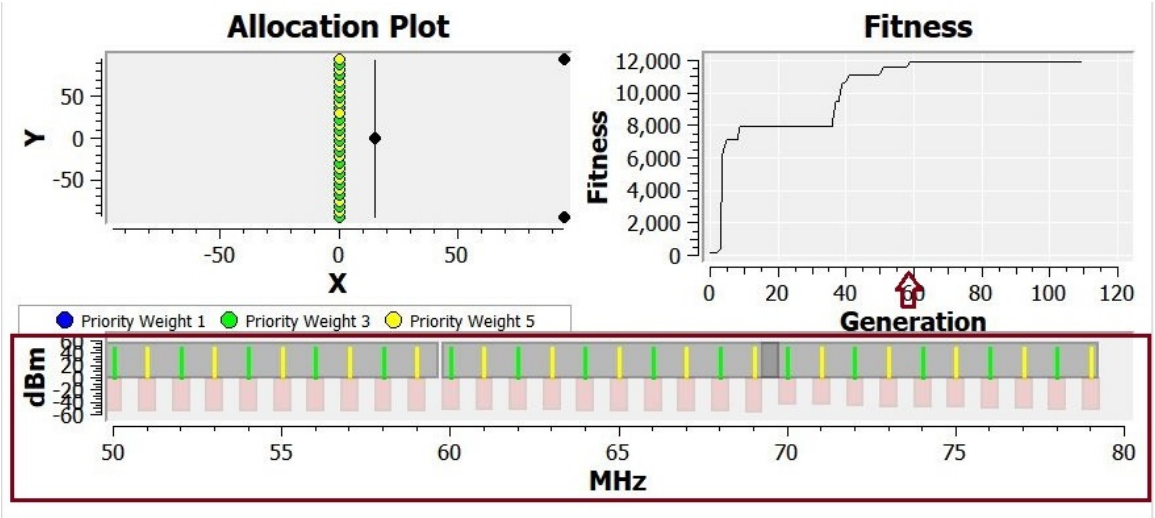

Fig. 3.5. Result of Global Solution Analysis for PSO in Normal Settings

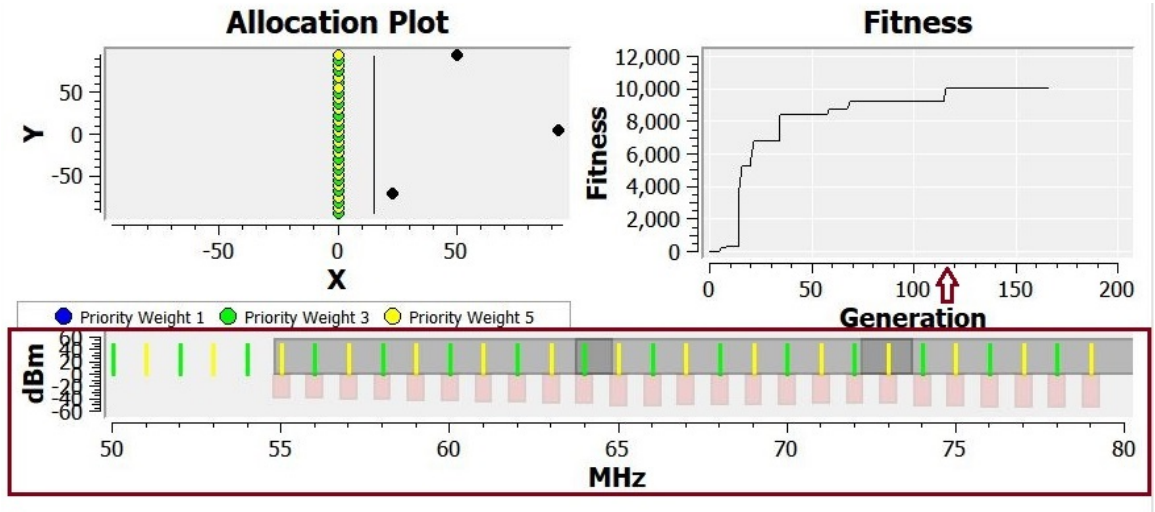

Fig. 3.6. Result of Global Solution Analysis for GA in Normal Settings

Particle Swarm Optimization and Genetic Algorithm can find a global solution, but for Genetic Algorithm we need extended settings, where termination conditions need to be more flexible.

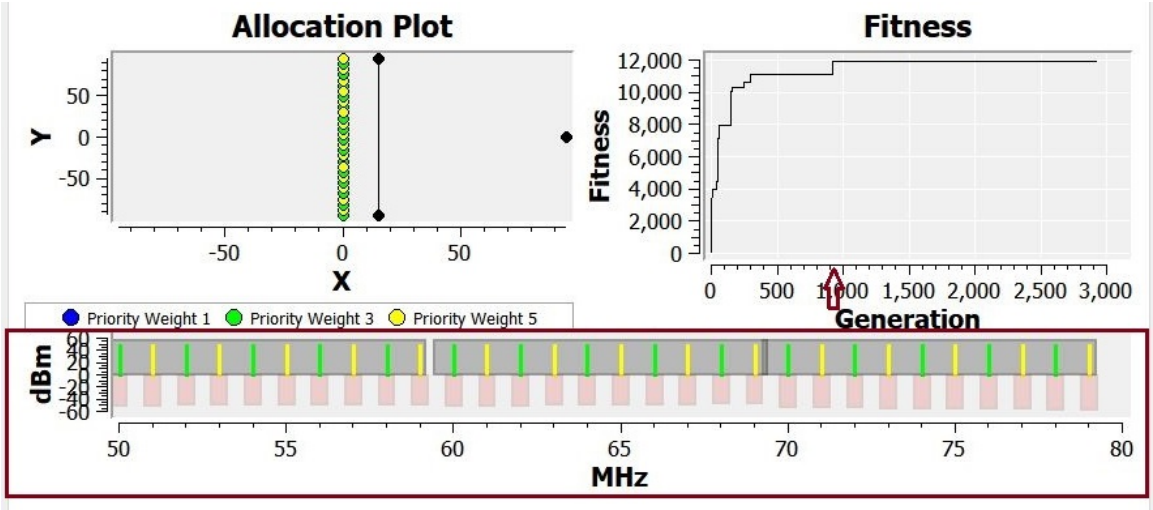

Fig. 3.7. Result of Global Solution Analysis for GA in Extended Settings

3.3.3 Convergence Rate Result Comparison

The "Convergence rate" is how quickly both algorithms can reach a global solution. From Figure 3.4 in extended settings, the Particle Swarm Optimization reaches its global solution in approximately 60 generations and from Figure 3.5 in normal settings, generations required is also close to 60.

But from Figure 3.7, although Genetic Algorithm reaches a global solution it takes many more generations: 900 to 1000. And from Figure 3.6, the Genetic Algorithm fails to reach a global solution where it achieves its own best solution in approximately 115 generations. In summary, Particle Swarm Optimization converges to a global solution more quickly than Genetic Algorithm.

3.3.4 Fitness Finding Result Comparison

Fitness finding result analysis is for testing the best overall fitness value reaching capability of the both algorithms. We define best fitness value as the fitness value for a global solution. From Figure 3.4 and 3.5, the Particle Swarm Optimization does reach a global solution for this problem and reaches the best overall fitness value.

In Figure 3.7, the Genetic Algorithm reaches a global solution and best fitness value only in extended settings. But from Figure 3.6, the Genetic Algorithm fails to reach a global solution and best fitness value with normal settings.

3.4 Time Limiting Comparison Analysis

Time Limiting analysis measures performance of Particle Swarm Optimization and Genetic Algorithm and make a comparison of them. For this purpose both algorithms are run for a time span of 1 second. In this time span, how both algorithms behave is the concern of this analysis.

3.4.1 Implementation

Thirty transmitters are set up in a straight line in an extended settings. In extended settings, max generations is set as 5000 and swarm termination window size as 2000 wherein normal settings these parameters have value of 1000 and 50 respectively. All other settings are kept same which is shown in Table 2.1.

3.4.2 Time Limiting Result Comparison

From the experimental result, Particle Swarm Optimization can cover all 30 transmitters with only three receivers. For this reason, the fitness value of 11908 is close to maximum fitness value of 12,000 in 1000 ms runtime. PSO converges the fitness value of 11908 within approximately 25 generations. Hence, Particle Swarm Optimization can optimize in a short time and few generations. The Genetic Algorithm covers almost all transmitters but unable to cover all 30 transmitters with three receivers. So its fitness value of 11114.1 is little less than maximum fitness value of 12,000 in 1013 ms runtime. It takes approximately 25 generations to converge to fitness value of 11114.1.

We can conclude that, since we set up termination condition of this analysis to 1000 ms the Genetic Algorithm did not get enough time to reach its the optimal solution. In regarding performance measurement, both algorithms have the potential to solve the research problem correctly, but in respect of speed or time, Genetic Algorithm is slower than Particle swarm optimization.

3.5 GUI Environments Comparison Analysis

This research problem has implementation with different graphical user interface (GUI) environments. The basic form is a representation in three-dimensions. In addition to three-dimensions, pheromone and terrain can be added to it. With different complex GUI environments, time complexity or cost to find a possible global solution also varies.

3.5.1 Implementation

For this test, transmitters are set up in random places in three-dimensional space. With normal settings both Particle Swarm Optimization Algorithm and Genetic Algorithm is run 50 times. The overall fitness values and run-time values are averaged respectively. It is for basic representation in three dimensions. Then the pheromone is added to three-dimensional representation and tested in the same way. Likewise, terrain is added to three-dimensional representation separately and tested. Finally, both pheromone and terrain are added to three-dimensional representation and tested. The more complex is the environment, the more the time cost to evaluate the fitness function hence, more generations are needed for both PSO and GA.

3.5.2 GUI Environments Result Comparison

Figure 3.8 shows the average fitness graph and Figure 3.9 shows the average run-time graph for both Particle Swarm Optimization Algorithm and Genetic Algorithm

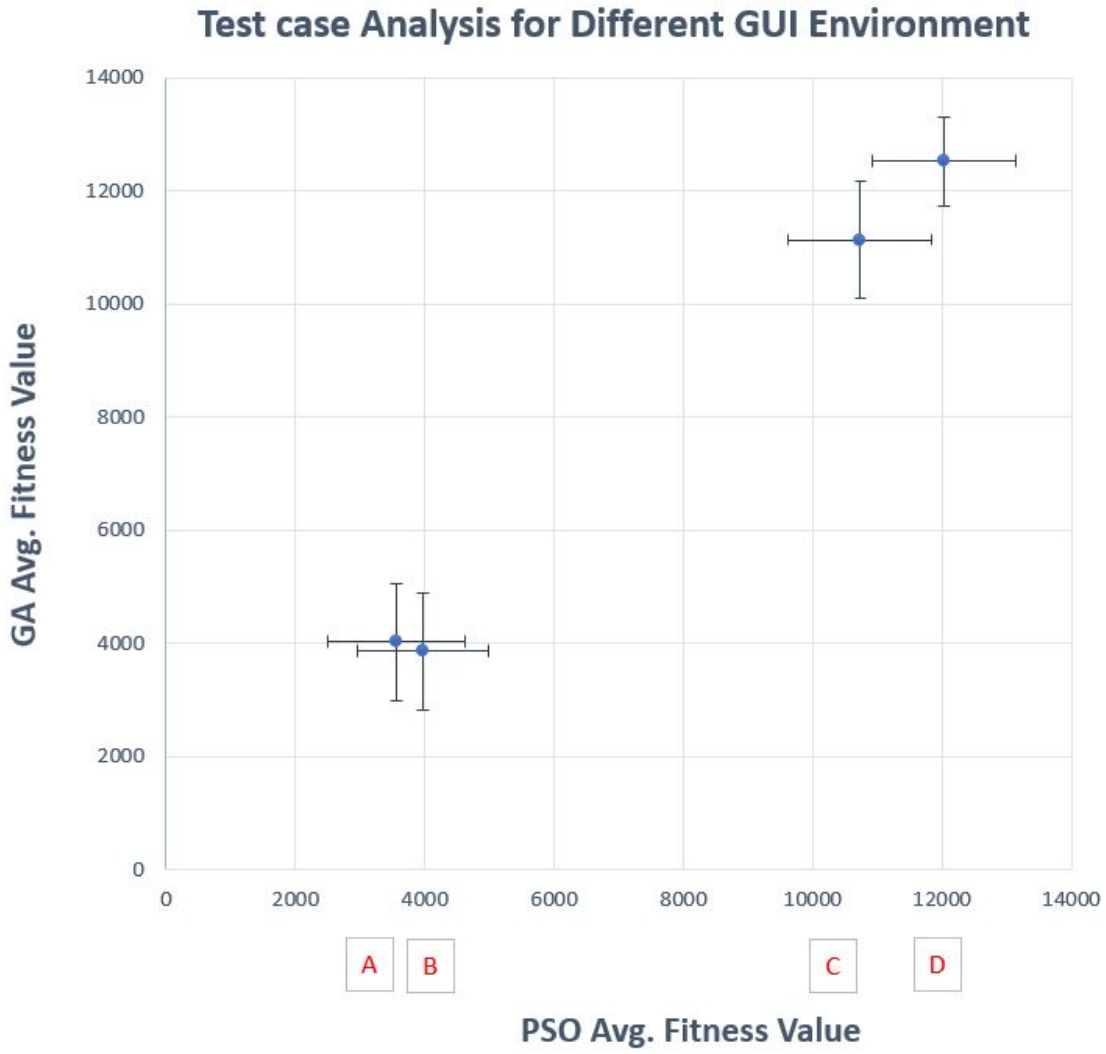

Fig. 3.8. Fitness Graph for Test Case Analysis for Different GUI Environment.

for different GUI environments. In Figure 3.8 the horizontal and vertical axis represent an average fitness value for PSO and average fitness value for GA respectively. We have four data points A, B, C, and D that represents four GUI environments: 3D, 3D+Pheromone, 3D+Terrain, and 3D+Pheromone+Terrain. From the graph, GA has equal or higher average fitness value than PSO. From two-dimensional error bars in the graph, the standard deviation of GA is equal to PSO and in some cases for PSO variations in fitness value is little higher than GA. For Figure 3.9, we have similar four data points A, B, C, and D. Average run time of GA is higher than PSO

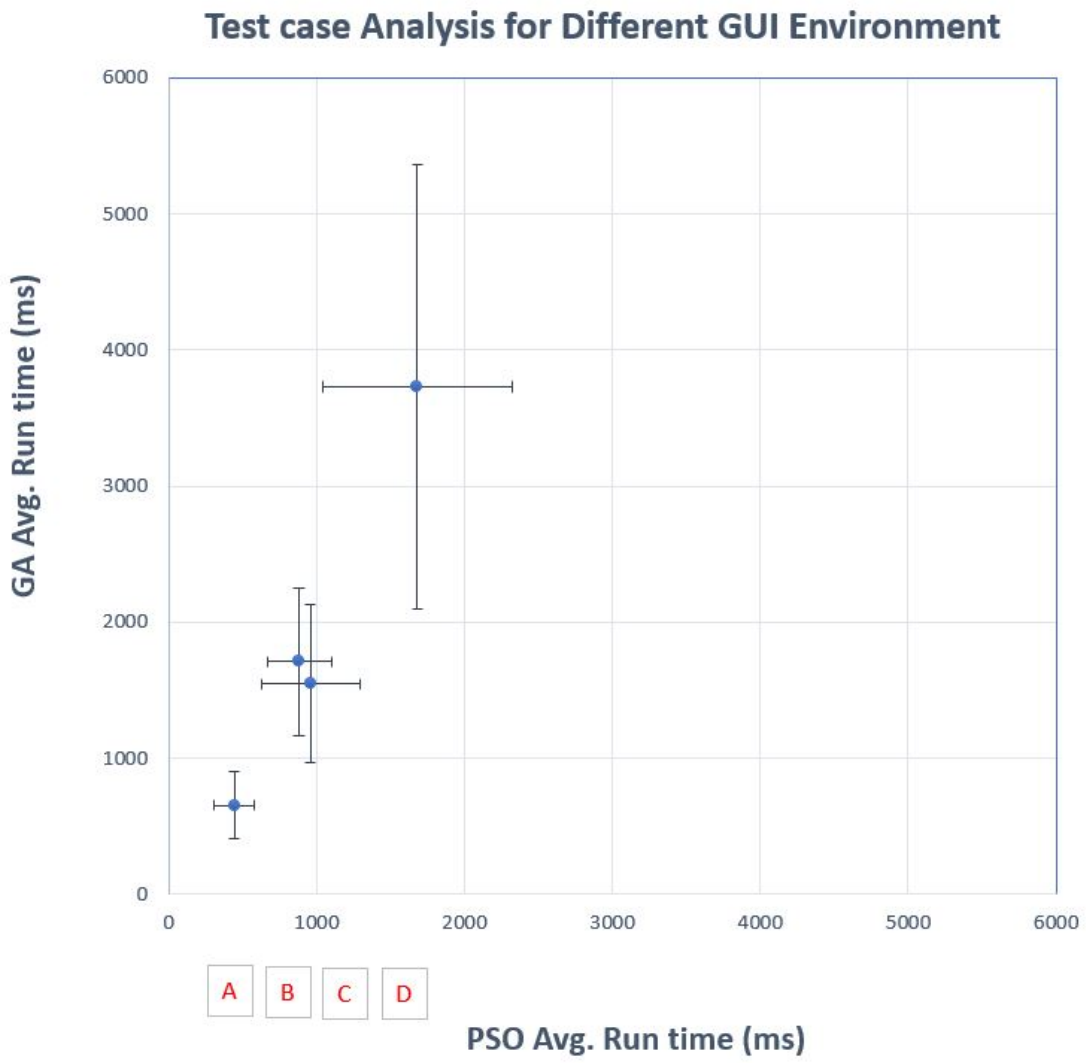

Fig. 3.9. Runtime Graph for Test Case Analysis for Different GUI Environment.

in most of the cases and GA has a higher standard deviation than PSO. Except for basic three-dimensional representation, in all other representations, GA has higher fitness hence, a better solution than PSO. Moreover, in all GUI representations, GA has a higher time cost value than PSO. Although the Genetic Algorithm has a better solution than Particle swarm optimization, it cannot compensate very higher time cost for more complex situations.

4. SUMMARY

Implementing dynamic Electronic Warfare Asset Allocation Problem (EWAAP) with the Genetic Algorithm and performance analysis of the Genetic Algorithm with the Particle Swarm Optimization Algorithm were two parts of this research. Previous C++ coding style in the back-end, and QT data visualization for displaying three-dimensional space, pheromone, and terrain were maintained in the Genetic Algorithm implementation. For receiver numbering 3 or more, the Genetic Algorithm has a higher fitness value, hence, better global solution but it costs more run time. Statistically, for asset numbers from 3 to 6 the GA has (15-30%) higher fitness value with (26-82%) more computational time than PSO. For complex set up, with three-dimensional space, pheromone, and terrain; time complexity increases both for Particle Swarm Optimization and Genetic Algorithm but it impacts the Genetic Algorithm more. With 3D space, pheromones, and complex terrains, the GA has 3.71% better fitness value but the speed of GA is 121% slower than PSO. The Genetic Algorithm can find a global solution like Particle Swarm Optimization, sometimes with a higher fitness value. But the Genetic Algorithm has lower convergence speed than particle swarm optimization. In conclusion, the Genetic Algorithm proves to give a better solution in some cases but the time cost is higher for the Genetic algorithm.

5. RECOMMENDATIONS

The Genetic algorithm has different variants for selection, crossover, and mutation process, but not every technique was tried due to the structural complexity of the code. The interconnection of code among different sections makes it a large complex Object-Oriented style. It was not possible to analyze with different GA variants. It may happen that with different parameters and settings, Genetic Algorithm can perform better than Particle Swarm Optimization.

Lastly, because of the structure of the code, this research could be implemented with even more complex topographical constraints, defining the assets on land, sea, or undersea; and providing limitations on asset physics and spatial movements.

REFERENCES

REFERENCES

[1] F. Johansson and G. Falkman, "Real-time allocation of defensive resources to rockets, artillery, and mortars," in *13th Conference on Information Fusion (FUSION), 2010*. IEEE, 2010, pp. 1–8.

[2] J. Reynolds, "Particle swarm optimization applied to real-time asset allocation," Purdue University MSECE Thesis, Indianapolis, IN, 2015.

[3] J. Crespo, "Asset allocation in frequency and in 3 spatial dimensions for electronic warfare application," Purdue University MSECE Thesis, Indianapolis, IN, 2016.

[4] P. R. Witcher, "Particle swarm optimization in the dynamic electronic warfare battlefield," Purdue University MSECE Thesis, Indianapolis, IN, 2017.

[5] W. Boler, "Electronic warfare asset allocation with human-swarm interaction," Purdue University MSECE Thesis, Indianapolis, IN, 2018.

[6] C. Wieczorek, "3D terrain visualization and cpu parallelization of particle swarm optimization," Purdue University MSECE Thesis, Indianapolis, IN, 2018.

[7] L. Christopher, J. Reynolds, J. Crespo, R. Eberhart, and P. Shaffer, "Human fitness functions," in *Swarm/Human Blended Intelligence Workshop (SHBI), 2015*. IEEE, 2015, pp. 1–4.

[8] L. Christopher, W. Boler, C. Wieczorek, J. Crespo, P. Witcher, S. A. Hawkins, and J. Stewart, "Asset allocation with swarm/human blended intelligence," in *2016 Swarm/Human Blended Intelligence Workshop (SHBI)*, October 2016, pp. 1–5.

[9] J. Kennedy and R. C. Eberhart, *Swarm Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[10] Y. S. Russell C. Eberhart, *Computational Intelligence: Concepts to Implementations*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2007.

[11] R. C. Eberhart, J. Kennedy *et al.*, "A new optimizer using particle swarm theory," in *Proceedings of the sixth international symposium on micro machine and human science*, vol. 1. New York, NY, 1995, pp. 39–43.

[12] R. C. Eberhart and J. Kennedy, "The particle swarm: social adaptation in information-processing systems," in *New ideas in optimization*. McGraw-Hill Ltd., UK, 1999, pp. 379–388.

[13] "Introduction to genetic algorithms," (last date accessed: 10/06/2018). [Online]. Available: https://tech.io/playgrounds/334/genetic-algorithms/history

[14] R. Hassan, B. Cohanim, O. de Weck, and G. Venter, "A comparison of particle swarm optimization and the genetic algorithm," *American Institute of Aeronautics and Astronautics*, 2004.

[15] S. Chaturvedi, P. Pragya, and H. Verma, "Comparative analysis of particle swarm optimization, genetic algorithm and krill herd algorithm," in *2015 International Conference on Computer, Communication and Control (IC4).* IEEE, 2015, pp. 1–7.

[16] S. Shabir and R. Singla, "A comparative study of genetic algorithm and the particle swarm optimization," *Int. J. Electr. Eng.*, vol. 9, no. 2, pp. 215–223, 2016.

[17] J. Sharma and R. S. Singhal, "Comparative research on genetic algorithm, particle swarm optimization and hybrid ga-pso," in *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom).* IEEE, 2015, pp. 110–114.

[18] K. O. Jones, "Comparison of genetic algorithm and particle swarm optimization," in *Proceedings of the International Conference on Computer Systems and Technologies*, 2005, pp. 1–6.

[19] J. Chen, B. Xin, Z. Peng, L. Dou, and J. Zhang, "Evolutionary decision-makings for the dynamic weapon-target assignment problem," *Science in China Series F: Information Sciences*, vol. 52, no. 11, p. 2006, 2009.

[20] X. Zeng, Y. Zhu, L. Nan, K. Hu, B. Niu, and X. He, "Solving weapon-target assignment problem using discrete particle swarm optimization," in *The Sixth World Congress on Intelligent Control and Automation, 2006. WCICA 2006.*, vol. 1. IEEE, 2006, pp. 3562–3565.

[21] B. Niu, T. Xie, Q. Duan, and L. Tan, "Particle swarm optimization for integrated yard truck scheduling and storage allocation problem," in *2014 IEEE Congress on Evolutionary Computation (CEC).* IEEE, 2014, pp. 634–639.

[22] S. N. Ohatkar and D. S. Bormane, "Hybrid channel allocation in cellular network based on genetic algorithm and particle swarm optimization methods," *IET Communications*, vol. 10, no. 13, pp. 1571–1578, 2016.

[23] H. Jiang, Y. Zhang, and H. Xu, "Optimal allocation of cooperative jamming resource based on hybrid quantum-behaved particle swarm optimization and genetic algorithm," *IET Radar, Sonar & Navigation*, vol. 11, no. 1, pp. 185–192, 2016.

[24] R. Hassan, B. Cohanim, O. de Weck, and G. Venter, "A copmarison of particle swarm optimization and the genetic algorithm," *American Institute of Aeronautics and Astronautics*, 2004.

[25] "A c++ library of genetic algorithm components," (last date accessed: 10/06/2018). [Online]. Available: http://lancet.mit.edu/ga/

[26] "Current versions of galib," (last date accessed: 10/06/2018). [Online]. Available: http://lancet.mit.edu/ga/dist/

[27] S. Sivanandam and S. Deepa, "Genetic algorithms," in *Introduction to genetic algorithms*. Springer, 2008, pp. 15–37.

[28] D. Hermawanto, "Genetic algorithm for solving simple mathematical equality problem," *Cornell University Library*, last date accessed: 10/06/2018. [Online]. Available: https://arxiv.org/abs/1308.4675

[29] S. G. B. Rylander *et al.*, "Optimal population size and the genetic algorithm," *Population*, vol. 100, no. 400, p. 900, 2002.

[30] "Introduction to genetic algorithms," (last date accessed: 08/16/2018). [Online]. Available: http://www.obitko.com/tutorials/genetic-algorithms/recommendations.php