

# A Study of the Application of Chaos to the Genetic Algorithm

by

Olawale David Jegede

A Thesis submitted to the Faculty of Graduate Studies of  
The University of Manitoba

in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg, Manitoba

Copyright © 2014 by Olawale D. Jegede

## Abstract

This work focuses on the use of a genetic algorithm for optimization in a search-based problem. The Genetic Algorithm (GA) is a subset of evolutionary algorithms that models biological processes to optimize highly complex functions. A GA allows a population composed of many individuals to evolve under specified selection rules to a state that maximizes the “fitness” (i.e. minimize the objective function). A major advantage of using GA over most stochastic techniques is its parallelism, which speeds up the simulation results leading to faster convergence. With mutation, the GA is also less likely to get stuck in local minima compared to other stochastic techniques.

However, some notable drawbacks of the Standard GA (SGA) include slow convergence and a possibility of being stuck in local optimum solution. The SGA uses a random process to generate parameter values for the initial population generation, crossover and mutation processes. Random number generators are designed to result in either uniform distributions or Gaussian distributions. We conjecture that the evolutionary processes in genetics are driven by a random non-linear deterministic dynamic process rather than a random non-deterministic process. Therefore, in the GA evolutionary process, a *chaotic* map is incorporated into the initial population generation, the crossover and mutation processes of the SGA; this is termed the Chaotic GA (CGA).

The properties of a chaotic system that provides additional benefits over randomly generated solutions are sensitivity to initial conditions, topological density and topological transitivity (robust diversity). These properties ensure that the CGA is able to explore the entire solution space. Introducing chaos into the whole process of a standard genetic algorithm may help improve convergence time and accuracy. Simulation was done using Matlab and Java.

## **Acknowledgements**

I would like to acknowledge the support of my advisor Dr. Ken Ferens. He has been more than an advisor to me. His patience, leadership, understanding and constructive criticism throughout the course of my studies have been exceptional. I appreciate his financial support and trust; indeed I am honored to have worked with him.

I would also like to thank Dr. Witold Kinsner for helping me throughout the course of my studies. He was there for me not only as a tutor but also as a father. He helped regain my confidence in my first few months in the program. Thank you for believing in me.

I would like to thank my examination committee members Dr. Bob McLeod and Dr. Chuang Deng for finding time to participate in my M.Sc. qualifying exam. The advice and corrections are very well appreciated.

A special thank you goes to the Department of Electrical and Computer Engineering for nominating me for the University of Manitoba Graduate Fund. This fund provided me with the financial stability I needed to complete my M.Sc. I would also like to thank the University of Manitoba for nominating me for the prestigious Manitoba Graduate Scholarship with which I had more financial stability. I am honored to have been a recipient. I am also very grateful to Dr. Douglas Buchanan for his role in getting an engage grant to support my studies. My M.Sc. story cannot be complete without recognizing the role Amy Dario played in gaining admission and then completing this work during the duration. Thank you for your guidance, objectiveness and effectiveness. I also want to say thank you to my undergraduate lecturers in Nigeria – Dr. Awodele and Dr. Omotosho for providing me with recommendation letters for the awards. Thank you to a colleague Ernest Onuiri for ensuring the recommendation letters got to me on time for the award.

I will also like to appreciate my colleagues in the same research group. It was a pleasure working with you. I am grateful to Taimoor Siddique and Thomas for helping with questions regarding part of the thesis and for technical support. A big thank you to Hasnain Khan for the sleepless nights we spent on java tutorial and translating the pseudo-code into the real code.

I am also grateful for the moral and financial support of my uncles and aunties to make this dream come true. The Fasesins', the Dadas', the Adeyemos', the Ogunlolus', the Shokunbis', the Ogundares', the Adedapos', the Georges', the Oyeyemis', the Adegunjus', the Babafemis' and the Jegedes'. I appreciate the motivation from Chukwuma Amobi, Dasola Oluge, Muyiwa Adelakun, Kazeem Adeogun and Enoch A-iyeh during the course of the program.

Last but not least, I acknowledge the love of God Almighty and my family. Thank God for a patient, caring, understanding and loving wife "OlutolaMi" who not only shared the vision but supported me through this adventure. Thank you to our prince Aseoluwa Olumoroti for sharing your Dad with UofM. I am grateful to my parents Arc. & Mrs. J.K. Jegede for being my pillar, my foundation and for supporting the dream. My sister Olaide Jegede is appreciated for the financial sacrifice and for sharing me with UofM. My in-laws, the Afilakas', have been wonderful and I am grateful for the understanding and support. A special thank you goes to my church family – Lighthouse of Hope Seventh-day Adventist Church – for being there to support me emotionally and spiritually since I stepped into Winnipeg.

## **Dedication**

I dedicate this work to God Almighty: the Author and Giver of all Wisdom and Understanding.

Through Him All Things were made;

Without Him nothing was made that has been made.

## Table of Contents

<b>Abstract.....</b>	<b>i</b>
<b>Acknowledgements.....</b>	<b>ii</b>
<b>Dedication .....</b>	<b>iv</b>
<b>List of Tables .....</b>	<b>ix</b>
<b>List of Figures.....</b>	<b>xi</b>
<b>Chapter 1.....</b>	<b>1</b>
<b>Introduction.....</b>	<b>1</b>
1.1 Thesis Motivation .....	1
1.2 Thesis Statement and Objectives.....	3
1.3 Contribution of the Thesis.....	4
1.4 Thesis Outline .....	6
<b>Chapter 2.....</b>	<b>8</b>
<b>Literature Review and Related Works .....</b>	<b>8</b>
2.1 Genetic Algorithms.....	8
2.1.1 Introduction .....	8
2.1.2 Selection.....	10
2.1.3 Crossover .....	17
2.1.4 Mutation .....	18
2.2 Drawbacks of Standard GA.....	19

2.3	Methods to Improve the Performance of GA .....	19
2.4	Intrinsic GA Defect .....	30
2.5	Chaos Theory.....	31
2.5.1	The Logistic Map .....	32
2.5.2	Characteristics of a Chaotic System .....	36
2.6	Motivation for Incorporating Chaos into Genetic Algorithms .....	39
2.7	Chaotic Genetic Algorithm .....	40
<b>Chapter 3.....</b>		<b>43</b>
<b>Genetic Algorithm, Chaos and Application .....</b>		<b>43</b>
3.1	Problem Classification .....	43
3.2	Application of GA to the Multiprocessor Task Scheduling Problem .....	45
3.2.1	Scheduling .....	45
3.2.2	Multiprocessor Task Scheduling .....	46
3.2.3	Computational Complexity .....	47
3.2.4	Related Work .....	48
3.2.5	Methodology: GA Approach to Multiprocessor Task Scheduling.....	58
3.3	Application of GA to the Radio Spectrum Allocation Problem .....	64
3.3.1	Fundamental Concept in Cognitive Radio .....	64
3.3.2	Cognitive Radio Spectrum Allocation .....	69
3.3.3	Computational Complexity of the Radio Spectrum Allocation Problem .....	71
3.3.4	Related Work .....	72
3.3.5	Methodology: GA Approach to Spectrum Allocation .....	73
3.4	Chaotic GA Optimization Approach .....	86

## **Chapter 4..... 90**

### **Experiments and Results ..... 90**

4.1	Introduction.....	90
4.2	Multiprocessor Task Scheduling.....	90
4.2.1	GA Parameter Setup .....	91
4.2.2	Results .....	91
4.3	Spectrum Allocation .....	94
4.3.1	GA Parameter Setup .....	95
4.3.2	Results .....	95
4.4	Convergence and Variance (Diversity) .....	103
4.4.1	GA Parameter Setup .....	103
4.4.2	Convergence: SGA vs. CGA.....	104
4.4.3	Variance: SGA vs. CGA.....	106
4.5	Effect of Population Size .....	109
4.5.1	Convergence: SGA vs. CGA.....	110
4.5.2	Variance: SGA vs. CGA.....	111

## **Chapter 5..... 115**

### **Conclusion and Future Work ..... 115**

5.1	Conclusion .....	115
5.2	Contributions.....	117
5.3	Future work .....	120

## **References ..... 121**

## **Appendix A ..... 1**

Software .....	1
----------------	---



A.1	Running the Task Graph Generator .....	2
A.2	Running the Task Scheduling GA Code .....	2
A.3	Running the Spectrum Allocation Code .....	3
<b>Appendix B</b>	.....	<b>1</b>
Source Code Files	.....	1
B.1	Task....	1
B.2	Task...	1
B.3	Task .....	1
B.4	Task .....	1
B.5	Spectrum Allocation .....	1
B.6	Spectrum Allocation .....	1
<b>Appendix C</b>	.....	<b>1</b>
Application of GA to Localization in Wireless Sensor Networks	.....	1

## List of Tables

1. Chaotic Genetic Algorithm Procedure	42
2. Height and Execution Time of Tasks	52
3. Task Order based on NTD	57
4. Genetic Algorithm Procedure	59
5. Spectrum Allocation GA Procedure	74
6. Chromosome Structure	75
7. Data Rate Gene	76
8. Signal Power Gene	76
9. Bit Error Rate Gene	77
10. Operating Frequency Gene	77
11. Modulation Technique Gene	78
12. Chromosome Configurations	78
13. Signal Power to Frequency Configurations	83
14. GA Parameters	91
15. Results	92
16. Results	93

17. QoS requirements of an application given as input to the process	94
18. GA Parameters	95
19. Resultant Spectrum and Corresponding Fitness Measure	96
20. Best and Worst Member of the Initial Population	101
21. Summary of Parameters' Performance	102
22. GA Parameters	104
23. GA Parameters	109

## List of Figures

1. Genetic Algorithm Flowchart	9
2. Plot of logistic map when $r$ lies between 0 and 3	33
3. Plot of logistic map when $r$ lies between 3 and $(1+\sqrt{6})$	33
4. Plot of logistic map when $r$ lies between $(1+\sqrt{6})$ and 3.54409	34
5. Plot of logistic map when $r$ lies between 3.56995 and 4	34
6. Plot of logistic map when $r$ lies between 2.5 and 4	35
7. A Task Graph	51
8. Height-based Task Schedule	53
9. Max-EST NTD-based Task Schedule	57
10. Min-EST NTD-based Task Schedule	58
11. Crossover Technique on Schedules $C_1$ and $C_2$	62
12. Mutation Technique on Schedules $C_1$ and $C_2$	63
13. The Cognitive Radio Cycle	67
14. Mobile Communication System	81
15. Two-Point Crossover	85
16. Probability Distribution of Logistic Map	88

17. Probability Distribution of New Chaotic Map	88
18. Initial Population Points with New Chaotic Map	89
19. Initial Population Points with Random Generator	89
20. Fitness measure of initial population in descending order	96
21. Fitness measure of parameter 1 over ten runs	97
22. Fitness measure of parameter 2 over ten runs	97
23. Fitness measure of parameter 3 over ten runs	98
24. Fitness measure of parameter 4 over ten runs	98
25. Fitness measure of parameter 5 over ten runs	99
26. Total fitness measure of spectrum solution over ten runs	99
27. SGA average fitness measure over 50 runs	100
28. CGA average fitness measure over 50 runs	100
29. SGA and CGA average fitness measure over 50 runs	101
30. SGA fitness per generation	104
31. CGA fitness per generation	105
32. SGA variance	107
33. CGA variance	108

34. SGA fitness measure of the last generation in descending order	108
35. CGA fitness measure of the last generation in descending order	109
36. SGA fitness per generation	110
37. CGA fitness per generation	111
38. SGA variance	112
39. CGA variance	113
40. SGA fitness measure of the last generation in descending order	113
41. CGA fitness measure of the last generation in descending order	114

# Chapter 1

## Introduction

The first section of this chapter provides the motivation for this thesis with a focus on the use of genetic algorithms. The second section describes the thesis statement and objectives while the third section outlines the contribution of the thesis. In the fourth section, we have described the thesis outline.

### 1.1 Thesis Motivation

The inherent problems associated with an exhaustive search within a large solution space for the purpose of optimization cannot be overemphasized. A search space is a collection of all possible solutions for specific problems. Depending on several variables for some particular applications, the computational complexity of the search may belong to any of the class of **nondeterministic polynomial time** (NP), NP-hard or NP-complete. The implication is that there are no known algorithms that can obtain the optimum solution in polynomial time. Intuitively, polynomial time implies that the execution time of the computation is no more than a polynomial function of the problem size. Therefore for enormously large search problems with time complexity regarded as NP-Hard or NP-Complete, the computation time for an exhaustive search is non-polynomial. It is practically impossible, application-wise, to seek for optimum solution in non-deterministic polynomial time.

It is desirable to obtain the optimum solution in polynomial time. However since the optimum solution cannot be guaranteed in polynomial time, a near-optimal solution obtainable in polynomial time is acceptable. Therefore, there is a need for such mechanisms that can guarantee

near-optimum solution in polynomial time. “Many computational problems require searching through a large number of possibilities for the optimum solution” [1]. The quests for efficacious algorithms is about finding intelligent ways around the exhaustive search process, using intelligent dynamics within such algorithms to narrow down the search space to obtain a near-optimum solution. Most of such algorithms developed to bypass the exhaustive search in approaching the problems are categorized as heuristics. Heuristics are stochastic search method introduced to obtain a near optimum solution in polynomial time. Examples include the Simulated Annealing, Particle Swarm Optimization, Greedy Algorithm, Artificial Neural Networks, Tabu Search, Hill Climbing and Evolutionary Algorithms (EA). These methods are able to obtain solutions regarded as good enough (near-optimal). Evolutionary algorithms have the capability to obtain more than one solution at a time whereas others can obtain only one solution at a time.

Evolutionary algorithms were independently developed and studied in the 1950s and 1960s by several computer scientists with the goal of solving optimization problem for several engineering problems. Darwin’s theory of evolution formed the basis upon which the idea of evolution is based. Evolution involves the generation of a set of population of possible solutions in the search space to a given problem with the aid of evolutionary operators that mimic the concept of reproduction in nature. The cost (fitness) function represents a heuristic estimation of the quality of any solution within the search space; the search process is driven by the variation (crossover and mutation) and the selection operators. The process of using the variation and selection operators is iterated until a solution with sufficient fitness (quality) is found or a set computational condition is met [2]. Rechenberg was first to introduce “evolutionary strategies” (*Evolutionsstrategie* in the original German), an idea that was further developed by Schwefel [1].



The various dialects of evolutionary computing only differ in technical details in terms of problem representation. They are Genetic Algorithms (GAs), Genetic Programming (GP), Evolution Strategies (ES), Evolutionary Programming (EP). When the search space consists of solutions represented by strings over a finite alphabet, then the appropriate EA to use would be the Genetic Algorithm [2]. Compared to other heuristics, the evolutionary algorithms, the simulated Annealing and the particle swarm optimization algorithm have an advantage called “*parallelism*” in that there are sets of potential solutions obtained at a time. This accounts for the speed with which solutions are obtained compared to other heuristics.

Genetic algorithms (GA) were developed by John Holland together with his students and colleagues in the 1960s. A Genetic Algorithm is an evolutionary algorithm that allows a set of initial population, each representing potential solution, to evolve under specified selection rules to a state that optimizes the cost (fitness) function. “Standard GA applies genetic operators such as selection, crossover and mutation on an initially generated random population within a search space; this is in order to compute a whole generation of new strings” [3]. This process is continued until the optimal solution is found or until a set stopping criteria is reached by which time a good solution would have been found. Genetic algorithm has been applied to solve many engineering optimization problems and is still widely used today.

## **1.2 Thesis Statement and Objectives**

In this thesis, the focus is on reducing the computational complexity involved in obtaining global optimum solution within a search space for NP-hard and NP-complete problems. This thesis proposes genetic algorithm as an evolutionary approach to reduce the computational complexity of such problems within polynomial time. The genetic algorithm is able to obtain

near-optimal solutions within polynomial time. Two different optimization problems have been examined in this thesis from a search-based perspective; genetic algorithms have been applied to these problems. The two problems are:

- a. Tasks scheduling in a multiprocessor system.
- b. Spectrum allocation in a cognitive radio networks.

We have also applied the GA to solve the localization problem in wireless sensor network [3]. A short description of the approach and the results obtained has been provided in Appendix C of this thesis. This work focuses primarily on how to improve the performance of the genetic algorithm using the chaos theory. Random sequences within a standard genetic algorithm were replaced with chaotic sequences using a logistic map.

### **1.3 Contribution of the Thesis**

The main contributions of this thesis are as follows:

- a. Spectrum Allocation in a Cognitive Radio Networks: The problem of finding optimal spectrum allocation in a cognitive radio networks has been proven to be NP-Complete [4][5][6]. Genetic algorithm has been used to obtain a ‘good enough’ spectrum allocation in polynomial time. Unlike previous works that have applied chaos to any one or two of the evolutionary processes of the GA, we have replaced the random sequences of all the evolutionary processes of the standard genetic algorithm with chaos using a logistic map. It is hoped that the inherent characteristics of a chaotic process will improve the ability of the genetic algorithm to find a near-optimal solution.

- b. **Tasks Scheduling in a Multiprocessor System:** The goal of task scheduling in a multiprocessor system is to schedule tasks on processors such that the processing time is minimized. This ensures optimal usage of the processing systems. The problem of obtaining optimal task scheduling in the multiprocessing system is reported to be NP hard [7] and heuristic based techniques can be used to obtain a good schedule in polynomial time. Several scheduling algorithms exists which can be used to make the initial set of schedules that is to be evolved by the genetic algorithm. Task scheduling based on the number of descendants of each task and their earlier start time has been found to be more optimal compared to task scheduling based on the height of each task in a task directed acyclic graph (DAG) [8]. In a task graph, some tasks can have more than one earliest start time as a result of multiple path of reaching such tasks in the graph. Our algorithm ensures the minimum of the multiple earliest start times is chosen, thereby increasing the ability of the genetic algorithm to obtain a near-optimal solution (schedule). We have assumed that the completion of the execution of one of the parent tasks satisfies the precedence relation in the graph. The genetic algorithm then uses its selection and variation operators in search of a near-optimal solution.
- c. We have also developed a user friendly software which can generate random task graphs. In generating the task graph, the user can specify the following parameters:
- the number of tasks
  - the number of levels desired
  - the maximum number of descendants possible per task
  - the range of random execution time

- d. We have also provided equations that can be used to obtain the actual time that a task may start execution on any of the processors and the updated available time at which an allocated task may start execution on any of the processor. The actual execution starting time was computed using the earliest start times and the updated available processor time. The updated available processor time was computed using the earliest start time and the execution time of each task.
- e. The thesis examines the idea of incorporating the logistic map into the evolutionary operations of a typical genetic algorithm with the aim of increasing the diversity within the solution space. Literature reveals that the more diverse a population is, the higher the chances of the genetic algorithm obtaining a near-optimum solution in polynomial time.
- f. This work further provides an analysis of the diversity of the solutions at every generation of the genetic algorithm using the ‘variance’ of the population. We have compared the variance of the standard genetic algorithm with the chaotic genetic algorithm.

## **1.4 Thesis Outline**

This thesis is organized into five chapters. Chapter two gives a detailed description of genetic algorithms and its mechanism of operation. The chapter also describes possible drawbacks of standard genetic algorithms and various ways on how to improve the performance of the algorithm. The chapter ends with a description of chaos theory and how it can be incorporated into the genetic algorithm processes. Chapter three describes the application of the genetic algorithms

to the multiprocessor tasks scheduling problem and the radio spectrum allocation problem. It also describes the application of the chaotic genetic algorithm to the radio spectrum allocation problem. Chapter four describes the experiments used to evaluate the performance of the standard genetic algorithm and the chaotic genetic algorithm. Finally, Chapter Five concludes the thesis by stating the notable contribution of this work to the application of genetic algorithm generally and specifically to the two optimization application areas.

## Chapter 2

### Literature Review and Related Works

This chapter provides a literature review of GA and some techniques that aid the operation of the algorithm. We have identified the drawbacks of the standard GA as well as various methods used to improve its performance. We have also discussed chaos theory and the motivation for using it to improve the performance of the GA. This we have termed Chaotic GA.

#### 2.1 Genetic Algorithms

In this section we review the genetic algorithms and the evolutionary processes it uses for the purpose of optimization.

##### 2.1.1 Introduction

Genetic algorithms are stochastic search algorithms that take their concept from nature [9][10][11][12]. They are heuristic search algorithms based on the Darwinian's concept of evolution – “survival of the fittest”. They belong to a class of evolutionary methods designed for the purpose of optimization for computationally cumbersome problems. They are suitable for search-based problems. GA maintains a population set of candidate solutions called *chromosomes*. Each chromosome is a collection of one or more components called *genes*. Each gene is encoded to form candidate solutions for a specific problem in a form that can be processed by a computing machine. Typically encoding of such candidate solutions can be done using *binary strings*.

The objective (cost) function of any particular problem is used to determine how well a solution solves the problem. In GA terminology, a *fitness function* represents the objective function while a

solution's *fitness value* represents how well the solution solves the problem. The objective is to find a near-optimal solution in polynomial time. The optimum solution is defined as the minimization/maximization of the fitness function. In this work, the near-optimal solution will be referred to as “good” solution. A standard genetic algorithm process/flowchart is shown in Fig.1.

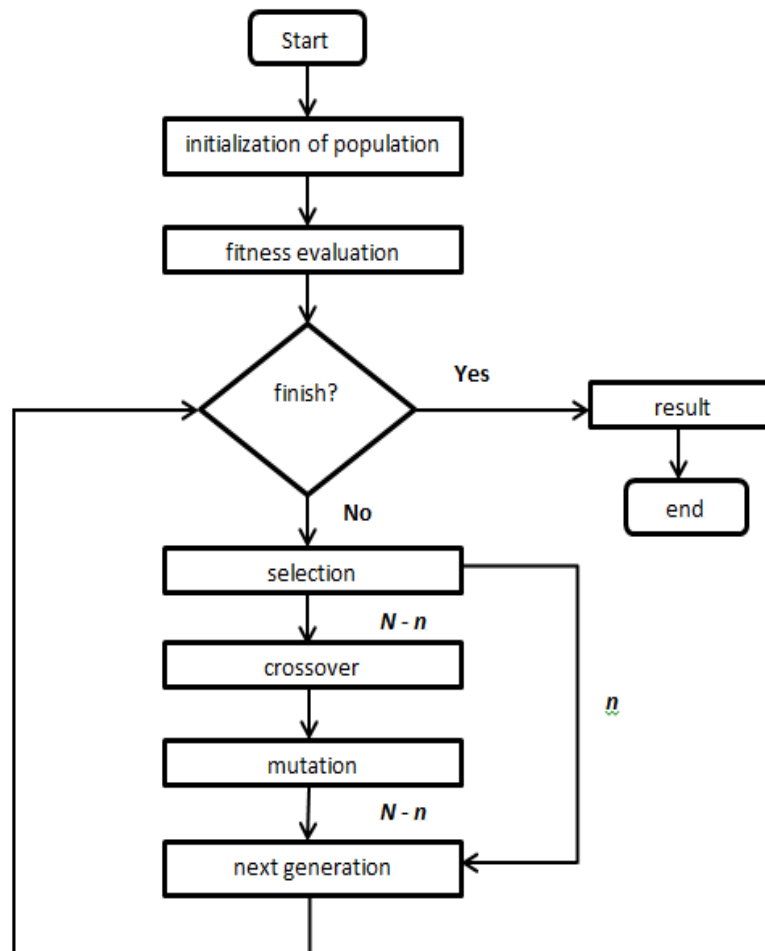


Fig. 1 Genetic algorithm flowchart

At the beginning of the algorithm, an initial population of candidate solutions is randomly generated. The fitness value of each of the candidate solutions is then determined using the fitness function. In the event that none of these fitness solutions is optimum or that the predetermined

computational criterion for stopping the algorithm is not met, the genetic algorithm then proceeds to generate a new set of solutions with the goal of obtaining a good solution. The new set of solutions is generated with the aid of the *selection* and *variation* operators. The *selection* operator is used to select a small number (usually two) of solutions on which the *variation* operators will operate. The *variation* operators comprise of the *crossover* (*re-combination*) and *mutation* operator. The *crossover* operator is used to exchange genes (properties) of candidate solutions, thereby producing new candidate solutions with different fitness values; this is analogous to *mating* in nature. The *mutation* operator is used to randomly change the genetic make-up of any one solution. The new solutions generated by the crossover and mutation operators are put back into the population to form a new population. These may represent unexplored points in the search space and may aid optimization in search of the best (optimum) solution.

Based on the fitness values, weaker solutions from the older population are replaced with stronger solutions in the new generation. The heuristic for doing this is termed the *replacement strategy*[13]. Each member of the new generation of solutions formed by the genetic operators is then evaluated using the fitness function. The genetic process (selection, crossover and mutation) of modifying the solutions in order to form a new population of solutions continues through generations until the stopping criterion has been reached. Examples of such stopping criteria may include *set number of generations*, *set value of fitness*.

### **2.1.2 Selection**

The fitness evaluation may be the most computationally intensive aspect of the GA. This evaluation depends on the complexity of the fitness function as well as the chosen population size. The ‘selection’ is the next stage after the fitness evaluation. Selection is the mechanism that



begins the process of genetically adjusting/modifying the candidate solutions for the purpose of seeking the optimum solution. At this stage, the individual candidate solutions are chosen from a population for the purpose of recombination or crossover. Intuitively, selection starts up the ability of the genetic algorithm to maneuver its way through the search space. The GA aims to improve the average quality (fitness) of the population set by giving candidate solutions that have relatively high quality (fitness value) a greater chance to be copied into the next generation of population. Selection concentrates the genetic algorithm search on promising regions within the search space [14]; thus, targeting potentially useful candidate solutions.

There are several selection methods that have been developed for efficient exploration of the search space. The designs of the variation operators influence the rate of convergence of the algorithm. The choice of selection method also influences the speed of the algorithm. Bickel and Thiele [14] carried out their work, compared and analyzed the various selection schemes used for the GA. Their work describes the influence of the various selection methods on the distribution of the fitness values of solutions. The various selection methods are described in this section. A generic selection procedure may be implemented as follows [15]:

1. The fitness value of each candidate solution is obtained using the fitness function, and then normalized by dividing the fitness value of each candidate solution by the sum of all candidate solutions' fitness values, so that the sum of all resulting fitness values equals 1.
2. Sort the population in descending order according to fitness values.
3. Compute the accumulated normalized fitness values of candidate solutions (the accumulated normalized fitness value of a candidate solution is the sum of its own

fitness value plus the fitness values of all the previous candidates). The accumulated normalized value of a candidate is the *probability* of the individual being selected.

4. Generate a uniformly distributed random number  $R$  between 0 and 1.
5. The selected candidate is the first candidate whose accumulated normalized value (probability) is greater than  $R$ .

#### 2.1.2.1 Roulette Wheel Selection

The roulette wheel selection method is the original method proposed for genetic algorithms by Holland. The method is also called the *fitness proportionate* selection. It is one of the earliest proposed methods of selection. In this method, the probability of a candidate solution being selected  $p_{cs}$  is proportional to the candidate solution's fitness value. It is proportional to the ratio of the candidate solution's fitness value to the sum of the fitness of the entire population of candidate solutions. It is implemented by repeating the generic selection procedure described in section 2.1.2 until there are enough candidates selected. This probability is given by:

$$p_{cs}(i) = \frac{f_i}{\sum_{i=1}^N f} \quad (2.1)$$

This method is popular among the users of genetic algorithm because of its notable advantage in that every candidate solution has a finite chance of making it to the next generation. A candidate solution that is poor (weak) in one generation may have some advantageous genes that may have been dominated by other “weak” genes. Thus, a poor candidate solution has a chance of making it to another generation where it is able to pass on its advantageous genes [16]. This results into a greater *genetic diversity* in the population, an obvious tradeoff with the number of good solutions within the population. Because this method involves sorting, the time

complexity for N population is given as  $O(N \ln N)$ [14]. Razali and Geraghty [16] have shown that although the tournament and roulette wheel methods can be superior to the rank-based roulette wheel method for smaller size problems, they become susceptible to premature convergence as the problem size increases. This stems from the fact that candidate solutions with better fitness have a higher chance of being chosen for crossover, and this can lead to premature convergence as a result of the dominance of solutions with similar fitness values. Genetic diversity or variation is a necessity for an improved performance of the genetic algorithm. The rate of evolution of the GA depends on the variance of the population's fitness values [1].

#### **2.1.2.2 Stochastic Universal Sampling (SUS)**

The stochastic universal sampling method was developed by James Baker [17]. It was developed based on the roulette wheel method with the aim of giving every candidate solution within the population the same probability of being selected for crossover. Unlike the roulette wheel method where a candidate's chance of being selected is proportional to its fitness, the SUS gives every candidate in the wheel equal chance of being selected. This gives "weaker" members of the population, based on their fitness, a chance to be selected, thus striking a balance between "exploitation" by highly fit candidates and "exploration" of the other regions of the search space. This approach reduces the bias nature of the roulette wheel method.

#### **2.1.2.3 Rank-based Roulette Wheel Selection**

In a rank-based selection strategy, the probability of a candidate solution being selected for crossover depends on its ranking (based on fitness) relative to the entire population. This method involves the sorting of the candidate solution according to their fitness. The selection probability is then computed according to the candidate's rank. The rank-based strategy allows the genetic

algorithm to explore the solution space and prevents exploitation by candidate solutions with higher fitness values. The method eliminates the disadvantage of the roulette wheel selection method by preventing premature termination of the algorithm, but it can be computationally expensive because of the need to sort the population. As a result of the sorting involved, the time complexity for N population is given as  $O(N \ln N)$ [14]. Two major divisions of the rank-based method are the *linear ranking* and *exponential ranking* method. Whereas these two algorithms are similar, they differ in the calculation of the selection probabilities.

#### **2.1.2.4 Truncation Selection**

The truncation method of selection is as the name '*truncation*' suggest. After sorting the candidate solutions with respect to their fitness, truncation method simply selects a fraction of the high-fitness candidate solutions. Compared to other selection methods, it is less sophisticated although can be computationally expensive because of the need to sort the population. As a result of the sorting of the population, the time complexity for N population is given as  $O(N \ln N)$ [14]. Another disadvantage of this method is that by cutting off “weak” candidate solutions that are below a set threshold from proceeding to the next generation, it automatically results to the loss of genetic diversity. This is a major reason why it is not commonly used in practice.

#### **2.1.2.5 Tournament Selection**

The tournament selection method is a method of selecting a candidate solution from a population of candidates within the search space. It represents a middle ground between the truncation and roulette-wheel selection methods. It involves randomly selecting a sub-population and then selecting the best fit candidate in the sub-population. This best fit candidate is the “winner” of the tournament. A compromise can be reached between selection intensity and

genetic diversity by changing the tournament size (i.e. sub-population size). A large sub-population size reduces the chance of “weak” candidates from being chosen, leading to loss of genetic diversity; but a small sub-population size can increase the genetic diversity. There is no sorting involved for this method; therefore, the time complexity of an  $N$  candidate population is given as  $O(N)$ [14]. Some advantages of this method include: it allows selection intensity to be easily adjusted depending on application, and it is easy and efficient to implement. In their work on GA performance with different selection strategies in solving the travelling salesman problem (TSP), Razali and Geraghty [16] have shown that although the tournament and roulette wheel strategies can be superior to the rank-based roulette wheel method for smaller size problems, they become susceptible to premature convergence as the problem size increases.

#### **2.1.2.6 Elitism**

Elitism is a selection strategy in which the best candidate or a percentage of candidates with better fitness is carried over to the succeeding generation unaltered. This prevents losing these better candidates to the activity of the variation operators (crossover and mutation). Elitism has been reported to significantly improve the performance of the genetic algorithms [1].

#### **2.1.2.7 Multi-Objective Selection**

The selection methods discussed earlier are designed for problems that have a single objective. For optimization problems where multiple criterion need to be simultaneously satisfied (multiple objectives), the fitness function becomes more complex. There exists selection strategies used to approach such multi-objective problems.

The simplest way is to use the *weighted sum approach* where a weight is assigned to each objective and then summed up into a single fitness function [14]. This method may be efficient for

problems with few objectives but becomes computationally complex with more objectives. Usually the objectives are contradictory and cannot have optimum solutions at the same time, thus one objective converges to optimum solution at the expense of all others. Thus there is a need for other methods that solve this problem. Selection methods that solve the convergence towards one objective's optimal solution belong to the group of Pareto methods. These types of methods obtain a set of non-dominated solutions from which any solution that best satisfies the requirements and needs can be chosen. Two methods exist for this group: the *non-dominated sorting* method and the *Pareto-optimal sorting*. For these two methods, before finding the Pareto-optimal candidates for a current generation, the Pareto-optimal candidates from the previous generation are added [18].

The non-dominated sorting method sorts the population and finds the Pareto-surface. All non-dominated candidates in the population are given the same rank (1), and are taken out of the population while the rest of the population is sorted again. A Pareto-surface is again found and the candidates forming it are given rank 2. This process continues until all the candidates get rank. The ranks are recalculated so that the rank of the candidates on the first Pareto-surface gets maximal [18]. The roulette wheel method is used. A typical advantage of this method is its fast selection and convergence. For the Pareto-optimal sorting, only the Pareto-optimal is/are taken from all candidates in a current generation. They are then recombined (crossover) with each other to generate the next generation. In the event there is only one candidate that dominates the whole generation the second Pareto-surface will be obtained and used for recombination (crossover). A disadvantage of this method is that several candidates "influence" the search process leaving the algorithm in danger of getting stuck in local minima [18].

### 2.1.3 Crossover

The crossover operation is one of the most important features of the genetic algorithm. It is performed immediately after the selection operation. Crossover also known as recombination is the mechanism by which the genetic algorithm is able to exchange the genetic properties (in blocks/segments) among candidate solutions in search for better solution. It is a major operator that propels the genetic algorithm search process. The crossover operator is engaged the most because it has a high rate of occurrence within the algorithm ranging from 80-95 percent depending on choice of the user.

There are several crossover techniques; the simplest one is the *single-point crossover*. In the single-point crossover, a single crossover position is randomly chosen thereby dividing a solution into two parts say A and B. In performing crossover, part A of one solution is merged with part B of a second solution while part A of the second solution is merged with part B of the first solution. This leads to the formation of two new candidate solutions which are likely to have differing fitness function from one another and from the parent solutions. Eshelman et al [19] pointed out a drawback of the single point crossover known as “positional bias” which results from the inability of the single-point crossover to deal with all possible points combinations. The positional bias implies “the schemas (blocks) that can be created or destroyed by a crossover depend strongly on the location of the bits in the chromosome” [19]. Also it has been noted that the single point crossover treats some loci preferentially: the segments exchanged between the two parents always contain the endpoints of the strings. [loci: the chromosomal position(s) of a gene as determined by its linear order relative to the other genes on that chromosome]. The *two-point crossover* was developed in order to overcome the drawbacks of the single-point crossover. The two point

crossover involves the random selection of two positions and the exchange of the segment between them. It is less likely to destroy blocks having long length and can combine more points than the single-point counterpart. Moreover, the segments that get exchanged do not necessarily contain the endpoints of the strings [19]. However the more the number of possible blocks there are to be exchanged, the less efficient the 2-point crossover can be.

There are several n-point crossover points that have been proposed by GA researchers; the choice of which is best to use will depend largely on several factors including the encoding of the candidates as well as the fitness function. Spears and De Jong [20] proposed a parameterized uniform crossover that sets up a probability of exchange happening at each bit position. This has no “positional bias” and can work well in situations of possible large block exchange. However this scheme can hinder co adapted allele (alternative form of a gene) from ever forming in the population.

#### **2.1.4 Mutation**

The mutation process is the final step of the genetic algorithm. This process occurs just before evaluating the new generation of population. Mutation is a genetic operator used to maintain genetic diversity from one generation of a population to the next. It alters one or more gene (characteristic) values in a candidate solution from its initial state. Mutation is applied to any random “gene” of the “chromosome” obtained after crossover, altering a binary bit from 0 to 1 or vice versa [21]. In mutation, the solution may change entirely from the previous solution; hence, the algorithm can come to better or worse solution by using mutation. Mutation occurs during evolution according to a user-definable mutation probability. This probability should be set as low as possible. If it is set too high, the search will turn into a primitive random search. The mutation



rate should be set as low as 1%-3%. In some cases, depending on the application, the candidate solution is converted to binary form for the purpose of mutation and converted back after mutation is done. Mutation reduces the chance of the GA getting stuck in local optima.

## **2.2 Drawbacks of Standard GA**

The genetic algorithms can be used for a wide variety of optimization problems because of its versatility and robustness. However, there are two major weaknesses associated with the standard genetic algorithm. The first is that, depending on the complexity of the fitness function and the method of encoding of the candidate solution, genetic algorithms can converge very slowly in their quest to finding the global optimum; there is no guarantee of finding the best solution within polynomial time, thus leading to *premature termination*. A second weakness, common to all evolutionary algorithms, is that the algorithm does not know for certain when to stop searching except for the preset time it is allowed to explore by the user; therefore, determining a stopping criterion is not straight forward [22]. Several works has been done and research is still ongoing to address these two weaknesses. Some of the works that have been done are described in the next section.

## **2.3 Methods to Improve the Performance of GA**

There are various techniques that have been used to improve the performance of the genetic algorithm. Majority of these techniques focus on *diversity control*; a more diverse solution may lead to a better solution. As the GA evolves, the diversity within the population is bound to reduce over the generations and this can lead to premature convergence. Premature convergence leads to the algorithm getting stuck in a local optima, thereby further preventing it from obtaining a better solution.

### 2.3.1 Diversity Control

The various techniques proposed to improve the performance of the genetic algorithm with focus on diversity control are discussed in this section.

#### 2.3.1.1 Modified Restricted Mating with Multiple Subpopulations

Jassadapakorn and Chongstitvatana [23] proposed a diversity control mechanism to solve the premature convergence problem. Intuitively, the more diverse the candidate solutions are, the more progress will be made by the evolutionary process in coming to a better solution. However, the formulation of the diversity control mechanism will vary depending on the complexity of the optimization problem. The main idea is based on a *modified restricted mating* concept where only candidate solutions with high diversity are selected for crossover, thus aiding the algorithm to better solutions within the search space. Jassadapakorn and Chongstitvatana have used two mechanisms to control diversity: *a modified restricted mating* and *multiple subpopulations*.

In the *modified restricted mating*, each candidate has a preference for its partner and this depends on “*preference type*” which is a parameter for controlling the degree of the difference of two candidate solutions to be crossed-over. Thus by controlling the preference type, the algorithm is able to influence the diversity of the population. The first candidate is selected based on any of the selection methods discussed in section 2.1.2 (tournament selection was used by [24]), and then the preference type is used to calculate the chance of another candidate being selected as the first candidate’s partner. The criterion for selecting the second candidate depends on the difference function given by (2.3) and the fitness value expressed in (2.2) [23].

$$x_2 = \arg \max_{i \in S_t} [f(c_i) \cdot D(\tau, d_i)] \quad (2.2)$$

where  $x_2$  represents the second selected partner,  $c_i$  is the  $i^{th}$  candidate randomly selected from the population,  $f$  is the fitness function, and  $s_t$  is tournament size (tournament selection),  $d$  is the difference between the first selected candidate and its partner,  $\tau$  is the preference type, and  $D$  is a function of  $d$  and  $\tau$  called the “*difference function*”. The candidate with a higher  $D$  value has a higher probability of being selected as the second partner. The linear difference function used is given by (2.3), where  $\tau_{max}$  represents the maximum preference type,  $0 \leq \tau \leq \tau_{max}$ ,  $d_i$  is the difference between the first selected candidate and the candidate  $c_i$ .

$$D(\tau, d_i) = 0.5 + \frac{\tau}{\tau_{max}}(d_i - 0.5) \quad (2.3)$$

The equation to calculate  $d_i$  is given by (2.4), where  $h$  is the Hamming distance of two individuals, and  $l$  is the length of chromosome.

$$d_i = \frac{h(c_i, x_l)}{l} \quad (2.4)$$

From (2.3), it is observed that when  $\tau$  is 0, the probability of selection is not determined by  $d$ ; thus, (2.2) becomes equivalent to the standard method of selection where the probability of being selected is determined only by the fitness value of each candidate. Also, a higher value of  $\tau$  accords more weight to the difference between candidates, thus influencing the population towards more diversity. This method is computationally expensive. Multiple subpopulations are evolved using different diversity (thus *preference type*) in order to determine the appropriate diversity value for a problem [23]. In effect, the multiple subpopulations compete for computational resource. Once each subpopulation’s effectiveness in solving the problem is evaluated, the sub-population that performs well will be maintained while the remaining (inferior) ones will be eliminated; thus leading to the concentration of computational resource on the promising subpopulation.

Experiment performed for this proposed method shows that although the subpopulation with lower diversity performed better at the early generations compared to those with higher diversity, the subpopulations with the higher diversity eventually outperforms those with lower diversity as time (generation) progressed.

### **2.3.1.2 Hashing of Fitness Values**

Povinelli and Feng [24] approached the convergence problem from the perspective of the computational problem of the standard GA. They argue that in a genetic algorithm, most of the computational time is spent on the evaluation of the fitness function. In a study of the convergence criteria and diversity characteristics of an evolving GA, it was found that the fitness evaluation of the same candidates was *frequently recalculated*. This is an opportunity for improvement of the algorithms performance.

Povinelli and Feng [24] proposed a method to improve the performance of the genetic algorithm. This method is also based on the idea of diversity control similar to the ones proposed by [23]. However, unlike the method proposed by [23] where the diversity is maintained using a modified restricted mating and subpopulation with varied diversity, [24] proposed a diversity control method based on the use of a hash table to store the most recently evaluated candidates. This is to ensure that the fitness values of such candidates are not reevaluated once again in the event that any of them makes it to subsequent generations. Experiments performed using the hashing method revealed that for simple search based problems (such as first-order and second-order polynomial), where the costs of calculating the fitness is simple, hashing has no effect on the performance of the genetic algorithm compared to the no hashing version of the genetic algorithm. Therefore the introduction of hashing does not change the genetic algorithm's

optimizing characteristics. But for complex search problem (such as data mining) where the cost of evaluating the fitness is overwhelming, hashing was found to reduce the computational demand by over 50 percent. Therefore the hashing of fitness values method is suitable for complex real-world problems.

### 2.3.1.3 Adaptive Mutation Rate Control

Another technique developed to address the issue of loss of genetic diversity is the *adaptive mutation rate control*. Several works [25][26][27][28][29] has been done to show the benefit of varying the mutation probability as compared to a fixed mutation rate. An adaptive mutation rate allows the algorithm to do a better search because the mutation rate will be varied based on feedback information extracted on the performance of the algorithm as the population evolves. This frees the user from making pre-determined decision on the mutation rate beforehand and is particularly useful since the fitness characteristics of the population with evolution cannot be known beforehand. In his work, Thierens [30] proposed two simple adaptive mutation rate control schemes principally adapting the mutation rate to the fitness characteristics of the population as they evolve with time (generation). The two schemes are: the *constant gain* adaptive mutation rate and the *declining* adaptive mutation scheme.

The *constant gain* adaptive mutation rate control scheme's idea was borrowed from the stochastic Manhattan learning algorithm from the field of stochastic approximation. The "stochastic learning algorithms provide recursively refined estimates of optimal model parameters" [30]. It uses indices such as the *learning rate* and *input data* to estimate optimal model parameters at recursive time step. Thierens [30] introduced this concept into the evolutionary process. For a current mutation parameter value of  $p_m(t)$ , two new candidates are

produced by mutating the current candidate with a mutation rate  $p'_m(t) = \omega p_m(t)$  and  $p'_m(t) = p_m(t)/\omega$  where  $\omega$  is a constant called the *exploration factor*. Evaluating the fitness value of the new mutated candidates indicates whether the current mutation rate should be modified (decreased or increased). A multiplicative constant learning factor  $\lambda$ , which is proportional to the current mutation probability, is used to make the modifications. The learning factor  $\lambda$  and the exploration factor  $\omega$  typically have different values ( $\lambda, \omega > 1$ ); typical appropriate values are  $\lambda = 1.1$  and  $\omega = 1.5$ .

The *declining adaptive* mutation control is a variant of the constant gain method. This variant “aims for a more aggressive step size while retaining a smooth dynamics” [30]. Every time a candidate produces an offspring by mutation, the mutation rate is reduced by a small factor known as the *declination factor*  $\gamma$ . Also, the exploration towards lower mutation rate values is replaced by an exploration towards higher values. The adaptation of the step size is also made more aggressively than in the constant gain method. The *declination factor*  $\gamma$  has typical appropriate values  $0.9 \leq \gamma < 1$ . A formal definition of the algorithm is explained in [30]. Experiment was carried out by applying the two techniques and a fixed mutation rate method to the Counting Ones problem and the Zero/one multiple knapsack problem. Results obtained showed that these two techniques converge faster than the fixed mutation rate method. Also the declining adaptation method was seen to converge faster than the constant gain counterpart.

#### **2.3.1.4 Social Disaster Technique (SDT)**

The *Social Disaster Technique* (SDT) was developed by Kureichick et al [31] with the aim of avoiding premature convergence to local optima. The goal is to prevent loss of genetic diversity with evolution. The algorithm modifies the standard greedy crossover which is ‘less’ greedy. The standard greedy crossover operates in such a way that it recombines ‘good’ candidate solutions

with very bad ones producing a candidate with almost exactly the same fitness as the ‘good’ one. This does not actualize the objective of preventing loss of genetic diversity. Therefore a modification was made such that when a very good candidate is selected to be crossed-over with a very bad candidate, the algorithm randomly generates new candidates to represent the offspring.

The manner in which the random candidate is generated is *catastrophic* in nature and deemed similar to *social disasters* in the real world. Two different operators used to generate the random candidate are:

- a. *Packing*: In a population, when it is detected that a certain number of candidates have the same fitness value, only one of such candidates is kept unchanged while the others are completely randomized to produce new population.
- b. *Judgment Day*: In a population, the candidate with the best fitness is kept unchanged while the others are completely randomized to form new population.

This proposed technique was applied to the Traveling Salesman Problem (TSP) and it was reported that results obtained showed significant improvements both for small and large population size compared to the standard genetic algorithms. It was conjectured that the catastrophic operator was useful in getting the population out of local minima.

#### **2.3.1.5 Random Offspring Generation (ROG)**

Rocha and Neves [32] proposed another technique to approach the premature convergence of solutions coding local optima of the objective function. This problem as usual occurs as a result of loss of genetic diversity as the genetic algorithm evolves over generations, leading to a decrease in the performance of the algorithm.

The proposed technique is very much similar to the *modified restricted mating with multiple subpopulations* and the *hashing of fitness values* described previously. The main idea of the Random Offspring Generation technique is to ensure that no two candidate solutions with similar genetic characteristics are allowed for crossover operation. In the event of similar characteristics, one candidate (*1-RO*) or two candidates (*2-RO*) will be randomly generated to be the new offspring and offspring respectively to form part of the next generation of population. The two different cases (*1-RO*) and (*2-RO*) differ only on the number of offspring generated. Experiment was carried out by applying this technique and two other well-known techniques (Adaptive Mutation Rate and Social Disasters Techniques) to the *travelling salesman problem* (TSP). The results obtained revealed that the ROG performed better than the other two techniques; this is because of the fact that the ROG ensures maintenance of the genetic diversity.

### **2.3.2 Tuning of the Genetic Algorithm Parameters**

Angelova and Pencheva [33] proposed a method which involves changing the sequences of implementation of the three main genetic operators (selection, crossover and mutation) in the genetic algorithm. Their proposal also considered the impact of the important genetic algorithm parameters -*generation gap*, *crossover*, and *mutation rates* on the convergence time.

The standard genetic algorithm (SGA) searches for a global optimum solution with the genetic operators in the following sequence: selection, crossover and mutation. Goldberg [9] introduced the basic multi-population genetic algorithm (MpGA) which has the same sequence of implementation of the genetic operators as the SGA. The MpGA differs from the SGA in that the MpGA has many populations referred to as *subpopulations*. The evolution of each of the subpopulations occur independent of each other for a specific number of generations, after which



a number of candidate solutions are distributed between the subpopulation. The SGA and MpGA with the standard sequence of selection, crossover and mutation are denoted as SGA-SCM and MpGA-SCM respectively. Each of these operators plays a crucial part in helping the algorithms to obtain a near-optimum solution.

Over the years, improvements have been made to the SGA-SCM and MpGA-SCM and this involves modifying the sequence of the main genetic operators within the algorithm, leading to some variation of the SGA and MpGA. One of such variations is the modified genetic algorithm with a sequence *crossover*, *mutation*, and *selection* [34], otherwise denoted as SGA-CMS and MpGA-CMS. The main motivation for this sequence is to prevent current (found) good solution from being crossed-over or mutated, thereby preventing the loss of such good solution. However, the *elitism* technique is known to take care of the prevention of loss of good solution. Another variation is a modified genetic algorithm with a sequence that reverses the CMS order; that is *selection*, *mutation* and *crossover* (SMC). Thus we obtain the following modifications: SGA-SMC and MpGA-SMC. Another variation puts the genetic operators in this order: *mutation*, *crossover*, and *selection* thus leading to the modifications SGA-MCS and MpGA-MCS. These various modifications of the SGA and MpGA, with varied rate settings of the parameters: generation gap (GGAP), crossover (XOVR) and mutation were applied to a fermentation process in order to optimize parameter identification of a fed-batch cultivation of *S. cerevisiae*. The results were analyzed and it was found that up to 40 percent computation time can be saved in cases of the SGA-MCS and the MpGA-SCM application using GGAP = 0.5 instead of 0.9 without loss of model accuracy. Meaning that the sequence of genetic operators in the order of: *mutation*, *crossover* and *selection* is the most optimal in terms of convergence time; this sequence also

ensures high decision accuracy. Furthermore, it was concluded that using different values of mutation and crossover rates has no effect on the convergence of any of the variations of the SGA and MpGA, but a crossover rate of 0.85 was assumed to be more appropriate for optimal performance. The mutation rate typically ranges between 0.01 and 0.1.

### 2.3.3 Solution Acceleration

Wong and Li [35] developed the *solution acceleration* technique in order to improve the convergence characteristics of the genetic algorithms by improving the speed of the evolutionary algorithm. They have applied this technique to a constrained-genetic algorithm load flow (CGALF) algorithm for solving the problem of evaluating the voltage profile and power flow in electric power networks.

The solution acceleration techniques have been previously used in iterative methods to determine solutions of unknown variables in a set of simultaneous equations. From the initial set of solutions at the first generation up to the last generation, accelerated solutions are produced per generation. The accelerated solutions are generated using Equation 2.5, and then used to generate the population at subsequent generations. The accelerated solution of a variable  $x$  at the  $i^{th}$  generation (iteration) is estimated from its solutions at the  $i^{th}$  and the  $(i-1)^{th}$  generations according to (2.5).

$$x_{ac}^i = x^{i-1} + \alpha (x^i - x^{i-1}) \quad 2.5$$

where  $x_{ac}^i$  represents the accelerated solution of a variable  $x$ ,  $x^i$  is the solution at the current generation,  $x^{i-1}$  is the solution at the previous generation,  $\alpha$  is a constant coefficient. “The acceleration mechanism is started by setting the value of  $\alpha$  to a value greater than unity [35]”.

In a genetic algorithm, the candidate solutions at any generation  $i$  that make up a population  $P(i)$  are considered to be possible solutions for any particular optimization problem. The best fit candidate solution among the population can be taken as the most appropriate solution  $S_b$  in the current generation  $i$ . In order to accelerate the process of obtaining the optimum solution, the other candidate solutions in a population  $P(i)$  are moved closer to  $S_b$  as outlines in the following three steps:

- a. Let  $S$  represent the candidate solution in  $P(i)$
- b. Obtain the difference between  $S_b$  and  $S$
- c. Scale the *difference* and add it to  $S_b$  to obtain a new solution

These three steps ensure that all of the candidate solutions are accelerated. The accelerated candidate solutions are then used to generate the candidate solutions for the next generation  $P(i+1)$  through the use of the genetic operators. For this method however, it is important to know how to appropriately set the constant coefficient  $\alpha$ , used to scale the difference between  $S_b$  and  $S$ , in order to ensure optimal performance of the algorithm for any specific application. The authors [35] explain that this coefficient can be estimated using any of two means:

- a. By experiment
- b. By randomly setting the value within a specific range in the entire evolutionary solution process.

This technique has been applied to examine the light and heavy load conditions of the load flow problem, specifically for solving the problem of evaluating the voltage profile and power flow in electric power networks. It was found that for light load, convergence occurred at the 12<sup>th</sup> iteration with the solution accelerator, whereas convergence occurred at more than 200 iterations without the solution accelerator. For heavy-load case, convergence occurred around the 5<sup>th</sup> and

6<sup>th</sup> iteration using the solution accelerator, whereas convergence occurred at more than 200 iterations without the solution accelerator. Therefore the proposed method of solution acceleration can greatly reduce the computing requirement of a genetic algorithm.

## 2.4 Intrinsic GA Defect

The various conventional improvement strategies explained above are not without setbacks. There still remains a tendency to get stuck in local optima and converge prematurely; this can be explained by several reasons.

### i. Random Walk in Diversity Control

The most important factor is the ability of the algorithm to maintain a consistent diversity within the population as the algorithm evolves. An advantage of the diversity is that it ensures the GA does not get *stuck* in the local minima. Although most of the conventional improvements discussed focused on the *diversity control* method of generating solutions within the search space, the manner of generating such diverse solutions is *random* in nature. This *random walk* within the search space implies that obtaining the optimum solution is *probabilistic* in nature and as such there is no guarantee that this optimum solution will be found.

### ii. Genetic Operators' Effect

Each of the diversity/convergence improvement methods above makes use of the genetic operators to achieve diversity and/or speed up convergence. However, comprehensive diversity of the population cannot be guaranteed as a result of the probabilities associated with each of the genetic operators. This is also irrespective of whether an adaptive or fixed rate approach is employed for any of the operators. There is a need for a mechanism that aids the operators in overcoming this shortcoming.

### iii. Effect of Evolution

The standard GA and its improvements have a common defect – they are ignorant of the candidates’ experiences during evolution[36]. As a result of the random-nature of the search, there exist no necessary connections between the current and next generations except for some controlling parameters such as the choice of selection strategy, as well as the crossover and mutation probabilities. Thus, the feedback information from former population is discarded.

These intrinsic defects observed in a standard GA, largely attributed to the random nature of the search, has led to further research in finding an efficient approach to “*walking*” through the search space. The aim is for the algorithm to have the ability to *walk* through the search space in such a way as to overcome the setbacks mentioned above. This is the motivation for employing the concept of *chaotic walk* wherein the manner of searching through the search-space is *chaotic* rather than *random*.

## 2.5 Chaos Theory

Chaos is a phenomenon exhibited by deterministic dynamical system. Chaos is a state whereby a dynamical system, which can be described by a deterministic equation, behaves unpredictably [37]. This unpredictable behavior is also a characteristic of random influence. However a random influence is non-deterministic in nature. The study of chaos started in the 1800 by Henri Poincare and was continued by other researchers such as John von Neumann, Edward Lorenz, and George D. Birkhoff. Others were T.Y. Li, James York and A.M. Sharkovskiy. The deterministic model used to study this behavior evolved from the introduction of a model of population growth without feedback to one with feedback. The model with feedback is what is popularly known today as the *logistic map*.

### 2.5.1 The Logistic Map

The logistic map is a polynomial map of the second order. It is used to explain how a simple nonlinear dynamical system exhibit complex, chaotic behavior. The logistic map is given by:

$$x_{n+1} = r x_n (1 - x_n) \quad (2.6)$$

where  $x$  is a number ranging from *zero* to *one*,  $x_n$  is the initial (existing) value of  $x$  at the iteration  $n$ ,  $x_{n+1}$  is the new value of  $x$  at the iteration  $n+1$ ,  $r$  is the net  $x$  growth rate (“net” refers to a combined rate between reproduction and extinction). The behaviors exhibited by the logistic map are a function of the value of the parameter  $r$ . Figures 2-5 show these behaviors. When  $r$  lies between 0 and 3, the logistic map will converge to some value  $x$ . For instance, Figure 2 shows that when  $r$  is 2, the map converges to a value “0.5” of  $x$  for 4 iterations. The map exhibits *periodic-doubling*, termed *bifurcation*, when  $r$  is greater than 3 and less than or equal to approximately 3.56994. Bifurcation implies the permanent oscillations of the map between specific values of  $x$ . These bifurcation points are also periodic depending on the interval of  $r$  within the bifurcation band. The bifurcation process follows the order below while each of Figures 3 and 4 is an example of bifurcation described by items a and b respectively.

- a. For  $3 < r \leq (1+\sqrt{6})$ ,  $x$  cycles between 2 values depending on  $r$ .
- b. For  $(1+\sqrt{6}) < r \leq 3.54409$ ,  $x$  cycles between 4 values depending on  $r$ .
- c. For  $3.54409 < r \leq 3.564407$ ,  $x$  cycles between 8 values depending on  $r$ .
- d. For  $3.564407 < r \leq 3.568759$ ,  $x$  cycles between 16 values depending on  $r$ .
- e. The period doubling goes on until  $r_\infty = 3.569944\dots$

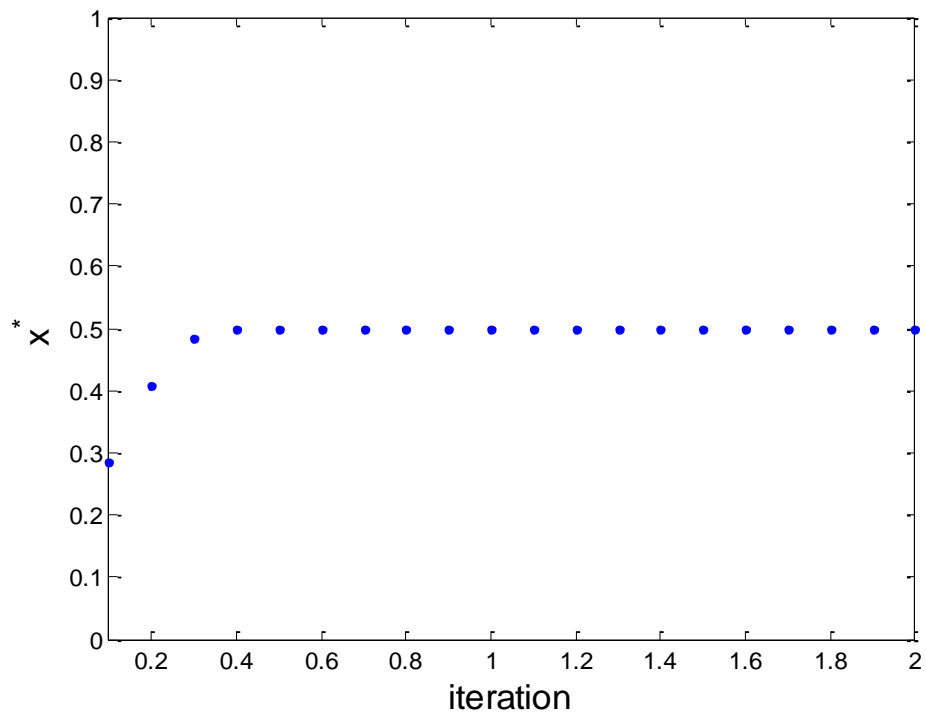


Fig. 2 Plot of logistic map when  $r$  lies between 0 and 3

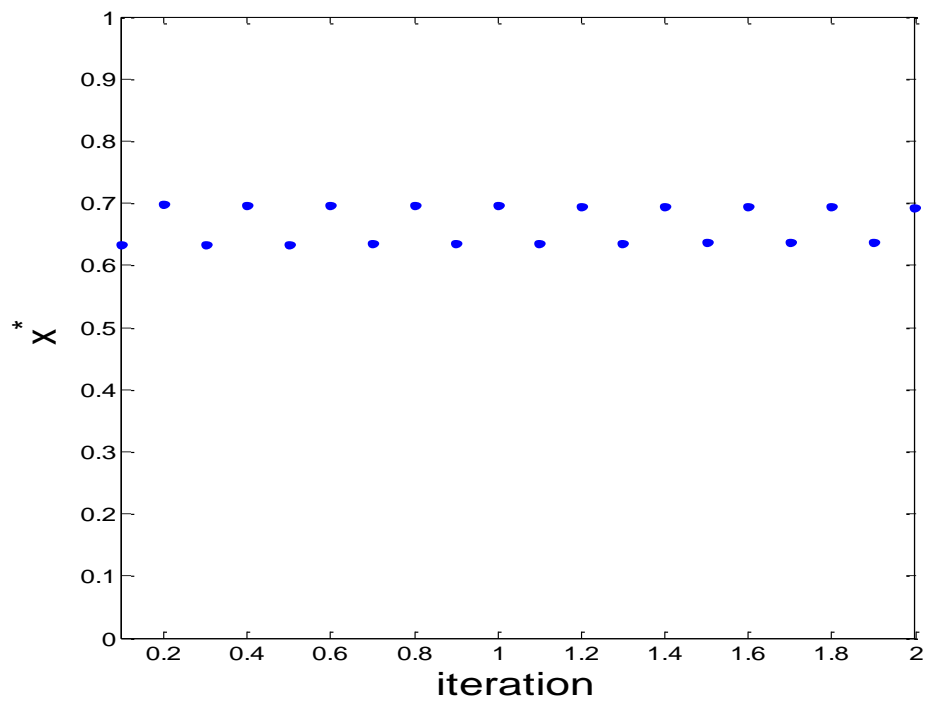


Fig. 3 Plot of the logistic map when  $r$  lies between 3 and  $(1+\sqrt{6})$

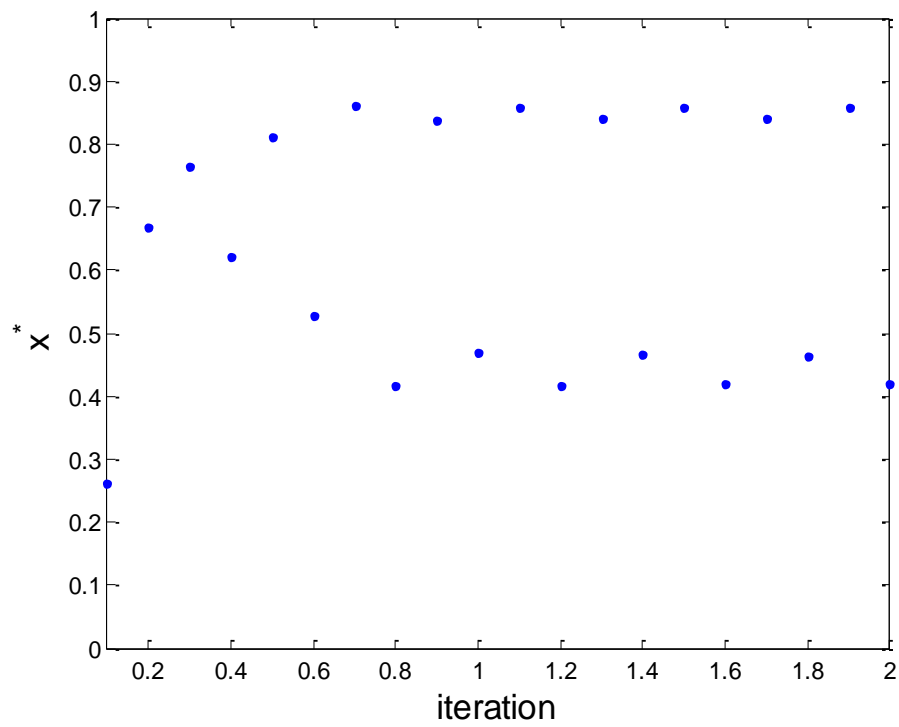


Fig. 4 Plot of logistic map when  $r$  lies between  $(1+\sqrt{6})$  and  $3.54409$

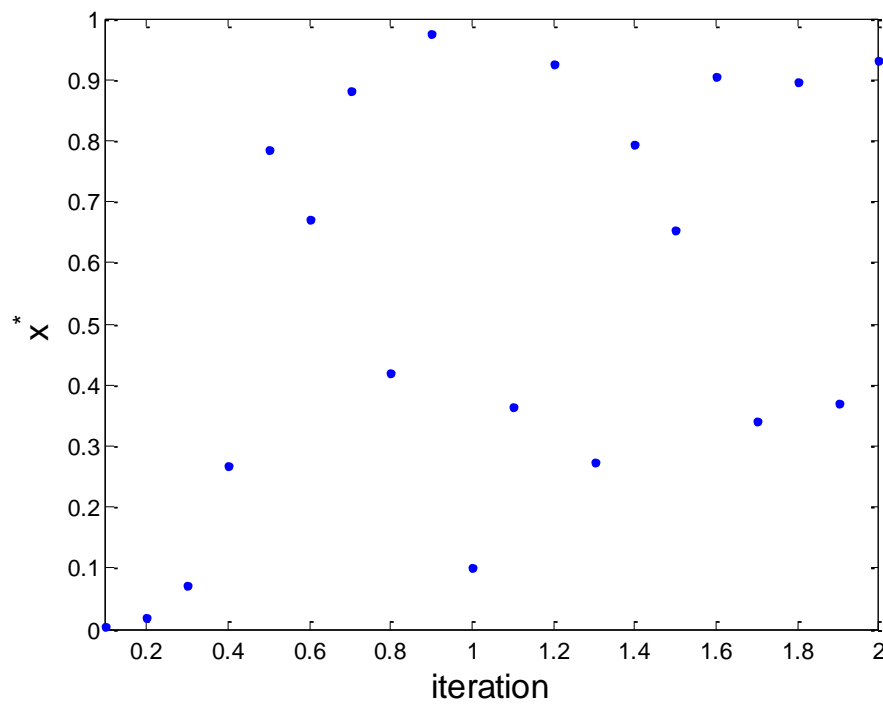


Fig. 5 Plot of logistic map when  $r$  lies between  $3.56995$  and  $4$



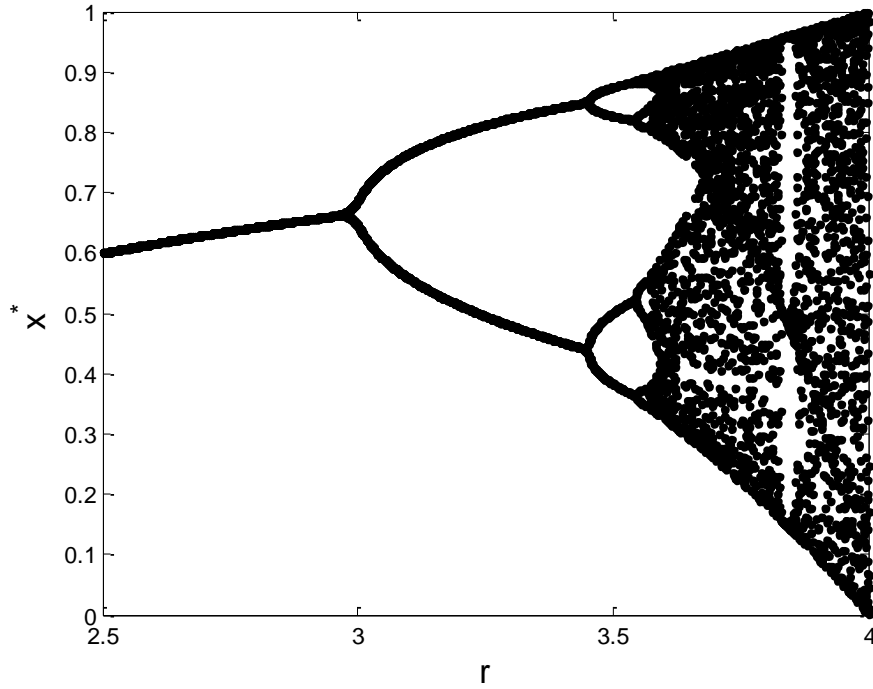


Fig. 6 Plot of logistic map when  $r$  lies between 2.5 and 4

It is observed that the spacing between successive bifurcation intervals becomes smaller and smaller till it reaches the limit value  $r_{\infty} = 3.569944$ . The ratio between these successive bifurcation intervals has been established to converge to a constant  $\delta$  known as the *first Feigenbaum constant*. This constant has been estimated to be:

$$\delta = 4.669\ 201\ 609\ 102\ 990\ 671\ 853\ 203\ 820\ 466\ 201\ 617\ 258\ 185\ \dots$$

This constant has been discovered to be *universal* in that it does not depend on any specific type of map, provided the map is *unimodal* and of *the same degree in the same number of embedding dimensions*. A unimodal map is one that has the ability to remap itself to an initial starting point after going through all the possible trajectories. Examples of unimodal maps are the tent map, the sine map and the logistic map. As the bifurcation points get closer, the ratio/densities become indistinguishable. This is the route that leads to *chaos*. This phenomenon known as Chaos occurs

when  $r$  lies between 3.56995 and 4 as shown by Fig. 5. Figure 6 represents the logistic map behavior when  $r$  lies between 2.5 and 4. The map is convergent when  $r$  lies between 2.5 and 3, while it bifurcates for  $3 < r \leq r_{\infty} = 3.569944$ . Beyond the point  $r_{\infty} = 3.569944$ , the map becomes chaotic. The characteristics of a chaotic system are described next.

## 2.5.2 Characteristics of a Chaotic System

According to Robert Devaney [38][39], chaos can be defined in terms of three topological properties. A map  $q: X \rightarrow Y$ , where  $X$  is a metric space, is said to exhibit chaos if:

1.  $q$  is *transitive*;
2. the set of periodic points  $P$  is dense in  $X$ ; and
3.  $q$  has sensitive dependence on initial conditions.

These three characteristics must all be satisfied before it can be concluded that a dynamical system is chaotic. Each of the characteristics is elaborated further in this section.

### 2.5.2.1 Sensitivity to Initial Conditions

In 1961, while running a climate model which consists of twelve differential equations with the objective of forecasting weather long-term, Edward Norton Lorenz discovered the climate's "sensitive dependence on initial conditions". This property implies that little differences in initial conditions (e.g. due to rounding errors in numerical computation) produces completely divergent outcome for such dynamical systems, thereby making long-term prediction of the system's behavior impossible[39]. It is this *loss of predictability* due to small errors in the initial conditions that is called the *butterfly effect*. Though a chaotic system is governed by a deterministic nonlinear dynamic equation, the sensitivity to initial conditions is the sole reason for this unpredictability. Considering the logistic map in (2.6) for instance, any slight change to the initial set value of the

parameter  $x_n$  will lead to a completely different value of  $x_{n+1}$ . Thus with the slightest difference to  $x_n$ , the behavior of the system cannot be predicted in the distant future [37]. This is a very good source of ensuring diversity in the population of solutions from one generation to another in the genetic algorithm. While linear maps may have sensitive dependence on initial conditions, they do not exhibit chaos since they do not satisfy the remaining conditions of a chaotic system. However, quadratic and high-order maps may exhibit chaos.

### 2.5.2.2 Topological Density

Topology is the most basic form of geometry. It deals with the mathematical study of objects (shapes and spaces). It studies the properties of objects that remain the same irrespective of continuous deformation or distortion. Topology allows any continuous change to geometry that can be continuously undone. Topological density can be explained using the concept of a *dense* set. In the context of dynamical systems, “a set  $S$  is a collection of points  $\sigma_n$  in a phase space,  $\mathbb{R}^m$ . The set  $S$ , together with its topology defined by the subsets,  $\{s_n\}$ , define a topological space [37]”.

A subset  $s_n$  of a topological space  $S$  is called **dense** (in  $S$ ) if any point  $s$  in  $S$  belongs to  $s_n$  or is a limit of  $s_n$  [43]. Formally, a subset  $s_n$  of a topological space  $S$  is dense in  $S$  if for any point  $s$  in  $S$ , any **neighborhood** of  $s$  contains at least one point in  $s_n$ . In other words,  $s_n$  is dense in  $S$  if and only if the only closed subset of  $S$  containing  $s_n$  is  $S$  itself. Thus we can say that the **closure** of  $s_n$  is  $S$ , or that the **interior** of the complement of  $s_n$  is empty. The definition implies that  $s_n$  approximates  $S$  to any degree of accuracy such that every point in  $S$  not in  $s_n$  has a point of  $s_n$  arbitrarily close to it [37]. For example every real number is either a rational number or has one arbitrarily close to it.

### 2.5.2.3 Topological Transitivity (Ergodicity)

This is the most essential condition of chaos and it is exclusively a property of chaos. According to Kinsner's definition [37]:

“Let  $g$  be a map on metric space  $X$ . Then  $g$  is said to be topologically transitive if for any pair of nonempty open sets  $G$  and  $H$  there exists a positive integer  $k$  such that  $g^k(G) \cap H \neq \emptyset$ . Thus, under a transitive map, a point can wander all over the space  $X$  where its orbit gets as close as we wish from every point in  $X$ ”.

Transitivity of points means that any future trajectory will get into the vicinity of another trajectory, without ever intersecting it. This concept is popularly called *mixing*. Thus any single point is able to get to any other point on the trajectory, arbitrarily close, without ever intersecting it. This behavior is closely related to the concept of *ergodicity*. Ergodicity is a property of any stochastic or dynamical system. It describes a system whose probability distribution is independent of its initial conditions. This implies that the statistical property is invariant with initial starting point in the trajectory. Ergodicity describes a dynamical system that has “the same behavior averaged over time as averaged over the space of all the system's states (phase space)” [40]. Thus the concept of mixing also implies ergodic orbits.

The unique feature of this property is that for any map that exhibits chaos, during one trajectory, no one point will be visited more than one. This implies that no point will be repeated in the order of generation per iteration (trajectory). This is a powerful tool that can be used to improve the performance of search-based algorithms since it can guarantee that no solution will be revisited in the trajectory of the search space and equally ensure a better chance of not getting stuck in a local optima.

## 2.6 Motivation for Incorporating Chaos into Genetic Algorithms

The standard genetic algorithm uses a random process to generate parameter values for the initial population generation, the crossover and mutation processes. This randomization leads to limitations in the performance of the GA such as *premature convergence*, probability of getting stuck in local optimum and inability of the algorithm to ascertain when to stop searching for the global optimum solution. And despite efforts aimed at militating against these problems by developing various techniques to improve the performance of the GA as discussed in section 2.3, the drawbacks of the GA are never fully addressed. One of the reasons responsible for these intrinsic drawbacks is attributable to the inherent *random walk*, which may lead to the revisiting of solutions within the search space. This may reduce the probability of obtaining the global optimum solution. This *random walk* is also responsible for the below-par performance of the evolutionary processes of the algorithm.

One reason for premature convergence may be the revisiting of points in the search space leading to frequent re-evaluation of the fitness value of the same candidate solution. The ergodic property of a *chaotic walk* is a very powerful tool that ensures that no one point is revisited for a complete walk through the search space. Thus, the probability of obtaining the global optimum solution is increased depending on the number of iterations (generations) set as the finishing criterion. Also the sensitivity to initial condition of a chaotic system may help the GA to avoid been stuck in local optimum. This property is another powerful effect that ensures that any slight change in the diversity of the population of any generation (and thus fitness value) leads to a completely different population which differs greatly in fitness values from one generation to another.

We conjecture that the evolutionary processes of a GA –initial population generation, crossover and mutation in genetics - are driven by a deterministic non-linear dynamic process rather than a random deterministic process. Thus, a chaotic logistic map will be incorporated into the initial population generation, the crossover and mutation processes of GA. Introducing chaos into the whole process of a standard GA (SGA) may help improve convergence time and accuracy. This is the motivation for introducing chaos into the GA’s evolutionary processes – otherwise called Chaotic GA (CGA). The coupled logistic chaotic map will be used in the process of the GA such as *population generation, crossover and mutation*.

## **2.7 Chaotic Genetic Algorithm**

Yun-Xiao et al [41] introduced chaos into GA processes and applied it to the cognitive radio resource allocation problem. The coupled CGA succeeded in reducing the total transmission power, bit error rate, and convergence speed in the cognitive system compared to the simple GA (SGA) and dynamic allocation algorithm. Min-Yuan and Kuo-Yu [42] also proposed K-means clustering and Chaos Genetic (KCGA) for Non-linear optimization. The KCGA was shown to enhance the diversity of the GA as well as improve on the limitations of the SGA in terms of convergence time and local optima. The KCGA has an improved accuracy and faster convergence time compared to the SGA. There are several works [43][44][45][46] that have applied chaos to other stochastic methods for different applications. Cook et al [43] applied chaos to simulated annealing to optimize the multiprocessor task allocation problem. They took “*chaotic walks*” in the solution space with the goal of obtaining a “good-enough” task to processor allocation solutions. Their approach was to use chaotic sequences to generate the chaotic variables that were used to set the number of perturbations made in each iteration of a chaotic simulated annealing. They also adjusted the parameters of a chaotic variable generator to create different chaotic

distributions that were used to search through the solution space. Their experiments show that the chaotic simulated annealing converged faster than the conventional simulated annealing when the solutions are far apart in the solution space. Shaw and Kinsner [44] applied the chaotic simulated annealing to the training of multilayer feedforward neural networks with the goal of avoiding and escaping the local minima. Their result for the said application show that the chaotic simulated annealing, using the logistic equation, was 600 percent faster than conventional simulated annealing that uses Gaussian random numbers. Mingjun and Huanwen [45] also reported that the chaotic simulated annealing improved the convergence of a typical complex function optimization compared to the conventional (Gaussian-based) simulated annealing. The chaotic counterpart was also reported to be more efficient. Meng et al. [46] incorporated a chaotic search into the evolutionary process of the particle swarm optimization algorithm and applied it to the Schwefel's function, the Rosenbrock's function and the Schaffer's function. When compared with other methods like the standard particle swarm algorithm, the standard genetic algorithm and an improved particle swarm algorithm, the chaotic particle swarm algorithm was reported to converge faster and less susceptible to get stuck in local optimum solutions. The inherent properties of chaos ensure that the CGA is able to explore the solution space in such a way as to obtain better fit solutions. The CGA process flow is described in Table1.

Table 1 Chaotic Genetic Algorithm Procedure

Step	Action
1	Generate a chaotic initial population of n solutions, where n is the population size
2	Evaluate the fitness of each of the solutions in the initial population
3	Generate new population of solutions using processes in steps 4-6
4	Selects two solutions among the current population using the roulette wheel method (based on fitness of each solution)
5	Crossover the two selected solutions using chaotic sequences and considering the crossover probability, to form the solutions for the next generation
6	Mutate one of the selected solutions at each defined chaotically generated mutation point, considering the mutation probability and place it in the new population
7	Evaluate the fitness of each of the solutions in the new population
8	Repeat steps 3-7 until the stopping criteria have been met



## **Chapter 3**

### **Genetic Algorithms, Chaos and Application**

This chapter focuses on the application of GA to specific problems. The first section of the chapter discusses problem classification based on the computational complexity of obtaining the optimum solution. The second and third section describes the application of GA to the multiprocessor task scheduling problem and spectrum allocation respectively. The Chaotic GA optimization approach is described in the last section.

#### **3.1 Problem Classification**

A problem for which a solution is sought could be a decision problem, a search problem or an optimization problem. This thesis focuses on seeking a solution within a search space. For some search problem, there are efficient algorithms that give solution within polynomial time. The problem size could be that we seek solution from among an exponential population of possibilities. An exhaustive search will guarantee the optimum solution, but for a large problem size the exhaustive search will not run in polynomial time. Thus, there is a need for efficient algorithms that can cleverly bypass this process of exhaustive search using clues from the input in order to drastically narrow down the search process. The problem of seeking solution can be classified based on the difficulty that is encountered in seeking such solution. The classes are:

- **P**
- **NP**
- **NP-hard**
- **NP-complete.**

**P:** The P stands for the class of all search problems that can be solved in polynomial time. In other words, there are known algorithm that can solve the problem in polynomial time. A polynomial time can be explained as the *time requirement* of any algorithm that grows only as a polynomial function (such as  $n$ ,  $n^2$ ,  $n^3$ , or  $n^4$ ) of the size of the input, where ‘ $n$ ’ is the size of the input. Polynomial-time problems are deemed as *easy problems* when compared to some other *very hard* ones. Examples of **P** problems include the shortest path problem, Euler path, unary knapsack, and bipartite matching.

**NP:** The NP stands for “nondeterministic polynomial time”. It denotes the class of all problems because not all problems can be solved in polynomial time. An NP problem solution can be found using a special kind of algorithm known as a *non-deterministic algorithm*. The solution can also be verified in polynomial time. We see that  $P \subseteq NP$  although  $P \neq NP$ .

**NP-hard:** These are a class of problems that are at least as hard as the ‘hardest’ problems in NP, but do not have to be in NP. A problem  $S$  is NP-hard if a polynomial-time algorithm for  $S$  would mean a polynomial-time algorithm for every problem in NP. Thus, we can reduce any problem in NP to an NP-hard problem. If any NP-hard problem can be solved in polynomial time, then  $P = NP$ .

**NP-complete:** NP-complete problems are the ‘hardest’ NP problems. This means that if any NP-complete problem can be solved in polynomial time, then all problems in NP can be solved in polynomial time i.e.  $P=NP$ . Formally, we say a problem  $S$  is NP-complete if it satisfies these two conditions:

- $S \in NP$
- NP-hard i.e. every problem in  $NP$  can be reduced to  $S$  in polynomial time.

## **3.2 Application of GA to the Multiprocessor Task Scheduling Problem**

In this section we discussed task scheduling based on the height of tasks within a task graph and pointed out the problem with this method. The scheduling method based on the number of task descendant was also discussed. We pointed out the advantage of this method and discussed a way to improve the method. Finally we have described the application of the genetic algorithm to the improved scheduling method.

### **3.2.1 Scheduling**

Scheduling is a process of decision-making, which is employed regularly in many industries. Typically, scheduling involves the decision of how to allocate resources to tasks over a given period of time with the goal of optimizing one or more objectives. Scheduling plays a significant role in many applications today such as in product manufacturing and information processing systems. Thus, the role of scheduling in a real world application cannot be over-emphasized.

The resources to be allocated depend on the environment or industry. The resources could be processing units in a computing environment, landings and takeoffs at an airport, gate assignments at an airport, teller officers' assignment to customers within a banking hall, reservation of reading rooms in a University library. Each task may have more than one attributes, each of which can be used as a criteria for allocation. Examples of the attributes include execution time and cost/implication of task waiting-time. Example of an objective can be the minimization of the total processing time of a set of tasks arranged in a task graph.

### 3.2.2 Multiprocessor Task Scheduling

Example of resources in a multiprocessing system will be the processors, while the task would be the computational work to be done. This kind of scheduling is also known as multiprocessor task scheduling. The goal of task scheduling in a multiprocessor system is to schedule tasks on available processors with the goal of minimizing the total task-processing time. The total task-processing time is also known as *makespan*. For each task, there is an execution (processing) time  $e_{ij}$  that represents the execution of task  $j$  on processor  $i$ . The distribution of the execution times may or may not be known in advance, including their mean and their variance. Furthermore, priority (weight) levels can be assigned to each task; the weight  $w_j$  of a task  $j$  specifies the importance of the task  $j$  relative to the other tasks in the systems. Thus, the priority levels specify which tasks are more important in the creation of the schedule order. The goal remains the same: to minimize the expected makespan.

The completion time of each processor may vary for different schedules. Thus, the objective to be minimized depends on the *completion times* of all the processor in a schedule. Within a schedule, if the completion time of the operation of task  $j$  on processor  $i$  is represented by  $C_{ij}$ , then the time task  $j$  leaves the system (i.e. task  $j$ 's completion time on processor on which it is processed) is represented by  $C_j$ . Two examples of possible objectives to be minimized are:

**Makespan** ( $C_{max}$ ): This is defined as  $\max(C_1, \dots, C_n)$  and is equivalent to the maximum completion time of the processors in the system (schedule). A minimum makespan is an indication of good utilization of the processor(s).

**Total weighted completion time** ( $\sum_1^k w_j C_j$ ) [47]: This is the sum of the weighted completion times of  $k$  jobs, where  $w_j$  represents the relative weight of  $C_j$  with respect to others. A minimum total weighted completion time usually signifies a good utilization of the processor(s).

A typical attribute of a task graph is the precedence based-relation between the tasks. This attribute implies that some tasks have to complete execution before the execution of other tasks. Two approaches to task scheduling are the pre-emptive or non-pre-emptive scheduling. Pre-emption refers to the interruption of the execution of a task. Pre-emption ensures that tasks having higher priority and longer execution time do not prevent tasks having lower priority and relatively short execution time from having a share of a processing unit. The scheduler divides the execution time of each task into smaller chunks and then rotates these smaller chunks on the processing unit. The non-pre-emptive relation implies that any task in execution cannot be interrupted until it completes execution. There are different special forms of precedence constraints: a *chain* constraint is when a task has at most one predecessor and at most one successor. An *intree* constraint is when a task has at most one successor while an *outree* constraint is when a task has at most one predecessor.

### 3.2.3 Computational Complexity

The impact of a formulated method of scheduling on the set objective(s) may not be clear at the onset. The question is how sensible it is to spend time and effort searching for a good schedule, within a search space rather than just choosing a schedule at random. According to Pinedo [47], in practice the choice of schedule *does* have a significant impact on the performance of the system; thus, in practice it is sensible to spend some time and effort to search for a suitable

schedule. Hou et al [7] reported that the problem of obtaining optimum task scheduling for any task graph with task dependencies is NP-hard.

If we consider ‘ $n$ ’ number of tasks and  $m$  number of processors, sorting each task (in non-decreasing order) according to a relative deadline order will require  $\Theta(n \lg n)$  time. Fisher [48] estimated the run time of their proposed algorithm in mapping all  $n$  tasks on  $m$  processors as not more than that given by (3.1), where  $m \leq n$ .

$$\sum_{i=1}^n \mathcal{O}(i + m) \quad (3.1)$$

### 3.2.4 Related Work

There have been several approaches to the task allocation problem in a multiprocessing system. Jin et al [49] carried out a comprehensive survey of nine scheduling algorithms, which are frequently used to solve the multiprocessor task scheduling problem and they compared the performance of each of the algorithms. The nine algorithms considered included tabu search, simulated annealing, genetic algorithms, chaining, min-min, A\*, Insertion Scheduling Heuristic (ISH), Highest Level First Known Execution Times (HLFET), and the Duplication Scheduling Heuristics (DSH) with task duplication. They evaluated the performance of each of the nine algorithms and benchmarked them against two well-known problems of linear algebra: LU decomposition and the Gauss-Jordan elimination. It was observed that with task duplication the DSH performed best while the ISH performed best in the absence of task duplication. The GA and tabu search were reported to have obtained the best solution out of all the iterative search algorithms considered. In order to avoid unnecessary computational overhead, task duplication was not considered in this thesis.

There are other works in the literature which reports on the performance of GA in this area. The GA is used to search for the best schedule out of all the possible schedules. However, the key to increasing the ability of any heuristic to finding the best schedule is the choice of method of scheduling. Majority of these works [7][50][51][52][53] have assumed a non-pre-emptive approach to schedule precedence constrained task graph. However, Wu et al [54] assumed task duplication. Until recently, the most utilized priority-based task scheduling is the height-based method. This method prioritizes task allocation based on the position (in height) of tasks in a task graph. The task graph is generated using a Directed Acyclic Graph (DAG). A DAG for tasks is the graph that represents the precedence constraints among the tasks along with their execution time. The higher the height of a task is in the graph, the higher the priority assigned to it. However as the size of the DAG increase, the height-based method becomes less efficient principally because of variation in task descendants. “Task descendants” implies the number of tasks that depends on the completion of any task before starting their own execution. A notable drawback of the height-based method is that any task at a higher height but with no descendant will be scheduled ahead of tasks at a relatively lower height that have descendant(s). This may increase the makespan of the processing system. Abdeyazdan and Rahmani[8] proposed a new scheduling algorithm which prioritizes task allocation based on the *number of children* and the *earliest start time*(EST) of each task. We have replaced the notation ‘*number of children*’ with ‘*number of task descendants*’ (NTD). This new method solves the above mentioned problem encountered with the height-based counterpart by ensuring that tasks having a greater number of descendants are given higher priority regardless of their height on the graph. This can further decrease the makespan of the processing system.

In this thesis, we have considered the new algorithm proposed by [8]. However, we have also observed that within a DAG, it is possible that some tasks will have multiple earliest start time(s) as a result of multiple paths from which the ESTs can be calculated. Our contribution to this research ensures that for tasks that have multiple ESTs, the minimum EST will always be chosen as against the observed maximum EST used by [8]. The assumption is that the completion of the execution of one of the parent tasks satisfies the precedence relations in the graph; a method similar to an ‘OR’ relation. Our method is akin to choosing the shortest path in a routing problem, thus reducing the makespan of the processing system.

#### 3.2.4.1 Height-based Task Scheduling

The goal of task scheduling in a multiprocessing system with  $m$  processors is to assign  $n$  tasks to the processors such that the precedence relations are maintained and all of the tasks are completed in the shortest possible time. For a schedule with  $m$  processors, the completion time of each processor is termed the *Finishing Time* (FT) of such processor. Most times the FT of each of the  $m$  processors varies one from another, and depends also on schedule. The maximum FT among the  $m$  processors of a schedule is termed the *Total Finishing Time* TFT of that schedule. This is also referred to as the *schedule length*. The TFT of a schedule is given by (3.2).

$$TFT = \text{Max} \{FT \text{ of Processor}_i\}; \text{ for } 1 \leq i \leq m \quad (3.2)$$

Hou et al [7] developed the height-based method to convey the precedence relations between the tasks in a DAG. A typical DAG is represented by Fig.7. A height function was developed to ensure that tasks that are higher up the task graph are given more priority compared to tasks on lower levels. In a task graph where there is a sequence of directed edges from task, say  $t_i$  to  $t_j$ , then  $t_i$  is higher up the graph while  $t_j$  is lower down the graph.



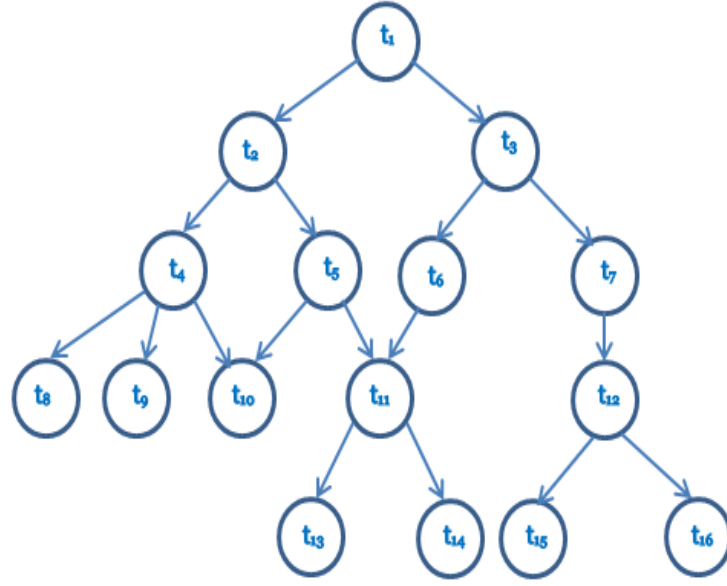


Fig. 7 A task graph

This precedence relation implies that the execution of task  $t_i$  has to be completed before the execution of tasks  $t_j$  and other tasks that have directed edges from task  $t_i$ . If  $PRED(t_i)$  stands for a set of preceded tasks of  $t_i$ , then the height of any of the preceding tasks can be obtained using (3.3). This equation is a recursive representation of the precedence relations between the tasks in the task graph.

$$height(t_i) = \begin{cases} 0 & \text{if } PRED(t_i) = \emptyset, \\ 1 + \max\{height(t_j)\} & \text{otherwise.} \\ t_j \in PRED(t_i) \end{cases} \quad (3.3)$$

The task height increases from 0 to a finite length, where the succeeding tasks to the task(s) at height 0 will have heights of value(s) greater than 0. Therefore the lower the value of the height of any task, the higher up is the task on the DAG. In other words, if task  $t_i$  precedes tasks  $t_j$  in a task graph, then  $t_i$  will be executed before  $t_j$  and  $height(t_i) < height(t_j)$ . However if there is no precedence relation between any two tasks, the order of execution can be random. For example, in

Figure 7, the height-based function gives a higher priority to scheduling task  $t_0$  before any of task  $t_1$  or  $t_2$ . An algorithm to produce schedules based on the height of task is shown below:

1. Arrange the tasks in ascending order according to their height.
2. Generate a random number  $r$  between 1 and  $m$  (where  $m$  is the number of processors).
3. Put tasks with the same height in a single group and perform steps a and b for all the groups in order of height until every group is empty.
  - a. Randomly select a task from the group.
  - b. Allocate the selected task to the  $r^{\text{th}}$  processor and then delete it from the group.
4. Repeat steps 'a' and 'b' until the queue is empty.

Table 2 shows the arrangement of the tasks in ascending order based on their height. The problem inherent with this approach is discussed in the next section. An example of a possible schedule that can be generated using the height-based scheduling algorithm with respect to Table 2 is presented in Fig. 8.

Table 2 Height and Execution Time of Tasks

Task	Height	Execution time
t1	0	4
t2	1	2
t3	1	3
t4	2	3
t5	2	11
t6	2	3
t7	2	5
t8	3	3
t9	3	3
t10	3	10
t11	3	7
t12	3	5
t13	4	8
t14	4	13
t15	4	15
t16	5	12

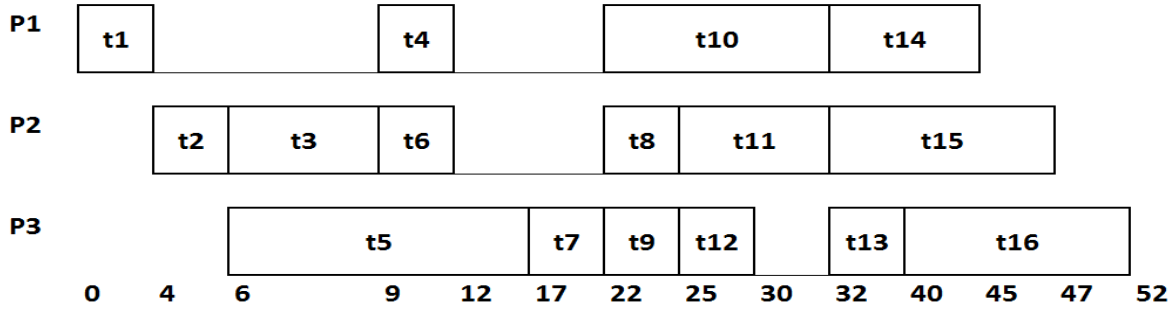


Fig. 8 Height-based Task Schedule

### 3.2.4.2 Problem with Height-Based Scheduling

A major drawback of this approach can be explained using the DAG of Fig. 7. Using the height function we can assign corresponding height to each of the tasks  $t_1$  to  $t_{16}$ . For instance, if task  $t_1$  is set to be at height 0 (highest), then each of tasks  $t_2$  and  $t_3$  will have height 1. Also each of tasks  $t_4$ ,  $t_5$ ,  $t_6$ ,  $t_7$  will be assigned height 2. This way every task in the graph have their height assigned. For this example, a random *execution time* (ET) is assigned to each task. Table 2 shows the heights and ET for each of the tasks in the graph. For this graph we assume that the ET values range from 0 to 15. The height based approach schedules tasks at higher height before tasks at relatively lower height. For tasks with equal height values, any of such tasks will be randomly picked for scheduling. Based on this approach, we observe the following:

*“tasks such as  $t_8$  may be scheduled before tasks such as  $t_{11}$ ”*

We note that the task  $t_8$  has no descendant, which means that there is no task that needs task  $t_8$  to complete before it can start. The algorithm states that any of tasks  $t_8$  and  $t_{11}$  can be randomly selected for scheduling since they both have the same height. However, it is observed that tasks  $t_{11}$  has two descendants: tasks  $t_{13}$  and  $t_{14}$ , each of which cannot be scheduled unless task

$t_{11}$  completes execution. A better method will be to schedule task  $t_{11}$  that has descendants first before task  $t_8$  that has no descendant. Therefore, scheduling based on height may increase the FT of a processor and consequently the TFT of a schedule. For a task graph with a high percentage of task descendants, the height-based scheduling will not be suitable for optimum performance of the multiprocessing system. This is the motivation for a new scheduling algorithm to be designed to achieve a better task scheduling in the system.

### 3.2.4.3 Task Allocation based on Number of Task Descendants

This new method proposed by Abdeyazdan and Rahmаниm [8] was developed to further reduce the TFT (makespan) of the processing system. It prioritizes task scheduling based on the number of descendants (NTD) and the earliest start time (EST) of each task. Tasks with higher NTD(s) are given higher priority irrespective of their height on the task graph. Assume a task  $t_i$  with a  $j$  number of outgoing edge(s), i.e. descendant(s). The descendants represent tasks whose execution depends directly on the completion of the task  $t_i$ . The Number of Task Descendants (NTD) for such task  $t_i$  is given by (3.4).

$$NTD(t_i) = \begin{cases} 0 & ; \text{if } NTD(t_i) = \emptyset, \\ \sum_{j=1}^j \{1 + NTD(t_j)\} & \text{otherwise.} \\ t_j \text{ is a descendant of } t_i; \end{cases} \quad (3.4)$$

The NTD function above represents the total number of tasks that depend on a task  $t_i$  whether directly or indirectly; this was termed *number of children* by [8]. In this work, we have replaced the term '*number of children*' with '*number of task descendants*' (NTD). A task with more NTD will be scheduled earlier than any other tasks with lower NTD. The concept of Earlier Start Time was introduced to ensure that tasks are scheduled with respect to the earliest time at which they

are available. The EST of a task is a function of the sum of the execution times of all the tasks that precede that task. However there could be more than one path on the task graph along which a task could be executed. This implies that there will be multiple EST for such task. Our algorithm selects the minimum EST (min-EST) whereas [8] selects the maximum EST (max-EST). An algorithm to produce the schedule based on the number of task descendants and minimum EST is shown below:

1. Arrange the tasks in descending order based on the number of task descendants of each task.
2. Put tasks with the same NTD in a single group and perform steps a and b for all the groups in order of higher NPD until every group is empty.
  - a. Randomly select a task from the group and then delete it from the group.
  - b. Allocate the selected task to one of the processors based on the EST method such that the starting time of the task on that processor is less than other processors.
    - i. For tasks with multiple EST, choose the minimum EST in performing *b*.
3. Repeat steps 'a' and 'b' until all the tasks have been selected.

The EST method is based on (3.5) and (3.6). Equation 3.5 shows how the algorithm obtains the actual EST for each task. The actual  $EST(t_i)$  on processor  $P_j$  is the time at which a task  $t_i$  starts execution on processor  $P_j$ . It is based on estimations made from other parameters. These other parameters include  $EST(t_i)$  and  $A_t(P_j)$ .

$$Actual\ EST(t_i)\ on\ P_j = \begin{cases} 0; & \text{if } EST(t_i) = \emptyset, \\ EST(t_i); & \text{if } A_t(P_j) \leq EST(t_i) \\ A_t(P_j); & \text{if } A_t(P_j) > EST(t_i) \end{cases} \quad (3.5)$$

where  $A_t(P_j)$  represents the current available time on processor  $P_j$  at which an allocated task may start execution,  $EST(t_i)$  is the earliest start time of a task  $t_i$  based on the task graph,  $n$  is the number of tasks and  $m$  is the number of processors.

The equation that can be used to compute the  $A_t(P_j)$  is given by (3.6).

$$A_t(P_j)_{i=1 \text{ to } n; j=1 \text{ to } m} = \begin{cases} 0; & \text{if } EST(t_i) = \emptyset, \\ EST(t_i) + ET(t_i); & \text{if } A_t(P_j) \leq EST(t_i) \\ A_t(P_j) + ET(t_i); & \text{if } A_t(P_j) > EST(t_i) \end{cases} \quad (3.6)$$

where  $ET(t_i)$  represents the given execution time of task  $t_i$ . We see that  $A_t(P_j)$  is a function of the EST and the execution time of each task.

The algorithm assigns a task only once to a processor since there is no repetition of same task on the task graph. From the task graph in figure 7, we can arrange the tasks according to the NTD of each task. The total execution time of all tasks that precedes a particular task along any path in the task graph is used to compute the EST of the task. Table 3 shows the arrangement of the tasks according to the descending order of their NTD. It also shows the EST of each task. From this table, we see that each of tasks  $t_{10}$ ,  $t_{11}$ ,  $t_{13}$ , and  $t_{14}$  have two ESTs because there are two possible paths from which the EST can be obtained. For instance task  $t_{10}$  has an EST of '9' along the path:

$$t_1 \rightarrow t_2 \rightarrow t_4$$

However, task  $t_{10}$  has an EST of '17' along the path:

$$t_1 \rightarrow t_2 \rightarrow t_5$$

The min-EST algorithm always chooses the minimum EST. We have assumed an 'OR' precedence relation in the completion of any of the tasks  $t_4$  or  $t_5$ . Since EST 9 is smaller, the

completion of task  $t_4$  maintains the precedence relation. This further reduces the makespan of the schedule. Figure 9 shows a schedule that can be generated using the Max-EST method with a makespan of 44; this is better when compared with the height-based schedule of Figure 8 which has a makespan of 52. Our method - Min-EST - can generate a schedule with a reduced makespan of 41 as shown in Fig. 10.

Table 3 Task Order based on NTD

Task	NTD	EST
t1	15	0
t2	10	4
t3	6	4
t4	5	6
t5	4	6
t6	3	7
t7	2	7
t8	2	9
t9	1	9
t10	1	9,17
t11	0	10, 17
t12	0	12
t13	0	24,17
t14	0	24,17
t15	0	17
t16	0	17

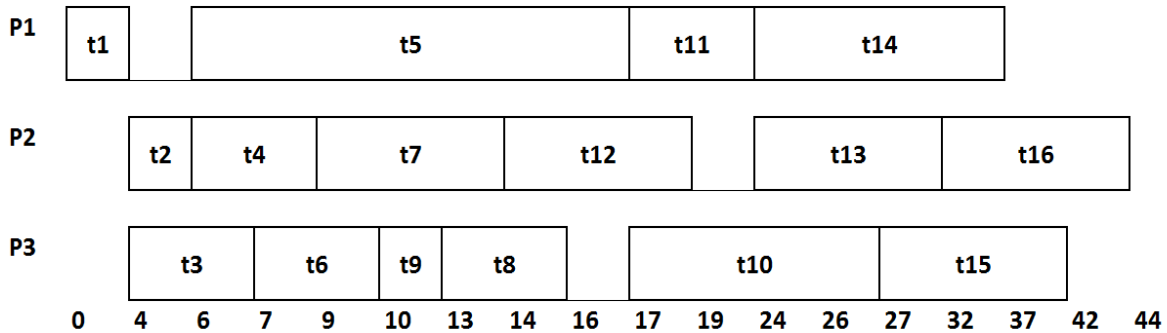


Fig. 9 Max-EST NTD-based Task Schedule

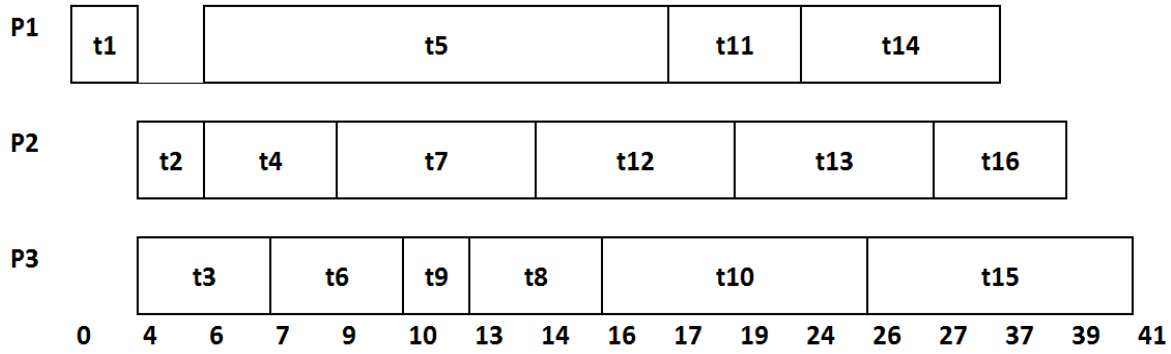


Fig. 10 Min-EST NTD-based Task Schedule

### 3.2.5 Methodology: GA Approach to Multiprocessor Task Scheduling

The goal of this combinatorial optimization problem is to find the optimum task schedule in polynomial time. The genetic algorithm has been chosen to obtain a near-optimum schedule within the search space of this NP-complete problem. GA is a subset of evolutionary algorithms that model biological processes in optimizing complex functions. In GA terms, a ‘schedule’ is a possible ‘candidate solution’ also known as a ‘chromosome’. Within a search space of possible candidate solutions, the GA aims to obtain the best ‘candidate solution’ through the use of its evolutionary processes. The GA procedure is shown in Table 4. The GA allows a population comprising of many candidate solutions to evolve under specified selection rules to a state that maximizes the fitness (i.e., minimizes the objective function). Because time is critical in a real-time multiprocessing system, a suitable schedule must be found in polynomial time. An important advantage of the GA over most stochastic techniques is the ability to generate multiple solutions per time (iteration) – a property called *parallelism*. This offers better exploration of the search space and makes the GA less local in nature [12]. The multiple population and variation operators of the GA (*crossover* and *mutation*) make the GA less likely to get stuck in local optima. All these attributes make the GA suitable for this search problem.



Table 4 Genetic Algorithm Procedure

Step	Action
1	Generate a random initial population of n schedules, where n is the population size.
2	Evaluate the fitness of each of the schedule in the initial population
3	Generate new populations using processes in steps 4-6
4	Selects two schedules among the current population using the roulette wheel method based on fitness of each schedule.
5	Crossover the two selected schedules considering the crossover probability, to form the schedules for the next generation
6	Mutate the one of the selected schedules at each defined mutation point, considering the mutation probability and place it in the new population.
7	Evaluate the fitness of each of the schedules in the new population
8	Repeat steps 3-7 until the stopping criteria have been met.

An initial population (i.e., set of candidate solutions) is generated after which the GA is run over many iterations to find a near-optimum solution in regard to the given objective function. This initial population consists of possible candidate solutions within the search space. The size of the population will be chosen depending on the size of the problem. An example of the problem size could be the number of tasks to be scheduled and the number of processors. Each solution in a population is evaluated using a fitness function, which is derived from the objective function. Since the objective is to find the smallest schedule length possible, the objective function for  $k$  generated schedules is given by (3.7):

$$\min\{TFT(s)_j\} \text{ where } j = 1, \dots, k \quad (3.7)$$

A near-optimal schedule will represent a good schedule; this good schedule would be the smallest TFT with a fitness value larger than the other schedules. The GA's fitness function is similar to

the objective function except that GA seeks to maximize the fitness function. Therefore, the fitness function can be formulated as in (3.8). Thus the higher the fitness value of a schedule, the better.

$$1/\min \{TFT(s)_j\} \text{ where } j = 1, \dots, k \quad (3.8)$$

After the fitness evaluation of each of the candidate schedules has been computed and the stopping criteria of the algorithm have not been met, a new solution population will be generated using *selection*, *crossover* and *mutation* techniques. These new sets of solution are generated with the goal of obtaining a good solution.

### 3.2.5.1 Selection

The selection process has to do with the survival of the fittest candidate within the pool. It is a probabilistic technique where solutions with higher fitness values have more chance of moving to the next generation compared to those with lower values. Candidates are selected for the next generation based on their fitness values while the ones that are not selected are discarded. There are various selection methods [14][15][16]; we have proposed the roulette-wheel selection method because it offers diversity – an important focus of this thesis. The method gives every schedule a chance to survive to the next generation, although the probability of a solution being selected is proportional to the solution's fitness value. The goal is to increase the diversity of the solutions with the hope that the diversity increases the chances of the GA finding a near-optimum solution in polynomial time. For instance, in this application, the lower the TFT of a schedule, the larger the slot the schedule occupies in the roulette wheel; this leads to an increased chance of being selected at a spin of the wheel. In this thesis, the roulette wheel method is used to select one task and two schedules which will be used in performing the crossover and mutation operations.

### 3.2.5.2 Crossover

The crossover operator, also known as *recombination* operator, derives its meaning from the exchange of *genes* between *chromosomes* in biological reproduction. It is considered to be the most important process of evolutionary algorithms. Crossover occurs in reproduction at a certain rate. In optimization, the selected crossover rate will depend on the user and/or application. However, in general, the rate should be high enough to allow a diverse set of solutions in the population. In literature this rate varies between 0.80 and 0.95. However, for some problems, this range can lead to space *exploitation* which may lead to premature convergence. Therefore some authors have proposed a rate of 0.6 to avoid this problem. At this rate, the GA exchanges one or more components of the two schedules selected every time the selection process is called. In determining which components are exchanged, for one of the schedules, the algorithm chooses the tasks that have equal or lower NTD to the selected task. For every processor in the first schedule, the chosen tasks are exchanged with tasks that have equal or lower NTD on the corresponding processor in the second schedule. By exchanging only tasks that have equal or lower NTD, the precedence order of tasks in the schedule is maintained. A typical example is shown in Fig. 11.

In Figure11, we assume two schedules  $C_1$  and  $C_2$  and the task  $t_{14}$  were selected during the selection phase. Since the task  $t_{14}$  has an NTD of 0, then the tasks selected for crossover will have NTD equal to that of task  $t_{14}$  or less. These tasks include tasks  $t_8, t_9, t_{10}, t_{13}, t_{14}, t_{15}, t_{16}$ . In performing the crossing over operation, the tasks  $t_9$  and  $t_{14}$  (in this order) on processor  $P_1$  of schedule  $C_1$  are exchanged with the task  $t_{14}$  on corresponding processor  $P_1$  of schedule  $C_2$ . The tasks  $t_8, t_{10}$ , and  $t_{15}$  (in this order) on processor  $P_2$  of schedule  $C_1$  are exchanged with the task  $t_{13}$  and  $t_{16}$  on corresponding processor  $P_2$  of schedule  $C_2$ . Finally, the tasks  $t_{13}$  and  $t_{16}$ , (in this order) on processor  $P_3$  of schedule  $C_1$  are exchanged with the task  $t_9, t_8, t_{10}$  and  $t_{15}$  on corresponding processor

$P_3$  of schedule  $C_2$ . The operation is done in the hope that a schedule with lower TFT is obtained. The new schedule obtained by performing crossover on schedule  $C_1$  is  $C_{1\text{-new}}$  and it has a TFT of 42, while the new schedule obtained by performing crossover on schedule  $C_2$  is  $C_{2\text{-new}}$  and it has a TFT of 44. Thus we see that with crossover the algorithm can come to a better or worse solution depending on the tasks that are being exchanged between the schedules.

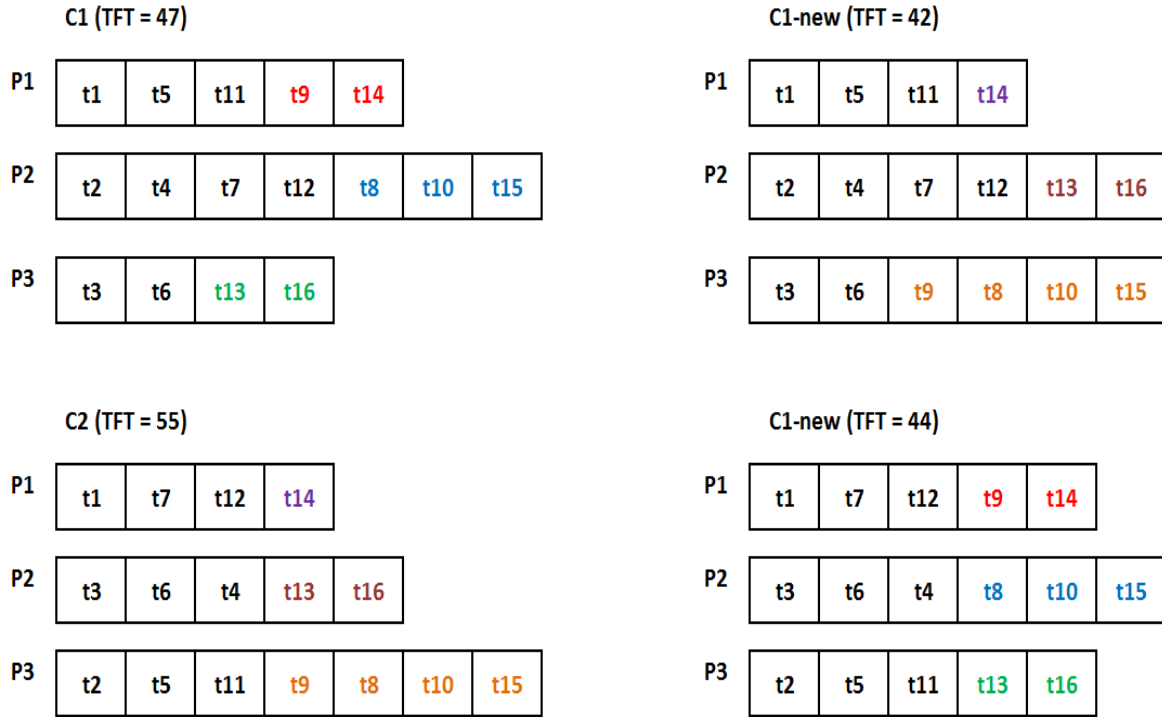


Fig. 11 Crossover Technique on Schedules  $C_1$  and  $C_2$

### 3.2.5.3 Mutation

Mutation is a genetic variation operator which is used to force diversity within the population. It alters one or more gene (task) values in a schedule from its initial state. In mutation, the solution may change entirely from the previous solution. Hence GA can come to a better or worse solution through mutation. Thus, rate of mutation should be set low, say within the range 0.01 – 0.03. It should not be set too high otherwise the search will turn into a primitive random

search. In effect mutation ensures that the algorithm does not get stuck in local optima. For this application, mutation is done by using the selected task and the two schedules obtained in the selection process. A typical example of the mutation operation is shown by Fig. 12.

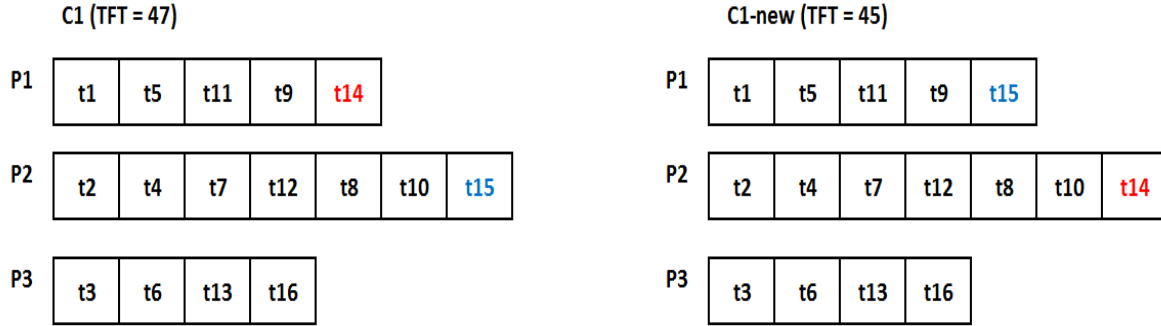


Fig. 12 Mutation Technique on Schedules  $C_1$  and  $C_2$

Mutation is performed on only one schedule at a time. In one of the schedule, the selected task is exchanged with another task with the same NTD on another processor within that same schedule. The same procedure is performed in the second schedule. This procedure generates two new schedules that will most likely have differing TFT from the initial ones. By exchanging the task  $t_{14}$  on processor  $P_1$  with the task  $t_{15}$  on processor  $P_2$  within schedule  $C_1$ , a new schedule  $C_{1\text{-new}}$  with a TFT of 45 is generated. Thus, just like the crossover operation, the algorithm may come to a better or worse schedule in terms of TFT when compared to the TFT of the initial schedule before mutation depending on the tasks that are being exchanged between processors.

After each cycle of selection, crossover and mutation, the newly generated set of candidate solutions (schedules) is termed a *new generation*. Each schedule in a generation is evaluated using the fitness function until the stopping criteria has been met. These stopping criteria could be the number of generations, evolution time, fitness threshold, fitness convergence or population convergence. In this thesis, the ‘number of generations’ was set as the stopping criteria. This is

because it is not clear how long the algorithm will take to obtain the best solution. However, it is expected that after certain iteration of the algorithm, the algorithm should have come to an acceptably good-enough solution.

### **3.3 Application of GA to the Radio Spectrum Allocation Problem**

In this section we have discussed the fundamental concept in cognitive radio and the motivation for cognitive radio in spectrum allocation. We have also described the problem of spectrum allocation in terms of computational complexity. Finally we have described the application of the genetic algorithm to the process of spectrum allocation.

#### **3.3.1 Fundamental Concept in Cognitive Radio**

In the past two decades, the use of wireless applications has increased rapidly, eventually leading to an increased demand of bandwidth. This higher demand of bandwidth has resulted in two main problems: spectrum scarcity and underutilization. The Cognitive Radio (CR) concept was introduced to solve this problem. The development of the CR was a result of earlier work done by Joseph Mitola in the early 90s. He introduced a radio called the Software Defined Radio (SDR). SDR is defined as a radio terminal communication system where components that have been usually implemented in hardware (e.g. modulators/demodulators, amplifiers, filters, etc.) are rather implemented using software on embedded systems or computer, i.e. some physical layer functions are software implemented. This design allows the SDR to support different communication standards like GSM, OFDMA, and CDMA. This design also allows it to perform reception and transmission of different radio protocols based on the software. SDRs along with software defined antennas are the enabling technology of the cognitive radio. J Mitola and G.

Maguire described the CR as “a radio that understands the context in which it finds itself and as a result can tailor the communication process in line with that understanding”[55]. A more analytically-oriented definition of CR is offered by Professor Simon Haykin [56]:

*Cognitive radio is an intelligent wireless communication system that is aware of its surrounding environment (i.e., outside world), and uses the methodology of understanding-by-building to learn from the environment and adapt its internal states to statistical variations in the incoming RF stimuli by making corresponding changes in certain operation parameters (e.g., transmit-power, carrier-frequency, and modulation strategy) in real-time, with two primary objectives in mind:*

- *Highly reliable communications whenever and wherever needed;*
- *Efficient utilization of the radio spectrum.*

A CR is aware of its internal state and environment such as location and the RF frequency spectrum utilization at the location. By mapping information about their radio operating behavior against predefined objectives, an SDR is able to make decisions. The utilization of the underlying technologies of a CR is critical in allowing end-users to make optimal use of available spectrum. “The CR has two major subsystems: a cognitive unit which makes decisions based on various inputs and a flexible SDR unit whose operating software provides a range of possible operating modes” [56]. According to Fette [57], key applications required to be integrated with an SDR to make it a CR are:

- Spectrum management and optimization.
- Wireless networks interface to provide network resources management and optimization.
- Human interface to provide electromagnetic resources to aid human activities.

### 3.3.1.1 Cognitive Capabilities and Behavior

Cognitive radio enables secondary users to ‘borrow’ free spectrum not being used by the primary users without degrading the quality of service of the primary user’s communication. The CR therefore must be able to sense available spectrum, establish and maintain quality of service (QoS) requirements for user’s application, meet service level agreement (SLA) and understand its own operational capabilities such as radio parameters [58]. The behavior of a CR is adaptive with respect to the varying conditions of wireless network communications. Link adaptation is an issue of special interest in wireless communications. The CR can adapt effectively to numerous network parameters; parameters such as operating frequency, data bit rate, signal power, bit error rate, modulation technique, antenna beam pattern, coding technique, processor utilization. The CR has the ability to recognize the wireless radio environment, learn from the environment and predict future event occurrences [55]. The cognition capabilities are a consequence of the adaptive characteristics of the CR. It is this capability that allows the CR to be aware of its internal state and environment as well as user’s requirements and network regulatory policies. These cognition capabilities include *learning*, *sensing*, *awareness* and *reasoning* [57].

### 3.3.1.2 The Cognitive Cycle

A cognitive radio refers to both an engineering model for designing wireless systems and the implementations of that model. The cognitive radio community has adopted various terms related to human psychology to explain the cognitive radio concept because the term ‘cognition’ is usually associated with the human thought process. Cognitive radio can be described as “an approach to wireless communication engineering where the radio, radio network or wireless system is endowed with *reason*, *awareness*, and *agency* to *intelligently* adapt operational aspects



of the radio, radio network or wireless system”[56]. As a result of these capabilities, we can say that the CR works in cycle i.e. *observe, decide and act* [55]. Observation has to do with the awareness (learning and sensing) of the radio environment and these observations are fed as input to the CR where a decision is made on the basis of a mechanism and then the CR takes appropriate action. The computational complexity of the cycle will depend largely upon the kind of observations and corresponding decisions taken. The decision making process may be simple or complex; complex procedure may need knowledge of the past or future analysis and probabilities [58]. A cognitive cycle can be described by Fig.13.

According to Doyle[55] , while it may appear as though the cognitive cycle begins at the ‘observation’, the cycle actually begins from the ‘act’ end. The explanation for this is that it is imperative to first ascertain what actions are possible and their significance; then it will be clear what observations need to be made and the corresponding decision to be taken.

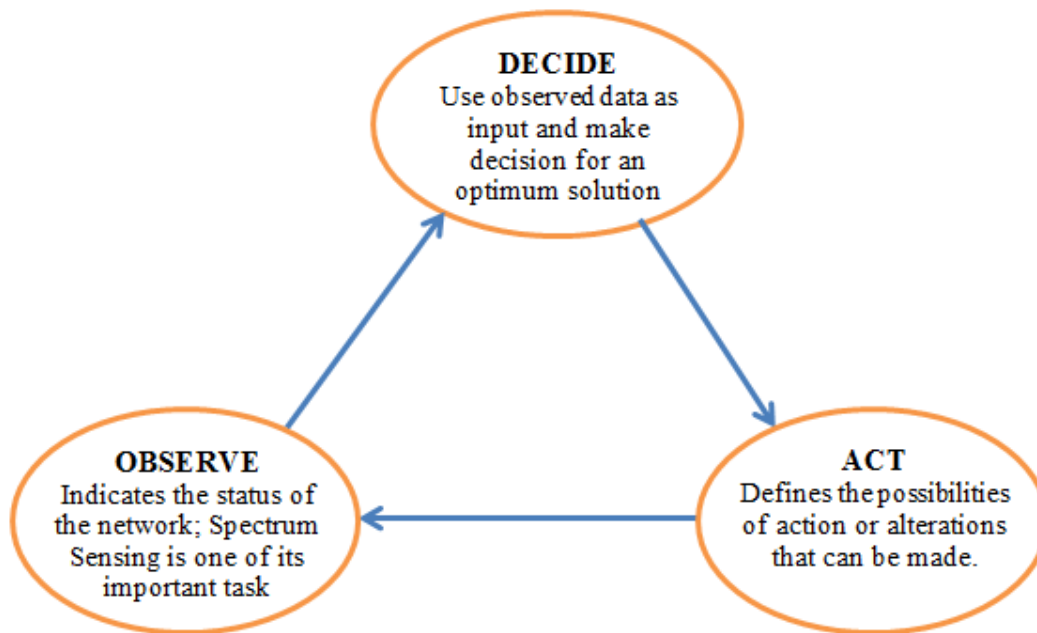


Fig. 13 The Cognition Radio Cycle [58]

Thus the three cognitive cycle is in the following order:

- Act
- Observe
- Decide

**Action :** Taking an action has to do with all the possible actions that can be taken to ensure optimal performance of the CR for specific application. The actions to be taken depend directly on the decision that has been taken by the CR; this is also indicated by Fig.13.

**Observation :** Making an observation has to do with acquiring, classifying and organizing information for the purpose of feeding the CR engine in order to make necessary decisions. This phase involves understanding the environment in which the CR operates, understanding the user requirements and regulatory policies and the awareness of its own capabilities. The CR can acquire information from various sources which can be classified as either internal or external. The ability to learn from the environment is key to optimum performance of the CR. By understanding the environment, the CR is aware of the availability of spectrum through a mechanism that does *spectrum sensing*. This phase is considered synonymous with spectrum sensing[62]. A CR requires current information regarding its awareness of its environment, its internal state, node capabilities, and current needs of its user. Spectrum sensing refers to the action of a wireless device measuring characteristics of received signals such as rf energy levels as part of the process of detecting if a particular section of the spectrum is free or occupied. Current research related with spectrum sensing include: how to *accurately* perform sensing, the *range* at which to perform sensing, and the *in time* sensing of primary users.

**Decision :** This process is regarded as the the *heart* of the CR. In order to take a decision, the CR must take inputs from the observation phase and use such inputs in taking the optimum decision possible. The important decision is about the optimum distribution and usage of the radio resources. This is where optimization comes in.

### 3.3.2 Cognitive Radio Spectrum Allocation

The primary goal of a cognitive radio is to facilitate secondary users borrowing free spectrum that are not being used by the primary users without degrading the quality of service of the primary user's communication. Cognitive radio (CR) was developed to meet the increasing demands of QoS in wireless communications [59]. The QoS of a network application can be defined as "the set of quantitative and qualitative characteristics of the communication system required to achieve desired functionality of that application" [60]. By definition, a CR is a radio that understands the context in which it finds itself and as a result can tailor the communication process in line with that understanding [55]. The main objective of the CR is to address the underutilization of the electromagnetic (EM) spectrum to meets today's increased needs in wireless communications. A CR can also recognize the radio environment, can predict the future events, and can learn from previous behaviors. The four main cognition capabilities of the CR include- *learning, sensing, awareness and reasoning*.

Optimization has been defined as "*the process involved in selecting the 'best' choice from the list of available choices in order to reach some kind of goal or at least get as near as possible to the goal*"[55]. The goal of optimization is to either *maximize* or *minimize a set objective(s)* depending on particular application. Optimization seek to address the efficiency of how the frequency spectrum is being used. Some CR techniques include spectrum sharing and spectrum

pooling, dynamic frequency selection, adaptive bandwidth control, transmit power control, etc. The ‘available choices’ refer to the set of all possible solutions to satisfy certain objectives; these set of ‘available choices’ is what is called the *search space* where each choice is a potential solution. Cognitive radios will attempt to adapt such as to optimize their performance or to optimize their usefulness for users. The objective is mathematically formulated and this formulation will be used to test the fitness of each of the available choices so as to obtain the best fit solution. There is a whole bunch of possible spectrum parameters that can be allocated in a wireless networks depending on the activities of the primary users. Spectrums that are vacated or not being used by the primary users are opportunistically allocated to secondary users till they are needed again by the primary users. Thus this process of spectrum allocation is a combinatorial problem that requires optimization. A framework must be designed to implement the spectrum allocation process in which the decisions to assign spectrum are made according to the radio user defined QoS requirements.

In literature, given the huge search space involved in spectrum allocation, it has been proven that the problem of finding the optimal spectrum allocation to CR users is NP-complete [4][5][6]. Thus any chosen framework must ensure that spectrum is allocated in polynomial time given the sensitivity of the application in regard to time. The nature of NP-complete or NP-hard problems make heuristics the only viable option for problems that need to be routinely solved in real-world applications. A heuristic is better defined as a method designed for solving a problem more quickly when classic methods are too slow or fail to find the exact solution. The objective is to produce quickly enough a solution that is ‘good’ enough for solving a problem at hand. The emphasis here is on speed at the expense of optimality, precision and completeness. Some

examples of heuristics approaches include ant colony optimization (ACO), tabu search, hill climbing, genetic algorithm, greedy algorithms, simulated annealing, particle swarm optimization (PSO) etc. Out of these only the genetic algorithm, the ant colony optimization and the particle swarm optimization maintain multiple solutions per iteration (generation) thus making them better at obtaining solutions faster. The ACO and PSO are evolutionary computing algorithms classified as Swarm Intelligence. Compared to the GA, the ACO and PSO are more likely to get stuck in local minima as a result of the random sequencing which dominates their operation. The GA, although makes use of random sequencing, uses its *mutation* operator to reduce the chance of being stuck in local optima. This is the motivation for using GA to search for the best solution.

### 3.3.3 Computational Complexity of the Radio Spectrum Allocation Problem

It has been proven that the problem of finding the optimal spectrum allocation to CR users is NP-complete [4][5][6]. Depending on how many parameters involved in the search process, there will be several possible frequency combinations. The complexity of such search could be a permutation (where order matters, Equation 3.9) or combination (where order does not matter, Equation 3.10) of selecting  $k$  element(s) from a set of all possible  $n$  elements.

$$P(n, k) = {}^nP_k = \frac{n!}{(n - k)!} \quad (3.9)$$

$$C(n, k) = {}^nC_k = \frac{P(n, k)}{k!} = \frac{n!}{k! (n - k)!} = \binom{n}{k} \quad (3.10)$$

It can be easily seen that for a large size problem, an exhaustive search of the  $k$  element(s) will not run in polynomial time. Thus there is a need for efficient algorithms that can cleverly bypass this process of exhaustive search using clues from the input in order to drastically narrow down the

search process – a motivation for deploying heuristic in searching for a good solution that if not exactly the expected  $k$  element(s), will be at least very close to it.

### 3.3.4 Related Work

Genetic Algorithm has been applied to spectrum optimization in cognitive radio networks. Siddique and Azam [58] used GA as the mechanism to optimize spectrum allocation whereby a secondary user specifies the QoS and the GA is then used for the spectrum allocation. Kaur et al [61] also proposed an Adaptive Genetic Algorithm (AGA) to optimize QoS parameters in a cognitive radio. Unlike the SGA that uses a constant crossover and mutation rates throughout the evolution process (iterations), the AGA allows different crossover and mutation rates so that the algorithms can transverse different directions in the search space. This ensures improved performance as well as represents a response to the cognitive radio's need to adapt to a changing environment. GA has also been used by Hauris [62] to implement an adaptive process for a cognitive radio on an autonomous vehicle. GA was specifically used to optimize radio frequency (RF) parameters for wireless communications. An objective function which corresponds to the GA's fitness measure was formulated as a benchmark to evaluate the performance of the GA in relation to the overall RF performance. The chromosome structure consisted of genes each of which is a binary string representation of the individual parameter of the RF environment. The GA was used to determine the set of RF parameters for optimal radio communications in the varying RF environment. In a related work, Rondeu et al [63] also used GA to find a set of RF parameters that optimize the CR for user's current needs. The parameters considered were *power*, *frequency*, *pulse shape*, *symbol rate*, and *modulation*. Experimental results showed a successful GA performance. Withall et al [64] also used the GA to optimize the parameters for a sequence of

packets sent over the internet. The Fitness Measure was based on the delay time returned by the traceroute program. The GA was reported to have adapted to different conditions on the network as well as different fitness requirements but was inconsistent in its ability to outperform the control settings performance - the benchmark. Future work was suggested to improve the GA's performance. Newman et al [65] introduced a population adaptation scheme for GA-based CR engine. The objective focused on further reducing the computational time requirement it takes for the GA to obtain the best solution within a large search space while optimizing RF parameters. In this scheme, the GA takes advantage of information from previous cognition cycles so as to reduce the time required to reach a near-optimal decision. They demonstrated that the amount of information from the previous cognition cycle can be determined from the environmental variation factor (EVF), which represents the amount of change in the environment parameters since the previous cognition cycle. These EVFs can be classified as either an *internal information* about the radio operating state or *external information representing* the wireless environment. The EVF considered were signal power, noise power, delay spread, battery life, power consumption, and spectrum information. These EVF and the control parameters made available by underlying SDR system are used as inputs into the cognitive engine. Their simulation showed that the proposed method can reduce the convergence time of the GA.

### **3.3.5 Methodology: GA Approach to Spectrum Allocation**

In this section we describe how the standard GA (SGA) algorithms can be applied to find a solution to the spectrum allocation problem. It is assumed that sensing has been done and that the secondary users within the network are looking to opportunistically use white spaces (free spectrum) which are not been used by primary users in the wireless network without causing

interference to other users. We can assume that the possible number of secondary users is finite and the spectrum resources (QoS) will always be countable thus making the spectrum allocation problem that of combinatorial optimization.

### 3.3.5.1 Introduction

The problem of spectrum allocation is a combinatorial optimization problem with an objective function and a solution space. The solution space for our problem is a set of parameters of the QoS. The objective function is the difference between the available QoS parameters and that requested by the secondary user. The closer this difference is to zero, the closer the optimization process is to the optimum solution. Since sensing is assumed to have been done, the sensed information represents a pool of available solutions for spectrum allocation for the secondary user, and from this pool the initial population for the GA can be selected randomly. After selecting the initial population, spectrum allocation decision takes place following the genetic algorithm procedure in Table5.

Table 5 Spectrum Allocation GA Procedure

Step	Action
1	Generate a random initial population of n spectrum, where n is the population size
2	Evaluate the fitness of each of the spectrum in the initial population
3	Generate new populations using processes in steps 4-6
4	Selects two spectrums among the current population using the roulette wheel method based on fitness of each spectrum
5	Crossover the two selected spectrum considering the crossover probability, to form new spectrums for the next generation
6	Mutate the new spectrum at each defined mutation point, considering the mutation probability and place it in the new population
7	Evaluate the fitness of each of the new spectrums in the new population
8	Repeat steps 3-7 until the stopping criteria have been met



### 3.3.5.2 The Chromosome Structure

We have considered five radio frequency (RF) QoS parameters in the optimization process as designed by Siddique and Azam [58]. Thus a solution consists of five RF parameters arranged in the order shown in Table6; this is the solution structure. In GA terminology, each parameter is referred to as a ‘gene’, while the solution is referred to as ‘chromosome’. The QoS requirement of the secondary user using the application is compared with several available solutions in the pool and then the GA searches for the best possible solution and assigns it. The QoS parameters considered are:

- Data Rate
- Signal Power
- Bit Error Rate
- Operating Frequency
- Modulation Technique

Table 6 Chromosome Structure

Data Rate	Signal Power	Bit Error Rate	Operating Frequency	Modulation Technique
-----------	--------------	----------------	---------------------	----------------------

#### Data Rate

This is the first gene of the chromosome. It is the rate at which bits are transferred per unit of time (seconds), i.e. bps. We have used a range of 0 – 2Mbps with a step size of 125kbps. This implies that we have 16 decimal values ranging from 0 – 15, where ‘0’ is assigned to the 1<sup>st</sup> data rate band (0-125 kbps), ‘1’ to the 2<sup>nd</sup> data rate band (126-250 kbps), etc. Table 7 shows this configuration.

Table 7 Data Rate Gene.

Index	0	1	2	....	15
Data Rate	0-125 kbps	126-250 Kbps	251 – 275 Kbps	....	1.876–2.000 Mbps

### Signal Power

This is the specific power range (transmit and receive) that allows the users to communicate without any error; it boosts the probability of successful communication. It is the second gene of the chromosome. This parameter was configured to range from -31dBm to 31dBm, step size of 1dBm, resulting into 63 decimal values of 0 – 62 required for chromosome representation. Table 8 shows this configuration.

Table 8 Signal Power Gene.

Index	0	1	2	...	62
Signal Power	-31 dBm	-30 dBm	-29 dBm	...	31dBm

### Bit Error Rate

This is the third gene of the chromosome. The BER quantifies the reliability of the entire radio system in terms of ‘bits transmitted’ to ‘bits received’. It is an end-to-end measure of the performance of the radio communication system. The BER can be defined as the number of bit errors divided by the total number of transferred bits during a time interval. This parameter was configured to range from  $10^{-1}$  to  $10^{-7}$ , step size of  $10^{-1}$  resulting into 8 decimal values required for chromosome representation. Table 9 shows this configuration.

Table 9 Bit Error Rate Gene.

Index	0	1	2	...	7
Bit Error Rate	$10^{-1}$	$10^{-2}$	$10^{-3}$	...	$10^{-7}$

### Operating Frequency

This is the fourth gene of the chromosome. It is the frequency at which data is transmitted and received. This parameter was configured to range from 0-20 MHz with a step size of 40 KHz producing 500 frequencies resulting in decimal values representation from 0 to 499. Table 10 shows this configuration.

Table 10 Operating frequency gene

Index	0	1	2	...	499
Operating Frequency	0-40 KHz	41-80 KHz	81-120 KHz	...	19.9 – 20 MHz

### Modulation Technique

This is the fifth gene in the chromosome. It is the process of varying one or more properties of a high-frequency periodic waveform, called the carrier signal, with a modulating signal which typically contains information to be transmitted. Eight modulation techniques have been considered and their equivalent decimal values range from 0 to 7 in the following order in which they are listed in Table 11.

Table 11 Modulation Technique Gene

Modulation Technique	Decimal Value
BPSK	0
QPSK	1
8PSK	2
16PSK	3
DBPSK	4
DQPSKMSK	5
16QAM	6
64QAM	7

The values of the respective parameters above have been coded in decimal for the purpose of initial population generation, selection and crossover. However, mutation process requires the binary form of any value encoding adopted. Therefore each of the genes to be mutated will need to be represented in their binary form. Table 12 shows the configuration of the chromosome in decimal and the number of bits used for the binary representation of each of the genes.

Table 12 Chromosome Configurations

Gene No.	Gene	Decimal Values	Number of Bits
1	Date Rate	0 - 15	4
2	Signal Power	0 – 62	6
3	Error Rate	0 - 8	4
4	Frequency Band	0 - 499	9
5	Modulation Technique	0 - 8	3

### 3.3.5.3 Fitness Measure

A pseudorandom initial population of 100 chromosomes was generated with a GA breeding rate of 50 generations. In formulating the fitness function (objective function) of the GA, Siddique and Azam [65] considered the magnitude of the difference between the values of each parameter (or gene) that is requested by the secondary user (QoS) and the corresponding values of the parameter available in the search space; The objective, given by (3.11), is to minimize the error.

$$G'_f = \min_y |G'_{gen} - G'_{su}| \quad (3.11)$$

where  $G'_{gen}$  is a randomly generated gene, and  $G'_{su}$  is the secondary user's requested QoS gene.  $Y$  is a vector of the five network parameters considered in the work. The fitness function tries to minimize the chances of the selection of the most terrible chromosomes for the next generation of population. Notably this work also considers the number of bits used to represent each gene in the chromosome as part of the fitness measure of each of the gene. The number of bits used to represent each of the genes is termed the *Weight* of the gene denoted by 'GW'. The weight of each gene is represented by GW1, GW2, GW3, GW4 and GW5 for the data rate (3.12), the signal power (3.13), the error rate (3.14), the operating frequency (3.15) and the modulation technique (3.16), respectively. The detailed weight for each gene represents the percentage ratio of the number of bits used to represent each gene to the total bits (26) of the chromosomes.

$$GW1 = (4/26) \times 100 \% \quad (3.12)$$

$$GW2 = (6/26) \times 100 \% \quad (3.13)$$

$$GW3 = (4/26) \times 100 \% \quad (3.14)$$

$$GW4 = (9/26) \times 100 \% \quad (3.15)$$

$$GW5 = (3/26) \times 100 \% \quad (3.16)$$

Another important constant used in calculating the fitness measure is a *fitness point* (FP). This FP will have an integer value within the range defined for each gene in their respective decimal representation part. This value is purely the developers own choice. The FP is meant to *limit* the search process of the algorithm on both sides of the required gene decimal value range. In fitness measure equations for each gene these fitness points are represented by FP1, FP2, FP3, FP4 and FP5 for the data rate, the signal power, the error rate, the operating frequency and the modulation technique respectively with chosen values being 6, 20, 7, 200 and 1 respectively. By denoting the fitness measure of a gene as  $fm$ , then  $fm$  is given by (3.17) and (3.18).

$$fm = [(GW \times G_f) / FP] \text{ for } FP > G_f \quad (3.17)$$

$$fm = [GW] \text{ for } FP \leq G_f \quad (3.18)$$

The total fitness of the chromosome  $Tfm$  is then calculated by summing up all the fitness of each of the genes and then subtracting it from 100. The  $Tfm$  is given by (3.19) and the aggregated weighted sum of each of the gene  $fm_{aws}$  is given by (3.20).

$$Tfm = 100 - fm_{aws} \quad (3.19)$$

$$fm_{aws} = fm_{dr} + fm_p + fm_{ber} + fm_{fb} + fm_{ms} \quad (3.20)$$

where  $fm_{aws}$  is the aggregated weighted sum of each parameter's fitness.  $fm_{dr}$  is the fitness measure of the data rate parameter.  $fm_p$  is the fitness measure of the signal power parameter.  $fm_{ber}$  is the fitness measure of the bit error rate parameter.  $fm_{fb}$  is the fitness measure of the frequency band parameter.  $fm_{ms}$  is the fitness measure of the modulation scheme parameter. The lower the value of the  $fm_{aws}$ , the higher the fitness measure  $Tfm$  of the chromosome.

In formulating the fitness function, it is important to consider the relationship between some of the specified parameters for real world applications. There are various relationships that exist between all of the parameters. In this thesis we have considered the relationship between the frequency band and the signal power simply because of the relative weights of these two with respect to other parameters used to formulate the fitness function. The frequency band has a weight of '9', while the signal power has a weight of '6'; these sums up to 15 which is approximately 57 percent of the total weight sum of 26. Figure 14 shows a typical real world application of the mobile communication system, where a mobile user wants to access the spectrum via a mobile base station. The relationship between the transmit power  $P_t$  and the transmit frequency  $f$  in the mobile communication system is given by (3.21). This is known as the Friis transmission formula; where  $P_r$  is the resulting received power at the other end of the transmission,  $G_t$  is the gain of the transmit antenna in the direction of the receive antenna,  $G_r$  is the gain of the receive antenna in the direction of the transmit antenna,  $c$  is the speed of light,  $R$  is the distance between the two antennas,  $f$  is the transmission frequency.

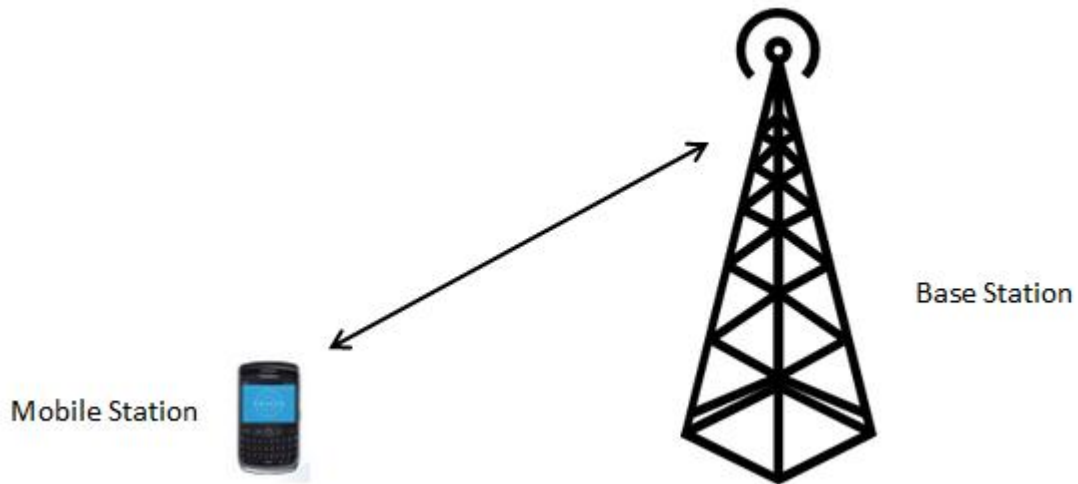


Fig. 14 Mobile Communication System

$$P_r = \frac{P_t G_t G_r c^2}{(4\pi R f)^2} \quad (3.21)$$

Equation 3-19 shows that the transmit power is directly proportional to the transmit frequency. The received power is inversely proportional to the transmit frequency; this means that the received power decreases with an increase in frequency. This reduction in the power density of the transmitted electromagnetic wave as it propagates through space is known as the path loss. Thus, the path loss is higher at higher frequencies. The consequence of (3.21) is that the secondary user's choice of signal power value determines the range of available operating frequency values.

The transmission power ranges from 0 to 62, which means there are 63 possible choices. The frequency ranges from 0 to 499, which means there are 500 possibilities. The relationship between the two parameters is given by (3.22) and (3.23). In each of the two equations, the transmission power value is represented by 'x' while  $f(x)$  represents the corresponding frequency range values for 'x'.

$$f(x) = x \left( \frac{500}{63} \right) : (x + 1) \left( \frac{500}{63} \right) \Big|_{x=0} \quad (3.22)$$

$$f(x) = x \left( \frac{500}{63} \right) + 1 : (x + 1) \left( \frac{500}{63} \right) \Big|_{1 \leq x \leq 62} \quad (3.23)$$

The equation (3.22) is applicable when  $x$  is '0', while equation (3.23) is applicable when  $x$  ranges from 1 to 62. By using (3.22), i.e. when the value of  $x$  is '0', the corresponding range of available frequency is 0 - 8. Also by using (3.23), if the value of  $x$  is '4', the corresponding range of available frequency will be 25 – 32. It is important to note that the value of  $f(x)$  is rounded to the nearest decimal value. Table 13 shows the signal power parameter and the corresponding



frequency range. It is important to note that each of the values in table 13 represents the decimal representation of the actual value of the parameters as shown in tables 8 and 10.

Table 13 Power to frequency configurations

Signal Power	Frequency Range
0	0 : 8
1	9 : 16
2	17 : 24
3	25 : 32
⋮	⋮
61	485 : 492
62	493 : 500

Although the values within the frequency range represent the ideal range for the corresponding signal power for transmission, the availability of the frequency range is not guaranteed. The availability of the spectrum depends on the behavior of the primary users within the network at any given time. Thus the genetic algorithm comes handy in searching for the closest spectrum to what is required with respect to what is available.

#### 3.3.5.4 Construction of New Population

After the fitness evaluation of each of the candidate solutions in the initial population has been done, the algorithm ascertains if the stopping criteria has been met. If met, the solution (spectrum) obtained is assigned to the secondary user, otherwise the algorithm proceeds to the evolutionary process in order to generate new sets of solutions. This evolutionary process includes

the selection, crossover and mutation in that order. There are various ways in which each stage of the evolutionary process can be implemented. The choice of how to implement each stage of the evolutionary process depends largely on the objective of the research and the complexity of the problem. The method of implementation chosen for this work at each stage of the evolutionary process is discussed next.

#### **3.3.5.4.1 Selection**

The major objective of the work is to find a way to improve the fitness diversity of solutions which the GA can come to before the stopping criteria is met. In order to be consistent with this objective we have chosen the *roulette wheel* method of selection. This method is also called the *fitness proportionate* selection because the probability of a candidate solution being selected  $p_{cs}$  is proportionate to the candidate solution's fitness value. It is popular among the users of genetic algorithm because of its notable advantage in that every candidate solution has a finite chance of making it to the next generation. After the selection process was completed, a technique called *elitism* was used to retain the best-fit solution of all the selected solutions, making it part of the next generation. This ensures that best fit chromosome is not lost during crossover and mutation thus making sure the performance of the GA is always on the increase.

#### **3.3.5.4.2 Crossover**

This is the mechanism by which the genetic algorithm exchanges the genetic properties (in blocks/segments) among candidate solutions in search for the optimum solution. In this case the blocks are represented by the parameters of the solution. The crossover is done by exchanging the same parameter(s) of two different solutions from the selected pool of solutions. There are several crossover techniques like *single point*, *2-point*, *multi-point* and *uniform crossovers*. In this work

we have adopted the *2-point crossover*. Hasancebi and Erbatur [66] argued that numerical testing of the aforementioned crossover techniques show that 2-point crossover technique produces better solutions as compared to others. For a proper crossover technique, the crossover rate is also important, which is usually set from 80% to 90%. However, for some problems, this range can lead to space *exploitation* which may lead to premature convergence. Therefore we have used a rate of 60% for this problem especially because the aim of our work is to promote diversity. Thus at the rate of 0.6, a 2-point crossover process is performed to produce new results. These results are then transferred to the next generation thus increasing the diversity within the solution space. The two-point crossover calls for two points to be selected on the parent solution strings as shown in Fig. 15. Everything between the two points is swapped between the parent solutions, rendering two child solutions: in our case, a ‘point’ represents a ‘gene’.



Fig. 15 Two-Point Crossover

#### 3.3.5.4.3 Mutation

Mutation is a genetic operator used to force or maintain genetic diversity from one generation of a population of algorithm chromosomes to the next. It is analogous to exploration. It alters one or more gene values in a chromosome from its initial state. Mutation is applied on genes of the child-chromosome after recombination or crossover, altering a binary bit of 0 to 1 or vice versa [23]. Solutions selected for mutation were converted from decimal form to binary form for the purpose of mutation and converted back to decimal form after mutation was done. In mutation,

the solution may change entirely from the previous solution; hence GA can come to better or worse solution. Mutation occurs during evolution according to a user-definable mutation probability. This probability should be set low to prevent the search from turning into a primitive random search. We have used a mutation rate of 2%.

### 3.4 Chaotic GA Optimization Approach

Chaos refers to apparent randomness (but definitely not true randomness), or irregularity, or unpredictability that arises in deterministic dynamical systems [39]. According to Kinsner [17], the properties of a chaotic system that provide additional benefits over randomly generated solutions are sensitivity to initial conditions, topological density and topological transitivity. Topological transitivity implies that within a cycle of trajectory no one solution is visited more than once. This guarantees diversity and ensures that the CGA is able to explore the entire solution space. The initial population of size  $N$  was generated using the coupled logistic chaotic sequence. The elitism method of selection is used to ensure that best fit  $n$  solutions are copied to the next generation. The decisions as to which of the genes to be crossed over and/or mutated of the remaining  $N-n$  chromosomes are also taken using a chaotic sequence. The important steps in the CGA include: establishing the logistic chaotic sequences, using the sequence to generate the initial population, and finally using the sequence to decide which solutions are selected for crossover and mutation. The behavior of any chaotic system is governed by deterministic equations. Chaotic systems have a sense of order or pattern even though they appear to be disorderly. The chaotic system can be generated by the well-known one-dimensional logistic map which is given by (3.24):

$$z_{k+1} = \mu z_k (1 - z_k) \quad \text{for } \mu = 4 \quad (3.24)$$

where  $z_k$  represents the value of the variable  $z$  at the  $k^{\text{th}}$  iteration;  $z_k$  is in the interval  $[0,1]$ ; and  $\mu$  is a so-called bifurcation parameter of the system. In this work, we have employed a new chaotic map proposed by Mingjun and Huanwen [49] because it has a better probability distribution. This new chaotic map is given by (3.25) as:

$$z_{k+1} = \eta z_k - 2 \tanh(\gamma z_k) e^{-3z_k^2} \text{ for } \eta = 0.9, \gamma = 5 \quad (3.25)$$

If we assume a search space having a total of 100,000 solutions, the probability distribution of the solutions as generated by the logistic map is shown in Fig.16 while that generated by the new chaotic map is shown in Fig.17. A GA combined with chaotic operator has several advantages such as large solution search space, reduced similarity among individual solution and fast convergence speed [45]. As explained by Mingjun and Huanwen [49], the logistic map of Fig.16 shows a lot of the points on the distribution are near the edges, meaning that it can escape local minima although it is difficult to seek for the global optimum solution. Figure 17 shows that point distribution of the new chaotic map is similar to uniform distribution with two peaks near -0.8 and 0.8. This means that the new chaotic map has the ability to escape local optimum as well as converge to the global optimum at the same time. This is the motivation for using the new chaotic map. We observe that for an initial population size of 5000 candidate solutions there is a significance difference pictorially between the ones generated by the new chaotic map (Fig.18) compared to that generated by a pseudo-random method (Fig.19). Thus the main difference between the standard GA approach and the Chaotic GA approach is that the random mechanism involved in the stages of initial population generation, crossover and mutation is replaced by a chaotic mechanism. The CGA was therefore used as the search mechanism with the goal of not revisiting any one solution and consequently improving the diversity of the candidates.

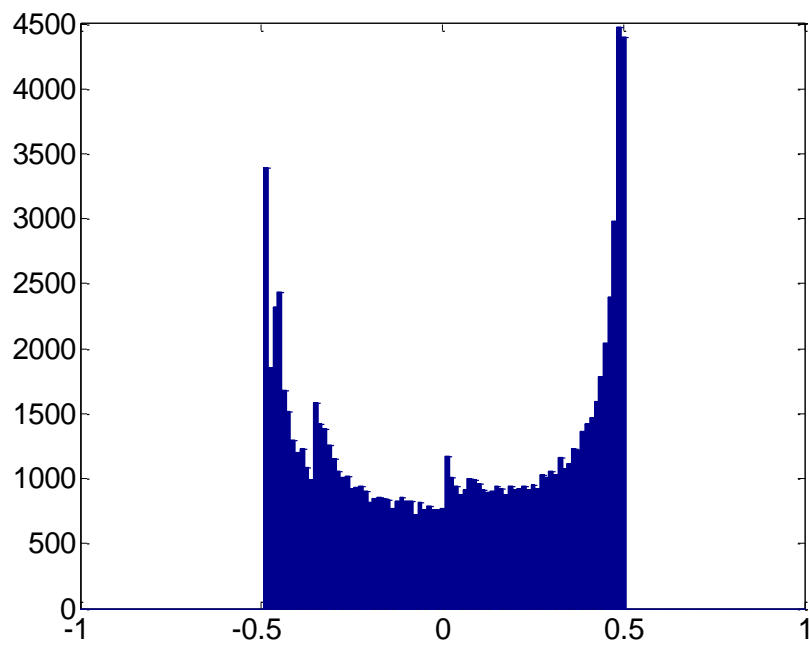


Fig. 16 Probability distribution of Logistic map

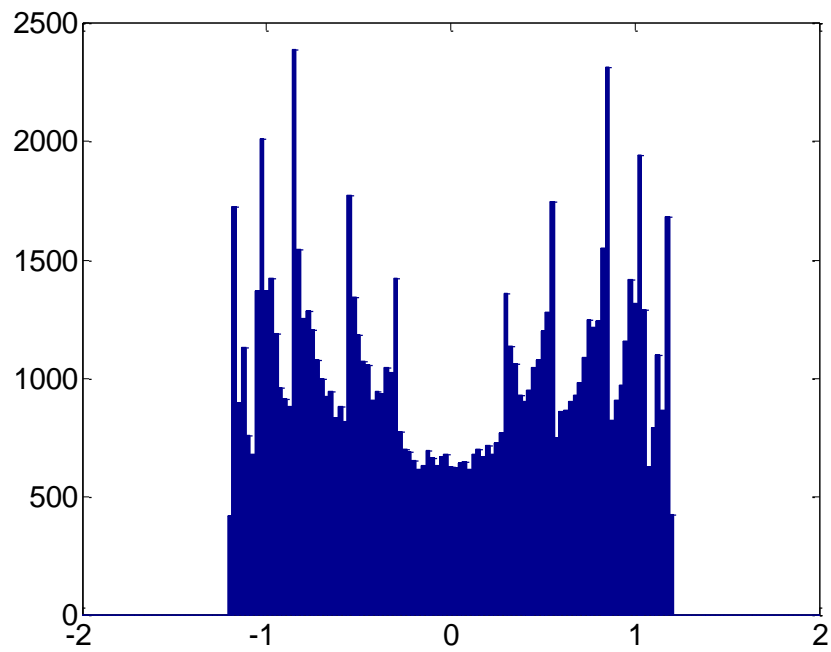


Fig. 17 Probability distribution of new chaotic map

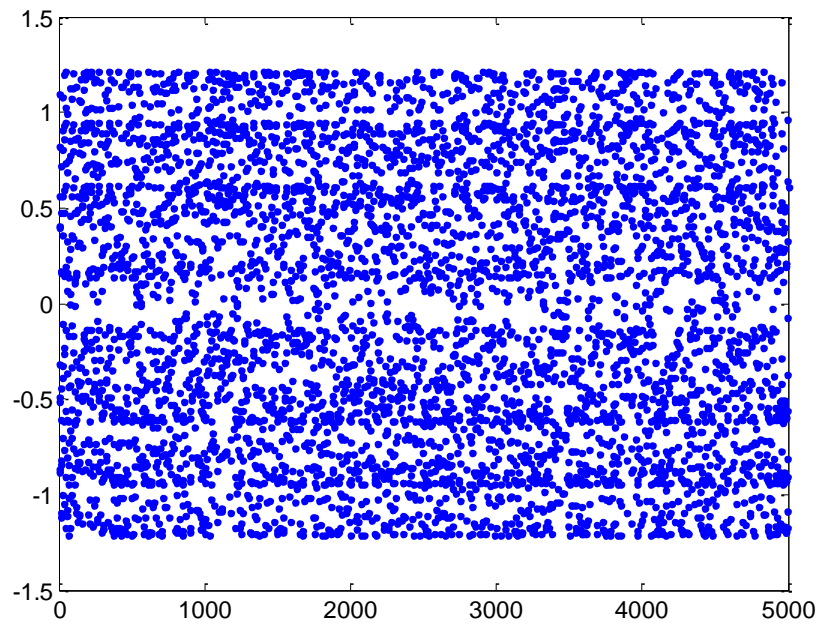


Fig. 18 Initial population with new chaotic map

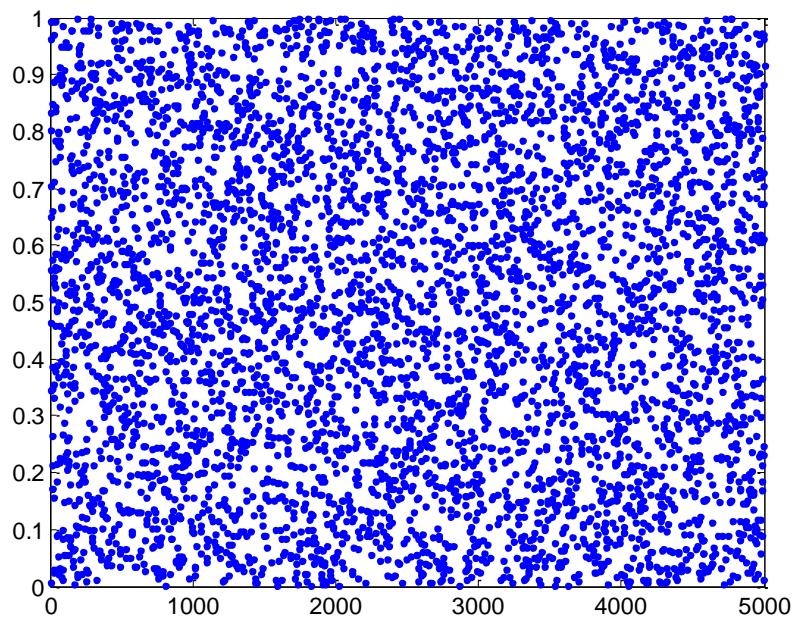


Fig. 19 Initial population with random generator

## **Chapter 4**

### **Experiments and Results**

#### **4.1 Introduction**

This chapter describes the experimental setup and results to verify the performance of the standard genetic algorithm (SGA) and the chaotic genetic algorithm (CGA) in obtaining good solution within polynomial time. The performance of the SGA was verified on the two different applications discussed in the thesis: multiprocessor task scheduling and spectrum allocation. The CGA was then applied to one of the applications to evaluate performance in comparison to the SGA. Specifically, the CGA was applied to the spectrum allocation problem application because of the many possibilities (larger search space) and the value encoding approach to the problem. We have used variance of the solutions produced over several generations to show the diversity patterns of both the SGA and CGA.

#### **4.2 Multiprocessor Task Scheduling**

One of the goals of task scheduling in a multiprocessor system is to schedule tasks on the processors such that the processing time is reduced to the possible minimum. The algorithm to evaluate our method has been implemented using Java. To verify the performance of our proposed scheduling method (min-EST) using GA, we have compared it with the max-EST method using GA [8], as well as the expected optimal schedule of the system. The parameter setup of the GA is discussed next. The scheduling method is used to generate an initial set of schedules and the GA is used to search for a good schedule within the search space. The experiment was performed using three processors.



### 4.2.1 GA Parameter Setup

The GA parameter values are varied depending on the amount of tasks to be scheduled. This is so because the larger the size of the DAG, the more time it will require for GA to obtain a suitable schedule, assuming a fixed population size. Table 14 shows the GA parameters for task graphs that have the number of tasks ranging from 16 to 30. Also, the execution time for each task is randomly generated from 1-15. The population size is set at 20; meaning 20 schedules. The crossover rate and mutation rate were carefully chosen to reflect the recommended ranges in literature. The crossover rate was set at 0.8, while the mutation rate was set at 0.01. The stopping criterion was chosen to be the number of generations. There is no rule guiding the number of generations the GA should run; we have set the number to one considered sufficient enough for the GA to come to a good solution. By setting the number of generations to 10, the simulation runs for less than 6 seconds.

Table 14 GA Parameters

POPULATION SIZE	20
CROSSOVER RATE	0.8
MUTATION RATE	0.01
NUMBER OF GENERATIONS	10

### 4.2.2 Results

Due to the random walk taken by the SGA, different results may be obtained at different runs of the algorithm. Therefore, in order to compare the Min-EST method with the Max-EST we have taken 10 results obtained at 10 different runs of the algorithm. This is to ascertain the average performance of both the Min-EST and the Max-EST scheduling method using the GA.

The results obtained from simulation are shown in table15. The results show the total finishing time (TFT) of the schedules obtained for each of the Max-EST and the Min-EST algorithm. The random task graphs are constructed in such a way that the optimal schedule is known. The results were compared with the known optimal schedule. Simulation shows that though both methods run in polynomial time, the Min-EST may obtain a lower TFT compared to the max-EST over the same number of generation.

Table 15 Results

Algorithms	Total Finish Time (seconds)		
	Number of Processors = 3		
	Number of Tasks		
	16	21	30
Max-EST	50	52	147
Min-EST	47	50	141
Optimal Schedule	32	42	114

Table 15 shows that the solutions obtained for the Min-EST and Max-EST can be very close to each other. Result shows that the Min-EST can produce schedules that have the almost the same TFT or a lower TFT compared to the Max-EST. This can be attributed to the small percentage of tasks that have multiple EST in the task graph. The three task graphs comprise of 16, 21 and 30 tasks respectively. For each of this graph, 20% of tasks within the graph have multiple EST. For tasks with multiple EST, they can be scheduled at any of the ESTs or later. The Min-EST will perform better than the Max-EST only when tasks are scheduled at the min-EST.

The performance of the algorithms when the ratio of tasks that have multiple EST to those with single EST increases is shown in Table 16.

Table 16 Results

Algorithms	Total Finish Time (seconds)		
	Number of Processors = 3		
	Number of Tasks = 30		
	Increasing Ratio of Tasks having Multiple ESTs		
	20%	40%	60%
Max-EST	133	210	243
Min-EST	131	200	232
Optimal Schedule	120	166	216

Table 16 shows that the Min-EST can produce better schedules with a relative increase in the number of tasks that have multiple EST. When the ratio of tasks that have multiple EST to tasks that have single EST is large, there is an increased probability that tasks with multiple EST will be scheduled at the minimum EST or before the maximum EST. If the multiple-EST tasks also have a relatively high NTD, then there is an additional increase in the probability of the tasks to be scheduled at the minimum EST. The minimum EST becomes insignificant when the multiple EST tasks have a lower scheduling priority; this will lead to an equal performance of the Min-EST and Max-EST algorithm. Thus, our proposed scheduling algorithm (min-EST) can obtain a better schedule compared to the max-EST if both of the following conditions are true:

- When the number of task with multiple EST is considerably higher than tasks with single EST.
- When the NTD of the tasks with multiple EST is relatively more than the tasks having single EST.

The initial schedules that make up the initial generation are randomly generated. As a result, the fitness quality of the schedules will vary every time the algorithm is run. The fitness quality of the initial generation can determine the fitness quality of the solution (schedule) produced by either of the Min-EST or Max-EST. Thus the performance of GA using the two schedules does depend largely on the quality of the initial population generated. However we have shown that the GA can be used to obtain a good schedule in polynomial time.

### 4.3 Spectrum Allocation

The primary goal of a cognitive radio is to facilitate secondary users to opportunistically make use of free spectrum that are not being used by the primary users without degrading the quality of service of the primary user. We have used the GA engine as the mechanism to allocate the free spectrum to the users. In running this experiment, we have assumed a secondary user's QoS requirement shown in table17. From table13, the frequency range that corresponds to the transmission power of '60' is from 477 to 484; therefore the value 480 has been chosen. We have also assumed that a communication link already existed meaning sensing has been done, this work therefore concentrates on using the GA as the spectrum adaptation mechanism. Simulation was done using MATLAB to test the proposed solution.

Table 17 QoS requirements of an application given as input to the process.

Data Rate	Power	Bit Error Rate (BER)	Frequency Band	Modulation Scheme
3	60	2	480	4

### 4.3.1 GA Parameter Setup

The GA parameter setup was varied to evaluate the performance of the algorithm with respect to the variation. Two of the GA parameters varied were the population size and number of generations. The crossover rate and mutation rate are constant; they were carefully chosen to reflect the recommended range in literature. The crossover rate was set at 0.6 while the mutation rate was set at 0.01. In order to mimic a random (SGA) walk in the crossover phase, a random number between 0 and 1 is generated and compared to '0.6' to determine if the crossover process can be performed. The value was chosen to be greater than the average of 0 and 1, i.e. 0.5. For the CGA, the logistic function is used to generate the random number. The stopping criterion was chosen to be the number of generations. Since there is no rule guiding the number of generations the GA should run, the value is set to one considered sufficient enough for the GA to come to a good solution. The parameter setup is in table18.

Table 18 GA Parameters

POPULATION SIZE	100
CROSSOVER RATE	0.6
MUTATION RATE	0.02
NUMBER OF GENERATIONS	100

### 4.3.2 Results

The assumed secondary user's QoS requirement inputted into the GA framework is shown in table17. In order to show the performance of the SGA algorithm over some time, the algorithm was run 10 times and results obtained are shown in table19. The numbers succeeding the letter R in the result row represent the number of times the algorithm was run. These results have been plotted in figures 20 to 26 and detailed analysis has been provided.

Table 19 Resultant Spectrum and Corresponding Fitness Measure

Results	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Data Rate	14	7	10	3	4	8	7	3	8	5
Signal Power	57	60	57	53	61	62	56	57	61	60
Bit Error Rate (BER)	4	3	1	6	7	1	2	3	6	2
Frequency Band	461	486	458	428	492	496	453	460	493	482
Modulation	6	3	5	1	2	4	3	1	4	2
Fitness(%)	67.32	85.28	74.96	68.16	77.76	84.48	81.56	81.20	79.64	87.76

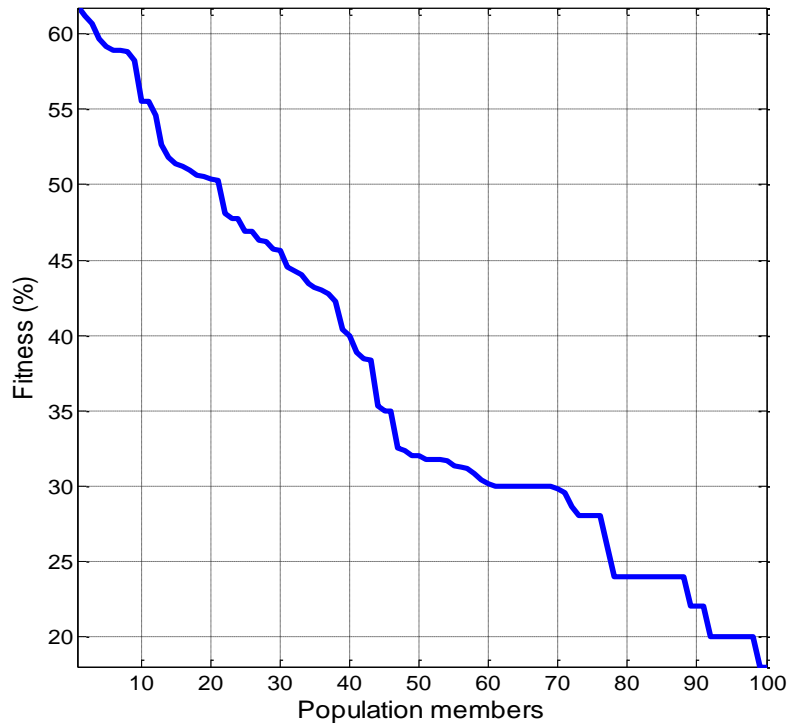


Fig. 20 Fitness measure of initial population in descending order

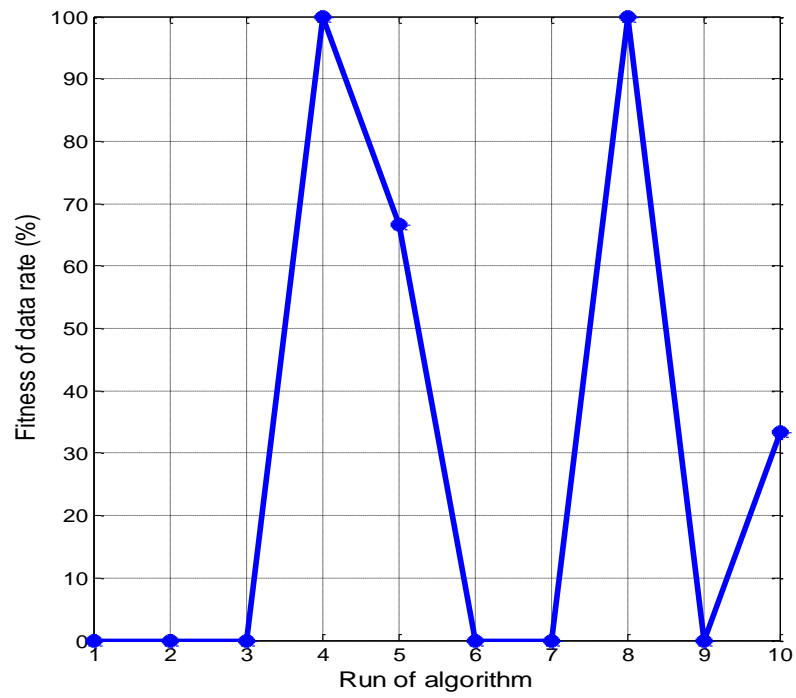


Fig. 21 Fitness measure of parameter 1 over ten runs

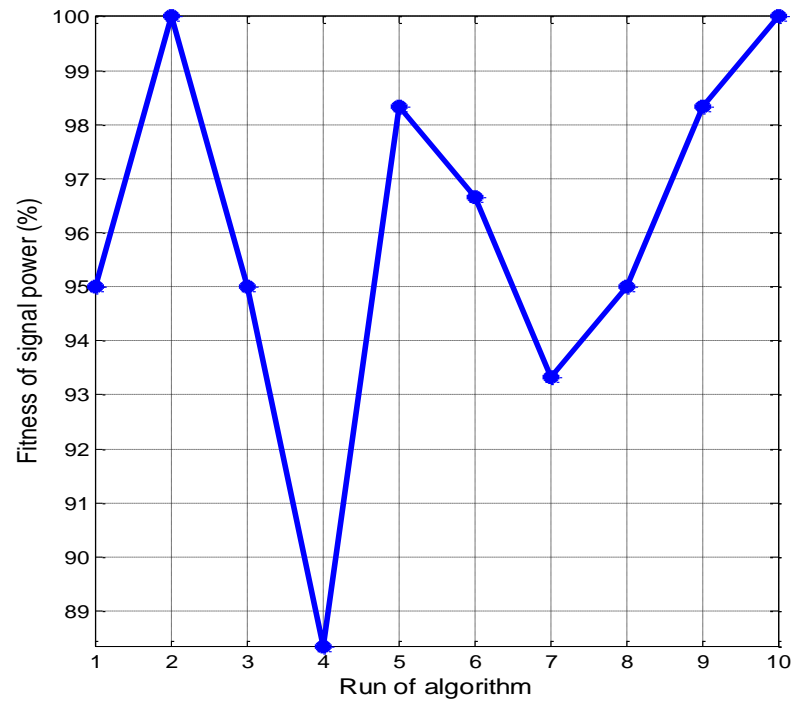


Fig. 22 Fitness measure of parameter 2 over ten runs

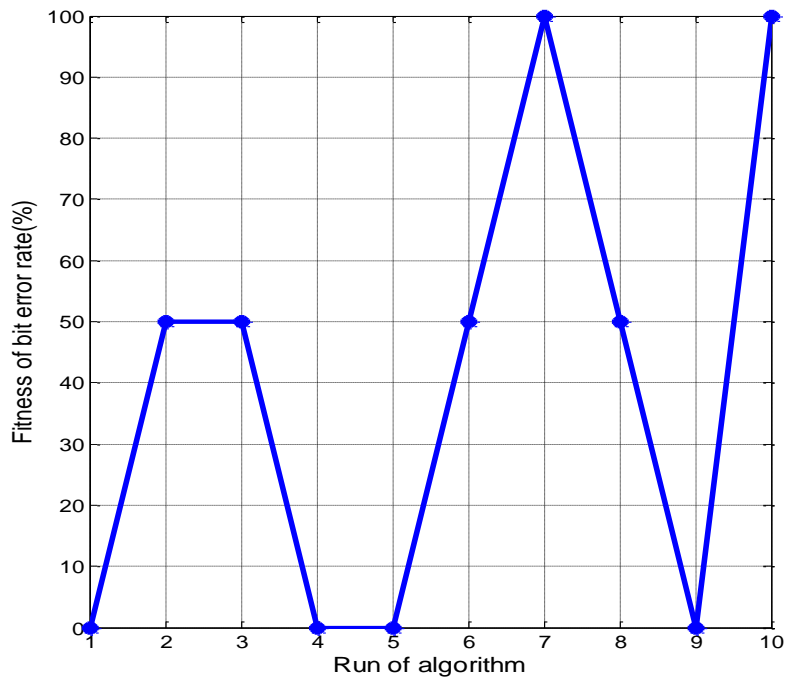


Fig. 23 Fitness measure of parameter 3 over ten runs

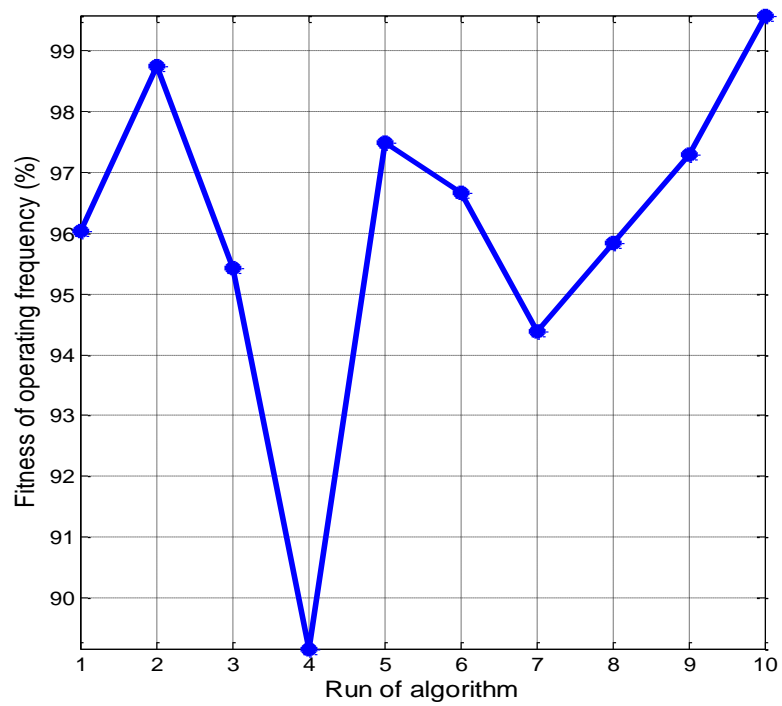


Fig. 24 Fitness measure of parameter 4 over ten runs



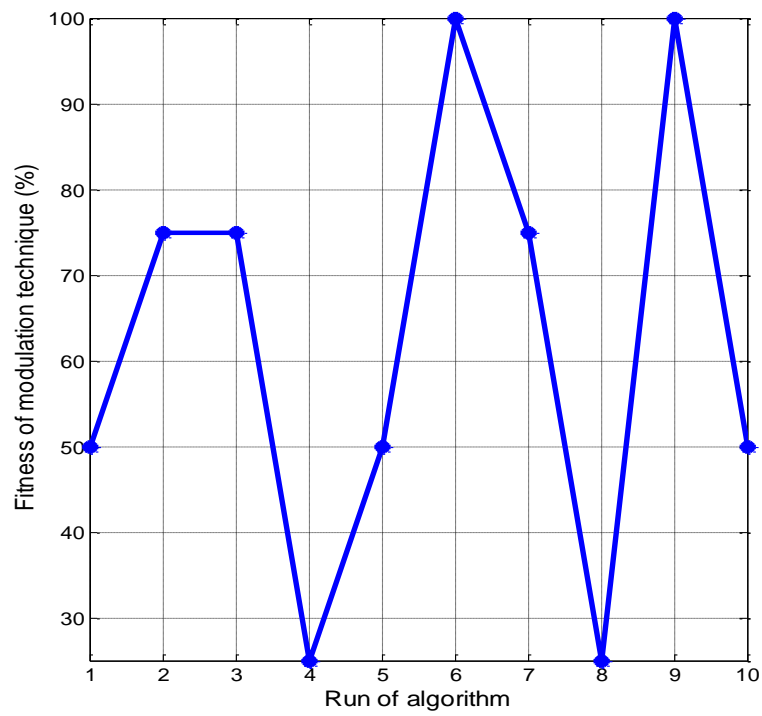


Fig. 25 Fitness measure of parameter 5 over ten runs

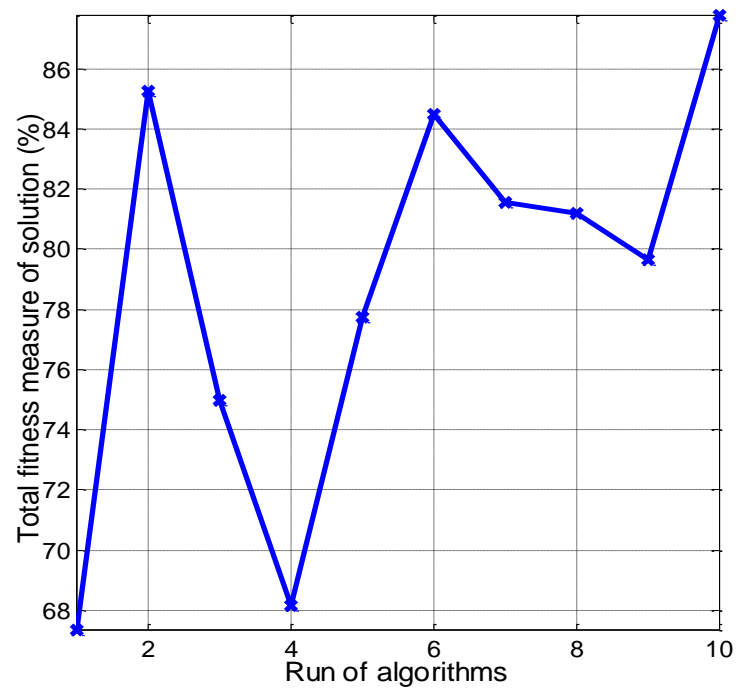


Fig. 26 Total fitness measure of spectrum solution over ten runs

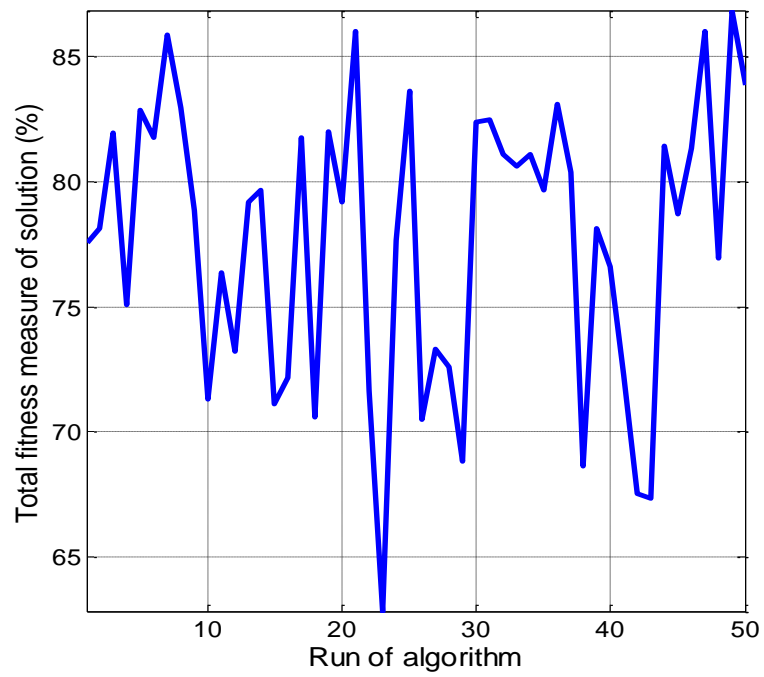


Fig. 27 SGA average fitness measure

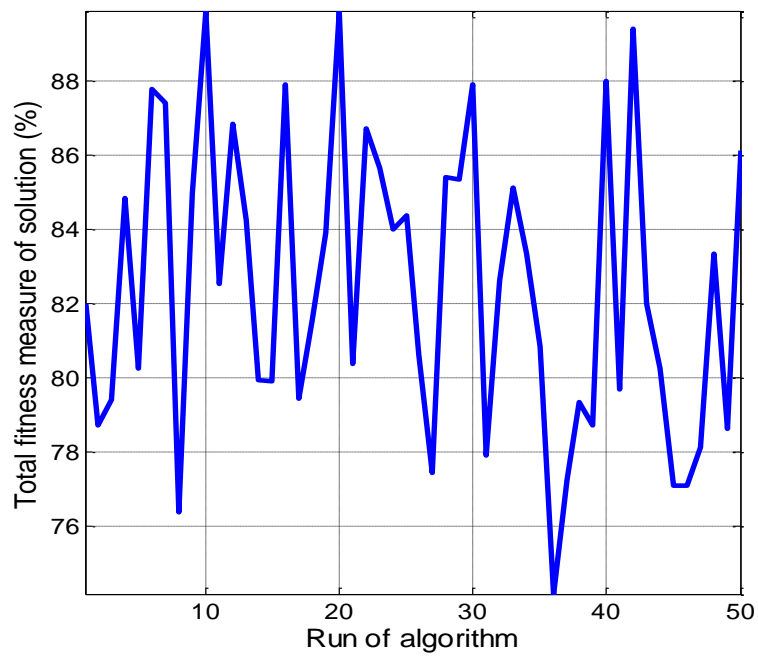


Fig. 28 CGA average fitness measure

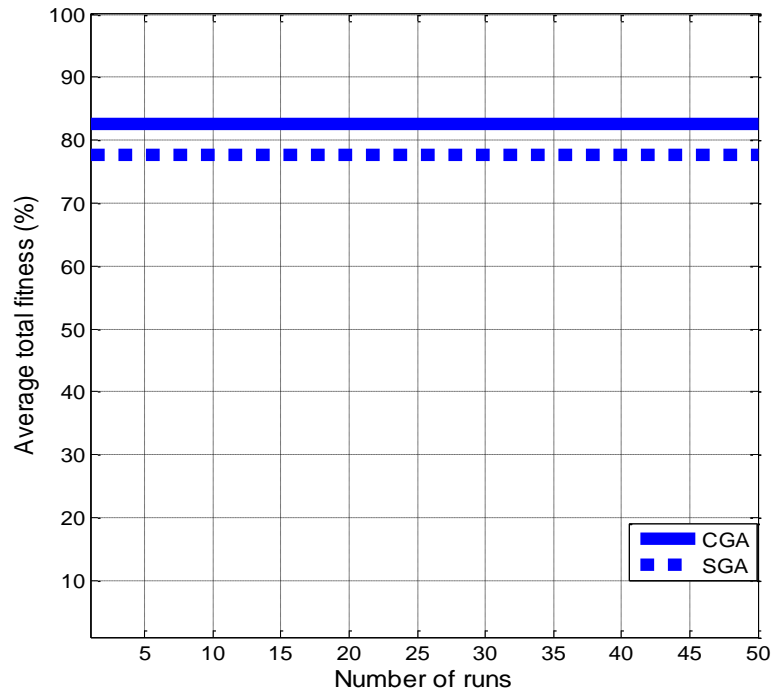


Fig. 29 SGA and CGA Average fitness measure for 50 runs

Figure 20 shows the fitness measure of the initial generation of the first run of the algorithm in descending order. The figure shows that the percentage fitness of the members of the population ranges from 18 to around 61.76. Table 20 shows the parameter values of each of the most fit and least fit member of the initial population solution. The importance of this figure is that it gives us an idea of the fitness diversity of the solutions in the initial population. We will then be able to compare the fitness diversity in the final generation with this initial generation.

Table 20 Best and worst members of the initial population

Initial Population	Data Rate	Power	Bit Error Rate (BER)	Frequency Band	Modulation Scheme
Most Fit	14	60	7	482	7
Least Fit	0	2	0	14	0

Figure 21 shows the range of the data rate parameter's fitness measure over the 10 different runs of the algorithm. The results obtained varied between 0 and 100%. The average performance of the GA on this parameter calculated over 10 runs is approximately 30.2%. It is important to note that the data rate parameter's contribution to the spectrum's fitness is equal to  $4/25$  which is about 16%. This is the degree at which the data rate impacts on the fitness of the solution. Figures 22 to 25 also show the results for each of the other four parameters' fitness measure over 10 runs; these results have been summarized in Table 21. For each of the parameters, Table 21 shows the range of the fitness measure, the average fitness over the 10 runs and the degree to which the parameter impact on the total fitness of the solution.

Table 21 Summary of parameters' performance

Parameter	Range (%)	Average Fitness (%)	Parameter Weight (%)
Data rate	0 - 100	30.20	0.16
Signal Power	88.4 – 100	96.03	0.24
Bit Error Rate (BER)	0 - 100	40.00	0.12
Frequency Band	89.1 – 99.5	96.09	0.36
Modulation	25 - 100	62.50	0.12

Figure 26 shows the range of the resultant solution's fitness measure over the 10 different runs of the algorithms. The results obtained varied between 67.32% and 87.76%. The average performance of the GA calculated over the 10 runs was approximately 78.81%. For each run of the algorithm, the spectrum solution obtained by the GA represents the best solution that was visited

by the GA among the set of all possible solutions in the pool. The random nature of the GA accounts for the different results that were obtained every time the algorithm is run. The optimum solution has a fitness of 100 percent and represents the requested QoS by the secondary user given in Table 17.

In order to verify the consistency of the two algorithms, we have taken the average fitness measure of the solutions obtained over 50 runs of the two algorithms. Figure 27 shows that the average fitness measure of the solutions obtained by the SGA over 50 runs is 77.70%; whereas Figure 28 shows that the average fitness measure of the solutions obtained by the CGA over 50 runs is 82.4832%. This represents about 4.78% increase in performance by the CGA compared to the SGA. Figure 29 shows the average fitness measure of both algorithms over the 50 runs.

#### **4.4 Convergence and Variance (Diversity)**

Literature review in this thesis shows that incorporating chaos into some or all of the evolutionary processes of a GA and other heuristics can speed up the rate at which heuristics come to a ‘good’ solution. In this section we present the analysis obtained from our simulation.

##### **4.4.1 GA Parameter Setup**

In order for the algorithm to be applicable in real time, it must process information (QoS input into the system) and produce a response in the form of solution (spectrum) within a specified deadline (time). Otherwise, the system is regarded not useful. This is a good motivation for using the GA because it can obtain a good solution within a specified time – in our case a set number of generations. Although the performance of the GA depends significantly on the quality of the initial solutions generated and the set population with respect to the problem size, we have set the number of generation to 100 in order to verify convergence and diversity. The other GA

parameters setup is as shown Table 22 below. We have assumed the same secondary user's QoS requirement in Table 17.

Table 22 GA Parameters

POPULATION SIZE	50
CROSSOVER RATE	0.6
MUTATION RATE	0.02
NUMBER OF GENERATIONS	100

#### 4.4.2 Convergence: SGA vs. CGA

The results obtained using the SGA and CGA framework are shown here. We have also observed that for each of the SGA and CGA, the diversity of the initial population in terms of fitness measures was similar every time the algorithm was run.

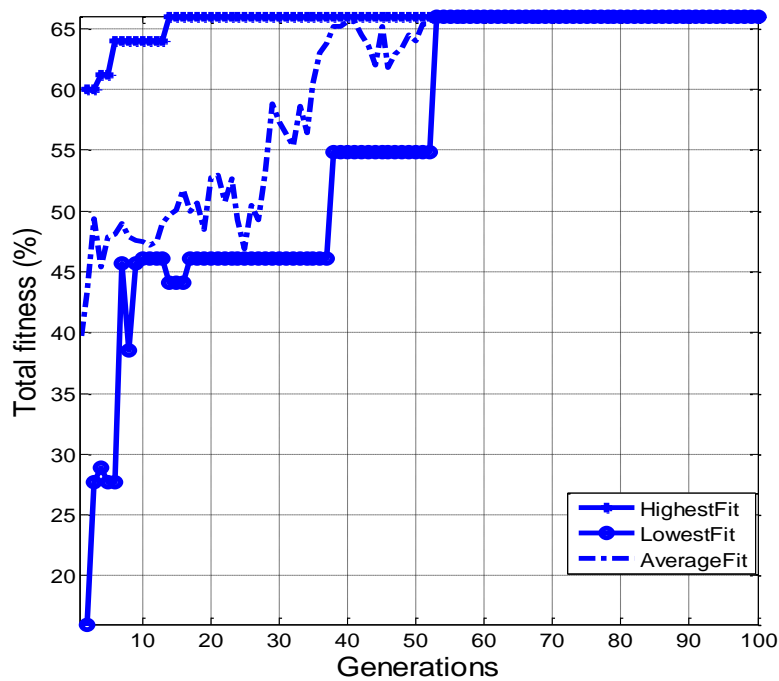


Fig. 30 SGA fitness per generation

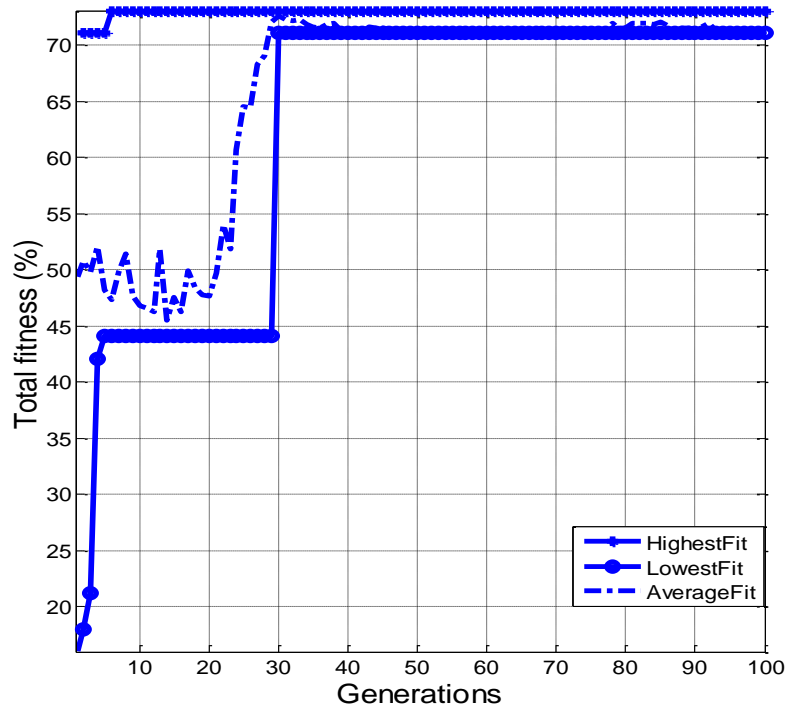


Fig. 31 CGA fitness per generation

The Figures 30 and 31 show the performances of the SGA and CGA over 100 generations. For the purpose of analysis, we have plotted the best fit, average fit and least fit solutions at every generation. Figure 30 shows that the SGA obtained a solution with fitness measure of 66.04% at the 14<sup>th</sup> iteration whereas Figure31shows that the CGA obtained a better solution with a fitness value of 73.08% at the 6<sup>th</sup> iteration. This means that the CGA obtained a more fit solution which is 7.04% better than that obtained by the SGA. The results also imply that the CGA converges to the better solution at a rate that is 133% faster than the SGA. Fig 30 shows that the SGA got stuck at the 53<sup>rd</sup> iteration and converges to a population where all the members have the same fitness measure of 66.04%. However the CGA maintained a population of members having a degree of diversity over the 100 generations. Also, the results show that the CGA obtained better average and least fit solutions compared to the SGA.

#### 4.4.3 Variance: SGA vs. CGA

The second and major objective of this work focuses on how to maintain robust solution diversity that the GA visits within the search space. In order to verify the diversity pattern of the solutions from the first generation to the last generations, we have estimated the *variance* over the generations. Variance describes the probability distribution of an observed population of numbers. It measures how far a set of number is spread out. A variance of zero typically indicates that all the fitness values are identical. A small variance indicates that the solutions tend to be very close to the average fitness value and hence to each other whereas a high variance is an indication that the solutions are very spread out from the average and from one another. It is this variance that distinguishes a chaotic phenomenon from a random phenomenon. With chaos, no one solution will be revisited or repeatedly generated thus leading to a guaranteed diversity. However, the random walk does not guarantee robust diversity as solutions with same fitness values can be revisited or repeatedly generated. Figures 32 to 35 show the variance of the population for the SGA and CGA over the set generations.

The Figures 32 and 33 show the variance pattern of the SGA and CGA respectively. From Figure 32 we see that the variance ranges from 147 at the initial generation to approximately 0 at the 53<sup>rd</sup> generation. Thus the SGA loses its diversity at the 53<sup>rd</sup> generation at which point we can technically say that the algorithm is stuck. However, figure 33 shows that the variance of the CGA varies from 180 at the initial generation to approximately 0 at the 30<sup>th</sup> generation. However the variance increases to around 2 at the 31<sup>st</sup> generation and stays at this value up till the 40<sup>th</sup> generation where it decreases to 0 again. It increases to around 2 again at the 43<sup>rd</sup> generation before decreasing to 0 again at the 50<sup>th</sup> generation. This behaviour of increasing to 2 and



decreasing to 0 continue at different interval up till the 100<sup>th</sup> generation. Though the variance value of 2 is considered very close to zero, at that point the diversity is not totally lost. This diversity gives the CGA the ability to obtain better solutions over the generations. A plot of the fitness values of the SGA solutions in the last generation arranged in descending order is shown in Fig. 34. The figure indicates that all members of the last generation have the same fitness measure of 66.04, thus diversity is completely lost. This is consistent with what is obtainable in Fig. 32. Figure 35 shows the plot of the fitness values of the CGA solutions in the last generation arranged in descending order. The range of the fitness values is from 73.08% to 71.08%. It is observed that only 4% (2 of 50) of the solutions actually have their fitness values greater than 71.08%. The remaining 96% of the solutions have the same fitness values of 71.08%. The 4% can allow the CGA to obtain better solution compared to the present solution of 73.08%.

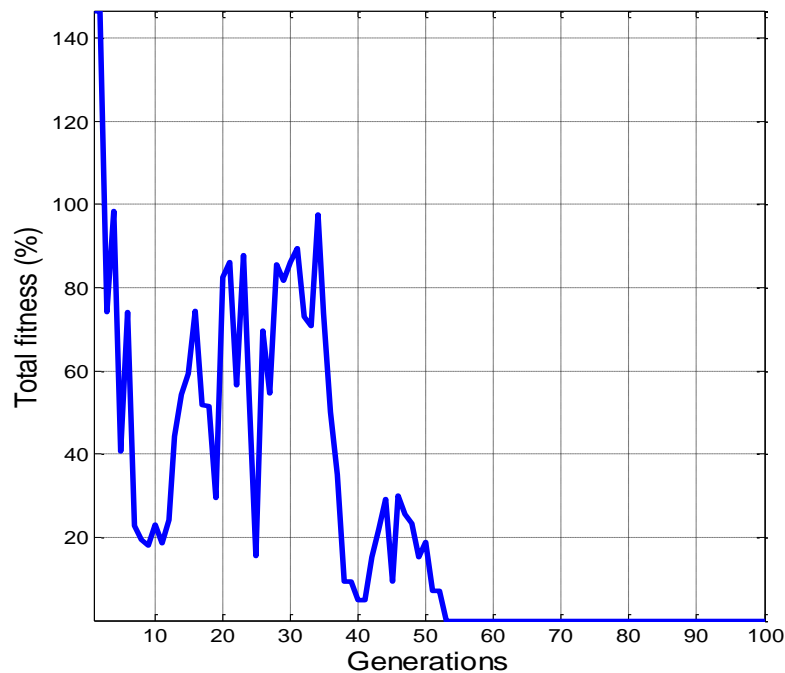


Fig. 32 SGA variance

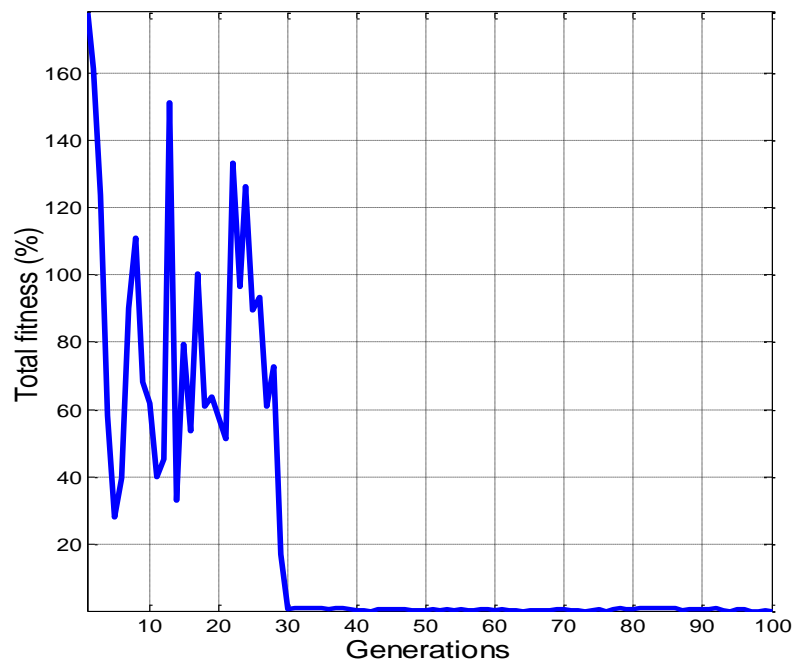


Fig. 33 CGA variance

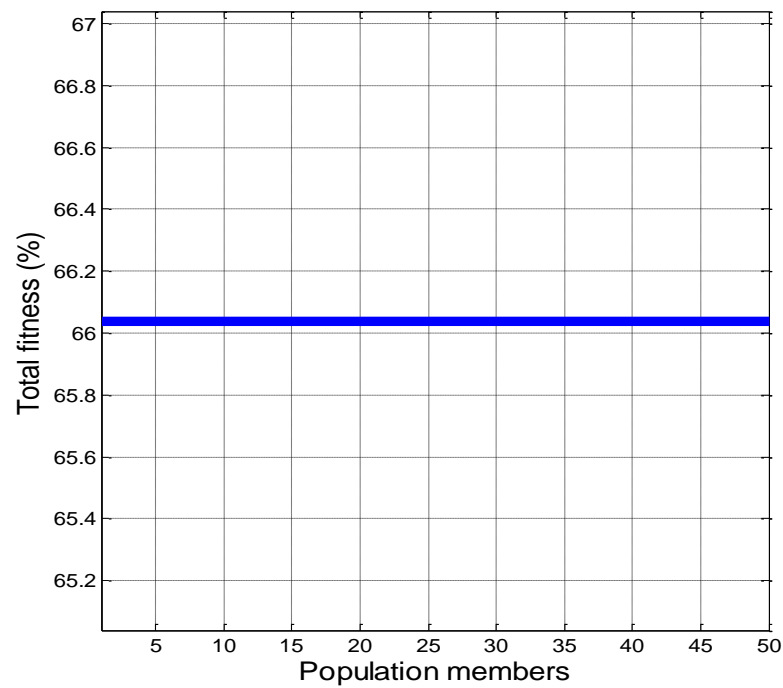


Fig. 34 SGA fitness measure of the last generation in descending order

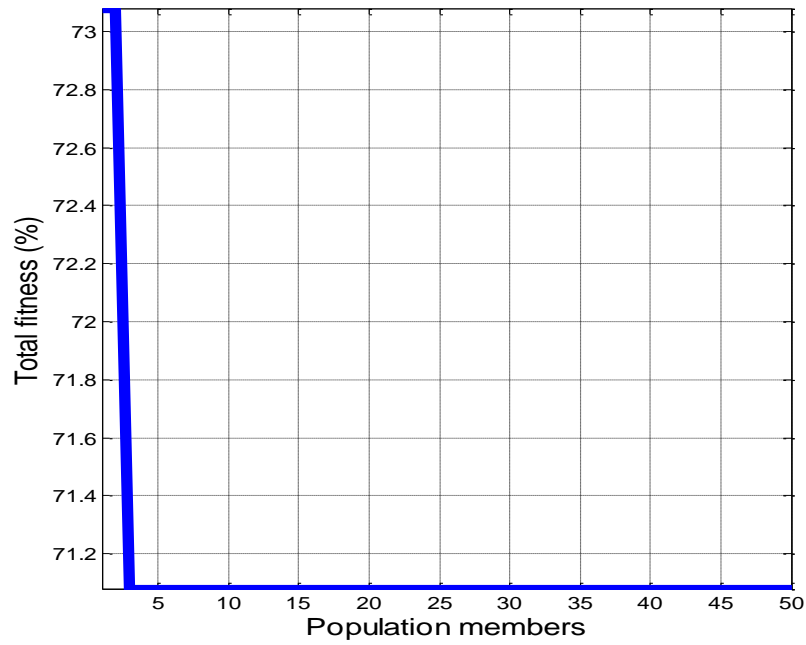


Fig. 35 CGA fitness measure of the last generation in descending order

#### 4.5 Effect of Population Size

In order to ascertain the effect of population size on the performance of the GA we have performed experiments with population size of 500 and generation of 200. The other GA parameters setup is as shown in Table 23 below. We have assumed the same secondary user's QoS requirement in Table 17. The results obtained are as shown in figures 36 to 41.

Table 23 GA Parameters

POPULATION SIZE	500
CROSSOVER RATE	0.6
MUTATION RATE	0.02
NUMBER OF GENERATIONS	200

#### 4.5.1 Convergence: SGA vs. CGA

Figures 36 and 37 show the best fit, average fit and least fit solutions for every generation. Figure 36 shows that the SGA obtained a solution with a fitness measure of 85.64% at the 26<sup>th</sup> generation and that the SGA gets stuck at the 195<sup>th</sup> generation where the diversity of the population is lost. Figure 37 shows that the CGA obtained a solution with a fitness measure of 95.08% at the 11<sup>th</sup> generation and maintains diversity within the population beyond the 200<sup>th</sup> generation. This means that the CGA obtained a more fit solution that is 9.44% better than that obtained by the SGA. Also, the results imply that the CGA converges to the better solution at a rate which is 136% faster than the SGA. Compared to the previous population size of 50 used in section 4.4 and figures 30 to 35, an increase in population size improves the performance of both the SGA and CGA. This is expected since the algorithm has more solutions to choose from.

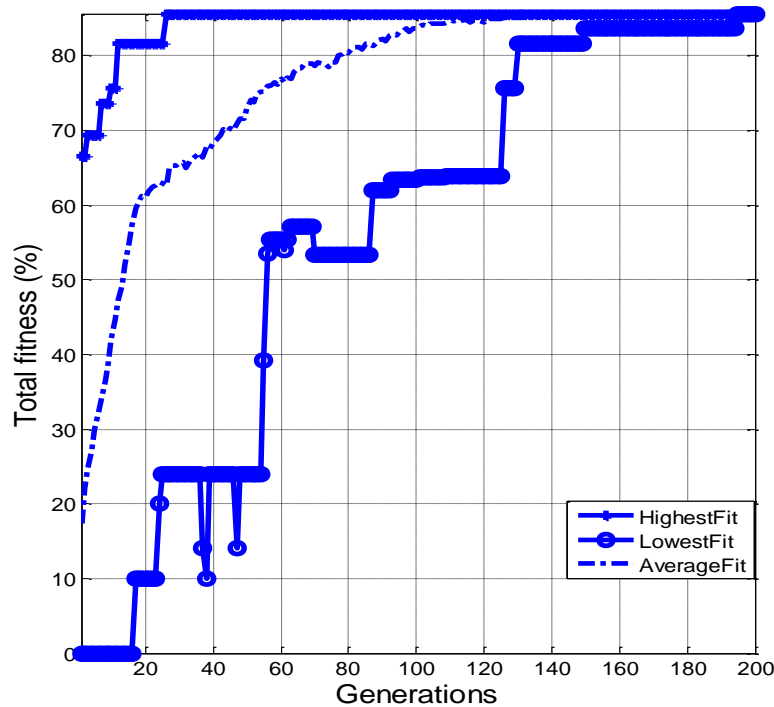


Fig. 36 SGA fitness per generation

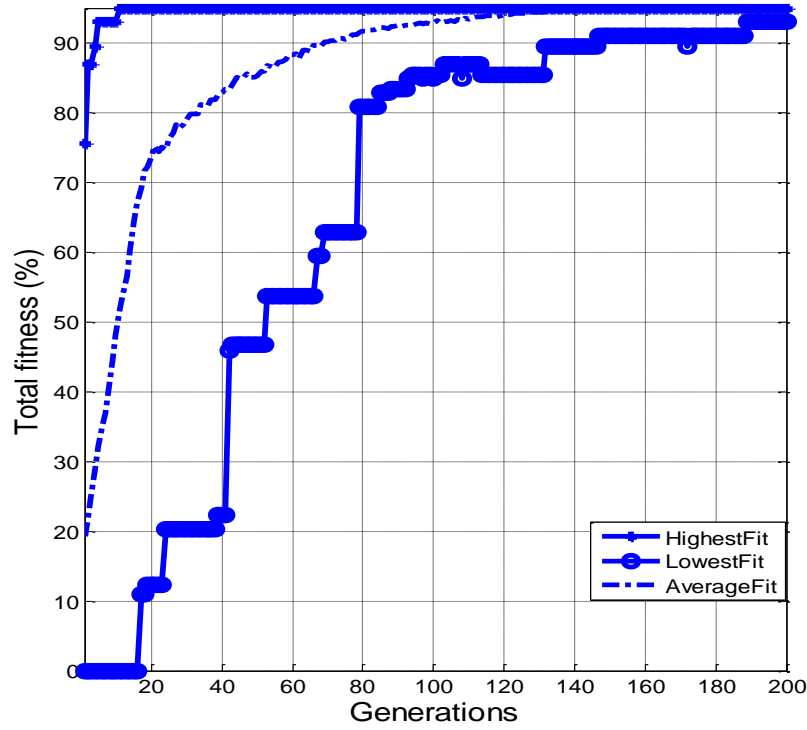


Fig. 37 CGA fitness per generation

#### 4.5.2 Variance: SGA vs. CGA

Figures 38 to 41 show the variance pattern of the two algorithms over the generations. From figure 38 we see that the variance of the SGA decreases from 858.52 at the initial generation to approximately 0 at the 195<sup>th</sup> generation. Thus the SGA loses its diversity at the 130<sup>th</sup> generation at which point we can technically say that the algorithm is stuck. Figure 39 shows that the variance of the CGA also decreases from a much higher value of 1092.7 at the initial generation to approximately 2 at the 123<sup>rd</sup> generation and stays at this value up until the 200<sup>th</sup> generation. Thus, with the CGA, the diversity is not lost until a later generation when compared to the SGA. A plot of the fitness measures of solutions obtained by the SGA in the last generation and arranged in descending order is shown in Fig. 40. The figure indicates that the members of the last generation

have the same fitness measures of 85.64%. This is consistent with what was obtainable in Fig. 38. Figure41 shows the plot of the fitness values of the CGA solutions in the last generation arranged in descending order. The values decrease from 95.08% to 93.08%. It is observed that all of the solutions actually have their fitness values within the said range. This is consistent also with what was obtained in Fig. 39. With a larger population size, the SGA and CGA thus have increased variance at same generation with respect to smaller population size; this is also expected. The results also show that the CGA performs better than the SGA at the increased population size. This can be attributed to the fact that an increased population size will lead to a higher chance of solutions been repeatedly visited by the SGA thus contributing to the loss of diversity in the available solutions. The CGA on the other hand ensures that no two solutions in the search space will be revisited.

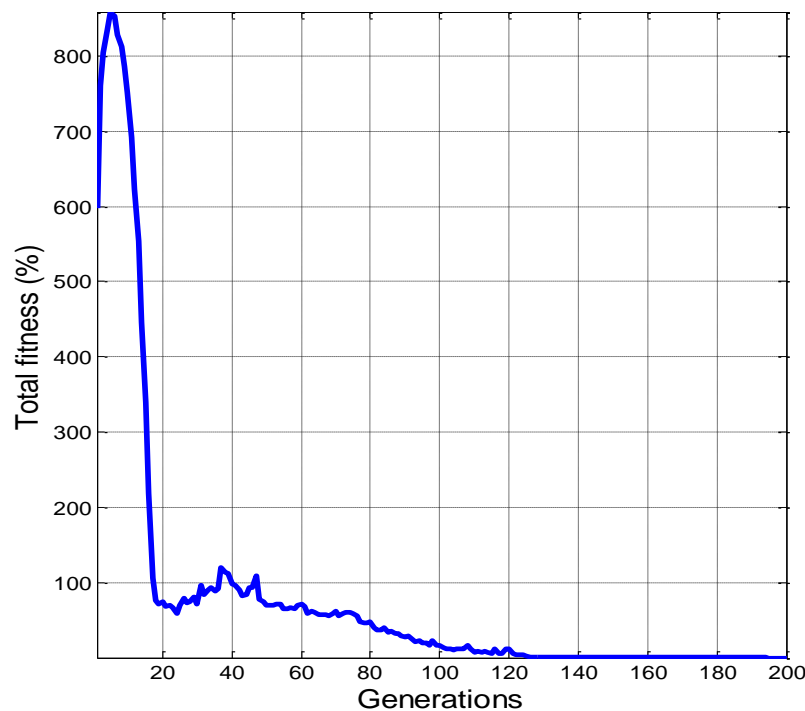


Fig. 38 SGA variance

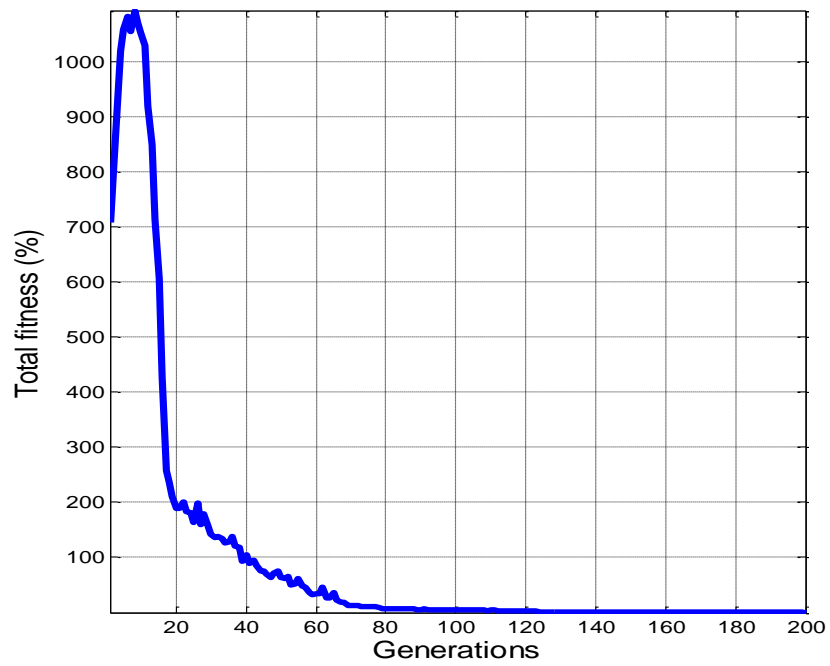


Fig. 39 CGA variance

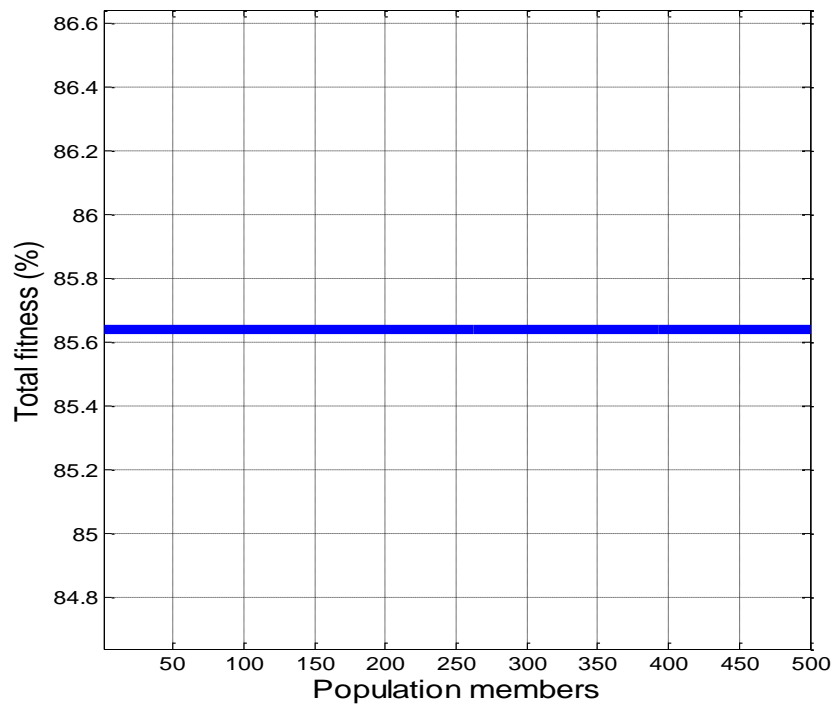


Fig. 40 SGA fitness measure of the last generation in descending order

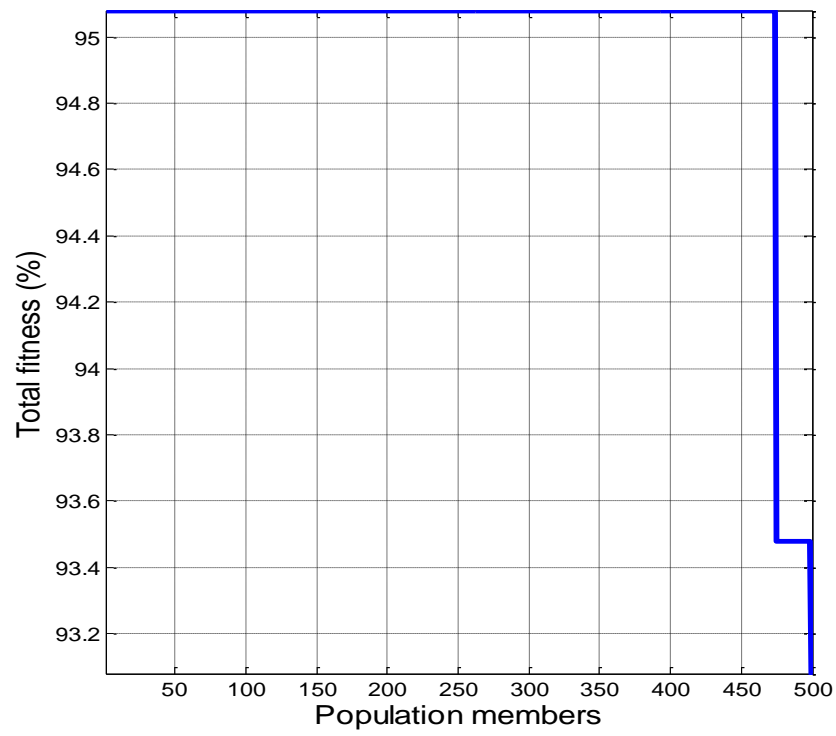


Fig. 41 CGA fitness measure of the last generation in descending order



## **Chapter 5**

### **Conclusion and Future Work**

#### **5.1 Conclusion**

In this thesis, the motivation for using genetic algorithm for combinatorial optimization problems has been presented. The genetic algorithm succeeds in reducing the computational complexity of problems that are either NP-Hard or NP-Complete. It is difficult to seek the optimum solution for problems of this nature because it is only an exhaustive search method that can guarantee the optimum solution requirement, an algorithm which runs in non-polynomial time. In many real world applications the optimum solution can be traded off for the time that a near-optimum (good) solution is obtained. The solution is termed ‘good’ in the sense that it is deemed acceptable enough for the application. The genetic algorithm has been used because it is able to maintain a population of solution at the same time while using its evolutionary processes to increase the diversity of solution points visited in the search space. This characteristic of maintaining a population of solutions aids the convergence of the GA compared to other non-evolutionary algorithms. The mutation property of the GA also reduces the chances of being stuck in the local minima compared to other evolutionary algorithms.

Despite the characteristic properties of the GA, a general problem remains inherent: the quick loss of genetic diversity leading to premature convergence; this is a major drawback of the algorithm. Premature convergence makes it difficult for the algorithm to have a chance of seeking the global optimum. This work examined the various factors and techniques that can be used to improve the diversity of solutions visited by the GA. The selection strategy has a great influence

on the diversity of the solutions; literature reveals that of all the various selection strategies available, the roulette wheel method of selection guarantees diversity the most. Our model therefore makes use of the roulette wheel method of selection as well as the elitism method of selection which ensures that the best fit solution found in any generation is not lost to the evolutionary processes. Several methods have been proposed in literature to solve this problem and incremental performances have been reported. All the methods have focused on increasing the ‘diversity’. However all of these methods have been unable to bypass the inherent problem of randomization in the genetic evolutionary processes. Randomization leads to the revisiting of the same solutions over time which can lead to the algorithms getting stuck in local optimal. This was the motivation for incorporating chaos into the GA evolutionary processes. Each of the random components of the GA is simply replaced with a chaotic component implemented using the logistic map. The characteristics of the logistic map which makes it suitable for this purpose are the sensitivity to initial condition, topological transitivity and topological density. The topological transitivity ensures that no one solution is revisited in the search space while both the sensitivity to initial condition and topological density guides against the algorithm getting stuck in a local optima solution. These characteristics are the reasons behind the improved diversity of the candidate solutions visited by the algorithm. The model has been termed the chaotic GA (CGA) in this work. This model has been reported in literature to improve the convergence of the GA. However in literature there has been no detailed analysis done to verify diversity. In this thesis we have provided the analysis using the variance characteristics of the system over the generations.

## 5.2 Contributions

In this thesis, we have applied the genetic algorithms to two major areas of research: the multiprocessor task scheduling problem and the radio spectrum allocation problem. The goals achieved in this work and our major contributions are detailed below:

- **Task Scheduling in a Multiprocessing System:** The goal of tasks scheduling in a multiprocessor system is to schedule tasks on processors such that the processing time is minimized. By finding the optimal schedule, the system is optimized for performance. Since this problem has been reported to be NP-hard, we have used the GA to search for a good solution in polynomial time. We have verified this through simulations using Java to develop the framework.
- **Literature survey** shows that the method of scheduling based on number of task descendants (NTD) can schedule tasks faster than scheduling based on the heights of the tasks in a task graph. However in this work we found that some tasks can have multiple earlier start time(s) due to multiple paths for which the earliest start time can be obtained. We have assumed that the completion of the execution of one of the parent tasks satisfies the precedence relation in the graph. We have used the principle of ‘open shortest path first’ to ensure that only the minimum of the multiple earliest start times is chosen. This further reduced the makespan of the processing system and thus increased the chances of the GA in obtaining the optimum

schedule. We have verified the effectiveness of this scheduling method through simulations using Java to develop the framework.

- We have also developed a user-friendly software which can generate random task graphs. The user of the software can determine certain parameters of the task graph to be generated. The parameters include: the number of tasks, the number of levels desired in the task graph, the maximum number of descendants possible per task and the range of the random execution time. This software can be used by researchers in related field.
- In this work we have provided equations that can be used to obtain the actual starting time that a task may begin execution on any processor to which it may be allocated, and the updated available processor time at which an allocated task may begin execution. The actual starting time was calculated using the earliest start times and the updated available processor time. The updated available processor time was computed using the earliest start time and the execution time of each task.
- Spectrum Allocation in Cognitive Radio Networks: Since the problem of finding the optimum spectrum allocation for secondary user has been reported to be NP-Complete, we have used the GA to search for a good solution within the search space of possibilities in polynomial time. The GA framework was developed using MATLAB. We verified that indeed

spectrum with percentage fitness of 75 and above can be obtained in polynomial time.

- Unlike previous works done that applied chaos to some of the evolutionary processes of the GA, we have incorporated chaos into all the evolutionary processes of the GA and termed the model chaotic GA (CGA). Simulation results showed that the CGA has the ability to converge faster than the standard GA (SGA) mainly due to the ability of the chaotic process to maintain a better diversity with time compared to the SGA.
- This thesis also provided a detailed analysis of verifying and measuring the diversity of solutions of both the CGA and SGA. Analysis showed that indeed the CGA has a better diversity compared to the SGA.
- We have published three conference papers demonstrating the performance of GA on three different combinatorial optimization problems. The publications are:
  - O. D. Jegede, K. Ferens and W. Kinsner. “A Chaotic Genetic Algorithm for Radio Spectrum Allocation.” in *Proc. Int. Conf. on Genetic and Evolutionary Methods*, Las Vegas, NV USA, pp. 118-125, 2013.
  - O. D. Jegede and K. Ferens. “A Genetic Algorithm for Node Localization in Wireless Sensor Networks.” in *Proc. Int. Conf. on Genetic and Evolutionary Methods*, Las Vegas, NV USA, pp. 126-132, 2013.

- T Kaiser, O.D. Jegede, K. Ferens, D. Buchanan. “A Genetic Algorithm for Multiprocessor Task Scheduling.” in *Proc. Int. Conf. on Genetic and Evolutionary Methods*, Las Vegas, NV USA, pp. 105-110, 2013.

### 5.3 Future work

This work opens up research opportunities applicable to both the CR and GA fields. Some suggested future work includes:

- The primary goal of optimization is to obtain the optimum result, thus the ability of the GA to obtain the optimal result in polynomial time can be further enhanced by combining the CGA with an adaptive adjustment of the algorithm’s parameters (crossover and mutation rate) proposed by Yun-Xiao [46]. This adaptive adjustment has been reported to reduce the vector distance between individual solutions. This should further reduce the convergence time for our proposed CGA.
- The spectrum allocation process of the GA can also be extended to multiple secondary users at a time. The GA can also be improved for smart re-allocation of the spectrum on detection of a primary user.

## References

- [1] M. Mitchell, *An Introduction to Genetic Algorithms*, Cambridge: Massachusetts Institute of Technology, 1999.
- [2] A. E. Eiben and J.E. Smith, *Introduction to Evolutionary Computing*, Germany: Springer, 2003.
- [3] O. D. Jegede and K. Ferens, "A Genetic Algorithm for Node Localization in Wireless Sensor Networks," in *Genetic and Evolutionary Methods*, Las Vegas, 2013.
- [4] F. Wu and N. Vaidya, "SMALL: A Strategy-Proof Mechanism for Radio Spectrum Allocation," in *IEEE International Conference on Computer Communications*, 2011.
- [5] D. Cox and D. Reudink, "Dynamic channel assignment in high capacity mobile communication system," *Bell System Technical Journal*, vol. 50, no. 6, p. 1833–1857, 1971.
- [6] W. Yue, "Analytical methods to calculate the performance of a cellular mobile radio communication system with hybrid channel assignment," *IEEE transactions on vehicular technology*, vol. 40, no. 2, p. 453–460, 1991.
- [7] E. S. Hou, N. Ansari and H. Ren, "A genetic algorithm for multiprocessor scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 2, pp. 113-120, 1994.
- [8] M. Abdeyazdan and A. M. Rahmani, "Multiprocessor task scheduling using a new prioritizing genetic algorithm based on number of task children," in *Distributed and Parallel Systems in Focus: Desktop Grid Computing*, Springer Verlag, 2008, pp. 105-114.
- [9] D. E. Goldberg, *Genetic Algorithm in Search, Optimization, and Machine Learning*, Cambridge, Massachusetts: Addison-Wesley Publishing Company, 1989.
- [10] L. Davis, *Handbook of Genetic Algorithm*, New York: Van Nostrand Reinhold, 1991.
- [11] J. Holland, "Some practical aspects of adaptive systems theory," in *Electronic Information Handling*, A. Kent and O. Taulbee, Eds., Washington, DC: Spartan Press, 1965, p. 209 – 217.
- [12] T. Lau, "Guided Genetic Algorithm," Phd Thesis, Department of Computer Science, University of Essex, Essex, UK, 1997.

- [13] A. Bethke, "Genetic algorithms as function optimizers," Technical Report No. 212, Computer and Communication Sciences, University of Michigan, USA, 1978.
- [14] T. Blickle and L. Thiele, "A comparison of selection schemes used in genetic algorithms," TIK Report, Second Edition, Computer Engineering and Communication Network Labs (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, 1995.
- [15] Wikipedia, "Selection," 2013. [Online]. Available: [http://en.wikipedia.org/wiki/Selection\\_\(genetic\\_algorithm\)](http://en.wikipedia.org/wiki/Selection_(genetic_algorithm)).
- [16] N. Razali and J. Geraghty, "Genetic algorithm performance with different selection strategies in solving TSP," in *World Conference on Engineering*, 2011.
- [17] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Second International Conference on Genetic Algorithms and their Application*, Hillsdale, New Jersey, 1987.
- [18] A. Popov, "Genetic algorithms for optimization," User Manual for the GAmin toolbox for Genetic Algorithms Optimization for MATLAB, 2005.
- [19] L. Eshelman, R. Caruana and J. Schaffer, "Biases in the crossover landscape," in *Third International Conference on Genetic Algorithm*, 1989.
- [20] W. Spears and K. D. Jong, "On the virtues of parameterized uniform crossover," in *Fourth International Conference on Genetic Algorithms*, 1991.
- [21] C. R. a. J. Rowe, *Genetic Algorithms: Principles and Perspectives A Guide to GA Theory*, AA Dordrecht: Kluwer Academic Publishers, 2003.
- [22] D. Zingg, M. Nemec and T. Pulliam, "A comparative evaluation of genetic and gradient-based algorithms applied to aerodynamic optimization," *European Journal of Computational Mechanics/REMNI*, vol. 17, no. 1-2, pp. 103-126, 2008.
- [23] C. Jassadapakorn and P. Chongstitvatana, "Diversity control to improve convergence rate in genetic algorithms," in *Intelligent Data Engineering and Automated Learning, Lecture Notes in Computer Science*, vol. 2690, 2003, pp. 421-425.
- [24] R. Povinelli and X. Feng, "Improving genetic algorithms performance by hashing fitness values," in *Artificial Neural Networks in Engineering*, St. Louis, Missouri, 1999.



- [25] T. Fogarty, "Varying the probability of mutation in the genetic algorithm," in *Third International Conference on Genetic Algorithms*, 1989.
- [26] J. Hesser and R. Manner, "Towards an optimal mutation probability in genetic algorithms," in *First Parallel Problem Solving from Nature*, 1991.
- [27] T. Back and M. Schutz, "Intelligent mutation rate control in canonical genetic algorithms," in *International Symposium on Methodologies for Intelligent Systems*, 1996.
- [28] G. Ochoa, I. Harvey and H. Buxton, "On recombination and optimal mutation rates," in *Genetic and Evolutionary Computation Conference*, 1999.
- [29] T. Jansen and I. Wegener, "On the choice of the mutation probability for the (1+1) EA," in *6th Parallel Problem Solving from Nature*, 2000.
- [30] D. Thierens, "Adaptive mutation rate control schemes in genetic algorithms," in *Proceedings of IEEE International Conference Evolutionary Computation*, 2002.
- [31] V. Kureichick, A. Melikhov, V. Miaghick, O. Savelev and A. Topchy, "Some new features in the genetic solution of the traveling salesman problem," in *ACEDC'96, 2nd International Conference of the Integration of Genetic Algorithms and Neural Network Computing and Related Adaptive Computing with Current Engineering Practice*, Plymouth, UK, 1996.
- [32] M. Rocha and J. Neves, "Preventing premature convergence to local optima in genetic algorithms via random offspring generation," in *12th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Multiple Approaches to Intelligent Systems*, Cairo, Egypt, 1999.
- [33] M. Angelova and T. Pencheva, "Tuning genetic algorithm parameters to improve convergence time," *International Journal of Chemical Engineering*, vol. 2011, no. 646917, pp. 1-7, 2011.
- [34] M. Angelova, S. Tzonkov and T. Pencheva, "Genetic algorithms based parameter identification of yeast fed-batch cultivation," in *Conference on Numerical Methods and Applications*, vol. 6046 of *Lecture Notes in Computer Science*, 2011.
- [35] K. P. Wong and A. Li, "A technique for improving the convergence characteristic of genetic algorithms and its application to a genetic-based load flow algorithm," in *Simulated*

*Evolution and Learning. Lecture Notes in Artificial Intelligence*, Springer, 1997, pp. 167-176.

- [36] Y. Sun and F. Deng, "Chaotic parallel genetic algorithm with feedback mechanism & its application in complex constrained problem," in *IEEE Conference on Cybernetics and Intelligent Systems*, 2004.
- [37] W. Kinsner, "Fractal and chaos engineering," Dept. Electrical and Computer Eng., Univ. of Manitoba., Winnipeg, 2010.
- [38] R. Devaney, *An introduction to chaotic dynamical systems*, Reading, Massachusetts: Benjamin/Cummings, 1986, p. 320.
- [39] R. Devaney, *A first course in chaotic systems: theory and experiments*, Reading, Massachusetts: Addison-Wesley, 1992, p. 302.
- [40] Wikipedia, "Ergodicity," 2013. [Online]. Available: <http://en.wikipedia.org/wiki/Ergodicity>.
- [41] Z. Yun-Xiao, Z. Jie and Z. Chang-Chang, "Cognitive radio resource allocation based on coupled chaotic genetic algorithm," *IOP Science Chinese Physics B*, vol. 19, no. 11, pp. 119501-1 - 119501-8, 2010.
- [42] C. Min-Yuan and H. Kuo-Yu, "K-means clustering and chaos genetic algorithm for nonlinear optimization," in *The 26th International Symposium on Automation and Robotics in Construction (ISARC)*, 2009.
- [43] D. Cook, K. Ferens and W. Kinsner, "Application of chaotic simulated annealing in the optimization of task allocation in a multiprocessing system," in *IEEE International Conference on Cognitive Informatics and Cognitive Computing*, 2013.
- [44] D. Shaw and W. Kinsner, "Chaotic simulated annealing in multilayer feedforward networks," in *Canadian Conference on Electrical and Computer Engineering*, 1996.
- [45] J. Mingjun and T. Huanwen, "Application of chaos in simulated annealing," *Chaos, Solitons & Fractals*, vol. 21, no. 4, pp. 933-941, 2004.
- [46] H. Meng, P. Zheng, R. Wu, X. Hao and Z. Xie, "A hybrid particle swarm algorithm with embedded chaotic search," in *IEEE Conference on Cybernetics and Intelligent Systems*, 2004.
- [47] M. Pinedo, *Scheduling - Theory, Algorithms, and Systems*, New York: Springer, 2008.

- [48] N. Fisher, "The multiprocessor real-time scheduling of general task systems," Phd Thesis, Department of Computer Science, University of North Carolina, USA, 2007.
- [49] S. Jin, G. Schiavone and D. Turgut, "A performance study of multiprocessor task scheduling algorithms," *Journal of Supercomputing*, vol. 43, no. 1, pp. 77-97, 2008.
- [50] M. U, C. Ho, S. Funk and K. Rasheed, "GART: A genetic algorithm based real-time system scheduler," in *IEEE Congress on Evolutionary Computation*, 2011.
- [51] R. M. Miryani and M. Naghibzadeh, "Hard real-time multiobjective scheduling in heterogenous systems using genetic algorithms," in *International CSI Computer Conference*, 2009.
- [52] D. Montana, M. Brinn, G. Bidwell and S. Moore, " Genetic algorithms for complex, real-time scheduling," in *IEEE Conference on Systems, Man, and Cybernetics*, 1998.
- [53] Y. Monnier, J. Beauvais and A. Deplanche, "A genetic algorithm for scheduling tasks in a real-time distributed system," in *Euromicro Conference*, 1998.
- [54] A. S. Wu, H. Yu, S. Jin, K.-C. Lin and G. Schiavone, "An incremental genetic algorithm approach to multiprocessor scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 9, pp. 824 - 834, 2004.
- [55] L. Doyle, *Essentials of cognitive radio*, New York: Cambridge University Press, 2009.
- [56] The SDR Forum, "Cognitive radio definitions and nomenclature," 2008.
- [57] B. Fette, "History and background of cognitive radio technology," in *Cognitive Radio Technology*, Burlington, Elsevier Inc, 2009, pp. 1-26.
- [58] T. Siddique and A. Azam, "Spectrum optimization in cognitive radio networks using genetic algorithms," Master's Thesis, Blenkinge Institute of Technology, Sweden, 2010.
- [59] F. Bruce, "Introducing adaptive, aware, and cognitive radios," in *Cognitive Radio, Software Defined Radio, and Adaptive Wireless Systems*, AA Dordrecht, The Netherlands, Springer, 2007, pp. 1-16.
- [60] A. Vogel, B. Kerherve, G. v. Bochmann and J. Gecsei, "Distributed multimedia and qos: a survey," *IEEE Multimedia*, vol. 2, 1995.

- [61] M. Kaur and M. Uddin, "Optimization of QoS parameters in cognitive radio using adaptive genetic algorithm," *International Journal of Next-Generation Networks (IJNGN)*, vol. 4, no. 2, pp. 1-15, 2012.
- [62] J. Hauris, "Genetic algorithm optimization in a cognitive radio for autonomous vehicle communications," in *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Jacksonville, FL., 2007.
- [63] T. Rondeu, C. Rieser, B. Le and C. Bostain, "Cognitive radios with genetic algorithms: intelligent control of software defined radios," in *SDR Forum Technical Conference*, Phoneix, FL, 2004.
- [64] M. Withall, C. Hinde, R. Stone and J. Cooper, "Packet transmission optimization using genetic algorithms," in *Lecture Notes in Computer Science*, Berlin/Heidelberg, Springer, 2003, pp. 119-138.
- [65] T. Newman, R. Rajbanshi, A. Wyglinski, J. Evans and G. Minden, "Population adaptation for genetic algorithm based cognitive radios," in *IEEE 2nd International Conference on Cognitive Radio Oriented Wireless Networks and Communications*, Orlando, FL., 2007.
- [66] O. Hasancebi and F. Erbatur, "Evaluation of crossover techniques in genetic algorithm based optimum structural design," *Computer and Structures*, vol. 78, no. 1-3, p. 435 – 448, 2000.

## Appendix A

### Software

This chapter shows the task graph generator software graphical interface which was developed to generate the task graph. The details from the task graph such as task number, height, execution time and earliest start times are then fed into the GA for the purpose of obtaining a good schedule for the multi-processor system. Figure A1 shows the graphical interface of the task graph generator.

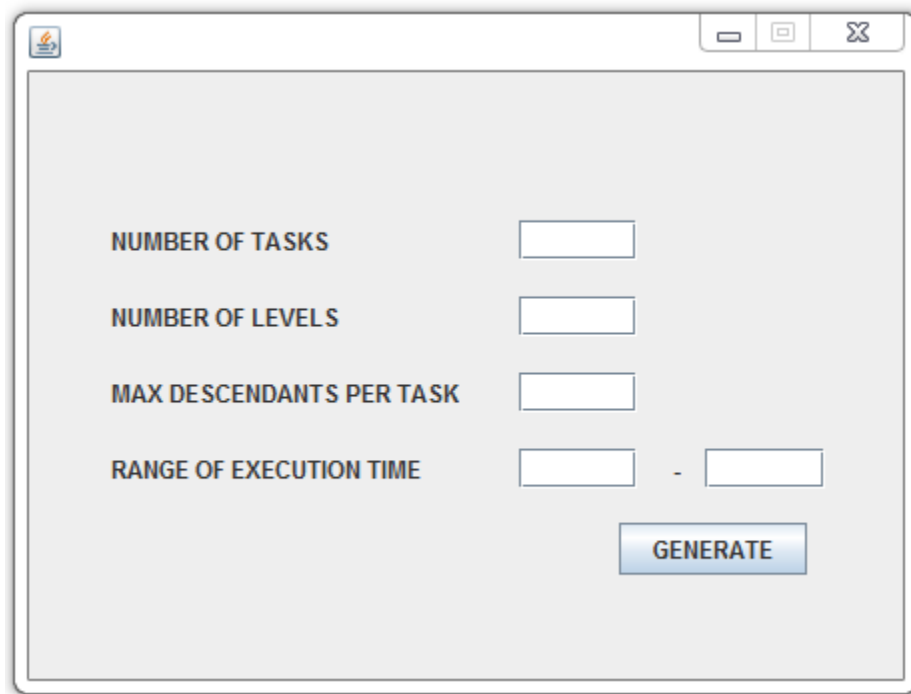
The image shows a graphical user interface for a task graph generator. It is a window with a standard title bar containing minimize, maximize, and close buttons. The main area is light gray and contains four input fields with labels to their left: 'NUMBER OF TASKS', 'NUMBER OF LEVELS', 'MAX DESCENDANTS PER TASK', and 'RANGE OF EXECUTION TIME'. The first three labels are in all caps. The 'RANGE OF EXECUTION TIME' label is followed by two input boxes separated by a hyphen. Below these inputs is a blue button with the word 'GENERATE' in white capital letters.

Fig. 1 Task graph generator

In this graphical interface, the users can specify the parameters of the task graph. The “Number of Tasks” refers to the set number of tasks in the task graph. We have limited the range of the possible number of tasks to 50; this is in order to ensure that the window size of the interface can accommodate the task graph diagram. The “Number of Levels” refers to the number of height in the task graph according to literature. The “Max Descendants per Tasks” refers to the highest possible number of

descendants that a task can have. The “Range of Execution time” refers to the range within which the execution time of each task will fall. For example, the user can fill in “0” – “15”; there is no set limit the values that can be chosen. After the input parameters have been filled in, the user can click on the “Generate” button to generate the task graph.

### **A.1 Running the Task Graph Generator**

1. Open new configuration files in any java emulator environment and paste each of the codes B.1, B.2 and B.3 listed in Appendix B.
2. Compile each of the file.
3. Run each of the code with the code B.3 the last to be run; the task graphical interface pops up.
4. Input the parameters and generate the task graph. Note that entering the same set of parameters every time will generate different task graphs because it is random.

### **A.2 Running the Task Scheduling GA Code**

1. Open the TaskScheduling\_GA.java file in any java emulator environment.
2. Ensure that you have transferred the details of the chosen task graph into the “tasks.txt” file before compiling the code.
3. Compile and Run the code.
4. In this experiment we have 6 different .txt files with different number of tasks and tasks dependencies. These are listed in B.4. The ‘tasks\_16.txt’ has 16 tasks and a task dependency of 20%; the tasks\_21.txt has 21 tasks and a task dependency of 20%; and the tasks\_30.txt has 30 tasks and a task dependency of 20%. The tasks\_20%\_30.txt has 30 tasks and a task dependency of 20%. The tasks\_40%\_30.txt has 30 tasks and a task dependency of 40%. The tasks\_60%\_30.txt has 30 tasks and a task dependency of 60%.

5. For each of these .txt files, compile and run the code.

### **A.3 Running the Spectrum Allocation Code**

1. The GA-based spectrum allocation experiment was implemented using MATLAB. There are two versions: the SGA and the CGA. These two versions have some files (code) in common while some files are exclusive.
2. To run the SGA version:
  - a. The files in B.5 should all be placed in the same folder (location).
  - b. The main file “Main\_SGA.m” should be opened and run.
  - c. Set the QoS input parameters as that in Table 17 of chapter 4 which was used for the experiment.
  - d. The text version of the results will be displayed in the command window while the graphical results will be displayed on different plotted graphs.
3. To run the CGA version:
  - a. The files in B.6 should all be placed in the same folder (location).
  - b. The main file “Main\_CGA.m” should be opened and run.
  - c. Set the QoS input parameters as that in Table 17 of chapter 4 which was used for the experiment.
  - d. The text version of the results will be displayed in the command window while the graphical results will be displayed on different plotted graphs.

## Appendix B

### Source Code Files

The file name of the source codes are listed in B.1 to B.6.

#### B.1 Task....

Display.java;

#### B.2 Task...

InputForm.java;

#### B.3 Task

TaskGraph.java;

#### B.4 Task

TaskScheduling\_GA.java;

tasks.txt

tasks\_16.txt

tasks\_21.txt

tasks\_30.txt

tasks\_20%\_30.txt

tasks\_40%\_30.txt

tasks\_60%\_30.txt

#### B.5 Spectrum Allocation

Main\_SGA.m;

initial\_population.m;

selection\_operation.m;

crossover\_operation.m;

mutation\_operation.m;

fitness\_measure.m;

#### B.6 Spectrum Allocation

Main\_CGA.m;

initial\_population\_chaotic.m;

selection\_operation.m;

crossover\_operation\_chaotic.m;

mutation\_operation\_chaotic.m;

fitness\_measure.m;



## Appendix C

### Application of GA to Localization in Wireless Sensor Networks

We have applied the GA to node localization in wireless sensor networks (WSN). WSN can be implemented for the purpose of monitoring the environment for agricultural and security purposes. A wireless sensor network is a collection of nodes organised into a cooperative network. The ability to detect the locations of each wireless sensor in a WSN is central to accurate information gathering. Example of conventional location detection techniques include the global positioning system (GPS) and infrared; they are however very expensive to deploy within a very large sensors network. In general, the localization problem has been shown to be NP-hard [3]. For very sensitive WSN applications such as in military operations, it is important to detect the location of the sensor nodes in polynomial time. Stochastic processes have been known to reduce the computational complexity involved in the localization problem. Therefore, we have proposed the use of a GA to learn the environment impairments within a 5m by 5m grid wireless sensor networks for the purpose of localization for data management within the network.

For each coordinate in the grid network area, random perturbations of received signal strength (RSS) from sensor nodes were obtained and supplied to the GA. The RSS obtained from a next-hop sensor node is used to determine the location of the sensor node. In our work [3], we have made use of a one-hop connection where a sensor node is directly connected to each of three anchor nodes. The anchor nodes are equipped with GPS capability. Since the sensor nodes are within transmission range of each of the anchors, the signal strength of the sensor nodes received at each of the anchor is used to find the coordinate of the sensor nodes in the network. The relationship between the transmit power of

a sensor node and the corresponding RSS at the anchor nodes is used to compute the possible location of the node within the network. The methodology and GA approach to this problem is described in details in our work [3]. Simulation was done in MATLAB to determine the location of the sensor nodes using the GA. The results obtained for five sensors (targets) showed that after 100 generations of the GA, the average error between the actual sensors location and the GA estimated locations was approximately 0.01. This is good for practical purpose in a wireless sensors network. The results are shown in tables I to IV and plotted in figures I to VI.

Each table shows the error in the predicted location of the GA for each target positions as well as the relative percentage error to the grid. Table I shows the results in the GA predictions for each target at each of the target's initial positions in the grid. Table II shows the results in the GA predictions for each target when each of the target's initial positions is increased by 10cm. Table III shows the results in the GA predictions for each targets when each of the target's initial positions is increased by 20cm. Table IV shows the results in the GA predictions for each targets when each of the target's initial positions is increased by 30cm.

Figure I shows the error in GA predictions for target 1 with increased distance. Figure II also shows the error in GA predictions for target 2 with increased distance. The error in GA predictions for target 3 with increased distance is shown by figure III. Figure IV shows the error in GA predictions for target 4 with an increased distance. The error in GA predictions for target 5 with increased distance is shown by figure V. The average error which shows the difference between the GA predicted results and the expected ones is shown in Fig VI. It is observed that the error decreases as the number of generations increases. At around the 33<sup>rd</sup> generations, the GA was able to converge to a good solution with an error of approximately 0.01. This is a practically acceptable result.

Table I

GA PREDICTED	ACTUAL Targets' Initial Positions	ERROR in GA Prediction	Percentage Error (Relative to total Grid )
(2.47, 1.35)	(2.5, 1.4)	0.05	1.25
(1.16,2.37)	(1.2, 2.4)	0.05	1.25
(0.4,2.96)	(0.4,3.0)	0.04	1
(2.51,3.23)	(2.5,3.2)	0.05	1.25
(3.54,3.56)	(3.6,3.5)	0.08	2

Table II

ACTUAL Targets' Initial Positions + 10cm	ERROR in GA Prediction	Percentage Error (Relative to total Grid )
Target 1	0.212	5.3
Target 2	0.191	4.78
Target 3	0.108	2.7
Target 4	0.095	2.38
Target 5	0.165	4.125

Table III

ACTUAL Targets' Initial Positions + 20cm	ERROR in GA Prediction	Percentage Error (Relative to total Grid )
Target 1	0.340	8.5
Target 2	0.332	8.3
Target 3	0.312	7.8
Target 4	0.275	6.88
Target 5	0.295	7.38

Table IV

ACTUAL Targets' Initial Positions + 30cm	ERROR in GA Prediction	Percentage Error (Relative to total Grid )
Target 1	0.481	12.03
Target 2	0.474	11.85
Target 3	0.453	11.33
Target 4	0.396	9.9
Target 5	0.433	10.825

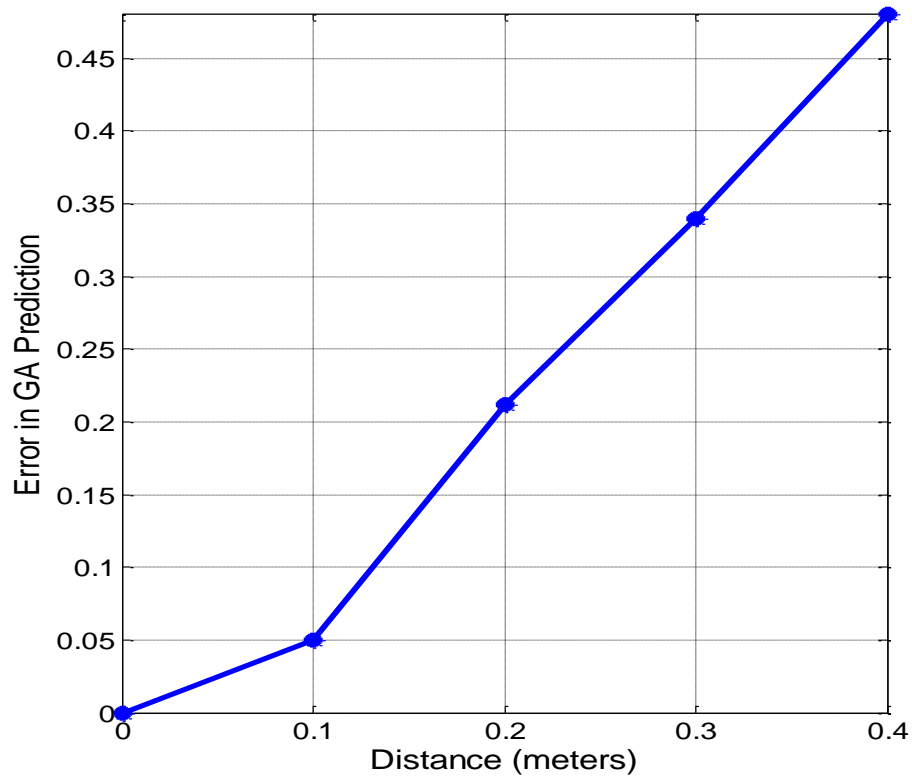


Fig. I Target I error with distance

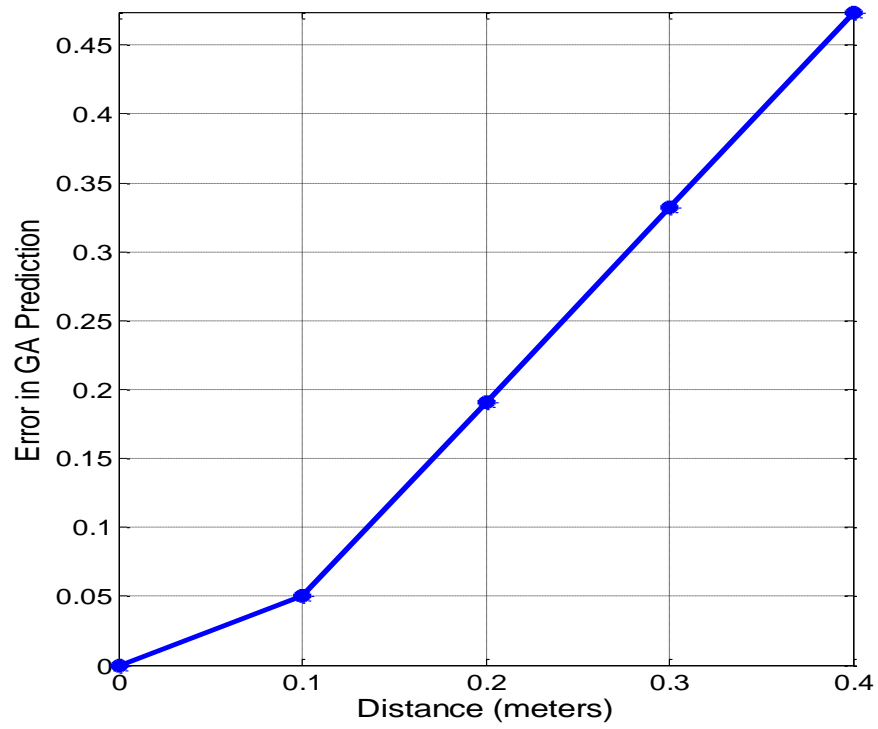


Fig. II Target II error with distance

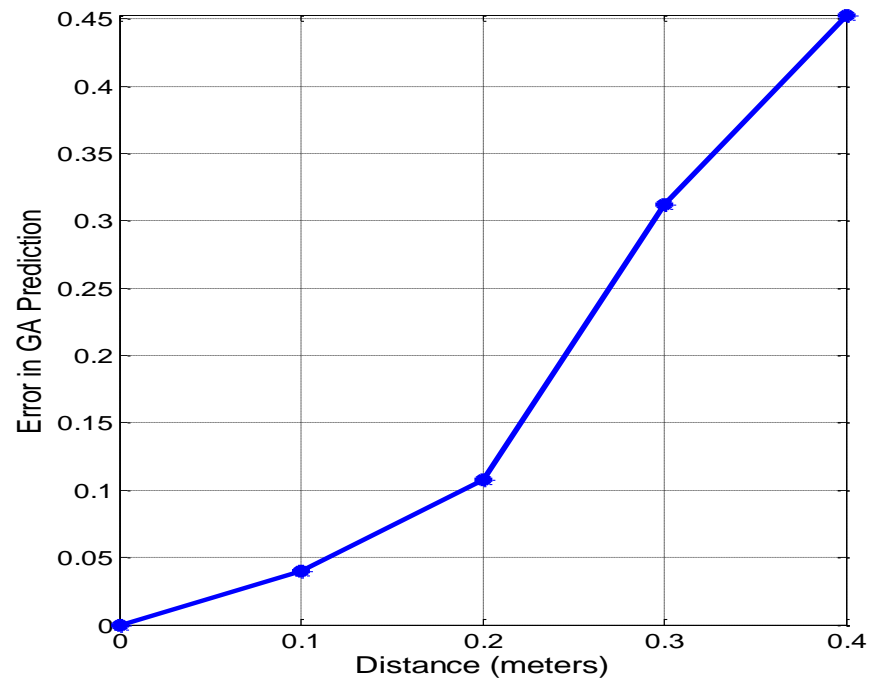


Fig. III Target III error with distance

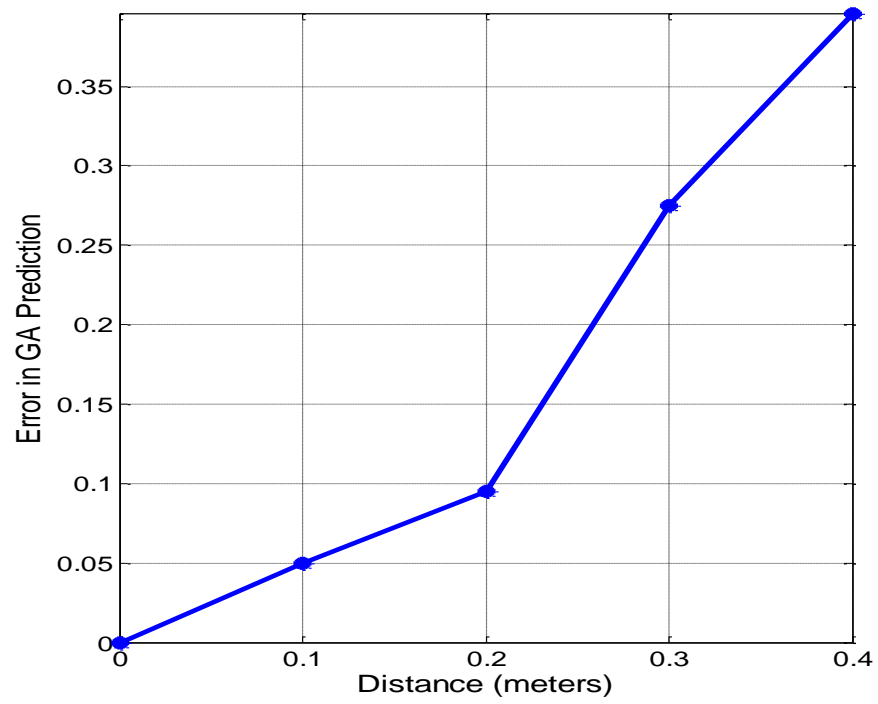


Fig. IV Target IV error with distance

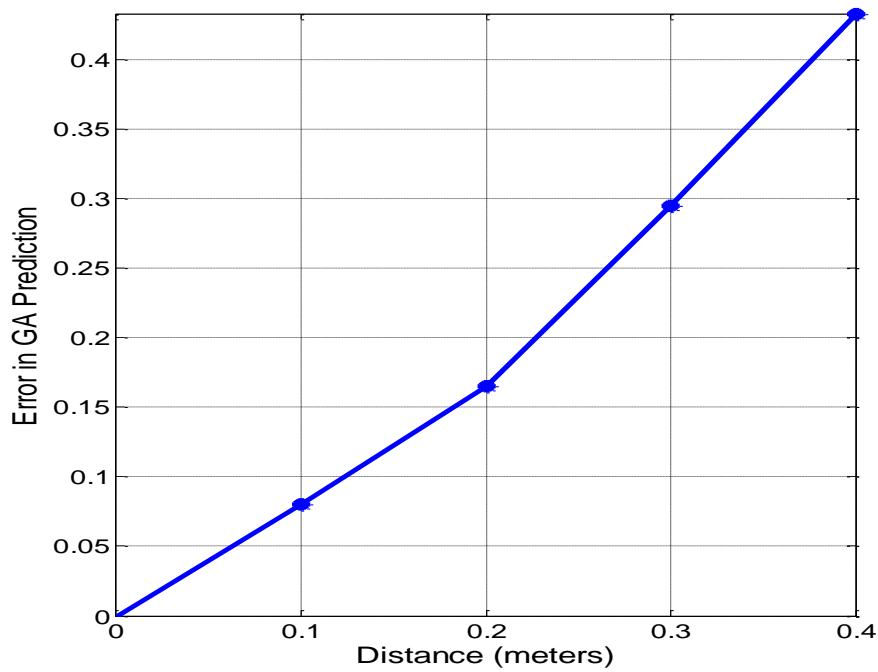


Fig. V Target V error with distance

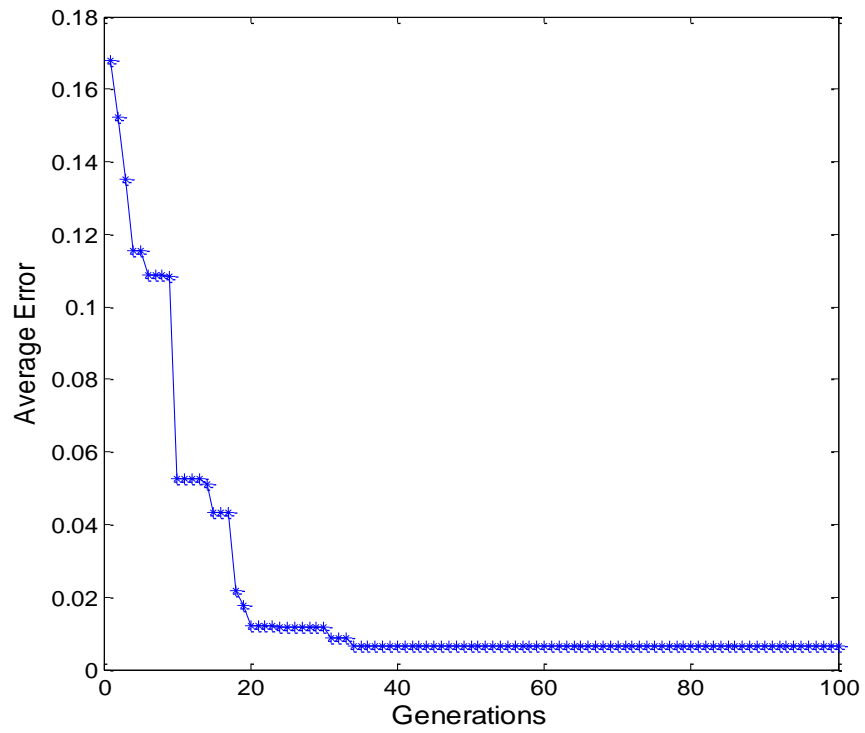


Fig. VI Average error for the 5 target nodes per generation