THE UNIVERSITY *of* EDINBURGH

Dynamic and Fault Tolerant Three-Dimensional Cellular Genetic Algorithms

Asmaa Al-Naqi

A thesis submitted for the degree of Doctor of Philosophy
The University of Edinburgh
June 2012

Declaration of Originality

I hereby declare that this thesis and all the work included herein was conducted and originated entirely by myself, in the School of Engineering at The University of Edinburgh.

Asmaa Al-Naqi

June, 2012

Acknowledgement

It is a pleasure to thank those who made this thesis possible; I would like to acknowledge the advice and guidance of Professor Tughrul Arslan. I also thank Dr. Ahment Erdogan for his advice and suggestions.

This thesis was examined by Professor Klaus Mcdonald-Maier (University of Essex) and Dr. Alister Hamilton (University of Edinburgh). Special thanks go to both.

I acknowledge the Public Authority for Applied Education and Training (PAAET) for their financial support for this project.

I would like to thank my family members, especially my husband, Mohammad Al-Harz for supporting and encouraging me to pursue this degree. Without my husband's encouragement, I would not have finished the degree.

I would like to grant this thesis to my family for their support and encouragement.

Abstract

In the area of artificial intelligence, the development of Evolutionary Algorithms (EAs) has been very active, especially in the last decade. These algorithms started to evolve when scientists from various regions of the world applied the principles of evolution to algorithmic search and problem solving. EAs have been utilised successfully in diverse complex application areas. Their success in tackling hard problems has been the engine of the field of Evolutionary Computation (EC). Nowadays, EAs are considered to be the best solution to use when facing a hard search or optimisation problem.

Various improvements are continually being made with the design of new operators, hybrid models, among others. A very important example of such improvements is the use of parallel models of GAs (PGAs). PGAs have received widespread attention from various researchers as they have proved to be more effective than panmictic GAs, especially in terms of efficacy and speedup.

This thesis focuses on, and investigates, cellular Genetic Algorithms (cGAs)–a competitive variant of parallel GAs. In a cGA, the tentative solutions evolve in overlapped neighbourhoods, allowing smooth diffusion of the solutions. The benefits derived from using cGAs come not only from flexibility gains and their fitness to the objective target in combination with a robust behaviour but also from their high performance and amenability to implementation using advanced custom silicon chip technologies. Nowadays, cGAs are considered as adaptable concepts for solving problems, especially complex optimisation problems. Due to their structural characteristics, cGAs are able to promote an adequate exploration/exploitation trade-off and thus maintain genetic diversity. Moreover, cGAs are characterised as being massively parallel and easy to implement.

The structural characteristics inherited in a cGA provide an active area for investigation. Because of the vital role grid structure plays in determining the effectiveness of the algorithm, cellular dimensionality is the main issue to be investigated here. The implementation of cGAs is commonly carried out on a one- or two-dimensional structure. Studies that investigate higher cellular dimensions are lacking. Accordingly, this research focuses on cGAs that are implemented on a three-dimensional structure. Having a structure

with three dimensions, specifically a cubic structure, facilitates faster spreading of solutions due to the shorter radius and denser neighbourhood that result from the vertical expansion of cells. In this thesis, a comparative study of cellular dimensionality is conducted. Simulation results demonstrate higher performance achieved by 3D-cGAs over their 2D-cGAs counterparts. The direct implementation of 3D-cGAs on the new advanced 3D-IC technology will provide added benefits such as higher performance combined with a reduction in interconnection delays, routing length, and power consumption.

The maintenance of system reliability and availability is a major concern that must be addressed. A system is likely to fail due to either hard or soft errors. Therefore, detecting a fault before it deteriorates system performance is a crucial issue. Single Event Upsets (SEUs), or soft errors, do not cause permanent damage to system functionality, and can be handled using fault-tolerant techniques. Existing fault-tolerant techniques include hardware or software fault tolerance, or a combination of both. In this thesis, fault-tolerant techniques that mitigate SEUs at the algorithmic level are explored and the inherent abilities of cGAs to deal with these errors are investigated. A fault-tolerant technique and several mitigation techniques are also proposed, and faulty critical data are evaluated critical fault scenarios (stuck at '1' and stuck at '0' faults) are taken into consideration. Chief among several test and real world problems is the problem of determining the attitude of a vehicle using a Global Positioning System (GPS), which is an example of hard real-time application. Results illustrate the ability of cGAs to maintain their functionality and give an adequate performance even with the existence of up to 40% errors in fitness score cells.

The final aspect investigated in this thesis is the dynamic characteristic of cGAs. cGAs, and EAs in general, are known to be stochastic search techniques. Hence, adaptive systems are required to continue to perform effectively in a changing environment, particularly when tackling real-world problems. The adaptation in cellular engines is mainly achieved through dynamic balancing between exploration and exploitation. This area has received considerable attention from researchers who focus on improving the algorithmic performance without incurring additional computational effort.

The structural properties and the genetic operations provide ways to control selection pressure and, as a result, the exploration/exploitation trade-off. In this thesis, the genetic operations of cGAs, particularly the selection aspect and their influence on the search process, are investigated in order to dynamically control the exploration/exploitation trade-off. Two adaptive-dynamic techniques that use genetic diversity and convergence speeds to guide the search are proposed. Results obtained by evaluating the proposed approaches on a

test bench of diverse-characteristic real-world and test problems showed improvement in dynamic cGAs performance over their static counterparts and other dynamic cGAs. For example, the proposed Diversity-Guided 3D-cGA outperformed all the other dynamic cGAs evaluated by obtaining a higher search success rate that reached to 55%.

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

1D	One Dimension
2D	Two Dimensions
2D-cGA	Two-Dimensional Cellular Genetic Algorithm
3D	Three Dimensions
3D-cGA	Three-Dimensional Cellular Genetic Algorithm
AFM	Ambiguity Function Method
ANOVA	Analysis of Variance
BT	Binary Tournament
CA	Cellular Automata
cEA	Cellular Evolutionary Algorithm
cGA	Cellular Genetic Algorithm
CGA	Genetic Algorithm Implemented on Cellular Automata
CR	Convergence Rate
CT	Convergence Time
dcGA	Distributed Cellular Genetic Algorithm
dGA	Distributed Genetic Algorithm
dssGA	Distributed Steady-State Genetic Algorithm
EA	Evolutionary Algorithm
EC	Evolutionary Computation
EHW	Evolvable Hardware
FMS	Frequency Modulation Sound
FT 3D-cGA	Fault-Tolerant Three-Dimensional Cellular Genetic Algorithm
GA	Genetic Algorithm
genGA	Generational Genetic Algorithm
GPS	Global Positioning System
IC	Integrated Circuits
MBU	Multiple Bit Upset
MISD	Multiple Instructions Single Data

MIMD	Multiple Instruction Multiple Data
NEWS	Von Neumann Neighbourhood
NGR	Neighbourhood to Grid Ratio
PE	Processing Element
PGA	Parallel Genetic Algorithm
SEE	Single Event Effect
SEU	Single Event Upset
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SLE	System of Linear Equations
SP	Speed of Convergence
ssGA	Steady State Genetic Algorithm
ST	Stochastic Tournament

Chapter 1

Introduction

Evolutionary computation is an area of artificial intelligence that is concerned with solving computational problems through the use of ideas from biological evolution. Computational problems commonly involve searching a massive potential solution space to find the best solution. In the real world, such problems require complex solutions that are usually too difficult to solve using traditional techniques. Nowadays, the dramatic increase in computer power enables many practical applications to become reality. Accordingly, evolutionary algorithms are efficiently used to optimise the design of systems and to solve high-dimensional problems. Since the nineteen-nineties, EAs have increasingly become a crucial part of system design and implementation.

The actualization of Darwinian principles, which later evolved into evolutionary computation, began in the nineteen-fifties for automated problem solving. However, the idea started to take root in different areas in the nine-sixties when the first two methods of evolutionary algorithms (the approach of evolutionary computation) were proposed. In the USA, Lawrence Fogel proposed evolutionary programming, while in Germany, Ingo Rechenberg and Hans-Paul Schwefel proposed evolution strategies. In the nine-seventies, John Holland introduced Genetic Algorithms (GAs), which has become the most widely used manifestation nowadays, and is the main focus of this research.

Genetic Algorithms are one of the most powerful tools for efficiently solving complex problems in different application areas. Genetic Algorithms search for the optimum solution among a large number of possible solutions, encoded as gene sequences (chromosomes), by allowing these organisms to survive and produce (evolve) in their environments. The evolution process occurs though random variation, crossover, and mutation operators. Following that, natural selection occurs and plays the vital role of enabling the fittest

chromosome to survive and reproduce. As a result, new genetic materials are produced to form a new population of potential solutions, which propagate to successive generations. This process stops in accordance with a predefined termination criteria, such as reaching a specific number of generations and/or finding the desired solution (Goldberg, 1989).

Standard GA models are inherently parallel; however, they require frequent communication, which is based on centralised control. Accordingly, parallel GA models were investigated with a view towards making parallel models efficient. The parallelisation of GA models occurs at either the computation or the population level. In the former, the operations applied to each encoded solution (individual) are performed in parallel. Master-slave GAs are examples of such a model. In the latter, the population is divided into subpopulations of coarse or fine grain size such that each subpopulation evolves in parallel. When implementing the GA model, these subpopulations are distributed over a selected grid structure, which in turn defines the ways in which each subpopulation interacts with others. Therefore, different parallel GA classes can be formed, and the main aim of all is to enhance the speedup and efficiency of the search (Cantu-Paz, 2000).

In fine-grained or cellular GAs (cGAs), the population is divided into a massive number of subpopulations, each consisting of one encoded solution (individual). The individuals are distributed over an *n*-dimensional grid structure with wraparound edges (toroidal). Research surroundings cGAs are more commonly concerned with their implementation on one- or two-dimensional grid topology. Therefore, the interactions between the individuals occur within their local neighbourhood. In coarse-grained or distributed GAs (dGAs), the population is divided into several subpopulations, each consisting of a number of individuals. Each subpopulation evolves in isolation from others, and the interaction between the subpopulation occurs according to the employed migration policy (Cantu-Paz, 1995). This research is concerned with the cGA models.

1.1 Motivations

The topology of the grid is the key in determining the performance of GAs. Different topologies induce different levels of exploration or diversification of the search space and exploitation or intensification of good solutions. Hence, the appropriate balance between exploration and exploitation is an important issue in the rapid identification of promising regions with high quality solutions in the search space. A typical cGA is implemented on a two-dimensional toroidal grid topology; research concerned with higher dimensional

topologies is very limited. Preliminary research on cellular dimensionality has shown promising results. Consequently, exploring and investigating the implementation of a cGA model on higher cellular dimensions, specifically three-dimensions (3D), is one of the motivations underlying this research.

Another motivation relates to the field of fault tolerance. Evolutionary optimisation engines are subject to failures; such failures target the main data structures, such as those that store the chromosomes or their fitness values. This type of fault is known as Single Event Effects (SEEs) and may result in either permanent or temporary errors. In this research, to maintain the functionality and the performance of cGA engines, fault mitigation techniques at the algorithmic level are investigated. A cGA's inherent features, such as the diversity of phenotype and genotype spaces, migration polices, and adapting the number of evaluations, are utilised. In addition, information gathered based on the population diversity is used in an attempt to automatically isolate the faults.

The last motivation relates to the balance between exploration and exploitation. This balance is mainly achieved by means of the grid topology and/or the genetic operations. However, real-world problems require a system to be adaptive in order to continue to perform effectively in a changing environment. As a result, dynamic adaptation of the exploration/exploitation trade-off is an emerging challenge in the field of evolutionary computation. This area is intensely investigated in an attempt to improve algorithmic performance without incurring additional computational effort. In this research the genetic operations of cGAs, particularly the selection and their influence on the search process are investigated in order to dynamically control the exploration/exploitation trade-off.

1.2 Objectives

The overall aim of this thesis is to investigate the inherent characteristics of cellular genetic algorithms in order to improve their performance when dealing with complex problems, and to introduce new techniques that add fault tolerance and dynamic adaptation features to the algorithms. To achieve this aim, this research is carried out in three main stages−with the primary focus being geared towards improving the performance and reliability of cGAs.

The first stage is concerned with the investigation of cGAs characteristics, in particular the cellular dimensionality and their implications on the performance of the algorithms. Grid topology plays a significant role in the determination of the performance of EAs. cGAs are commonly implemented on 1D or 2D toroidal grid structures. A lack of studies concerning

higher cellular dimensions provides an opportunity to further investigate the algorithm's behaviour and performance. Hence, the first objective of this research is to evaluate and compare the performance of cGAs when implemented on 2D and 3D grid structures. In an attempt to seek advantages with higher cellular dimensions, an experimental study is carried out to compare the behaviours of 2D-cGAs and 3D-cGAs while maintaining similar algorithmic properties. The findings will add significant benefits for future optimisation engines. Achieving better algorithmic performance with 3D-cGAs creates a promising opportunity to combine the algorithmic benefits with thoes of advanced custom silicon chip technology, 3D-IC.

The second stage is concerned with increasing not only the effectiveness, but also the reliability of cellular genetic engines. The significant reduction in system electronics and operation in hostile environments lead these systems to be subjected to different kind of failures. Accordingly, research on fault-tolerant and mitigation techniques is becoming increasingly interesting. The second objective of this research is to develop an algorithm-based mitigating technique to tolerate failures encountered, in particular SEE errors, by utilising cGAs' inherent features. To achieve this goal, explicit migration techniques as well as dynamic adaptation techniques are proposed as mitigation techniques. The success of the proposed techniques in maintaining system functionality and effectiveness will not only be advantageous at the algorithmic level, but also at the hardware level as there will be no hardware requirement such as is the case with hardware-based fault-tolerant techniques.

The last stage is concerned with developing cGAs to allow dynamic adaptation in order to obtain an appropriate balance between exploration and exploitation. As pointed out earlier, the exploration/exploitation trade-off is a crucial factor that determines the behaviour and performance of the algorithm. The nature of EAs as being stochastic force systems to operate in a changing environment; this creates another interesting area for research, that is, dynamic adaptation. Thus, the final objective of this research is to introduce dynamic cGAs by utilising the genetic operations, specifically the selection. Two different approaches are proposed and evaluated by comparing their performance with that of other dynamic algorithms. The attainment of an appropriate exploration and exploitation balance while maintaining the algorithms' performance will positively contribute to the field of dynamic adaptation.

1.3 Contribution to Knowledge

This research seeks to take advantage of the decentralised structural properties of cGAs in order to expand the applicability of cGAs to the fields of dynamic adaptation and fault tolerance. The unique structure of cGAs contributes to their success and universal applicability in various application areas over standard and other parallel GA models. This research utilises the available opportunities that are offered through the inherent features of cGAs; namely, the structural properties that define the topologies of the local neighbourhood and the grid and their consequences on the genetic operations; and the related exploration/exploitation trade-off. Several studies that contribute to knowledge were carried out and published (the publications are shown in the next subsection). The following points highlight how this thesis contributes to existing knowledge.

- High cellular dimensions are proposed as a computationally effortless way to improve the exploration/exploitation trade-off and the overall performance of cGA models. The vertical expansion of the population plays an important role in the speed and the way that solutions spread, while maintaining the population size.

- A fault-tolerant technique is developed to mitigate SEE errors, specifically SEUs that target phenotypes registers. The technique is based on a genetic diversity measure, specifically the genotypic entropy, as a way to automatically identify and therefore isolate solutions with faulty fitness values. This technique illustrates how the faults occurring in one space (in this study, phenotypic space) are reflected in the other (genotypic space).

- A process of excluding the isolated or faulty solutions from the evolution process is proposed. This is in order to mitigate their impact on the search process, although this potentially results in a lower number of potential solutions.

- The development of adaptive migration operations with different policies as a technique to mitigate the impact of faults that are triggered by the reduction in the number of potential solutions and, accordingly, lead to diversity loss.

The proposed policies include the first fault-free neighbourhood, the best fault-free neighbour, and a random fault-free neighbour.

- Adaptive fault-tolerant techniques to further improve the performance of cGAs are proposed. The technique adapts to the fault ratio and allows more evaluations to overcome the impact of an increase in the number of faults.

- A dynamic-adaptive mechanism is proposed to balance exploration and exploitation in order to improve the performance of cGAs. The mechanism uses a stochastic selection operation that dynamically and gradually tunes the rate for selection based on a diversity degree. Accordingly, different levels of exploration and exploitation are induced at different search phases.

- An adaptive approach to dynamically control the exploration/exploitation trade-off based on convergence speed is developed. A diversity measure is used to compute the convergence speed, in accordance with the selection rate is tuned.

1.4 Publications

Publications that have arisen from this work are as follows:

1. Morales-Reyes, A., **Al-Naqi, A.**, Erdogan, A.T. and Arslan, T. (2009). Towards 3D Architectures: A Comparative Study on Cellular GAs Dimensionality. *In Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS '09)*, San Francisco, California, USA. IEEE.

2. **Al-Naqi, A.**, Erdogan, A.T., and Arslan, T. (2010). Fault Tolerance through Automatic Cell Isolation Using Three-Dimensional Cellular Genetic Algorithms. *In Proceedings of the IEEE Congress on Evolutionary Computation (CEC '10)*, Barcelona, Spain. IEEE.

3. **Al-Naqi, A.**, Erdogan, A.T., and Arslan, T. (2010). Balancing Exploration and Exploitation in Adaptive Three-Dimensional Cellular Genetic Algorithm via

Probabilistic Selection Operator. *In Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS '10)*, Anaheim, California, USA. IEEE.

4. **Al-Naqi, A.**, Erdogan, A.T., and Arslan, T. (2011). Fault Tolerant Three-Dimensional Cellular Genetic Algorithms with Adaptive Migration Schemes. *In Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS '11)*, San Diego, California, USA. IEEE.

5. **Al-Naqi, A.**, Erdogan, A.T., and Arslan, T. (2011). Dynamic Fault-Tolerant Three-Dimensional Cellular Genetic Algorithms, *Journal of Parallel and Distributed Computing* (submitted).

6. **Al-Naqi, A.**, Erdogan, A.T., and Arslan, T. (2012). Adaptive Three-Dimensional Cellular Genetic Algorithm for Balancing Exploration and Exploitation Processes. *Special Issue on Bio-inspired Algorithms with Structured Populations–Soft Computing Journal* (submitted).

1.5 Thesis Structure

This thesis is divided into six chapters. The five remaining chapters are organised as follows:

Chapter 2 provides a broad overview on the field of Evolutionary Computation. This chapter is further divided into three main sections. A deep look at both standard and parallel Evolutionary Algorithms, in particular the Genetic Algorithms, is provided in Section 2.1. Section 2.2 gives a dedicated review of cellular Genetic Algorithms–the main topic of this thesis. The different aspects that characterise cGAs are discussed in detail from both model and implementation points of view. This section ends by characterizing three-dimensional cGAs (the focus of this research) and gives an empirical comparison to standard GAs. Finally, the field of fault tolerance is reviewed in Section 2.3. This section gives an introduction to fault tolerance with a specific focus on the faults that were tackled in this research.

Chapter 3 proposes a study of cellular dimensionality in an attempt to exploit the advantages inherent in higher cellular dimensions. An empirical comparison is carried out to evaluate

3D-cGAs versus 2D-cGAs while maintaining similar algorithmic properties. A set of test and real-world problems that induce different levels of complexity to the search are also used to assess the performance of the algorithms compared. This work initially began as a collaborative effort with a previous group member in the System Level Integration research group (SLIg) and was further investigated in this research (Morales-Reyes, 2010). This chapter establishes the basis of this research.

Chapter 4 proposes a study on fault tolerance that aims to provide a fault-tolerant technique to mitigate SEU errors. This chapter is divided into three main parts. The first part (Section 4.1) introduces a fault-tolerant approach that automatically identifies and isolates faulty solutions. This section also introduces an explicit migration operation as a mitigation technique. The approach is assessed with and without the proposed migration operation against two fault scenarios for different test and real-world problems. Two additional adaptive migration policies are defined in the second part of this chapter (Section 4.2). The fault-tolerant approach proposed in Section 4.1 is then evaluated against the different migration polices for an extended set of problems with various complexities. The last part of the chapter, (Section 4.3) introduces an adaptive approach to fault tolerance in an attempt to further improve the performance of the approach proposed in Section 4.1. The adaptive fault-tolerant approach is assessed with and without migration for a similar set of problems as that used in Section 4.2.

Chapter 5 presents a study on a dynamic adaptation that aims to induce an appropriate balance between the exploration and the exploitation search process without incurring additional computational efforts. Two different adaptive algorithms are proposed: in the first, the search is guided by a diversity degree (Section 5.2) while the convergence speed guides the search in the second (Section 5.3). The convergence speed measure is adopted from a previous research in the same field (Alba and Dorronsoro, 2005) that proposed several static and dynamic cGA approaches. Selected static and dynamic approaches that were proposed in that study are compared to the approaches proposed in Sections 5.2 and 5.3 (Section 5.4). To assess the proposed algorithms and to reach a valid conclusion, a benchmark suite of test and real-world problems with different characteristics is used.

Chapter 6 summarises this thesis, provides conclusions, and discusses avenues for possible future research.

Chapter 2

Evolutionary Computation

This chapter provides a broad insight into the field of Evolutionary Computation (EC). The aim of this chapter is to review the critical aspects of Evolutionary Algorithms (EAs), especially those of Genetic Algorithms (GAs), including competitive search models of parallel GAs in general, and theoretical and methodological contributions to cellular Genetic Algorithms (cGAs) in particular (the main family of cellular EAs—cEAs). This comprehensive view of previous research on the cGAs establishes the basis for this research.

Evolutionary Algorithms are meta-heuristic algorithms that combine basic heuristic methods with higher-level frameworks in order to provide (sub-)optimal feasible solutions in a reasonable search time. Meta-heuristic algorithms are approximate and non-deterministic, and range from simple local search to complex learning processes.

EAs work on a population of encoded potential solutions (individuals) by applying a number of genetic operators—namely, selection, crossover, and mutation—to the individuals at each iteration in order to generate a new population. The selection operator is the most powerful as it guides the search process based on the selected individuals' fitness. Two individuals are selected to generate new individuals based on a predefined criterion. The crossover operator recombines the selected individuals to generate offspring, which are then modified by the mutation operator to introduce self-adaptation of individuals; these operators are applied based on a probability distribution. The mutation is commonly applied with a very low probability; otherwise this operator may lead to an ad-hoc search (Muhammad, Bargiela, and King, 1997; 1999). The main aim of the genetic operators is to learn about the connections between decision variables in order to locate areas in the search space processing high-quality solutions.

Various major branches of EAs have evolved over the past 40 years. One of the most well-known and widely studied EA variants is Genetic Algorithms (GAs). Unlike other EA techniques, GAs preserve a population of tentative solutions that are updated competitively through the application of some variant operators to find the global solution. Other EAs include Genetic Programming (GP), Evolutionary Programming (EP), and Evolution Strategy (ES). These algorithms differ slightly in their use of the genetic operators, with the main difference residing in their implementation and the nature of the problem to solve. GP was proposed by John Koza in 1992 (Koza, 1992). It generates its initial population by creating computer programs as potential solutions. As in GAs, GP assigns a fitness value to each solution (i.e., program) and uses selection, mutation, and crossover operators to generate a new population. ES was proposed by Ingo Rechenberg in the early 1960s and was subsequently developed (Jacob, 2001, p.211). It represents the solution as a chromosome of real values and considers the individuals' phenotypes as the parameters to be optimised. In that same year, EP was proposed by Lawrence Fogel (Jacob, 2001, p.297). EP has no constraints on the representation that follows from the problem. EP differs from the other EAs in that it uses no recombination mechanism (i.e., no crossover mechanism) (Jacob, 2001; Alba, 2005; Alba and Dorronsoro, 2008).

The field of EC is continually growing and evolving (Alba and Cotta, 2006). New EA variants have recently emerged in an attempt to overcome EAs' weakness that results in less accurate solutions when tacking hard and real problems in some applications. In addition, parallelisation in EAs is intensely exploited in an effort to improve performance. This research views cGAs as being highly parallel models of GAs. Details about GAs are provided in the subsection 2.1.

This chapter is divided into three subsections. Subsection 2.1 focuses on sequential and parallel GAs, while a deep insight into cGAs is provided subsection 2.2. Subsection 2.3 gives a general overview of the field of fault tolerance.

2.1 Genetic Algorithms

Genetic Algorithms are possibly the most popular class of EAs. GAs were proposed by Holland, who aimed to design artificial systems that possess similar properties to those of natural systems, in the early nineteen-sixties (Holland, 1992). As a result of his advanced understanding and utilisation of natural adaptation processes, Holland successfully introduced GAs in 1975. Subsequently, GAs widely proved their efficiency in a variety of

application areas. At first, they were mainly used to optimise combinatorial problems (Back, 1996). Nowadays however, GAs are also used to solve other optimisation problems belonging to continuous and other similar domains (Michalewicz, 1996).

GAs are iterative search techniques that apply stochastic operators on a population of encoded solutions (individuals). GAs efficiently explore complex problem spaces (i.e., genotypic space) in order to find the optimum solutions. The search process is guided with minimal information on the problem (i.e., phenotypic space). Phenotype space is evaluated through the objective (fitness) function at which a mapping between the individuals' phenotypes and genotypes is established.

As illustrated in Algorithm 2.1, GAs start with a random generated population $P(0)$ (line 2), followed by fitness evaluation (line 3). The first iteration t then starts with parent selection (line 6) in order to generate offspring. The crossover and mutation operators are then applied on the selected parent (lines 7 and 8, respectively). An evaluation of the updated population is then carried out, followed by the replacement of individuals to generate a population for the next iteration $P(t+1)$ (lines 9 and 10, respectively). These steps are repeated until the predefined stop criterion is fulfilled (line 5).

Algorithm 2.1 Pseudo-code of a canonical GA

```
1: procedure GA
2: Generate_initial_population (P(0));
3: Evaluation (P(0));
4: t ← 0;
5:   while ! stop_condition do
6:     P'(t) ← Selection (P(t));
7:     P''(t) ← Recombination (P'(t));
8:     P'''(t) ← Mutation (P(t));
9:     Evaluation (P'(t));
10:    P(t+1) ← Replace(P(t), P'''(t));
11:    t ← t+1;
12:  end while;
13: end procedure GA;
```

The most commonly used stop criterion is that of reaching a predefined number of fitness evaluations and/or finding the optimal solutions—an optimum solution can be defined as

those individuals having the global fitness value (f^*) or a tolerable fitness value (e.g., $f^* \geq$ *threshold*). However, other stop criteria could be defined. This research uses a more restrictive stop criterion that is based on the average fitness values of the population.

A brief discussion of the basic GA operations (selection, crossover, and mutation) is provided in the following paragraphs. A large number of selection mechanisms has been developed, with the most common being proportionate and tournament selections (Rothlauf, 2006). With proportionate selection, the number of copies an individual possesses in the subsequent population is proportional to its fitness; and an individual x_i to be chosen for recombination has a probability computed as follows:

$$f(x_i) \Big/ \sum_{j=1}^{N} f(x_j) \tag{2.1}$$

where N is the number of individuals in a population. The probability of the individual to be chosen increases as its fitness increases.

With tournament selection, a number of individuals (t) are randomly selected for a tournament, which the fittest individual wins. There are two approaches for tournament selection: without replacement and with replacement. In the former, there are t rounds and each round has N/t tournaments. The selection of individuals for a tournament is made from those who are not involved in the current round of the tournament. In the latter approach however, all t individuals are selected for a tournament at the same round. This research uses the tournament selection as the local selection method in the experimental setups. The specific type used is the Binary Tournament (BT) selection, in which two random individuals are selected and the fittest individual wins the tournament (i.e., $t = 2$). Zhong *et al.* (2005) conducted a study that compared the performance of simple GA for different selection mechanisms, particularly tournament and roulette wheel selections. They found that the tournament selection mechanism resulted in the better algorithm performance.

Similarly, a large number of crossover and mutation operators have been proposed. Crossover simulates the role of sexual reproduction and is operated on the selected individuals in a population to generate offspring, while mutation imitates biological mutation and is operated on the generated offspring to induce slight changes in an individual's genotype. Typically, in a GA, crossover generates two offspring from two parents, whereas the mutation alters one or more genes (or alleles) in an individual. Both occur according to predefined crossover and mutation probabilities. Classical crossover mechanisms include one-point, two-point uniform, and arithmetic crossover, among others. In addition, traditional mutation techniques include bit-flip, uniform, and non-uniform mutation, among

others. The selection of a specific crossover or mutation technique highly depends on the individuals' encoding (binary, real, etc.) as well as the type of the problem to be solved. The main purposes of crossover and mutation are to improve an algorithm's performance and to prevent trapping in local minima areas by preserving and promoting population diversity. Unlike crossover, mutation focuses on local search as it only alters properties of individuals. Hence, the probability of mutation should be low otherwise many genes (or alleles) will be altered, leading to random search.

Finally, replacement strategies also play an important role in improving the performance of algorithms in general and in enhancing population diversity in particular. The most common standard GAs directly depend on replacement strategies. A brief discussion on non-decentralised (panmictic) GAs is provided next.

2.1.1 Non-Decentralised GAs

This section describes the two most popular panmictic GAs, which are characterised by their non-structured population–resulting in interactions between individuals occurring without restrictions; that is, an individual can mate with any other individual. A brief description, with pseudocodes, of steady state and generational GAs is provided.

Algorithm 2.2 Pseudocode of a ssGA

```
1: procedure ssGA
2: Generate_initial_population (P(0));
3: Evaluation (P(0));
4: t ← 0;
5:   while ! stop_condition do
6:      P'(t) ← Selection (P(t));
7:      P''(t) ← Recombination (P'(t));
8:      P'''(t) ← Mutation (P(t));
9:      Evaluation (P'(t));
10:     P(t+1) ← Replace(P(t), P'''(t));
11:     t ← t+1;
12:   end while;
13: end procedure ssGA;
```

Algorithm 2.3 Pseudocode of a genGA

```
1: procedure genGA
2: Generate_initial_population (P(0));
3: Evaluation (P(0));
4: t ← 0;
5:   while ! stop_condition do
6:     for i ← 1 to popSize do
7:       P'(t) ← Selection (P(t));
8:       P''(t) ← Recombination (P'(t));
9:       P'''(t) ← Mutation (P(t));
10:      Evaluation (P'(t));
11:      P_aux(t) ← Add(P(t), P'''(t));
12:    end for;
13:    P(t+1) ← Replace(P_aux(t));
14:    t ← t+1;
15:  end while;
16: end procedure GA;
```

Algorithm 2.2 illustrates the pseudo-code for steady state GA (ssGA). Typically, ssGA selects two parents according to a defined selection policy (line 6) to generate offspring for the next iteration. It recombines the parent and then mutates the generated offspring (lines 7 and 8, respectively). Next, the new offspring is evaluated and made to compete with the parents. The winner is then added to the population according to a defined replacement policy (lines 9 and 10, respectively). In a typical ssGA, if the offspring is better than the worst parent in the population (*Replace-if-better*), the latter is replaced by the former. Other replacement policies include *Replace-the-worst*, *Replace-the-oldest*, *Replace-random-individual*, among others. This process is reiterated until the stop condition is satisfied (line 5).

Generational GA (genGA) generates new offspring from individuals in the current population by applying the genetic operators: selection, recombination, and mutation. It then adds the offspring to an auxiliary population (see Algorithm 2.3). The auxiliary population is then replaces the current population, when the entire population has been generated, to be used for the next iteration.

Thus, the difference between ssGA and genGA is that with the former only one individual at a time is introduced into the current population, requiring a replacement strategy to vacate the place for the new offspring to occupy if it survives, while with the latter, a whole new population is generated to replace the current one. As a result, genGAs are also known as (μ,λ)–GAs, while ssGAs are known as $(\mu,1)$–GAs, where μ is the size of the population and λ is the size of the auxiliary population (Alba and Dorronsoro, 2008).

Next, a general discussion about decentralised (or parallel) GAs is provided followed by a specific discussion about cGAs. A comparison between panmictic and parallel GAs is provided.

2.1.2 Decentralised Genetic Algorithms

The complexity of most real-world problems and/or the limited resources available to solve them, led to the development of meta-heuristic algorithms. As mentioned earlier, meta-heuristics give optimal, or near optimal, solutions in an adequate time. However, the high dimension of many tasks results in a long execution time. Hence, parallelism of meta-heuristics was initiated to reduce resolution time as well as to improve the quality of the solutions (Alba, 2005).

The two phases beyond the introduction of panmictic GAs are the coarse-grained (distributed) and the fine-grained (cellular) parallel GAs. In the former, several large subpopulations evolve in parallel with limited interaction between subpopulations; while in the latter, several small subpopulations evolve in parallel with regular interaction. One motivation behind the parallelism is the potential decrease in the resolution time through the assignment of each subpopulation to a single processor in a multi-processor system. Another motivation is the ability to explore different areas of the search space in parallel by independently evolving each subpopulation, with the independent evolution of the subpopulation leading to enhanced the genetic diversity (Chambers, 1999).

From the above discussion, two ways to reduce the execution time can be identified. The first method is to directly run the algorithm in parallel hardware, while the second is to utilise the GA's inherent parallelism (Eklund, 2003). Next, a brief discussion about parallel hardware is provided, followed by a discussion on coarse-grained and fine-grained GA models.

2.1.2.1 Parallel Hardware

The objective of this section is to understand hardware concepts related to parallel computer architectures in order to establish a relation between parallel hardware and the implementation of parallel algorithm models. However, it is first necessary to understand that parallel models and parallel hardware are not the same. Parallel models describe the independent computation of multiple tasks and can be executed on both parallel and sequential computers, while parallel hardware requires physical divisions in the independent tasks (Alba, 2005).

Generally, parallel architectures are classified into Single Instruction Single Data (SISD), Single Instruction Multiple Data (SIMD), Multiple Instruction Single Data (MISD), and Multiple Instruction Multiple Data (MIMD) (Culler, Singh, and Gupta, 1998; Roosta, 1999).

SISD refers to a computer architecture in which a mono-processor runs single instruction on data stored in single memory. However, SISD has parallel characteristics, for example fetching and pipelined execution of instructions (Roosta, 1999).

SIMD corresponds to a parallel computer architecture in which the same instruction is executed by several processors over multiple data. Typically, a SIMD architecture has hundreds or thousands of simple processors, each with a local memory. Despite its ability to

exploit data level parallelism, the use of SIMD architectures is limited due to their complexities, inflexibilities, and dependence on synchronisation (Roosta, 1999).

Similar to SIMD, MISD refers to parallel computer architecture, but this architecture executes multiple instructions on the same data; an example of this architecture is a pipelined computer. MISD architectures are rarely found in practice due to their poor scaling and their excessive use of computational resources (Roosta, 1999).

MIMD is a technique designed to achieve parallelism and is the most useful one. Most parallel computers fit this mould. MIMD computers have several processors that operate independently and asynchronously and in which different processors run different instructions over different data. MIMD architectures have more classifications based on the way the processor accesses memory. These classifications are as multiprocessors and multi-computers (distributed system). In the former, processors access memory directly, while in the latter, processors need a message-passing mechanism in order to access remote memories. These two classes of MIMD are even further divided; multiprocessors are classified into uniform and non-uniform memory accesses (UMA and NUMA, respectively), and each is also classified based on the interconnection media between the processors (Bus-based or switched). Although multiprocessors are widely in use, they have a limited number of processors. Increasing the number of processors results in an exponential increase in their price. Distributed systems consist of several computers that are interconnected: each computer has a processor, a memory, and a network adapter. A distributed system can be a cluster of workstations (COW) or a massively parallel processor (MPP). In the former, the workstations are connected by a network technology; this technology restricts the number of workstations to a few hundred. Conversely, MPP has thousands of processors. The advantages of distributed systems are mainly presented in their easy build and extension, better price-performance trade-off, and more scalability and flexibility (Roosta, 1999; Alba, 2005).

2.1.2.2 The Islands Model

Islands or distributed GAs (dGAs) is one of the most popular parallel models. This model is also known as coarse-grained GA according to grain size. In a typical dGA, the population is divided into multiple and relatively large subpopulations (islands) that each evolves independently (Alba and Troya, 1999a). Each subpopulation runs the standard GA and the interaction between individuals in different subpopulations is introduced and managed

through a migration technique (see Figure 2.1). Figure 2.1 illustrates a dGA with 6 subpopulations; each evolves independently by running a standard GA. The interaction between the subpopulation occurs through individual migration over a predefined communication link.

This subsection provides a broad discussion about the main issues related to dGAs. These issues are homogenous and heterogonous models, migration policies, synchronism, speed-up, and implementation.

Homogenous and heterogeneous dGA

Each island or subpopulation applies the genetic operators (selection, crossover, and mutation) in isolation from other islands; therefore each island searches a different area in the search space. In addition, each island can have its own configuration (such as crossover and mutation probabilities, individual representation, among others). The different configurations among islands lead to the formation of a class of dGA called heterogeneous dGA, while in heterogonous dGA a similar configuration is used for each island (Alba, Nebro, and Troya, 2002; Alba, Luna, and Nebro, 2004). Although heterogeneous models are difficult to understand and implement, they show good results in practice (Tomassini, 2005). A promising heterogeneous dGA that uses different crossover operators in each subpopulation was proposed by Herrera and Lozano (2000). A comparison between the

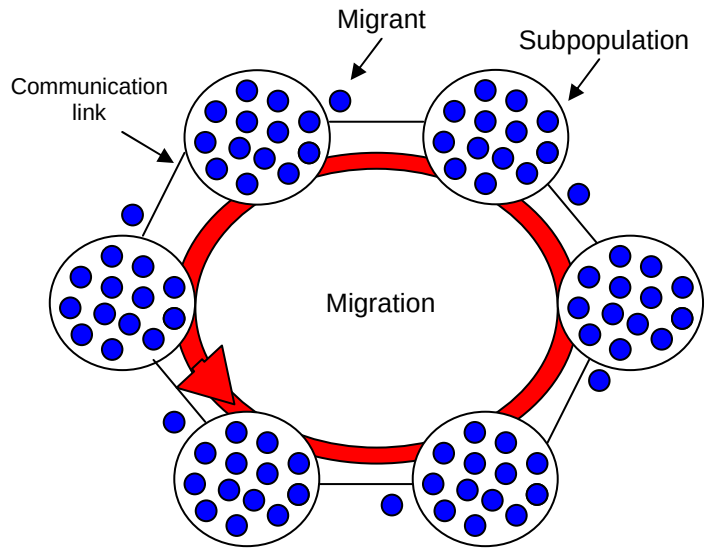


Figure 2.1. Islands or distributed GA with 6 multi-individual subpopulations.

proposed algorithm, panmictic GAs, and homogenous dGAs, among others, showed that the former outperforms the rest in terms of reliability and accuracy.

Migration

An essential issue in dGA is defining an appropriate migration policy due to its significant influence on the performance of the algorithm (Rebaudengo and Sonza Reorda, 1993). Typically, the parameters of the migration technique include migration gap, migration rate, selection/replacement of migrants, and topology (Alba, 2005). The migration gap or frequency defines how many generations in each island are between two successive migrations. The migration gap can either be set periodically or by defining a probability P_M. The migration rate, sometimes called migration size, defines the number of individuals involved in each migration, which can be a constant number or a percentage of the subpopulation size. The migration strategy or the selection/replacement of migrants is defined according to which migrants are selected and which individuals are replaced by migrants. Lastly, the topology defines the island's neighbours with which each island can communicate; in the islands model the interaction is geographically restricted to nearby neighbours.

In an early study, Rebaudengo and Sonza Reorda (1993) selected the problem of TSP to assess the performance of dGA against different migration frequencies, sizes, and strategies. They found that different migration parameters significantly affect the performance of the algorithm. In addition, they concluded that migration has a similar effect to that of mutation as both operations introduce new genetic information. However, they also found that mutation has an advantage in that the information introduced is better and new, which speeds up the algorithm without driving it to a local minimum area.

Matsumura *et al.* (1997), in a later study investigated the effects of migration on different multiprocessor system topologies (namely, ring, tours, and hypercube). In that study, Matsumura *et al.* used two types of migration to define the migration gap: namely, immigration and emigration types. In the former the migration operation is activated when the best fitness value is not updated, while in the latter the migration is activated when the best fitness value is updated. They found a relationship between solution quality, migration types, convergence speed, and topology. Thus, in general, the combination of specific migration type and topology may significantly affect solution quality and convergence speed.

A theoretical study on the scalability of parallel GA was proposed by Cantu-Paz and Goldberg (1999). The main aim of that study was to calculate the best possible number of processors needed to obtain the minimum execution time. The bounding cases (maximal and minimal values) in terms of topology degree, migration rate, and frequency were considered. Cantu-Paz and Goldberg concluded that the optimal number of processors needed to minimise the execution time is directly proportional to the square root of the population size and the time of fitness evaluation. They also suggested that a large number of processors could be integrated in parallel GAs while significantly reducing the execution time.

In a study similar to the previous one concerning island size, migration rates, and topologies, additional problems were considered to confirm the previous conclusion (Cantu-Paz, 1999a). In that study, Cantu-Paz established a relationship between island size, migration rate, and topology degree (number of neighbours of each island) with search success rate. He showed how to identify a configuration that obtains an appropriate execution-time/solution-quality trade-off by deriving an equation to calculate an accurate island size, which in turn is used to identify the migration rate and the topology degree. The conclusion arrived at is similar to that of the previous study.

At the same time, Cantu-Paz (1999b) also investigated the affect of different migration strategies on the selection pressure while migration rate, frequency, island size, and topology degree remained constant. He defined four combinations of random and fitness-based emigration and replacement of individuals; with the results showing that the selection/replacement of migrants significantly affect the convergence speed. Later, Cantu-Paz extended the latter study to quantify the increased selection pressure, which is an important issue in the avoidance of search failure (Cantu-Paz, 2001).

In summary, dGA introduced new algorithmic parameters such as number of subpopulations, frequency of migration, selection and replacement of migrants, and network topology. However, these parameters presented a major drawback of dGA as only few theories were proposed on how to tune these parameters (Eklund, 2004).

Synchronism

Besides migration, synchronism is another factor that influences the search time and speedup. In dGA, synchronism occurs through migration. If the migration uses asynchronous communication, then the migrants are inserted immediately when they arrive at the intended island. A major advantage of asynchronous communication is that it avoids blocking steps between the migration gaps. Conversly, synchronous islands wait for every migrant they

must add, consequently affecting the execution time because of the continuous waits. Several studies have reported faster execution and more flexibility for asynchronous communication (Alba and Troya, 1999b; Alba, Cotta, and Troya, 1999a; 1999b; Alba, Nebro, and Troya, 2002; Alba, Luna, and Nebro, 2004).

Alba and Troya (1999b) analysed the synchronism in the migration step of dGA with steady state or cellular modes of island evolution (dssGA and dcGA, respectively) as well as other panmictic and non-distributed GAs. The conclusion they reached reported that the asynchronous algorithms achieved considerably less search times and larger speedup than their synchronous counterparts. The tight coupling in dcGA demonstrated a drawback of the synchronisation for harder problems. However, dcGA showed better resistance to bad migration frequencies than dssGA. The same conclusion was obtained in (Alba, Cotta, and Troya, 1999a; b) in which more difficult problems were considered. In addition, they reported that in terms of effort and diversity, both synchronous and asynchronous versions of dGA with generational, steady state, and cellular islands showed no differences.

In addition, the influence of synchronisation in heterogeneous dGAs was analysed in (Alba, Luna, and Nebro, 2004) to further show the importance of synchronism in different dGA models. The results confirmed those obtained in previous studies. The wait constraints induced by the synchronous versions penalise the execution time, especially for a large number of islands. Consequently, better efficiency can be achieved by asynchronous parallelisation.

In summary, synchronisation in dGAs is determined through migration of individuals between panmictic or cellular subpopulations. The investigation on the advantages of asynchronous commutation showed high parallel efficiency and scalability. In addition, implementing parallel GAs with asynchronous communication on heterogeneous parallel hardware has the added advantage of parallelism that avoids the bottleneck induced by the slowest processor.

Speedup

Speedup is an important measure in parallel algorithms. In this measure, two times are compared: namely, the sequential and the parallel times needed to run the same algorithm. Thus, the speedup of m processors (s_m) is the ratio between execution time on a mono-processor (T_1) and the execution time on m processors (T_m) (Alba, 2005). For many years, this measure has been used to analyse the performance of deterministic algorithms.

However, replacing the absolute times (T_1 and T_m) by the average times ($\overline{T_1}$ and $\overline{T_m}$) enabled it to be used to analyse the performance of non-deterministic algorithms.

Alba and Troya (2002) identified two types of speedup: strong speedup and weak speedup. Researchers favour the use of the latter as the former considers the best (fastest) recent sequential algorithm, which is difficult to find. Furthermore, they suggest that the comparison of speedups between sequential and parallel GAs must be made by running both algorithms until similar quality solutions are arrived at.

The three levels of speedups are: sub-linear ($s_m < m$), linear ($s_m = m$), and super-linear ($s_m > m$). Many researchers suggest the possibility of parallel GAs being used to achieve super-linear speedup, for example, the work of Alba and Troya (1999b). However, the topic of super-linear speed is still controversial.

Obviously, the move from panmictic to distributed population plays an important role in enhancing speedup as a lower execution time is needed for smaller subpopulations. More interestingly, in addition to the previous speedup source, speedup can be gained from the same distributed algorithms. Alba and Troya (2002) showed that dGA running on several processors achieved a super-linear speedup when compared to its panmictic counterpart, while a sub-linear speedup is achieved when it is compared to the same dGA on one processor.

In addition, synchronism and migration in parallel GAs may significantly influence speedup (Alba, 2002). Alba and Troya (1999b) compared the speedups of asynchronous dGA with panmictic and cellular subpopulations to their synchronous counterparts. The result showed the ability of the compared algorithms to obtain super-linear speedups. In addition, an improvement was obtained when comparing asynchronous algorithms to the synchronous dssGA and a slight improvement was noticed when synchronous dcGA was considered (because of the highly coupled islands) for similar migration frequencies. Further, in their study, Alba and Troya investigated the effect of different migration gaps (1, 16, and 32) on the speedup. They found that there was better speedup for larger gaps (16 and 32) with super-linear speed for dssGA and almost linear speed for dcGA for the largest gap (32).

In conclusion, all the previous studies agreed on the possibility of parallel GAs to obtain super-linear speedup, in theory and in practice, both in homogenous and heterogeneous parallel hardware.

Implementation

A traditional (false) assumption about parallel GAs was the mapping of parallel GA models directly onto the parallel hardware, thereby making the model and its implementation equivalent terms. However, a parallel model can be implemented on either mono-processor or multi-processor machines.

From a hardware perspective, a dGA is very easy and efficient to implement in distributed memory MIMD computers, which partly contributes to its popularity. Despite the fact that a few independent subpopulations may limit the maximum speedup of this model, it is still faster than panmictic GA in terms of both run and convergence times. In addition, subpopulation structure, synchronism, and migration all influence the search time and speedup when running parallel GAs in a MIMD machine. Furthermore, cluster implementation of the island model is physically fairly large–resulting in the exclusion of many applications (Eklund, 2004).

In (Alba, Cotta, and Troya, 1999b) and (Alba, Nebro, and Troya, 2002) the islands model was implemented in homogenous and heterogeneous clusters of workstations, respectively. Super-linear speedup was experienced not only in the homogenous but also in the heterogeneous machine clusters. In addition, the results showed that the heterogonous cluster was more efficient. In the next subsection, the cellular GAs (diffusion) model is discussed in very broad terms, followed by a more profound discussion of this model in the subsequent subsections.

2.1.2.3 The Diffusion Model

The diffusion model is also called fine-grained, cellular, and massively parallel GA. This model distributes its population over the structure of the processing elements (nodes), commonly a two-dimensional grid with wraparound edges (toroidal), in which each processing element holds only a few individuals, typically one. This spatial distribution defines and restricts the interaction between the individuals to their local neighbourhoods (Baluja, 1993).

Figure 2.2 illustrates a diffusion or cellular GA with 5×5 subpopulations distributed over a 2D-toroidal grid: each contains one individual with its neighbourhood comprising four individuals located at the north, south, east, and west.

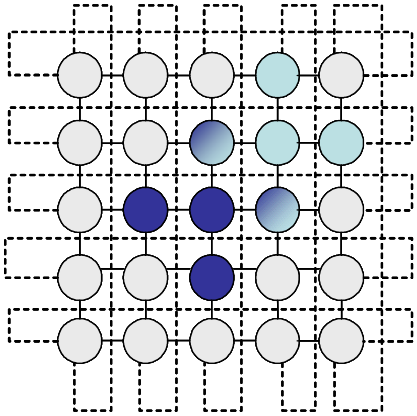


Figure 2.2. A cGA implemented over 5×5 toroidal grid. The neighbourhoods marked in dark and light blue show a possible overlapping of two neighbourhoods.

This model can also be viewed as a combination of standard GAs and Cellular Automata (CA) as the population is distributed over an n dimensional toroidal grid in which each individual occupies a position. Several researchers have investigated the performance and behaviour of a GA implemented on a CA (or CGA) (Kirley, Li, and Green, 1999; Back and Breukelaar, 2005; Olariu and Zomaya, 2006). The mutual conclusion is that CGA outperforms standard GAs with its ability to better escape local optima. Back and Breukelaar (2005) further investigated this model by considering multiple grid dimensions. The findings indicated promising benefits of algorithm performance for higher grid dimensions.

Unlike the island model, the number of subpopulations is quite large which makes the diffusion model massively parallel, consequently increasing the potential of obtaining higher speedups. In addition, the migration in the diffusion model implicitly occurs due to the overlapped neighbourhoods. However, an explicit migration could be defined (Lee, Park, and Kim, 2000). All steps of the GA (evaluation, selection, and genetic operations) are applied in parallel within each individual's neighbourhood in which only the current individual, the one at the centre, is updated. The massive parallelism and the absence of explicit migration are two advantages, among others, of the diffusion model (Eklund, 2004).

Another benefit of the diffusion model is its suitability for implementation in VLSI because of its simple, regular, and locally connected nodes. Despite the fact that cGAs were originally designed for work in massively parallel computers, they have also been adopted and implemented in distributed and mono-processor machines. Section 2.2 describes cGA in more detail.

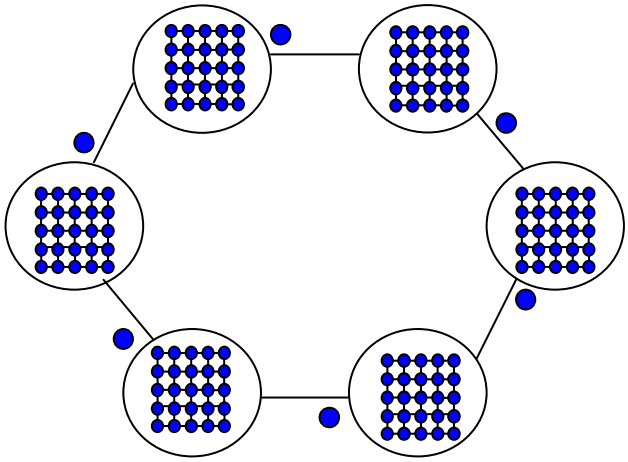


Figure 2.3. A hybrid parallel model of GA that combines cGA at the lowest level (each node) with dGA at the highest level to form what can be referred to as dcGA.

2.1.2.4 Hybrid Models

There have been various attempts to combine two of the parallel GAs in order to get the combined advantages of both (Cantu-Paz, 1995; Nowostawski and Poli, 1999); these are called hybrid models. One of the most well-known hybrid algorithms combines the diffusion model at the lowest level with the island model (see Figure 2.3). The discussion in Section 2.1.2.2 included some studies relating to this hybrid model, which was referred to as dcGA.

Although hybrid models may lead to the birth of new efficient algorithms, some of these models introduce more complexity to parallel GAs, for example the need for new additional parameters to manage a more complex topology structure (Alba, 2005).

2.2 Cellular Genetic Algorithms

The cellular model is a class of evolutionary algorithms with structured population that emphasises evolution at the individual level (Alba and Dorronsoro, 2008). cEAs are a kind of stochastic CA in which the number of points in the search space in cEAs is equivalent to the cardinality of the symbol alphabet in CA. Typically, in cGA each individual is assigned a grid position (cell); the topology of the grid is commonly implemented on an n-dimensional toroidal grid having a linear, square, or rectangular geometric shape. The concept of local neighbourhood is strictly enforced and an individual only interacts with its local neighbours. In a cGA, the diffusion of solutions occurs slowly with the aid of the overlapped local neighbourhoods, therefore offering exploration (diversification) of the search space, while

the genetic operations applied in each neighbourhood supports the exploitation (intensification) of good solutions. Hence, a major issue in determining the effectiveness of cGAs is the balance between exploration and exploitation, which is a direct effect of the selection pressure. The theory and practice surrounding this issue is discussed in the next subsection.

Algorithm 2.4 Pseudo-code of a canonical cGA

```
1. procedure cGA
2. Generate_initial_population (P(0));
3. Evaluation (P(0));
4. t ← 0;
5. while ! stop_condition do
6.    for i ← 1 to ROWS do
7.       for j ←1 to COLUMNS do
8.          neighbours ← Find_neighbours (position(i,j));
9.          parent1← position(i,j);
10.         parent2 ←Local_selection (neighbours);
11.         offspring ← Recombine (Pc, parent1, parent2);
12.         offspring ← Mutate (Pm);
13.         Evaluation ←Fitness(offspring);
14.         Replacement (position(i,j), offspring, Paux(t));
15.      end for;
16.   end for;
17. P(t+1) ← Paux(t); // updating
18. t ← t+1;
19. end while;
20. end procedure cGA;
```

In a cGA, the population is usually distributed over a two-dimensional toroidal grid topology, although lower or higher grid dimensions are possible. Algorithm 2.4 illustrates the pseudo-code of the canonical cGAs implemented on a two-dimensional grid. A cGA starts with a random population $P(0)$ followed by fitness evaluations (Lines 2 and 3). Next, each individual is updated by selecting a second parent from its neighbourhood according to a specified local selection method (Line 10), and the first parent is the individual itself (Line

9). This type of parent selection is referred to as 'current individual + local selection'; the other selection type selects both parents from the neighbourhood of the current individual with or without replacement through the defined local selection method. Several local selection methods that can be used include those implemented on standard GAs as well as selection methods specifically designed for implementation on parallel GAs such as anisotropic and centric selections (Simoncini *et al.*, 2006a; 2009). A crossover operator recombines the selected parents with a probability P_c to produce an offspring (Line 11), which is then mutated by a non-uniform mutation operator with a probability P_m (Line 12). The modified offspring is then evaluated and, according to the specified replacement policy, the current individual is either kept or replaced by the newly generated offspring (Lines 13 and 14).

This process continues until all of the individuals are updated. The current population $P(t)$ is then replaced by the auxiliary one $P_{aux}(t)$ to start the next generation (Line 17). The updating process defined here is synchronous, which means that the updated individual is inserted into an auxiliary population following a specified replacement policy. An alternative updating option is to apply an asynchronous update, in which the updated individual is directly inserted into the current population. (Subsection 2.2.2 discusses the synchronism in a cGA.) The algorithm terminates when the termination condition is met (Line 5).

2.2.1 Takeover Time and Selection Pressure

The structural properties of cGAs, including population (grid) and neighbourhood topologies, shape, and size, as well as genetic operations such as selection, replacement, and synchronisation may bestow several advantages on the effectiveness of the search. Two related and major issues that directly result from the abovementioned structural properties and operations are takeover time and selection pressure. Hence, careful attention to the takeover times and the selection pressure in the context of structural properties and operations is required.

The takeover time represents the speed needed by the best solution in the population to conquer the whole population when only activating the selection operator (i.e., the growth rate of the best individual). Goldberg and Deb (1991) theoretically derived and compared the takeover times for panmictic GAs for different selection methods. They found that most of selection methods considered had a similar convergence times for *order of growth* $O(\log n)$ generations, where n is the population size. The proportional selection method was an

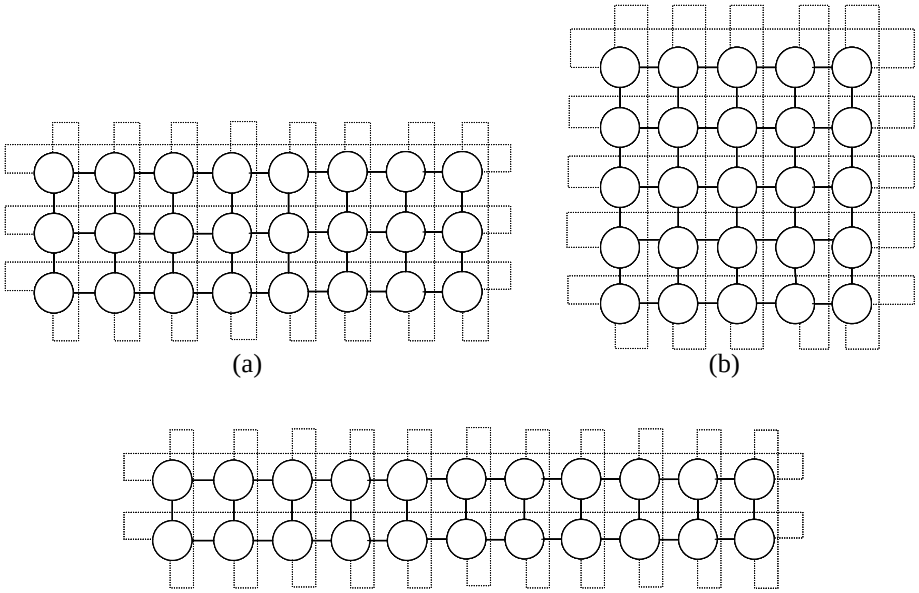
exception to this as a slower convergence time was obtained for it by a factor of n. Later, Rudolph (2000) proposed a theoretical study on takeover times in cellular EAs with one-dimensional array and ring topologies. He derived the takeover times as a function of population size and selection probability for both considered topologies. In addition, Rudolph suggested that the takeover time depends to a lesser extent on the selection method than on the radius of the neighbourhood.

A shorter takeover time denotes a higher selection pressure (intensity) leading to the promotion of more exploitation. High selection pressure leads to quick diversity loss. Therefore, the search may stagnate in the local minima area. Conversely, lower selection pressure promotes more exploration and therefore more diversity. Hence, careful attention to selection methods and other EAs settings is required. For other theoretical study on takeover time refer to Spiessens and Manderick (1991).

The next subsection discusses the selection pressure in cGAs with respect to the structural properties. Following that the influence of the genetic operations, specifically on the selection pressure is demonstrated.

2.2.1.1 The Influence of Grid-to-Neighbourhood Ratio

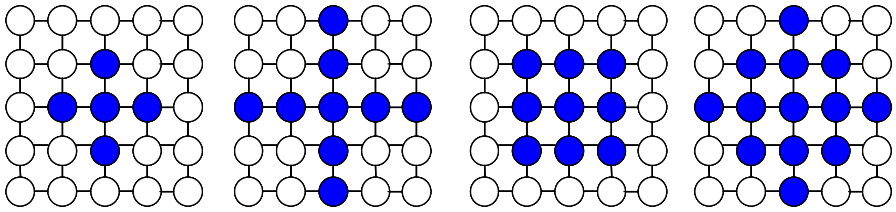
Before defining the Grid-to-Neighbourhood Ratio (*NGR*), a broad overview on grid and neighbourhood topologies is provided. As mentioned previously, a cGA is usually implemented on a two-dimensional grid topology with wraparound edges following a toroidal shape. Depending on the number of rows and columns a 2D toroidal grid can have a rectangular, a square, or a narrow topology; these configurations are illustrated in Figure 2.4(a)–(c), respectively. Whereas, several neighbourhood configurations can be defined, the various configurations are commonly classified into Von Neumann (NEWS) or Moore (X-net) neighbourhood (see Figure 2.5). In the former (also referred to as Linear (*L*)), the neighbourhood of an individual comprises those individuals located to its north, east, west, and south. With regard to the latter (also referred to as Compact (*C*)), in addition to the linear ones, the individuals located on the diagonal of the current individual are also included in its neighbourhood. The number of individuals in a neighbourhood is determined by the predefined neighbourhood radius (distance step). For example, the linear neighbourhood can have 5 individuals (*L*5) for 1 distance step (see Figure 2.5(a)), while a compact neighbourhood contains 9 individuals (*C*9) for the same distance step (see Figure 2.5(c)).



(a) (b)

(c)

Figure 2.4. Two-dimensional toroidal grid topologies in cGA: (a) with rectangular shape, (b) with square shape, and (c) with narrow shape.



(a) *L*5 (b) *L*9 (c) *C*9 (d) *C*13

Figure 2.5. Von Neumann neighbourhood: (a) with one distance step and (b) with two distance steps. Moore neighbourhood: (c) with one distance step and (d) with two distance steps.

It is now possible to proceed to theoretically define *NGR*. The concept of *NGR* establishes a numerical relationship between neighbourhood and grid radii, which is computed by measuring the dispersion of a point pattern (an individual position) with respect to the mean centre ($\bar{x}, \bar{y}$) of a neighbourhood (or grid) pattern of size *n* as follows:

$$Rad = \sqrt{\frac{\sum (x_i - \bar{x})^2 + \sum (y_i - \bar{y})^2}{n}}\ ,\ \bar{x} = \sum_{i=1}^{n} x_i \Big/ n\ ,\ \bar{y} = \sum_{i=1}^{n} y_i \Big/ n\ . \qquad (2.2)$$

where x_i is the row and y_i is the column of a location of the individual *i*.

Therefore the ratio between the neighbourhood and the grid radii is calculated as follows:

$$NGR = \frac{Rad_{neighbourhood}}{Rad_{Grid}} \tag{2.3}$$

Many researchers conclude that different grid/neighbourhood shapes and sizes impose different levels of selection pressure. Sarma and De Jong (1996) empirically analysed the effect of neighbourhood size, shape, and radius on the selection pressure. They showed in their study that *NGR* is a critical parameter in which different *NGR*s induce different global selection pressures. In other words, algorithms that have similar ratios, even if they have different population and neighbourhood sizes, show similar selection pressure. This conclusion was further investigated and confirmed in subsequent studies (Dorronsoro *et al.*, 2004; Giacobini *et al.*, 2005).

Alba and Troya (2002) analysed the effects of the *NGR* on the computational effort in terms of the number of evaluations, efficacy (number of hits), and scalability in cGAs. In summary, they found that thinner grids require more evaluations, provide better efficacy (especially when solving difficult problems), and scale adequately, while square grids scale slightly better as the size of a problem increases.

To empirically show the influence of different *NGR*s on the selection pressure, experiments that included combinations of different grid shapes (square, rectangular, and narrow) and different neighbourhood sizes (*L*5and *L*9) were carried out. The population contained 400 individuals arranged as 20×20, 10×40, and 4×100 for square, rectangular, and narrow grids, respectively. Figure 2.6 depicts the average growth rates of the best individual (of 50 independent runs) for a square grid with *L*5 (*NGR* = 0.1097) and *L*9 (*NGR* = 0.1828) neighbourhoods, rectangular grid with *L*5 (*NGR* = 0.0752) and *L*9 (*NGR* = 0.1253) neighbourhoods, and narrow grids with *L*5 (*NGR* = 0.0310) and *L*9 (*NGR* = 0.0516) neighbourhoods when applying the binary tournament selection only. Smaller ratio values induced lower global selection pressures in the population (longer takeover time), while larger ratio values induced higher selection pressures (shorter takeover time). Further, with regard to similar neighbourhoods, square grids obtained the highest selection pressure, leading to more exploitative search; while the lowest selection pressure was obtained by narrow grids, leading to more explorative search.

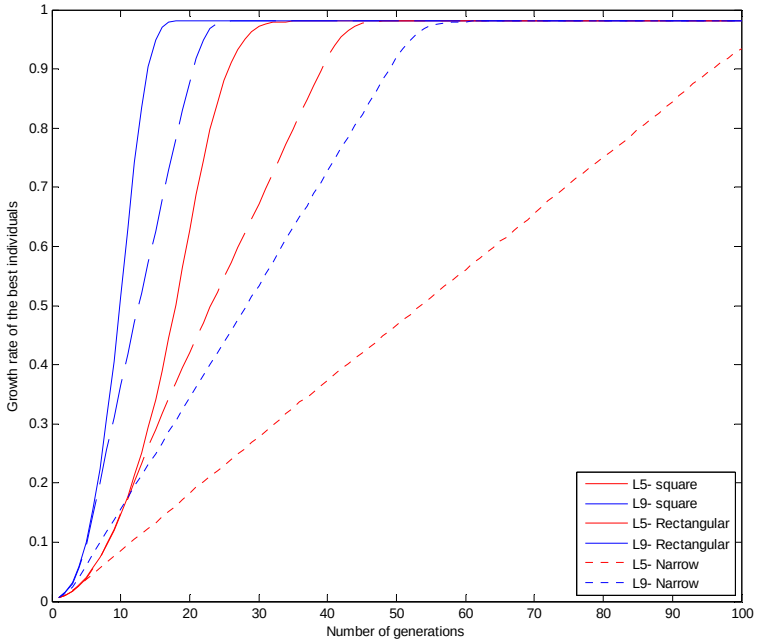


Figure 2.6. The best individuals' average growth rates for square, rectangular, and narrow grids, each with *L*5 and *L*9 neighbourhoods.

The latter findings led many researchers to investigate the effect of dynamic control of the selection pressure, which leads to balancing the exploration/exploitation trade-off. The switch between grid shapes is one way to dynamically tune the selection pressure; other ways are also possible (Ursem, 2002; Li and Kirley, 2002; Alba and Dorronsoro, 2005). Chapter 5 discusses this topic in more detail.

2.2.1.2 The Influence of Local Selection Method

In addition to *NGR*, the local selection method influences the selection pressure. De Jong and Sarma (1995) empirically studied this effect by considering standard selection methods (binary tournament, linear rank, and proportional selections). They found that different selection pressures were induced by the various selection methods and that binary tournament selection has the most desirable global search and communication overhead.

Subsequent studies investigated the effect of local selection methods as decentralising choices. The selection methods included standard and parallel-based techniques such as stochastic binary tournament, anisotropic, and centric selections (Simoncini *et al.*, 2006a; b; 2007; 2009). These methods introduced new parameters on which probabilities to select a

specific individual are computed. For example, considering the $L5$ neighbourhood, anisotropic selection assigns probabilities for the centre (p_c), north and south (p_{ns}), and east and west (p_{ew}) individuals based on anisotropic parameter ($\alpha, \alpha \in$ [-1 , 1]); see Equations (2.4).

$$p_{ns} = \frac{(1 - p_c)}{2}(1 + \alpha), \qquad p_{ew} = \frac{(1 - p_c)}{2}(1 - \alpha). \tag{2.4}$$

Hence, by tuning α, different selection intensities are induced. More discussions and experiments with respect to the influences of selection methods on selection pressure and takeover time are provided in Chapter 5, Section 5.1.

2.2.2 Synchronisation

Earlier, the topic of synchronisation was discussed in the context of dGA. This subsection discusses the same topic as it relates to cGA. In a cGA, the synchronism occurs through population updating policies (Tomassini, 2005).

In a synchronous cGA, the phases of evaluation, genetic operations, and selection take place at the same time for all cells before the next generation starts (refer to Algorithm 2.4). Implementing the synchronous cGA model on a single machine requires an auxiliary grid to keep the updated cells. The auxiliary grid then replaces the old population when all cells have been updated to start the next generation.

In an asynchronous cGA model, cells are updated in sequence. Different sequences are defined for asynchronous updating; with the most frequently used policies being fixed line sweep (LS), fixed random sweep (FRS), new random sweep (NRS), and uniform choice (UC). In LS the cells are updated successively according to their positions, either by row or by column. In FRS, each cell is selected randomly for updating with uniform probability and without replacement. Similar to FRS, NRS and UC select a cell randomly for updating; however in NRS, a new random cell distribution is used for each cell, while in UC a uniform probability with replacement (binomial distribution) is used.

Previous studies about synchronism in the field of Cellular Automata and dGA confirmed the advantages of asynchronous approaches over synchronous ones (Sipper *et al.*, 1997; Schofisch and de Roos, 1999; Alba and Troya, 2001). These findings led Alba *et al.* (2002) to investigate the respective advantages and disadvantages in synchronous and asynchronous

cGAs. Although asynchronous cGAs had faster convergence time while maintaining desirable search success rates, synchronous cGAs had higher search success rates—confirming the results of the previous studies.

Synchronisation and selection pressure

In addition to the shape and size of the neighbourhood (and/or grid) and local selection mechanisms, synchronism can influence the global selection pressure. Giacobini *et al.* (2003; 2005) investigated the selection intensity in synchronous and asynchronous cGAs. They successfully modelled the curves of the selection pressure on one- and two-dimensional cGAs with toroidal grids. Accordingly, Tomassini (2005) provided a mathematical background for understanding the models. An empirical investigation was then carried out to validate the models. Synchronous algorithms had the weakest selection intensity, followed by UC, NRS, FRS, and at finally LS, which had the strongest selection intensity. Thus, synchronous algorithms are more explorative than asynchronous ones.

Dorronsoro *et al.* (2004) further investigated the influence of synchronous and asynchronous update policies on the selection pressure. The results obtained confirmed those of previous studies in showing that it is possible to control the selection pressure without the need for additional parameters by synchronising updating policies. Moreover, asynchronous algorithms had faster convergence times than their synchronous counterparts. However, synchronous algorithms had higher search success rates.

The next subsection discusses the metrics most frequently used to measure the performance of the parallel algorithms.

2.2.3 Performance and Statistic Measures

As previously discussed, the most common measure of parallel algorithms is the Speedup. Speedups of meta-heuristics should be computed based on similar parallel and sequential accuracies (Alba, 2005). In this case, the average mean times of the parallel model on a single machine and the parallel model on *m* machines are compared in an orthodox (similar algorithm and accuracy), practical (the best, most recent algorithm is not required) manners. The definition and types of the speedup were discussed in Section 2.1.2.2.

Other metrics used to measure and analyse the performance of parallel algorithms include accuracy (quality of solution) combined with search success rate or hit rate (number of successful experiments), and computational effort (number of fitness evaluations and/or the

run time). To achieve a reliable conclusion and to gather sufficient data several independent experiments have to be carried out due to the stochastic nature of EAs. The first measure can be used if the optimum solution is known. Thus, the search success rate indicates the number of experiments that obtains the optimum solution. Knowing the optimum solution is not a necessity for the computational effort. This measure is computed using the convergence time (number of fitness evaluations or number of generations) and/or the convergence speed (execution time). Researchers recommend the use of both methods to compute the computational effort. The traditional assumption is that parallelism is mainly about reducing the time rather than the number of evaluations. However, using the execution time would bring the effects of hardware and software implementation. For empirical investigations on the influence of the measure, please refer to (Alba, 2005, p.54).

Statistical metrics are also important when measuring the performance of the algorithms. Common metrics include mean of solution accuracies and mean of computational efforts over all experiments. To illustrate the benefit of the statistical metrics consider obtaining low hit rate but with high mean accuracies, which indicates that the algorithm is robust. For global analyses, other statistical metrics such as standard deviations (or median absolute deviations, which is recommended for data with non-normal distribution) can also be used. To further assess the reliability and validity of the conclusion, significance statistical tests should be used to indicate the strength of the relation between performance measures (Alba, 2005).

Genotypic and phenotypic measures in cEAs

Capcarrere *et al.* (1999) introduced a number of statistical measures to analyse the behaviour of cEAs at the genotypic (structure of individuals) and phenotypic (fitness of individuals) levels. At both levels, the most important measure is the diversity, which can be computed using a variety of methods. The most common method is to calculate the entropy of the population based on individual fitness (phenotypic diversity) or structure (genotypic diversity). Phenotypic diversity (H_p) refers to the average number of different fitness values, while genotypic diversity (H_g) refers to the average values of the entropy of each variable (gene) in the population. See Equations (2.5) and (2.6).

$$H_g(P) = \frac{1}{N} \times -\sum_{j=1}^{N} g_j \log(g_j), \tag{2.5}$$

where N is the size of population P, and g_j is the fraction of individuals having a given distance from the origin.

$$H_p(P) = \frac{1}{N} \times -\sum_{j=1}^{N} f_j \log(f_j), \tag{2.6}$$

where f_j is the fraction of individuals having fitness j.

Population diversity plays a significant role in EAs. One of the drawbacks of panmictic EAs is their weakness in maintaining the population diversity, which causes the search to be trapped in a local optima area, particularly when tackling hard real problems. An implicit way to tackle this shortcoming is through the spatial structure of the population or the decentralised EAs (Tomassini, 2005, p.37). Besides being a main measure to analyse the performance of the algorithm, the population diversity is used to guide EAs. The use of genetic diversity in guiding search process introduces a new class of EAs, which is the dynamic model. This topic will be covered in detail Chapter 5.

2.2.3.1 Performance Measures and Statistical Tests used in this Research

This research uses the average number of generations to find a solution with a predefined accuracy for successful runs out of 100 independent runs. This measure is referred to as efficiency, or convergence time (CT). The second measure used is the efficacy or the convergence rate (CR), which defines the search success rate (% hits) to a solution of a predefined accuracy out of 100 independent runs. The final measure is the speed (SP), which is measured as the average run or execution times in seconds (s) for successful runs. In this thesis, CT appears first in a table cell, followed by CR, and then SP. The median absolute deviation (*mad*) is added to CT and SP and appears in tables after the symbol '±'. All experiments were carried out using MATLAB and GNU C compiler (Dev C++) on an Intel^{R} Core^{TM} 2 CPU at 2.4GHz with 3.12GB RAM, running Windows XP professional v. 2002.

With regard to statistical metrics and significance tests, this research uses the mean of the efficiencies and mean run times including the standard deviations (the median absolute deviations replace the standard deviations when data are not following a normal distribution). The Kolmogorov-Smirnov test is applied to identify the normality of the data. After which, the ANOVA test was applied when the data showed a normal distribution, and the Kruskal-Wallis test when the data failed to follow a normal distribution. The latter two tests are used to obtain the statistical significance in the efficiency and speed of the algorithm with a 95% confidence level (*p-value* < 0.05). For the efficacy, the Chi-square (χ^2) test for proportions was used to obtain the significance with a 95% confidence level.

2.2.4 cGAs from Hardware Perspectives

A number of comprehensive studies on parallelism in EAs were carried out to analyse the various features of the parallel EAs such as selection pressure, efficacy, efficiency, speedup, synchronism, among others, while stressing the difference between EA models and their implementation (Cantu-Paz, 1995; Alba and Tomassini, 2002).

Traditionally, dGAs are best-suited and usually implemented in distributed memory MIMD machines, while massively parallel computers (SIMD) are more suitable for the implementation of cGAs as these models can be directly mapped onto such architectures (Tomassini, 1999). Many massively parallel computers connect the processing elements (PEs) in a two-dimensional grid. However, other topologies can be implemented using a global router (Cantu-Paz, 1995). When implemented on a SIMD machine, a cGA places a single individual at each PE or cell. Each individual selects another individual from its local neighbourhood to mate with. The generated offspring may then replace the individual at the central cell according to the replacement policy. Hence, there is no need for any central control. Nevertheless, issues related to the neighbourhood topology and selection and replacement schemes should be considered when implementing a cellular model.

A cEA model may have more cells than PEs. In this case a PE has to deal with each cell sequentially. Today, the use of the theory of MPI message passing makes it easy to implement cEAs, particularly the synchronous model. Each PE synchronously updates its individuals in sequence and does not require the other's PEs memories except for communication involving edge values between neighbourhoods. In this case, those PEs need to send and receive the corresponding messages as different neighbouring regions are managed by different PEs (Tomassini, 2005, p.168).

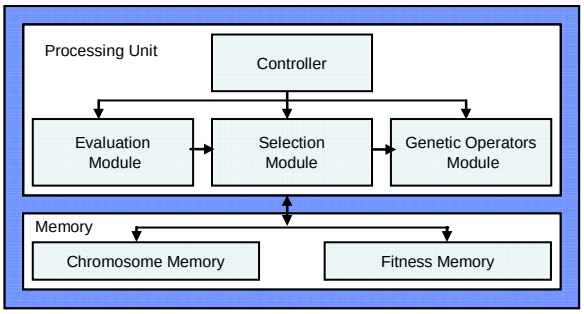


Figure 2.7. High-level hardware architecture of a cell of the cGA in the SIMD model.

In conclusion, the cGA model is well suited for VLSI implementation since the cells are simple, regular, have small local memories, need local communication links over the defined topology, and operate synchronously (Eklund, 2004). Figure 2.7 illustrates the two major components of a cGA cell in the SIMD model. They are the processing and the memory units. All cells are identical and are connected to their neighbouring cells over the defined topology (NEWS or X-net). Each cell evaluates its unique individual, selects, and applies crossover and mutation operators (i.e., perform the same cGA) in parallel with other cells.

A hardware architecture and implementation of a cGA for the application of the image registration was proposed by Turton *et al.* (1994). In the study, a two-dimensional captured image was compared to a reference image and a transformation between both images was required. From an algorithmic perspective, a step was added to the cGA model such that hill climbing was used to modify (increase or decrease) the transformation parameters by one unit for promotion. In a subsequent study, Turton and Arslan (1995b) improved the previous architecture to include data compression. They also proposed a cGA architecture for a disc-scheduling problem (Turton and Arslan, 1995a). The aim was to identify the best way to order tasks in order to minimise the access time. For more details about the previous studies please consult the cited references.

Xu *et al.* (2002a) proposed a technique based on GAs to determine the attitude of a GPS. The proposed technique resolved problems in existing methods such as the Ambiguity Function Method (AFM), making it more efficient and immune to cycle slips. The attitude parameters are determined using more than two antennas (of which one is used as a reference) attached to a vehicle. More details on the problem of GPS attitude determination are provided in Section 4.2.2. A VLSI implementation of the GPS attitude determination based on cGAs was subsequently proposed in (Xu *et al.*, 2002b). For implementation, issues related to functionality and practical performance restrictions, such as speed and scalability, were considered. The resulting architecture had low hardware complexity and the simulation results showed a linear speedup.

Later, Stefatos and Arslan (2004a) introduced a high performance, adaptive hardware architecture to alleviate the problem of GPS attitude determination based on cGAs. The aim of the proposed system was mainly to optimise the speed performance. As a result, the cGA employed a Coordinate Rotation Digital Computer (CORDIC) algorithm to further improve the system throughput rate. Results showed the system's potential to achieve the promised high throughput rates.

The previous studies showed different application areas of cGAs. These areas are extended to include the design of fault-tolerant systems in (Stefatos and Arslan, 2004b) and (Hounsell and Arslan, 2001). EAs and GAs in particular are also involved in the field of fault tolerance. GAs have been adopted to develop fault tolerant mechanisms by combining them with reconfigurable hardware devices. This combination leads to the concept of the evolvable hardware (EHW). For studies on EHW refer to (Thomson and Arslan, 2002; 2003; 2005; Stefatos, Arslan, and Hamilton, 2008).

Hounsell and Arslan (2001) presented a fault-tolerant system based on the EHW platform for the automated design and the adaptation of multiplierless digital filters. Filters were achieved using a dedicated programmable logic array (PLA). Three PLA initialization methods were investigated to identify the best fault recovery time. Results showed the ability of PLA to maintain the system's functionality despite an increasing number of faults reaching to 25% of the PLA area.

Stefatos and Arslan (2004b) further enhanced the GPS architecture to include a fault tolerant technique. This novel architecture consisted of two layers. The first layer related to the application while the second monitored the performance of the first layer and reconfigured its computational elements when appropriate. The class of faults considered in the study was Single Event Upsets (SEUs), which primarily originate from radiation effects (more details about SEUs are provided in Section 2.3). Results showed the capability of the first layer to tackle faults up to 40% of the PEs, while the second layer tackled up to 30% of faults.

2.2.5 3D-cGAs: Pseudo-code and Specification

This section emphasises the implementation of a cGA on three-dimensional (3D) toroidal grid. Previous studies focused on implementing cGAs on one-dimensional (1D), or most commonly, two-dimensional (2D) toroidal grids. Consequently, there is a lack of studies related to higher cellular dimensions. The research in this thesis is based on three-dimensional cGAs (3D-cGAs). Higher cellular dimensions show promising benefits at both hardware and software levels.

The previous discussion emphasised the importance of grid topology in determining the performance of the algorithm. In this research, a 3D cubic topology is utilised. A cubic topology allows good solutions to spread quickly to all PEs due to its shorter diameter (Cantu-Paz, 1995), as well as diverse degrees of exploration and exploitation.

In past works (Breukelarr and Back, 2005; Morales-Reyes *et al.*, 2009), a 3D architecture was utilised and investigated. The overall results showed improvements in the performance of the algorithm when compared with smaller grid dimensions. A further reason for using the 3D topology is its amenability to be implemented with new advanced custom silicon chip technologies to achieve added significant benefits, such as fast operation, reduction in power consumption, new design possibilities, heterogeneous integration, circuit security, and wide bandwidth (Das *et al.*, 2003).

Algorithm 2.5 Pseudo-code for a canonical 3D-cGA

```
1. procedure cGA
2. Generate_initial_population (P(0));
3. Evaluation (P(0));
4. t ← 0;
5. while ! stop_condition do
6.    for i ← 1 to ROWS do
7.       for j ←1 to COLUMNS do
8.          for k ←1 to LAYERS do
9.           neighbours ← Find_neighbours (position(i,j,k));
10.          parent1← position(i,j,k);
11.          parent2 ←Local_selection (neighbours);
12.          offspring ← Recombine (Pc, parent1, parent2);
13.          offspring ← Mutate (Pm);
14.          Evaluation ←Fitness(offspring);
15.          Replacement (position(i,j,k), offspring, Paux(t));
16.      end for;
17.    end for;
18.   end for;
19. P(t+1) ← Paux(t); // updating
20. t ← t+1;
21. end while;
22. end procedure cGA;
```

The pseudo-code for the 3D-cGA is shown in Algorithm 2.5, in which similar steps to other (lower or higher) cellular dimensions are followed. The steps are for finding the

neighbours (line 9), selection of parents (lines 10 and 11), recombination of selected parents (line 12), mutation of offspring (line 13), evaluation of offspring (line 14), and replacement (line 15). In comparing Algorithm 2.4 (2D-cGA) with Algorithm 2.5, the only difference is the addition of a third dimension (Line 8), which refers to the layers of the grid. An empirical study along with a detailed discussion of 3D-cGA is provided in Chapter 3. The next subsection empirically compares 3D-cGAs to panmictic GAs, while a comparison with 2D-cGAs is provided in Chapter 3.

2.2.5.1 3D Cellular versus Panmictic GAs

An experimental study was carried out in order to demonstrate the behaviour and the performance of the 3D-cGA with respect to panmictic algorithms (ssGA and genGA). The test bench selected to evaluate the algorithms included the problems of Rastrigin, Schwefel, Griewangk, Ackley, Michalewicz, Langermann, FMS, and SLE. The dimension of these problems consists of 10 variables, except for FMS (details about the problems are provided in Appendix A). The parameters used in all the experiments are summarised in Table 2.1. The population consisted of 343 individuals. One hundred independent runs were performed, allowing a maximum of 500 generations for each experimental case. The algorithms terminated when the difference between the average fitness values (*avgf*) and the optimum fitness value (*optf*) satisfied a specified threshold, or when the maximum number of generations was reached. Different thresholds were assigned for each problem based on its complexity. For all algorithms a non-uniform mutation and blended crossover operators were applied to generate offspring.

Table 2.1. Experimental parameters used for 3D-cGA, ssGA, and genGA

Population size:	343 individuals
Parent selection:	Current individual + BT (for 3D-cGA) BT + BT (for ssGA and genGA)
Recombination:	BLX-α (α = 0.5), P_c = 0.9
Mutation:	Non-uniform, P_m = 0.1
Replacement:	Replace-if-better
Neighbourhood:	NEWS
Lattice:	7×7×7 (for 3D-cGA) 1×343 (for ssGA and genGA)
Stop criterion:	$\|avgf - optf\| \leq Threshold$

For the 3D-cGA, the population was arranged over a 7×7×7 toroidal lattice with a NEWS neighbourhood containing the central individual plus those linearly positioned at one distance step. The first parent was always the central one, while the second parent was selected from the neighbourhood using BT selection. For panmictic GAs, the two parents were selected from the whole population using BT selection. For genGA, the size of the auxiliary population was equal to the size of the population ($\lambda = \mu$).

The algorithm performance measured as convergence time, rate, and speed are reported in Table 2.2, with the best values marked in **bold** (for more details about the performance metrics refer to Section 2.2.3.1).

Overall, 3D-cGA outperformed the panmictic GAs in terms of convergence rate as it had the best search success rates for 6 out of 8 problems (see Table 2.2), while it achieved the second-best convergence times and speeds following the ssGA. For Rastrigin's and Ackley's problems, all algorithms achieved almost similar efficacies. However, for more complex problems such as Langermann and FMS, 3D-cGA achieved significantly higher hit rates than

Table 2.2. Comparing 3D cellular to panmictic GAs' performances: Convergence time (CT), rate (CR), and speed (SP)* for test and real-world problems

Algorithms/ Problem	ssGA	genGA	3D-cGA
f_{Ras}	**128.11** ± 11.33 **100%** **14.23** ± 1.26	430.24 ± 00.0 99% 34.11 ± 0.98	323.77 ± 19.0 **100%** 49.81 ± 0.78
f_{Sch}	**70.01** ± 7.18 **100%** **7.62** ± 0.65	408.80 ± 1.0 73% 31.96 ± 0.43	200.96 ± 16.0 **100%** 21.01 ± 1.61
f_{Grie}	**72.45** ± 3.54 45% **7.29** ± 0.45	410.6 ± 1.0 **100%** 33.83 ± 0.70	290.51 ± 24.0 45% 30.91 ± 2.53
f_{Ack}	**77.66**± 0.95 **100%** **7.20** ± 0.11	472.90 ± 00.0 **100%** 42.02 ± 0.49	221.62 ± 1.0 **100%** 25.39 ± 0.42
f_{Mic}	**130.34** ± 13.9 **98%** **12.35** ± 1.47	– 0%	330.91 ± 17.0 37% 35.40 ± 1.68
f_{Lang}	**61.15** ± 7.19 61% **6.96** ± 0.85	– 0%	201.06 ± 14.0 **99%** 25.61 ± 1.76
f_{FMS}	**50.28** ± 4.60 67% **6.68** ± 0.57	399.20 ± 2.0 5% 49.81 ± 0.78	207.83 ± 17.0 **91%** 33.08 ± 2.40
f_{SLE}	– 0%	**413.00** ± 00.0 **5%** **34.37** ± 0.03	445.00 ± 6.00 **5%** 47.36 ± 0.75

* For more details about the performance measures, please refer to Section 2.2.3.1.

panmictic GAs. The difference in efficiencies and speeds may correspond to the variation in the achieved hit rates. For Schwefel and Griewangk, 3D-cGA and ssGA performed similarly in terms of hit rate, while the latter significantly outperformed the former by obtaining a lower number of generations reaching to 75%. The 3D-cGA showed its ability to solve all the problems, while genGA failed to solve Michalewicz and Lanagermann problems and ssGA failed to solve the SLE problem. However, the hit rate obtained for SLE was very low.

Section 2.2.4 provided a broad overview on the application of cGAs, particularly in the field of the fault tolerance. The next subsection provides more detailed overview on the topic of fault tolerance.

2.3 Fault Tolerance

The increasing use of electronic systems in critical areas such as space and medicine increases the importance and needs for reliable systems to remain functioning with the existence of failures. Systems operated in aggressive environments including space, ground, and water or where human life depends on their accurate functioning have to be fault-tolerant. Therefore, fault-tolerant systems can be defined as the ability of a system to operate correctly in spite of hardware and/or software failures (Avizienis, 1971).

This thesis focuses on radiation-induced failures; such failures are known as Single Event Effects (SEE). SEE errors occur when a system interact with high-energy particles at space level or low-energy particles at ground level (Label, 1996; Gong *et al.*, 2008). SEE are classified into hard errors and soft errors (Mastipuram and Wee, 2004). Hard errors are known as Single Event Latch-Ups (SELs), while soft errors are known as Single Events Upsets (SEUs); this research explores the effect of SEU errors.

In the nineteen-seventies, SEUs (also known as transient errors) were discovered in space (Normand, 1996). Systems operated in space are subjected to various anomalies including plasma and radiation, among others. Such anomalies have effects on systems, which result in different types of failures. Avionics (i.e., electronics in aircraft) SEU was first predicted in the nineteen-eighties and later severely demonstrated to occur in flight in the nineteen-nineties (Normand, 1996). Consequently, attention was paid to the radiation effects because the radiation was the main contributor to failure (45%), with SEUs having the highest impact of all possible radiation effects (80%) (Velazco *et al.*, 2005). In addition, the considerable reduction in the feature sizes of electronic circuits and increase in functional complexity and sensitivity increases the possibility of transient errors occurring (Normand, 1996).

Subsequently, radiation-induced SEUs have also been observed at ground level (Gong *et al.*, 2008. For these reasons, the demand for implementing efficient, reliable high-performance systems that can quickly adapt to different failures is a crucial concern. This is usually accomplished by a residual design that is resistant to, and tolerant of failures. To achieve fault tolerance, two essential processes must be considered, they are: fault detection and fault recovery (Greenwood, 2005; 2008). This section presents the major causes of system failures considered in this research.

SEUs occur as single-bit (SBUs) or multiple-bit (MBUs) flip in memory or data registers due to the passage of one or more energetic radiation particles (Mastipuram and Wee, 2004). SEUs do not cause permanent damage to system functionality, and can be handled by fault-tolerant techniques. There are various algorithms and approaches to fault tolerance are introduced including hardware techniques, software techniques, or a combination of both (Su and Spillman, 1977).

The most commonly used hardware technique to mitigate SEUs is Triple Modular Redundancy (TMR) (Layons and Vanderkulk, 1962; Lala, 1985). However, TMR is very area-extensive (general) and may not be able to cope with all the errors that occur. SEU hardware fault-tolerant techniques can rapidly detect and recover faults; however, they incur overhead, which increases the cost and complexity of the design. Further, in general, hardware techniques cannot handle all types of random and multiple-bit errors caused by potential transients (Pant and Joshi, 2007). These types of errors, specifically SEUs, cause functional impacts (software faults), rather than physical impacts. Consequently, many error-coding techniques have been proposed to solve the above-mentioned problems; however, they are seldom implemented due to their complexity.

Nowadays, fault-tolerant techniques to mitigate SEUs are being intensely researched, not only for aerospace applications, but also for terrestrial applications. Gong *et al.* (2008) proposed a hardware approach for tolerance to SEEs where two new structures were presented and compared with the traditional TMR. For a thorough discussion of SEEs, please refer to (Label, 1996). Conversely, Singh *et al.* (2006) presented a software approach to SEU tolerance that combined several techniques, such as checkpoint and TMR.

Pickle (1996) and Asenek *et al.* (1997) proposed a model to predict the rate of SEEs; however, the latter emphasised the SEU errors at system level rather than at device level. Asenek *et al.* analysed a telecommand system on a spacecraft; and found that around 50% of the SEUs that occurred resulted in errors observed at the system level.

In addition, several studies to explore the ability of cGAs to tackle SEUs have been conducted. Research studies related to the ability of a normal cGA and a parallel cGA to deal with SEUs that occur at fitness score registers were presented in (Morales-Reyes *et al.*, 2008a; 2009), while the ability of an adaptive cGA to handle SEU-targeted chromosomes registers was explored in (Morales-Reyes *et al.*, 2008b). In all the previous studies, EAs have proved their capability and power to tackle SEUs, as well as in improving the performance of the algorithm in terms of efficacy and efficiency.

Chapter 4 of this research deals with failures caused by SEUs when targeting individuals' phenotypes, particularly when fitness scores are stuck at 'one' or 'zero'. Although other possible memory or data registers, such as chromosome and finite-state machine (FSM) could be also targeted, this research focuses on fitness value registers due to the importance of fitness information in guiding the search.

2.4 Chapter Summary

This chapter covered previous studies and research on GAs in general and cGAs in particular. As mentioned previously, the focus of this thesis is on cGAs. As a result, more attention is paid to studies relating to cGAs. A review of the literature has shown only limited research on cellular dimensionality, in particular above two dimensions, although previous studies have shown that a grid topology is a key that determines the performance of GAs. Typical cGAs are implemented on 2D grid topology, while this thesis focuses on 3D grid topology. Preliminary research on grid dimensionality that were carried out in joint collaboration with another group member, showed promising results for higher grid dimensions, particularly 3D, that can lead to better performance. Hence, the work in this thesis is based on 3D cGAs.

Another area of interest is fault tolerance. Literature review showed major concerns by researchers about the effect of SEE errors on systems functionality. As a result, hardware- or software-based mitigating and fault tolerance techniques were intensely researched. Previous research in this area showed that SEEs, and in particular SEUs, affect systems functionality. However, studies on algorithm-based fault tolerant techniques are lacking. In addition, previous studies that were carried out by other group members investigating the ability of cGAs to tackle SEE errors show that cGA is capable of handling such errors. Therefore, this thesis aims to develop algorithm-based fault tolerance and mitigation techniques to tackle SEUs.

Moreover, previous researches were intensely targeted with the vital issue of exploration/exploitation trade-off, which also determines the effectiveness of GAs. These researches suggest that for GAs to continue performing effectively when tackling real-world problems, it should be adaptive. Previous researches proposed several techniques to dynamically control the exploration/exploitation trade-off in an adaptive manner and incurred lower computational costs. Consequently, this thesis also aims at developing effortless adaptive cGAs that dynamically control exploration/exploitation trade-off.

Chapter 3

3D Architectures

This chapter aims to investigate the behaviour and performance of cGAs when the cellular dimensionality is increased to 3D. Two-dimensional topologies are commonly employed in cGA investigations; however, this research employs 3D topology. In this chapter, a comparison between cGAs implemented on 2D with those implemented on 3D grids will be provided to show the advantages of increasing cellular dimensionality. The main reason for increasing cellular dimensionality is its amenability to being implemented efficiently with the new advanced custom silicon chip technology, in particular 3D integration technology (Das, Chandrakasan, and Reif, 2003; Topol *et al.*, 2006; Borkar, 2011).

Parallel EAs in general and cGAs in particular, offer a structure that establishes a powerful connection between both software and hardware levels, while offering high system's performance. In a cGA, the population is distributed over an *n*D grid structure with wraparound edges following a toroidal shape, such that each individual is assigned to a grid's position or a cell. This arrangement restricts the interactions between individuals within their defined local neighbourhoods. In this study, the population is arranged in a 3D toroidal grid. Therefore, the defined local neighbourhood consists of the central, the vertical north and south, the horizontal north and south, and the east and west individuals or NEWS (see Figure 3.1(b)).

cGAs offer numerous benefits over other GA models, in particular panmictic GAs. These benefits can be summarised as follows. First, an ability to maintain a high diversity level for much longer time in comparison with centralised models (Cantu-Paz, 2000). Second, an ability to achieve not only better efficiency, but also higher efficacy in combination with the accuracy of results.

Reproduction between individuals occurs when one individual is selected from a small neighbourhood composed of individuals located at a short distance and then mated with the chromosome currently being evaluated. This procedure is repeated for each individual at each grid position and in general all individuals can be updated either synchronously or asynchronously. Thus, it is necessary to define the number of individuals belonging to the local neighbourhood and within which radius or distance step would be contained. This also establishes a relationship between the size of the population, the shape of the grid, and size/shape of the local neighbourhood and the consequent effect in the search process.

In order to provide a thorough study of the behaviour of cGAs and model their performance, the dimension of the cellular is increased from 2D to 3D; which is the main objective of this study. Comparing 2D square and 3D cubic grid topologies while maintaining similar processing and interaction constraints among individuals will offer a wider overview of the effectiveness of cGAs as optimisation engines.

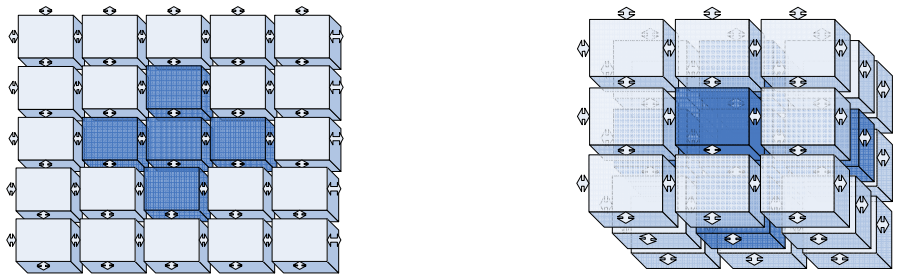
Breukelaar and Back (2005) did a study on evolving behaviour in multi-dimensional cellular automata using a GA. In that study different parameters in terms of crossover rate, mutation rate, number of iterations, tournament size, neighbourhood size, and cellular dimension (1D, 2D, and 3D) were explored. Three different problems—the majority, checkerboard, and evolving bitmaps problems—were solved in order to explore the potential of cellular automata. The overall results showed that with a multiple cellular dimension topology, in particular 3D, GA achieved a lower number of iterations and fitter objective values. However, there is a considerable difference between the number of individuals defined for 2D and 3D cellular automata. The conclusion drawn suggested that GA be used with multi-dimensional cellular automata as this combination shows great potential for effectively solving real-world problems.

A preliminary study that investigated and compared 2D and 3D cGAs was carried out in collaboration with another group member in SLIg (Morales-Reyes *et al.*, 2009). In that study, various population sizes and local neighbourhood radii were explored while maintaining similar population sizes for both grid dimensions. Four test functions with two each having similar characteristics were solved in order to investigate the effectiveness of increasing cellular dimensionality. The test functions used were Rastrigin, Schwefel, Ackley, and Griewangk (test function details are provided in Appendix A). Simulation results showed that 3D-cGA is more efficient than 2D-cGA in terms of convergence time, particularly when solving harder problems (i.e., Ackley and Griewangk). With regard to search success rate, both cellular structures achieved similar hit rates, however 3D-cGA had

a higher search success rate than 2D-cGA when a smaller local neighbourhood distance was applied.

This chapter extends this previous study in order to further investigate the performance of cGAs and obtain a wider overview. Like the prior study, different population sizes and local neighbourhood distances were explored while maintaining similar population sizes for 2D and 3D structures. However, a benchmark of six test functions and two real world problems were selected to offer diverse characteristics. In addition, a higher problem dimension was selected. The problems chosen were Rastrigin (f_{Ras}), Schwefel (f_{Sch}), Griewangk (f_{Grie}), Ackley (f_{Ack}), Michalewicz (f_{Mic}), Langermann (f_{Lang}), FMS (f_{FMS}), and SLE (f_{SLE}) (details are provided in Appendix A).

This research aimed to explore the benefits of 3D structures on cGAs at software (algorithmic) level and combine them with the benefits of the recently developed 3D integration technology. Recent advances in this area have presented optimistic results at the hardware level; therefore, combining the algorithmic approach of implementing 3D-cGAs as optimisation engines, in order to solve hard real time problems, would bring together the advantages that 3D integration technology has provided. Although 3D integration technology is not yet widely commercial, it is considered to be the future of coarse- and fine-grained reconfigurable architectures (Yarema, 2006; Xie and Ma, 2008). Moreover, 3D integration technology offers the following benefits: reduction of the routing length, decrease in interconnection delays, which affects not only the size of a fabric but also the performance of a device. In addition, a significant improvement in terms of logic and memory density has been reported. With respect to logic density, for fine-grained devices, it has been determined that $80\% - 90\%$ of their area is used for reconfigurable interconnections. Using 3D integration technology this ratio is reduced to $25\% - 60\%$ (Rahman and Das, 2003).



(a) (b)

Figure 3.1. (a) 2D *square* and (b) 3D *cubic* toroidal topologies when implemented in a cGA. A possible Von Neumann neighbourhood is marked in dark blue.

Future adaptive systems should offer characteristics such as fast adaptation, autonomous behaviour and fault tolerance. cGAs have been shown to be adaptive as well as fault tolerant for specific applications (Stefatos and Arslan, 2004; Morales-Reyes *et al.*, 2008). 3D cellular architectures will offer the added advantage of speed and package density. This study explores several 3D-cGA architectures and compares these to their 2D counterparts. Afterwards, a brief analysis of communication and computational complexities for both topologies is provided. In the next section the various cGA configurations for 2D and 3D topologies are provided.

3.1 Algorithm Configuration

cGAs are frequently implemented on 1D or 2D grid topologies. This study targets 3D grid topology as, with the recent advance in custom silicon chip technology, it can now be implemented efficiently. Moreover, this study compares the performance of cGAs when implemented on 2D and 3D grid topologies. A number of cGA configurations are defined in order to thoroughly investigate the effectiveness and compare fairly the performance of both topologies. Figure 3.1(a) and (b) illustrate a square grid shape in 2D and a cubic grid shape in 3D, respectively.

Several population sizes are defined for both grid dimensions. For 2D grids, the population is arranged as 5×5, 8×8, 11×11, 15×15, and 19×19, leading to a total of 25, 64, 121, 225, and 361 individuals, respectively. Conversely, for 3D grids the population is arranged as 3×3×3, 4×4×4, 5×5×5, 6×6×6, and 7×7×7, leading to a total of 27, 64, 125, 216, and 343 individuals, respectively. These sizes were selected to produce almost equal population sizes for both grid dimensions.

Although the neighbourhood topology considered is linear for both grids, the size of the neighbourhood differs based on the dimensions of the grid. Two different neighbourhood radii for both grid dimensions are defined. Considering one step distance from the central cell results in 4 neighbours, positioned to the north, east, west, and south, with a radius of 0.89 for the 2D grid; while it results in six neighbours, positioned to horizontal north and south, vertical north and south, east, and west, with a radius of 0.925 for the 3D grid.

Experiments were carried out to show the effect of grid dimensions and neighbourhood size on *NGR* and growth rate of the best individual. Figure 3.2 shows *NGR* considering two different radii and grid dimensions. Smaller neighbourhood size leads to smaller *NGR* than larger ones, which in turn decreases as the population size increases. Low *NGR* implies

weak global selection intensity and therefore promotes more exploration (Sarma and De Jong, 1996; Alba and Troya, 2000, Alba and Dorronsoro, 2008). As shown in Figure 3.2,

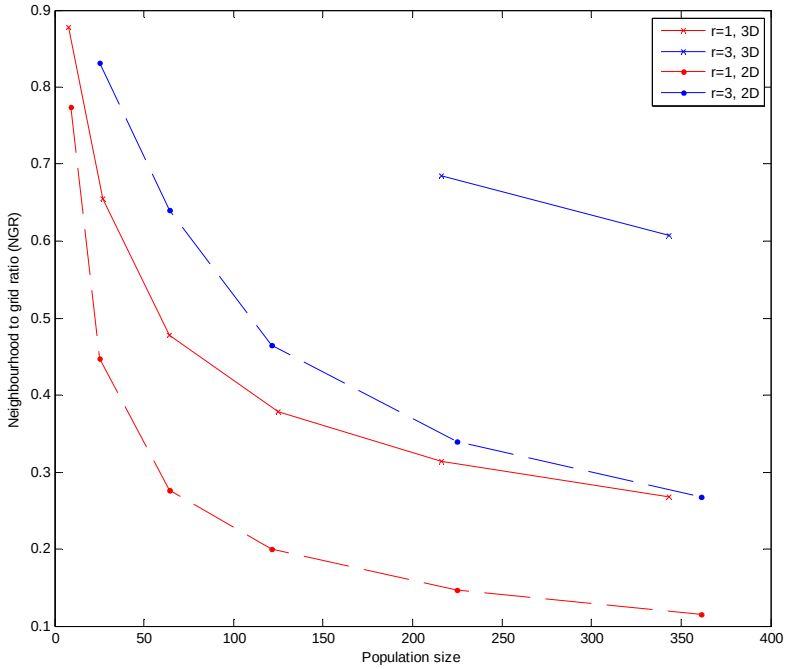


Figure 3.2. 2D/3D neighbourhood to grid ratio (*NGR*) versus population size.

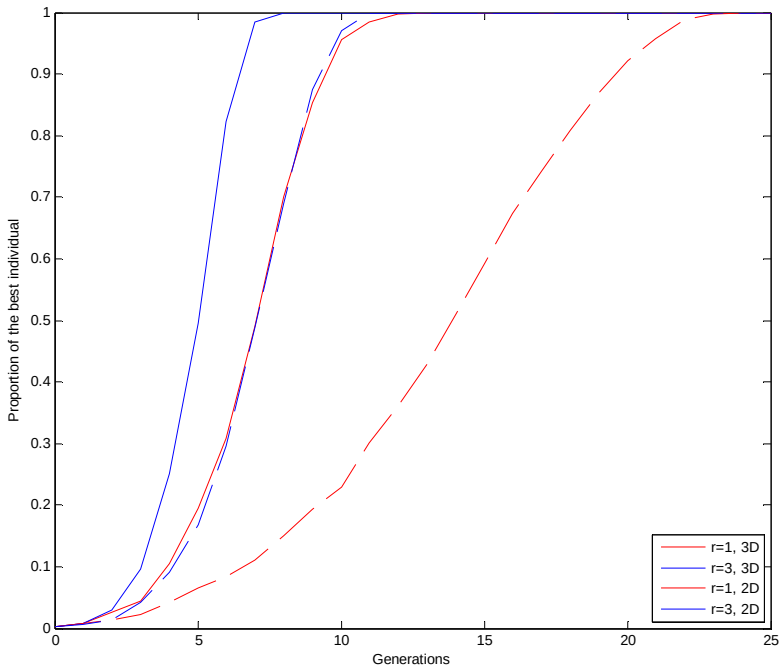


Figure 3.3. 2D/3D growth curves of the best individual.

NGR is not evaluated for a population size of 5× 5 (2D) individuals with three distance steps due to the low grid dimensions. Advancing three distance steps from an individual would result in an increase in its selection probability. Similarly, *NGR* is not computed for population sizes less than 5× 5× 5 (3D) individuals.

Figure 3.3 shows the growth curves of the best individual for population sizes 19×19 (2D) and 7× 7× 7 (3D) considering one and three distance steps. As the curves imply, 2D-cGA with one distance step has the slowest growth rate, while 3D-cGA with three distance steps has the fastest growth rate. 2D-cGA with three distance steps (*NGR* = 0.2679) produces an almost similar growth curve to the 3D-cGA with one distance step (*NGR* = 0.2673) due to similar *NGR* (Sarma and De Jong, 1996).

The pseudocodes of the canonical 2D-cGA and 3D-cGA were presented in Chapter 2, Section 2.3.4. The parameters used in the experiments and the experimental results are presented in the following section.

3.2 Experimental Results and Analysis

In order to achieve fair comparison, similar parameters were used during the experiments. Table 3.1 summarises these parameters.

Table 3.1. Parameterization used in the experiments

Population size	25, 64, 121, and 361 individuals (for 2D) 27, 64, 125, 216 and 343 individuals (for 3D)
Parent selection	Current individual + Binary Tournament
Recombination	BLX–α (α = 0.5), P_c = 0.9
Mutation	Non-uniform, P_m = 0.1
Replacement	Replace-if-better
Neighbourhood	*L5* and *L9* (for 2D-cGA) *L7* and *L13* (for 3D-cGA)
Lattice	5×5, 8×8, 11×11, 15×15, and 19× 19 (for 2D-cGA) 3×3×3, 4×4×4, 5×5×5, 6×6×6, and 7×7×7 (for 3D-cGA)
Stop criterion	$\lvert avgf - optf \rvert \leq Threshold$

The first parent was the current individual while the second parent was selected by using binary tournament selection. A blend crossover operator (BLX-α) with probability P_c = 0.9 was applied to generate an offspring (Herrera and Lozano, 2000; Dorronsoro and Alba, 2006). The offspring was then mutated by applying a non- uniform mutation operator, with

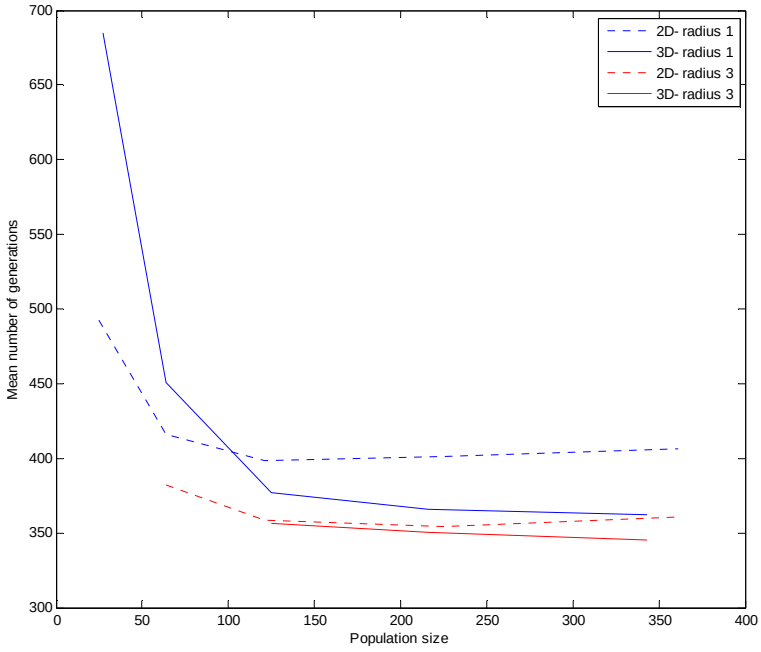


Figure 3.4. Average number of generations for f_{Ras}.

probability P_m = 0.1. The replacement policy defined here is *replace-if-better*, during which the current individual is replaced if its competitor (offspring) is fitter. Finally, the algorithm terminates if the difference between the average fitness values (*avgf*) and the optimum fitness value (*optf*) satisfies the defined threshold. Because of the different characteristics, different thresholds were defined for each problem. Similarly, the maximum number of generations assigned was 1000 generations for f_{Ras}, f_{Mic}, f_{Lang}, and f_{SLE}, while a number of 1500 generations was assigned for f_{Sch}, f_{Grie}, f_{Ack}, and f_{FMS}. The dimension of the considered problems was n = 10, except for f_{FMS} as the dimension was n = 6.

The shape of the local neighbourhood follows a linear topology with distance steps r = 1 and r =3 leading to a total of 5 (2D) / 7 (3D) and 9 (2D) / 13 (3D) individuals, respectively. The radii of the neighbourhood were 0.8944/ 2.0755 and 0.9258/ 2.1026 for 2D and 3D, respectively. These radii were selected to be almost similar for both topologies considering the same distance steps. The slight differences between the radii is due to a grid connection which assigned six neighbours in the 3D grid instead of four neighbours in the 2D grid considering one distance step (Breukelaar and Back, 2005).

The performance of the algorithms was measured using two metrics–the average number

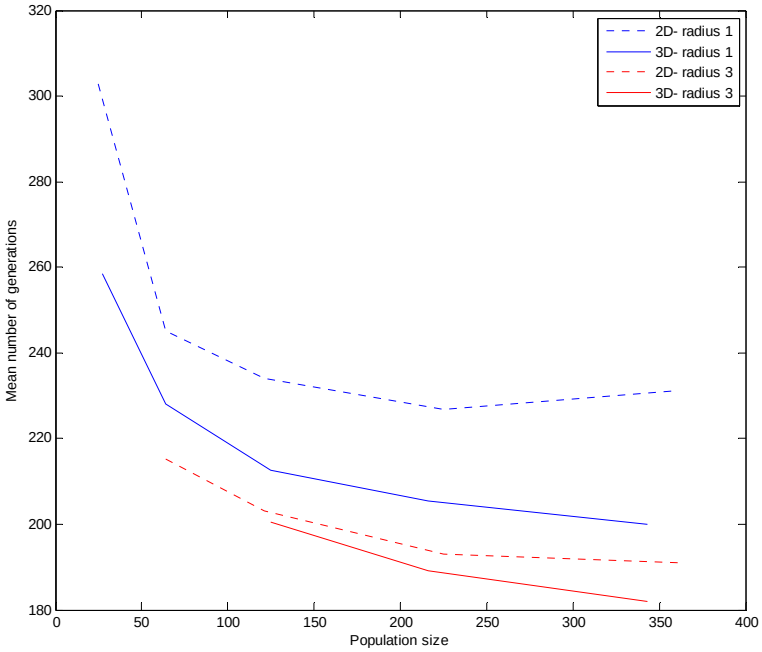


Figure 3.5. Average number of generations for f_{Sch}.

of generations or efficiency and the search success rate in combination with results accuracy or efficacy of 100 independent runs. Figures 3.4 to 3.10 illustrate the results obtained.

Figure 3.4 shows the average number of generations obtained while solving f_{Ras}. The results obtained show that 3D-cGA achieved a lower number of generations for population sizes greater than 64 individuals for r = 1. On the other hand, both algorithms (2D and 3D cGAs) obtained almost similar number of generations for r = 3. With regard to search success rate, all configurations achieved the best hit rate (100%), except for 2D-cGA with 5×5 individuals, which had a hit rate of 96%.

In Figure 3.5 the average number of generations obtained when solving f_{Sch} is illustrated. Significantly lower number of generations was achieved by 3D-cGA for the different configurations, in particular for r = 1. With regard to the search success rate, all the configurations compared achieved similar hit rates (99%–100%), except for those with small population sizes (5×5and 3×3×3 individuals) as there is a slight difference between the obtained hit rates (79% and 84%, respectively). Slight differences between the results obtained correspond to the slight differences between the population sizes of 2D and 3D lattices. Another reason is the difference in the local selection intensity, which was affected by the size of the neighbourhood, leading to a different exploration/exploitation trade-off.

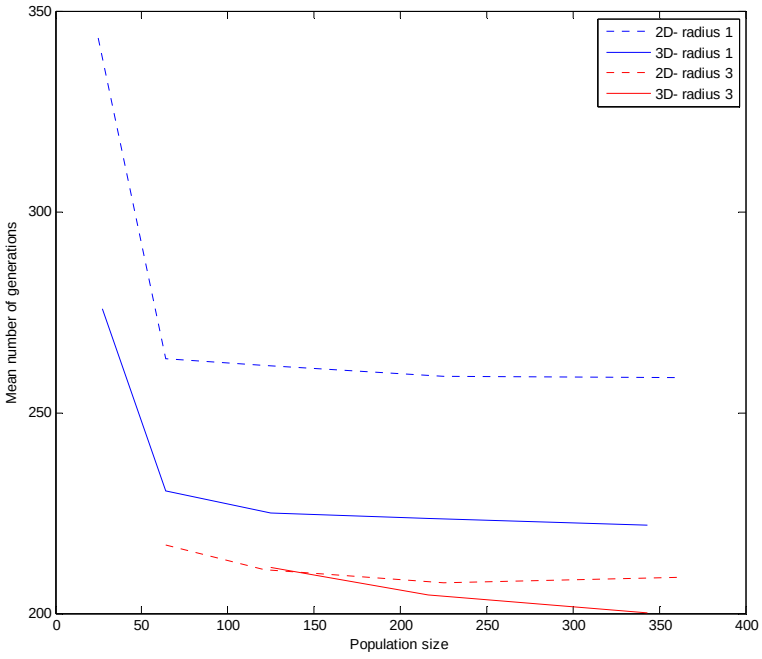
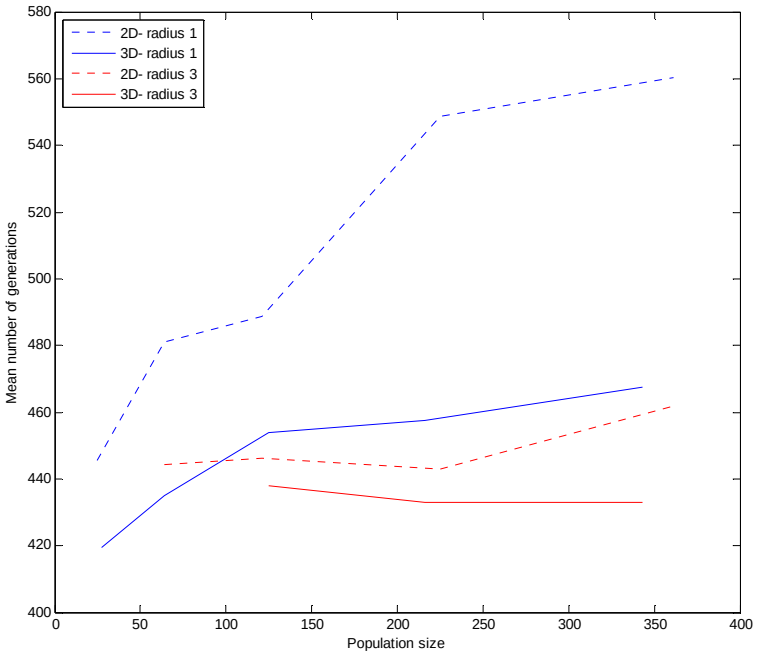


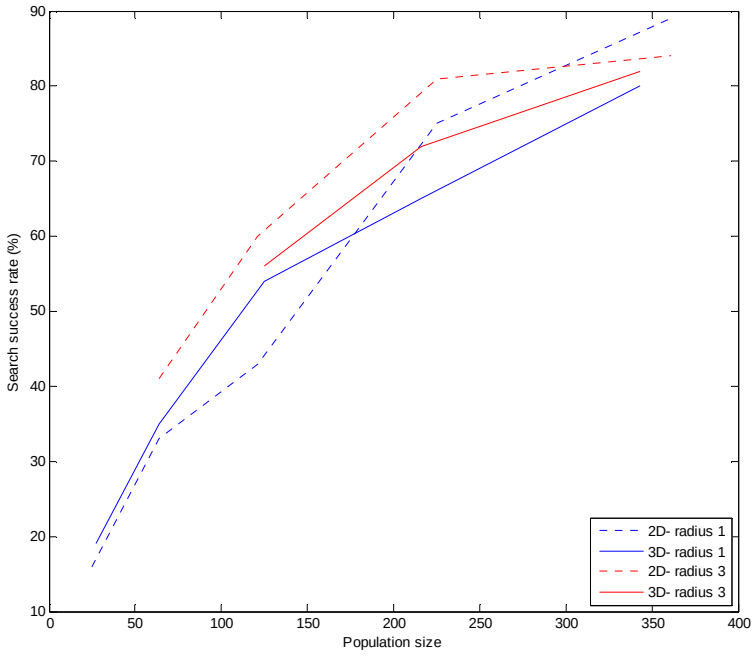
Figure 3.6. Average number of generations for f_{Ack} .

Figure 3.6 shows the average number of generations obtained while solving f_{Ack}. With respect to mean number of generations, a similar profile as that for the previous problem (f_{Sch}) was obtained. On the other hand, the best hit rate (100%) was obtained by all cGA configurations.

The average number of generations and search success rate obtained while solving f_{Mic} is shown in Figure 3.7(a) and (b), respectively. As can be seen in Figure 3.7(a), in contrast to the previous problems, the mean number of generations obtained increased as the population size increased due to the problem characteristics. f_{Mic} differed from the previous problems as it is not symmetric, which further complicates the search. In general, 3D-cGA significantly outperformed 2D-cGA. A significant difference between the mean numbers of generations was obtained for r = 1, especially for large population sizes; this improvement decreases for r = 3. In general, considering the search success rate (see Figure 3.7(b)), 2D-cGA achieved higher hit rates than 3D-cGA; however, the differences between the hit rates obtained by both algorithms were not significant.

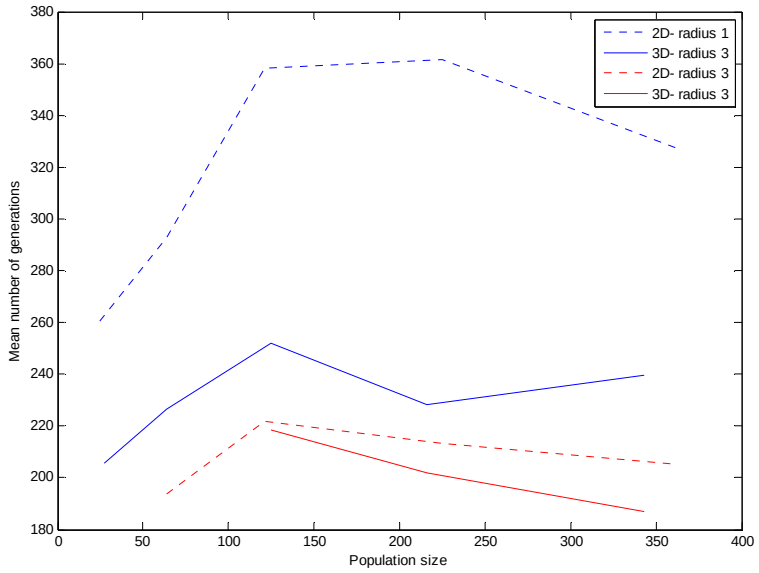


(a)

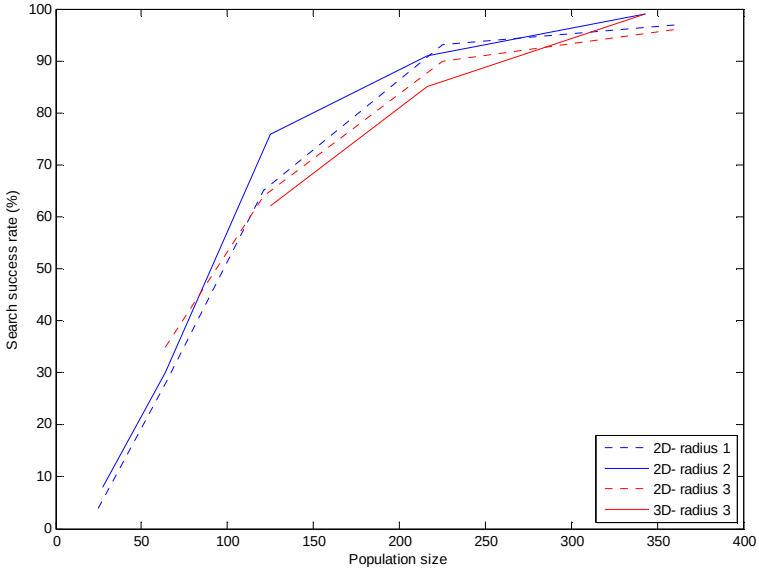


(b)

Figure 3.7. (a) Average number of generations and (b) search success rate for f_{Mic}.



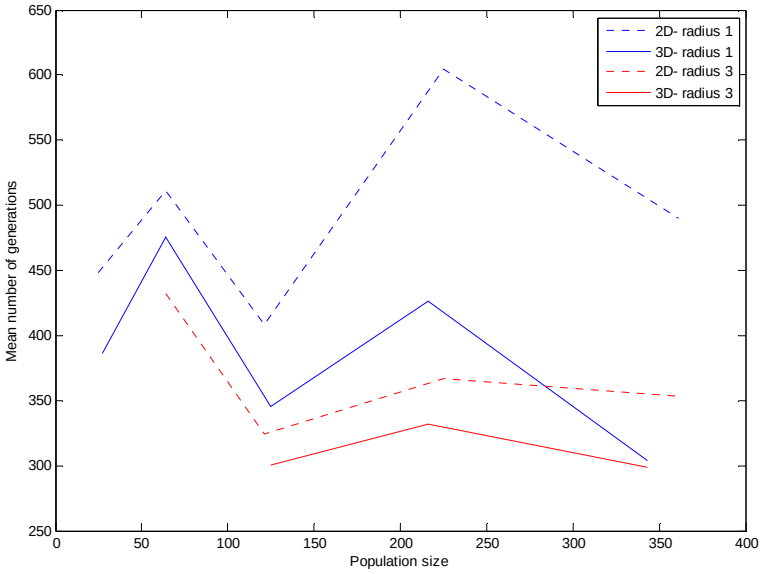
(a)



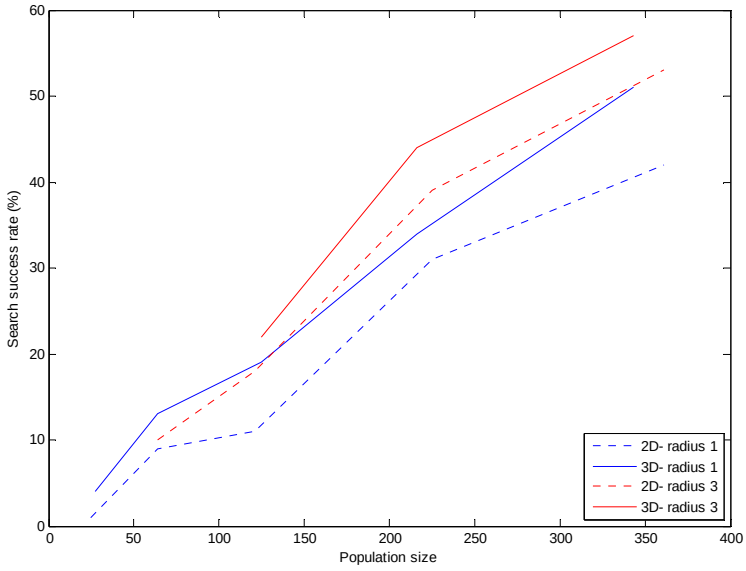
(b)

Figure 3.8. (a) Average number of generations and (b) search success rate for f_{Lang}.

For f_{Lang}, 3D-cGA achieved a significantly lower average number of generations than 2D-cGA, especially for r = 1 (see Figure 3.8(a)). However, both algorithms achieved almost similar search success rates (see Figure 3.8(b)).



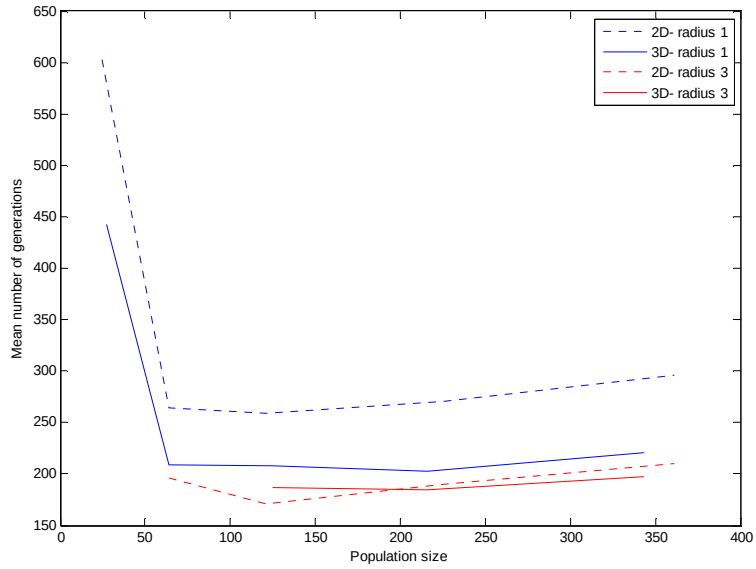
(a)



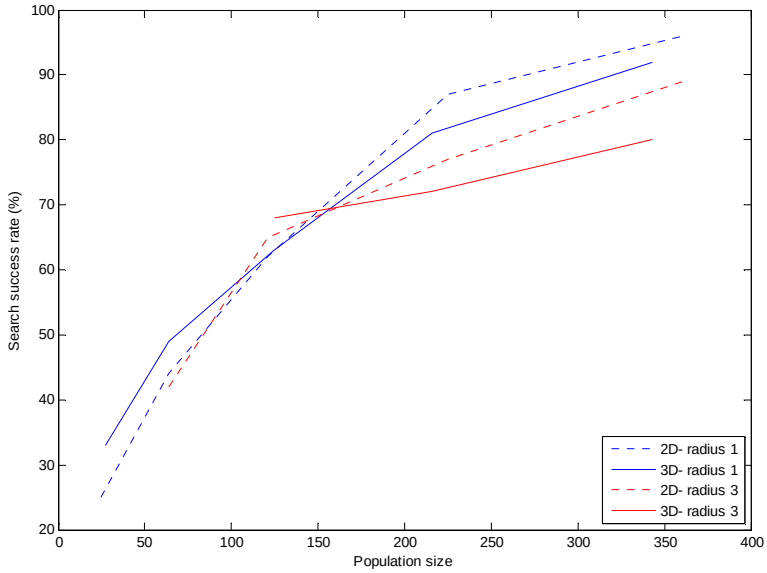
(b)

Figure 3.9. (a) Average number of generations and (b) search success rate for f_{Grie}.

With respect to the average number of generations, a similar profile as that for the previous problem was obtained while solving f_{Grie} (see Figure 3.9(a)). As can be seen in Figure 3.9(b), 3D-cGA achieved higher search success rates than 2D-cGA with similar distance steps for all the considered population sizes. In addition, it can be seen that the hit



(a)



(b)

Figure 3.10. (a) Average number of generations and (b) search success rate for f_{FMS}.

rates obtained by 3D-cGA for r = 1 were almost similar to those obtained by 2D-cGA for r = 3 as both algorithm configurations had similar selection pressure (refer to Figure 3.3).

Figure 3.10(a) and (b) show the average number of generations and search success rate obtained when solving f_{FMS}, respectively. In general, for the average number of generations,

3D-cGA outperformed 2D-cGA; while for the hit rate, 2D-cGA outperformed 3D-cGA. However, the differences were not significant.

The results obtained while solving f_{SLE} are omitted as both algorithms either failed or showed undesirable performance (very low search success rate). For the complete results including those omitted, take a look at Appendix B, Table B.1.

Overall, 3D-cGA significantly surpassed 2D-cGA in terms of algorithm efficiency as a lower average number of generations was achieved for all the problems considered. Regarding the efficacy of the algorithm, both algorithms achieved either equal (when solving less complex problems such as f_{Ras}, f_{Sch}, and f_{Ack}) or slightly different search success rates (when solving more complex problems such as f_{Mic}, f_{Lang}, f_{Grie}, and f_{FMS}). As mentioned previously, for similar distance steps, the 3D topology offers a bigger (denser) neighbourhood than the 2D topology due to the vertical expansion of the cells. For example, employing a liner neighbourhood topology with one distance step, the 2D grid results in 4 neighbours, while 6 neighbours are resulted for the 3D grid. In addition, the vertical expansion of the cells in a 3D grid allows shorter diameter compared to that of a 2D grid, which allows faster spreading of solutions. Therefore, the selection intensity of 3D-cGA is stronger than that of 2D-cGA, leading to a lower convergence time (i.e., the number of generations).

3.3 Analysis of Complexity for 2D and 3D Topologies

In a cGA, the topology of the grid defines the communication network that the individuals spread throughout the population over it. Different topologies induce different computational and communication complexities. The following paragraphs provide a brief analysis that aims to highlight the difference in the computational and communication complexities between the topologies under investigation (i.e., 2D and 3D grids with wraparound edges). The analysis is carried out at the level of GA basic steps, they are: evaluation, selection, and genetic operation (recombination and mutation). Before proceeding, it is important to make the following assumptions. First, for both grid topologies, the neighbourhood topology is assumed to be *Linear* with one distance step. Second, the local selection method is assumed to be tournament selection (the most appropriate mechanism for parallel implementation (De Jong and Sarma, 1995; Eklund, 2003)).

Evaluation

As the fitness evaluation of an individual is independent from other individuals, there is no communication required regardless of the grid topology used. Hence, there is no difference in communication complexities between 2D and 3D grids. On the other hand, the computational effort needed to evaluate an individual depends on the complexity of an individual (e.g., simpler and smaller individual requires fewer calculations than complex and lengthy ones) (Eklund, 2003). At the individual level, there is no difference between the computational efforts for 2D and 3D grids of similar population sizes. At the neighbourhood level, the amount of computation needed is more for 3D as the neighbourhood in a 3D grid consists of more individuals (in this case, 6 neighbours for 3D vs. 4 neighbours for 2D). However, the latter difference is not considered as the fitness computation of an individual is isolated from the others.

Selection

There are various selection mechanisms introduced, each requires different communication and computational complexities. This analysis focuses on one of the most common methods, which is tournament selection. Unlike other mechanisms, tournament selection does not depend on fitness proportionate or rank (i.e., no need for the gather-broadcast operations) as it randomly selects two or more individuals. However, this method needs access to the all individuals in a neighbourhood. As a consequence, 3D topology requires more (~0.66×) communications than 2D topology. Conversely, the computational effort needed by 3D grid are similar to those of the 2D grid as the complexity of a single tournament depend on the tournament size k (Commonly, $k = 2$). In other words, the time or computational complexity of tournament selection is $O(k)$, where k is the tournament size (Goldberg and Deb, 1991).

Genetic operations

With mutation, there is no communication needed as the mutation works over a single individual in isolation of the others. The computational complexity of mutation depends on the individual representation and the mutation technique used. In all cases, the mutation requires marginal computational effort and no communications (Eklund, 2004). Therefore, both grids (i.e., 2D and 3D) offer similar complexities of mutation. On contrary, crossover requires communication with limited amount as it recombines two individuals. For 3D topology more communications (~0.66×) is required than with 2D grid due to the need of

access to all individuals in a neighbourhood, which is bigger for the 3D case. On the other hand, similar amount of computations is needed for both topologies as the computational complexity of crossover depends on the individual representation and the crossover technique used.

Overall, in cGAs, the computational and communication complexities vary according to several parameters such as: grid topology, neighbourhood topology and size, genetic operations techniques, population size, among others. Comparing the complexities of 2D and 3D grids while other parameters remain similar has showed more communications needed for the 3D grid than 2D grid that reached to ~0.66×. Mainly, the difference in communications is due to the difference in the neighbourhood density; 3D grid offers a denser neighbourhood leading to more communications needed. With regard to the computational effort, both grids may require similar computational complexities. However a difference in computational complexities between both grids may encounter based on the selection and genetic operations employed, particularly those that need access to all the individuals in a neighbourhood (e.g., fitness proportionate selection).

3.4 Conclusion

This study aimed to compare and analyse the performance of cGAs when two different grid dimensions are employed, in particular 2D and 3D topologies. In order to thoroughly investigate the algorithm performance, a benchmark of problems with diverse characteristics and complexities was selected. Simulation results show that 3D-cGA is more efficient in terms of convergence time than 2D-cGA for all the considered problems. With respect to the search success rate, both algorithms achieved similar efficacy. In the 3D structure, the interconnection between the cells leads to vertical expansion rather than the horizontal expansion of the 2D structure. As a result, the 3D structure provides a larger neighbourhood size than the 2D structure for similar distance steps (Breukelaar and Back, 2005). Although a bigger neighbourhood size leads to more exploitative behaviour for the algorithm, the balance between exploitation and exploration was maintained by selecting an appropriate neighbourhood radius with respect to the grid topology (Alba and Troya, 2000). Thus, the control of the selection intensity through the size of the neighbourhood would lead to the attainment of a higher search success rate and lower convergence time.

If the benefits of the performance results obtained are combined with the benefits that 3D technology offers, the resulting architecture would offer significant advantages in terms of reduction in routing length and interconnection delay, as well as an increase in logic and memory density. Accordingly, it is possible to improve the performance of the current optimisation engines at software and hardware levels to fit the requirements of the future.

3.5 Summary and Contribution to Knowledge

In this chapter the first and most basic step towards increasing the cellular dimensionality of GAs was established. The aim was to investigate the performance of cGAs when implemented on 3D topology. A comparative study of 2D-cGA and 3D-cGA was conducted for similar parameters. However, with 3D topology, a higher selection intensity was achieved due to the vertical expansion of cells that leads to a larger neighbourhood size. 3D-cGA achieved significantly better performance results than 2D-cGA, especially in terms of convergence time. Further benefits and investigations of the performance of cGAs when implemented on 3D topology will be provided in subsequent chapters. The following points summarise what this study has contributed to knowledge.

- Increasing the dimension of cellular structure improves the performance of cGAs, mainly the convergence time, while maintaining high accuracy and search success rates. As a consequence, multi-dimensional evolutionary algorithm models such as 3D-cGAs can empirically offer robust and effective optimisation engines to tackle hard, real-time problems.

- cGAs with higher cellular dimensions, specifically 3D, achieves significantly less convergence time than their corresponding 2D algorithms when solving multimodal problems with diverse characteristics and complexities such as the considered problems. However, these improvements vary as each problem presents different difficulty for the search.

- cGAs with different cellular dimensions, particularly 2D and 3D, achieve similar efficacy as both algorithms achieved similar search success rates. However, 2D-cGA and 3D-cGA present different exploration/exploitation trade-off due to the way the cells are connected.

- 3D topology consists of multiple 2D layers stacked on top of each other, which results in vertical rather than horizontal expansion. Therefore, 3D topology has a shorter diameter and a denser local neighbourhood than the corresponding 2D topology; which leads to fast spread of good individuals. As a result, cGAs with 3D topology achieves less convergence time than 2D-cGAs.

- Although 3D-cGA shows more exploitative behaviour due to the stronger global selection pressure, it displays a more appropriate balance between exploring the search space and exploiting good solutions than 2D-cGA with similar distance steps r.

- 3D-cGA with $r = 1$ showed a similar *NGR* and growth rate for the best individual as 2D-cGA with $r = 3$. Consequently, a similar selection pressure could be obtained through the control of the size of the local neighbourhood.

Chapter 4

Fault Tolerant 3D-cGA

This chapter presents new cGA algorithmic approaches that introduce the essential feature of fault tolerance to real time systems based on cGA platforms. Electronic circuits in aggressive environments, such as space, are subjected to various anomalies, including plasma and radiation, among others (Velazco *et al.*, 2005). Such anomalies have effects on systems, which result in various types of failures. In this study, radiation effects are taken into consideration as radiation is the main contributor to failure. In particular, Single Event Upsets are considered because they have the highest impact of all possible radiation effects (Velazco *et al.*, 2005; Gong *et al.*, 2008). Radiation-induced SEUs have also been observed at ground level due to the fact that the decrease in the feature sizes of electronic circuits leads to increased functional complexity and sensitivity (Normand, 1996). Designing systems that are highly reliable and efficient has become increasingly important not only for aerospace applications but also for terrestrial ones. Therefore, designing new algorithmic models of cGAs that maintain system reliability, even with a growing number of faulty Processing Elements (PEs) is the main objective of this chapter. Another objective is to improve the performance of the algorithm by mitigating the impact of the faults.

Ensuring the reliability and validity of systems requires two main operations. The first is fault prevention which aims to avoid the occurrence of faults; and the second is fault tolerance which aims to ensure the proper functionality of the system. In this work, only the process of fault tolerance is considered, which in turn consists of the three complementary stages fault detection, fault isolation, and fault recovery.

Previous studies on fault tolerance were carried out by previous members in the SLIg. An evolutionary design based on evolvable hardware platform for the automated design and adaptation of digital filters that adapted to faults was introduced by Hounsell and Arslan

(2001). Subsequently, Stefatos and Arslan (2004) proposed a fault-tolerant VLSI architecture based on PGA, which tackled SEU errors when targeting an individual's phenotypes. The fault model "Stuck at 0" was considered in that study. In a later study, Stefatos and Arslan proposed a high performance adaptive VLSI architecture that achieved higher throughput rates. This architecture was an improved version of their previous effort. Subsequently, further investigations were carried out by Morales-Reyes *et al.* (2008a; 2008b; 2009) to explore the ability of cGAs to tackle SEU errors by assuming that the faulty PEs were isolated. Research studies about the ability of an ordinary cGA and a parallel cGA to deal with SEUs that occurred at fitness score registers were presented in (Morales-Reyes *et al.*, 2008a; 2009), while the ability of an adaptive cGA to handle SEUs-targeted chromosomes registers was explored in (Morales-Reyes *et al.*, 2008b). In all previous studies, EAs proved their capability and power to tackle SEUs, as well as in improving the performance of the algorithm in terms of efficacy and efficiency. In this research, a new cGA algorithmic model that automatically detects, isolates, and recovers SEU errors occurring at individual's phenotypes, as well as new migration schemes to mitigate the impact of faults are proposed.

This chapter consists of three main sections. In the first section, a three-stage 3D-cGA approach that tolerates SEU faults is presented. In addition, an explicit adaptive migration technique based on the first fault-free neighbourhood, which is integrated into the design, is proposed in order to mitigate the impact of faults and to improve the performance of the algorithm. The second section introduces two more migration schemes in order to further improve the reliability and the performance of the algorithm. In the third and final section, an improved dynamic 3D-cGA, which is tolerant to SEUs is introduced. This approach is designed to dynamically adapt to fault ratios encountered and mainly aims to improve the efficacy of the algorithm. As mentioned previously, the faults considered in this study are target individuals' phenotypes, particularly when fitness scores are stuck at either '1' or '0'. This study emphasises the phenotypic space due to the importance of the fitness information in guiding the search process. The proposed algorithms are tested against a benchmark of well-known real world and test problems to thoroughly investigate their effectiveness and reliability.

4.1 Automatic Isolation of Faulty Cells

This section presents a fault-tolerant approach proposed for 3D-cGAs to overcome SEU errors. The proposed approach detects, isolates, and recovers from errors encountered. The

design exploits the inherent features of a cGA and uses genetic diversity as the key factor in identifying and isolating faulty solutions. Further, an explicit migration operation is proposed and integrated into the fault-tolerant approach as a mitigation technique. Several configurations concerning the use of the migration operation and inducing different selection intensities were considered. The effectiveness of the algorithm was investigated using a benchmark of four test functions and two real-world problems, which presented different levels of search difficulty. They are: Rastrigin (f_{Ras}), Ackley (f_{Ack}), Michalewicz (f_{Mic}), Langermann (f_{Lang}), FMS (f_{FMS}), and SLE (f_{SLE}) problems (details of benchmark problems are provided in Appendix A). The initial investigation was proposed and carried out in (Al-Naqi *et al.*, 2010a). In that study, only stuck at '0' faults were considered. This section extends the previous study to include other fault scenarios. Section 4.1.1 specifies fault scenarios and design, while the pseudocode and the description of the algorithm are presented in Section 4.1.2. Section 4.1.3 illustrates and analyses the results obtained. Finally, Section 4.1.4 draws conclusions.

4.1.1 Faults Design

The proposed algorithm deals with SEU errors, specifically when targeting fitness score registers. SEUs occur as one or more bits in the fitness score registers flip, in a way that keep their fitness values stuck at either a very high value '1' or a very low value '0'. From an algorithmic perspective, with stuck at '0' faults, the local selection method selects faulty individuals as they are considered to be the fittest (i.e., individuals with very low fitness values are the fittest for the considered problem and have to be minimised; otherwise the fittest individuals are those having a very high fitness value), and spreads the poor solutions they provide over all the population, which results in system failure. Therefore, stuck at '0' faults is considered to be the worst fault scenario. Another critical fault scenario is when some individuals' fitness scores are stuck at '1'. In this case, locally, the selection method disregards those individuals and does not spread the solutions they provide, which results in a major increase in convergence time, as well as a reduction in the accuracy of the solutions.

In this study, the fitness values are normalised between 0 and 1 for all the problems in order to offer equal weights. Hence, the minimum and maximum fitness values are 0 and 1, respectively. The faults are induced by a random selection of individuals asserting their fitness values to '1' or '0' according to the fault scenario to be evaluated. For each fault scenario and rate, the same fault pattern is maintained over all 100 independent runs to

obtain an average, and for all the problems to achieve fair comparison. The rates of fault considered represent 10%, 20%, 30%, and 40% of the total population size.

Algorithm 4.1 Pseudo-code for Fault-Tolerant 3D-cGA

```
1: procedure FT 3D-cGA
2: Generate_initial_population(P_{t=0});
3: Evaluation(P_{t=0});
4: for g ← 1 to gen1 do                                   //First stage
5:   for i ← 1 to ROWS do
6:     for j ←1 to COLUMNS do
7:       for k ←1 to LAYERS do
8:         neighbours ← Find_neighbours(position(i,j,k));
9:         parent1← position(i,j,k);
10:        parent2 ←Local_selection (neigbours);
11:        offspring ← Crossover (Pc, parent1, parent2);
12:        offspring ← Mutate(Pm);
13:        Evaluation ←Fitness(offspring);
14:        Replace-if-Better (position(i,j,k), offspring ,P_{aux});
15:        if g>1 && g ≤ gen1 then
16:          Diversity← Genotypic_diversity(P_{aux}, P_{temp});   end if;
17:      end for;
18:    end for;
19:  end for;
20:  P_{temp} ← P_{aux};
21: end for;
22: Cond1 ← Compute_differences(Diversity);               //Isolation stage
23: Cond2 ← Count_changes(Diversity);
24: Isolation_list ← Diversity(Cond1, Cond2);
25: while !stop_condition do                              //Third stage
26:   for i ← 1 to ROWS do
27:     for j ←1 to COLUMNS do
28:       for k ←1 to LAYERS do
29:         neighbours ← Find_neighbours(position(i,j,k));
30:         parent1← position(i,j,k);
31:         parent2 ←Local_selection (neighbours);
32:         Migration(Isolated_list, neighbours);
33:         offspring ← Crossover(Pc, parent1, parent2);
34:         offspring ← Mutate(Pm);
35:         Evaluation ←Fitness(offspring);
36:         Replace-if-Better(position(i,j,k), offspring, P_{aux});
37:       end for;
38:     end for;
39:   end for;
40:   P_t ← P_{aux};                                      // updating
41: end while;
42: end procedure FT 3D-cGA;
```

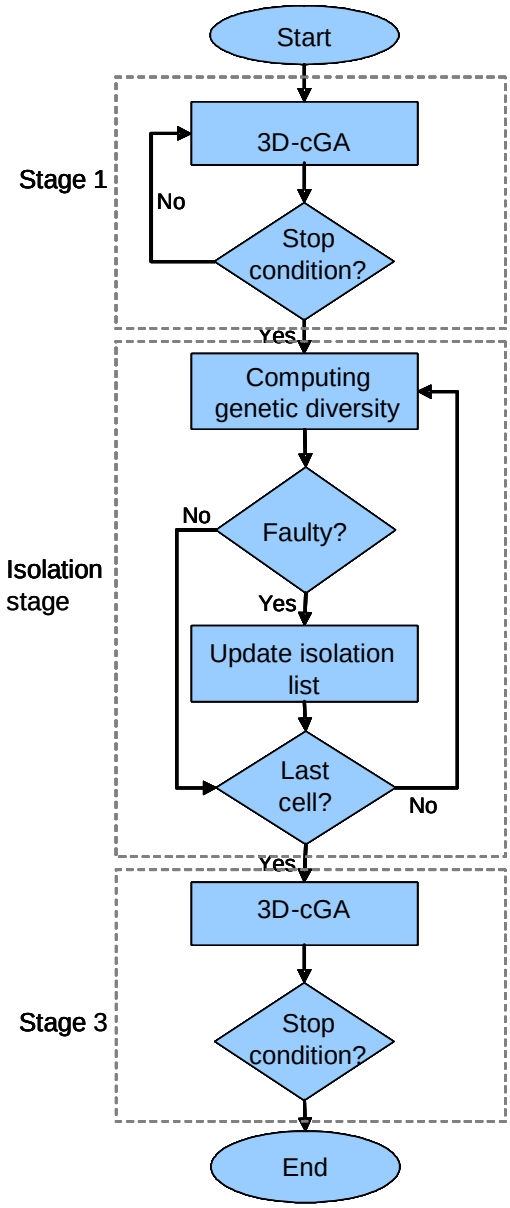


Figure 4.1. A high-level description of the Fault-Tolerant 3D-cGA showing the three stages.

4.1.2 Algorithm Description and Configuration

Algorithm 4.1 illustrates the pseudo-code for the fault tolerant 3D-cGA, which mainly consists of three stages. The first stage aims at monitoring the change in the genotypic diversities of each individual among successive generations by running a cGA for a very short period (i.e., a low number of generations). Next, the isolation stage identifies and isolates faulty individuals using feedback information from the first stage. Finally, in the third stage, another cGA is run to solve the given problem while excluding the faulty (i.e.,

isolated) individuals from the search process. In other words, the faulty individuals are not allowed to be updated or communicate with other fault-free individuals when it is a part of their neighbourhood.

As in canonical cGA (described in Section 2.3), the algorithm starts with a random generation of population followed by fitness evaluation (lines 1–3). Then the first stage begins (lines 4–21) and computes the genotypic diversities at an individual level rather than at the population level. The number of replacements for each individual is counted as well. This stage lasts for a few generations, *gen1*. Normally, the genetic diversity is expected to be high in the first few generations. During this stage, a normal updating process is carried out. It starts with an individual at a cell, identifying the neighbourhood of the current individual (line 8), choosing a second parent from the neighbours of the current individual, as the first one is the individual itself (lines 9 and 10). The genetic operators are applied to the selected individuals (i.e., parents) in order to generate an offspring, and either the current individual or the offspring is added to the auxiliary population following the defined replacement policy (lines 11–14). The genotypic diversities between successive generations are then computed for each individual (line 17). This process is repeated to update all cells. Before starting the next generation, a copy of the current updated population is maintained (line 20) in order to calculate the genotypic diversity.

The isolation stage starts by computing the differences in the individuals genotypic diversities obtained in the first stage (line 22). In addition, during this stage, the number of replacements for an individual throughout the first stage is assessed against the defined condition (line 23). Finally, in accordance with the isolation criteria, which are discussed later, faulty individuals are identified and a list of the isolated individuals is created (line 24). Figure 4.1 shows a high level diagram of the fault tolerant 3D-cGA pseudo-code presented in Algorithm 4.1.

The third stage starts following a similar updating process (lines 26–40) as in the first stage, and lasts until the termination condition is satisfied (line 25). In addition, an explicit migration operator could be applied (line 32) following the migration scheme presented below.

The local selection method used in this work is stochastic tournament (ST) selection. As mentioned in Section 2.3.1.2, two individuals are randomly selected and the best individual is then selected with a probability of $(1- r)$, while the worst one is assigned a probability of r; where $r \in [0 , 1]$. If r is 0, ST functions as a binary tournament selection, in which the best solution is always selected. This kind of selection offers a mean for controlling the selection

pressure and thus the diversity, which deeply affects the algorithm performance. As *r* increases, worse solutions are more likely to be maintained in the population; thus offering more diversity and weaker selection pressure (Simoncini *et al.*, 2007).

4.1.2.1 Genetic Diversity

The diversity of the population is one of the main issues in determining the performance of the algorithm and is widely used to analyse EAs. Several studies used genetic diversity to guide EAs (Ursem, 2002; Alba and Dorronsoro, 2005). In this study, a diversity measure based on genotypic entropy (H_t) is used to identify the faulty individuals (Tomassini, 2005).

As the main concern is to identify the faulty individuals, the genetic diversity is computed based on an individual's entropy rather than at the population level. Hence, the genotypic diversity can be defined as the average entropies of an individual in successive generations, which in turn is equal to the average of the entropies of different genes. The entropy of the j^{th} gene is expressed as:

$$H_j = -P \cdot \log P \tag{4.1}$$

where P (represented in 4.2) is the probability that the value of the j^{th} gene ($x_j \in [A, B]$) of a chromosome in generation t is different from that of the j^{th} gene of the same chromosome in generation $t-1$. A_j and B_j are variable (gene) limits.

$$P = 1 - \frac{\left| x_j(t) - x_j(t-1) \right|}{B_j - A_j} \tag{4.2}$$

Therefore, the average entropies of an individual consisting of n genes can be given as:

$$H = \frac{1}{n} \times \sum_{j=1}^{n} H_j \tag{4.3}$$

Figure 4.2 illustrates the process of computing the genotypic diversity of an individual in successive generations. For example, considering an individual of n genes, H_1 is the average entropies of the first gene in generations t–1 and t, and so forth. The genes entropies (i.e., H_1, …, H_n) are then averaged to compute the average entropy of an individual (H).

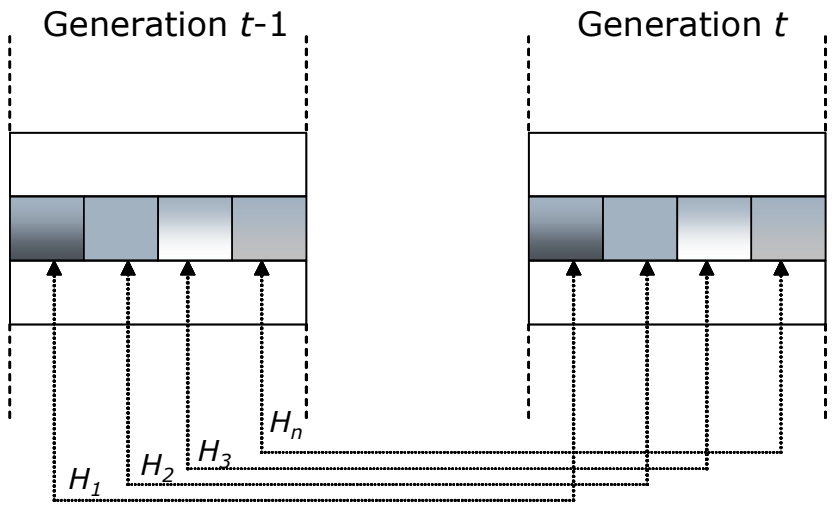


Figure 4.2. Computation of the genotypic diversity of an individual in generation t and $t–1$; H_i is the entropy of the i^{th} gene.

4.1.2.2 Isolation Criteria

Due to the impact of the fault models considered on the functionality of a system, the algorithm isolates the faulty individuals in order to mitigate their impact. That is to say, the aim of the isolation is to prevent the faulty individuals from spreading their poor solutions. On the other hand, good individuals are migrated in order to improve the performance of the algorithm.

In this research, two isolation criteria are proposed, with each one handling one of the fault models discussed above. Firstly, an individual is defined as faulty when its genotypic diversities computed in the first stage are found to be almost constant, taking into account that the first stage lasts for only the first few generations when an individual's genotypic diversity is expected to be fairly variable. Assuming a maximisation problem, this criterion handles the case where the individual fitness scores are stuck at '1'. Secondly, an individual can also be defined as faulty when the replacement rate of the individual considered during the first stage is too high. This situation occurs when the fitness score of an individual is too low (stuck at '0'). When minimising, the converse is applies

The above criteria are placed based on the following facts: the fittest individuals are always winning and thus not being replaced when they compete with other individuals following the defined replacement policy (*replace-if-better*). In this case, the genotypic diversities of those individuals are maintained. Conversely, the weakest individuals are always being replaced, leading to high frequency of changes.

4.1.2.3 Migration Technique

As mentioned previously, cGAs offer an implicit mechanism for migration that is inherent in their overlapping neighbourhoods; however, an explicit migration technique is defined in this research. The main objective behind employing an explicit migration operator is to mitigate the impact of the faults that occur. Another objective is to improve the performance and the reliability of the algorithm.

The migration technique is introduced by defining the migration parameters. The migration operator frequency is set to the highest (i.e., every generation) and is activated only when there is at least one faulty individual within the current individual's neighbourhood. The number of migrants or migration rate is adapted, and is computed whenever a migration is activated. This rate is equal to the number of faulty individuals, which varies from 1 to *no-of-neighbours*. In this study the *no-of-neighbours* is 7 (the central, vertical north and south, horizontal north and south, and the east and west individuals), as the defined neighbourhood topology is linear with one distance step from the central individual.

The migration scheme is defined as follows. The individuals to be replaced are all those who are faulty, and the individuals to be migrated (i.e., migrants) are chosen from the first fault-free neighbourhood found to replace the corresponding faulty individuals (see Figure 4.3). In the worst-case scenario, if there is no fault-free neighbourhood (i.e., there is at least one faulty individual within each possible neighbourhood), then a random neighbourhood is selected by allowing the selection of faulty individuals.

In the following section the experimental parameters and algorithm configurations are defined. In addition, the results obtained and their analyses are provided.

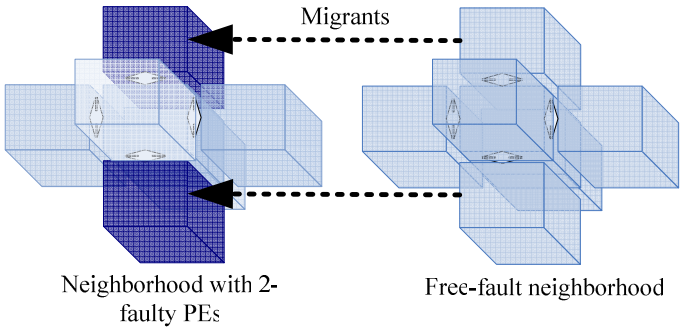


Figure 4.3. The replacement of two faulty PEs by the corresponding ones (migrants) from the first fault-free neighbourhood found through migration.

4.1.3 Experimental Results and Analysis

In this research, four algorithm configurations were defined. These configurations differed in the use of the migration technique in combination with the selection rate *r*. In the first configuration, a stochastic tournament selection with $r = 0$ was applied, while in the second configuration, a stochastic tournament selection with $r = 0.5$ was applied in order to offer equal chances for poor and good solutions to be involved. The migration operator was not introduced in the preceding configurations. Hence, the first and second configurations are represented by (ST, $r = 0.0$ + *noMigration*) and (ST, $r = 0.5$ + *noMigration*), respectively. The third and fourth configurations are similar to the first and the second, respectively; with the exception of introduction of the migration technique (ST, $r = 0.0$ + *Migration*; ST, $r = 0.5$ + *Migration*).

The same parameters were used for all the problems (see Table 4.1). The population size used here was 343 individuals, which were arranged into a 7×7×7 lattice. The defined local neighbourhood contained seven individuals—east, west, vertical north and south, and horizontal north and south individuals plus the central individual.

The first parent was the current individual while the second one was selected by stochastic tournament with rate *r*. An arithmetic crossover operator (AX) with a probability of $P_c = 0.9$ was applied to generate an offspring. The offspring was mutated by applying a non-uniform mutation operator with a probability of $P_m = 1/L$, where L is the length of the chromosome (the dimension of the problem). Although the dimension of the FMS problem is 6, the same mutation probability was used as with all the other problems. The replacement policy used here was *replace-if-better*, during which the current individual was replaced if its competitor (offspring) was better. The migration parameters used in the third and the fourth configurations are described in Section 4.1.2.3. Finally, the algorithm terminated if the average fitness value ($\overline{f}$) of the population satisfied a threshold (≤ 0.00005). This threshold was applied for all the problems, with the exception of the f_{SLE}, where a less precise threshold (≤ 0.0001) was used due to the problem's complexity.

During the experiment, similar fault rates and patterns were injected for all the configurations and problems. The performance of the algorithm was measured using two metrics—the search success rate, which represents the efficacy; and the average number of generations, which represents the efficiency of 100 independent runs. A different number of maximum generations was assigned to each problem due to their different complexities. f_{Lang}

was assigned 700 generations, whereas f_{Ras}, f_{Mic}, and f_{FMS} were assigned 1000 generations. Finally, f_{Ack} and f_{SLE} were assigned 2000 generations.

The algorithm was assessed against stuck at '0' and stuck at '1' faults with up to 40% faults. The results are presented in Tables 4.2, 4.3, 4.6, and 4.7, where the average number of generations with the median absolute deviation is included after the symbol '±' and the search success rate are shown for every fault rate. For each fault rate, the best values obtained among the four configurations are marked in **bold**.

Table 4.1. Parameters used in the algorithm

Population size:	343 individuals
Parent selection:	Current individual + ST, $P_s = r$
Recombination:	AX, $P_c = 0.9$
Mutation:	Non-uniform, $P_m = 1/L$
Replacement:	Replace-if-better
Neighbourhood:	*L*7
Lattice:	7×7×7
Termination criterion:	$\overline{f} \leq 0.00005$ (≤ 0.0001 for f_{SLE})

Additionally, the results obtained when there was no fault (0% faults) are shown in Tables 4.2 and 4.3. For 0% faults, the results shown for the first and third algorithms configurations were similar, while similar results were obtained for the second and fourth algorithm configurations due to the inactive migration operator in the absence of faults. The symbol '+' in the Tables means that there exists statistical confidence in the results of the compared algorithm configurations, while the symbol '•' means there is no statistical difference between the results obtained (for details about statistical tests refer to Section 2.2.3.1).

4.1.3.1 Stuck at '0' Faults

As mentioned previously, stuck at '0' is the most critical fault model as the problems considered all need to be minimised; for the case of maximisation the converse applies. Tables 4.2 and 4.3 show the results obtained for the test and the real-world problems, respectively.

With regard to f_{Ras}, the best search success rate (100%) was achieved by the second and the fourth configurations for which similar selection rates (r = 0.5) were applied with up to 30% faults (See Table 4.2). However, a major decline in the search success rate reached almost 87% and 40% faults were observed. With regard to efficiency, the minimum average generations were obtained by the third configuration (ST, r = 0.0 + *Migration*) with up to

20% faults. On the other hand, the best efficiency was achieved by the second and fourth configurations (these configurations achieved almost similar efficiency with a maximum difference of 9 generations) with faults more than 20%. Even though fewer generations were expected for ST, r = 0.0, a lower number of average generations were obtained for ST, r = 0.5, in particular the second configuration due to the impact of the faults. However, the introduction of migration with ST, r = 0.0 (ST, r = 0.0 + *Migration*) significantly improved the performance of the algorithm in terms of search success rates (≥ 13% with 20% and 30% faults) and average generations (≤ 116 generations with 10%–30% faults) when compared to the first configuration (ST, r = 0.0 + *noMigration*).

With regard to f_{Ack}, the third configuration (ST, r = 0.0 + *Migration*) achieved the highest search success rate with up to 30% faults and the minimum average generations with up to 20% faults. With 40% faults, the algorithm failed to solve the problem due to the high fault rate and higher problem complexity. Furthermore, from Table 4.2 it can be seen that there are no statistical differences in the efficiency of the algorithm between the four configurations.

The best efficacy (hit rate of 100%) was reached by all configurations with up to 10% faults when solving f_{Mic}, while the highest search success rates were obtained by the third configuration with up to 30% faults. Concerning the efficiency, the minimum average generations were obtained by the third configuration with up to 30% faults. Hence the third configuration (ST, r = 0.0 + *Migration*) significantly achieved the best performance of the algorithm. Similar to f_{Ack}, the algorithm failed to solve the problem with 40% faults.

When solving f_{Lang}, the highest search success rates were obtained by the fourth configuration (ST, r = 0.5 + *Migration*) with up to 30% faults; however, these hit rates differed slightly (≤ 2%) from those obtained when no migration was introduced (ST, r = 0.5 + *noMigration*). The best efficiency was reached by the third configuration (ST, r = 0.0 + *Migration*) with up to 30% faults; an exception was with 20% faults. Considering 40% faults, the algorithm is declined to solve the problem due to the high problem complexity and high fault rate.

Solving f_{FMS} the algorithm shows similar behaviour to that observed when solving the previous problem (f_{Lang}), as these two problems have similar characteristics. For all fault rates the best efficacies were achieved with ST, r = 0.5; minor difference not more than 3% hit rate were observed with and without the use of migration. The best efficiencies were achieved by the third configuration with up to 30% faults. Applying (ST, r = 0.5 + *noMigration*; ST, r = 0.5 + *Migration*) configurations on f_{Lang} and f_{FMS}, the algorithm shows a

Table 4.2. Experimental Results: Convergence time (CT) and rate (CR)* for test problems

Problems/ % of faults		BT + no Migration		ST, r = 0.5 + no Migration		BT + Migration		ST, r = 0.5 + Migration		Test
f_{Ras}	0%	**355.71 ± 29.5**	**100%**	453.78 ± 27.0	**100%**	**355.71 ± 29.5**	**100%**	453.78 ± 27.0	**100%**	+
	10%	539.41 ± 64.0	**100%**	522.73 ± 36.0	**100%**	**423.42 ± 24.5**	**100%**	523.23 ± 33.5	**100%**	+
	20%	784.95 ± 82.0	86%	616.01 ± 38.0	**100%**	**598.87 ± 64.0**	99%	624.78 ± 45.0	**100%**	+
	30%	887.10 ± 58.0	67%	709.61 ± 42.0	**100%**	728.83 ± 108	90%	**700.67 ± 42.0**	**100%**	+
	40%	–	0%	**948.69 ± 25.0**	**13%**	979.66 ± 3.00	3%	957.83 ± 14.5	12%	●
f_{Ack}	0%	**1840.1 ± 84.0**	**85%**	1846.1 ± 78.0	58%	**1840.1 ± 84.0**	**85%**	1846.1 ± 78.0	58%	●
	10%	1896.2 ± 64.5	66%	1862.7 ± 48.0	47%	**1860.2 ± 82.0**	**73%**	1908.7 ± 43.5	50%	●
	20%	1927.7 ± 34.5	50%	1929.6 ± 35.0	23%	**1872.5 ± 63.0**	**66%**	1917.9 ± 31.5	28%	●
	30%	1950.5 ± 19.5	16%	1916.0 ± 29.0	21%	1924.5 ± 25.0	**38%**	**1910.8 ± 21.5**	12%	●
	40%	–	0%	–	0%	–	0%	–	0%	–
f_{Mic}	0%	**534.48 ± 55.0**	**100%**	711.79 ± 61.5	**100%**	**534.48 ± 55.0**	**100%**	711.79 ± 61.5	**100%**	+
	10%	683.77 ± 57.0	**100%**	769.67 ± 60.0	**100%**	**607.22 ± 49.5**	**100%**	774.21 ± 59.5	**100%**	+
	20%	858.51 ± 53.5	86%	842.76 ± 68.0	91%	**764.95 ± 75.0**	**93%**	859.45 ± 58.0	87%	+
	30%	936.49 ± 40.0	59%	929.11 ± 30.0	70%	**850.27 ± 59.0**	**73%**	921.57 ± 33.0	59%	+
	40%	–	0%	–	0%	–	0%	–	0%	–
f_{Lang}	0%	**270.45 ± 35.0**	74%	359.28 ± 40.5	**82%**	**270.45 ± 35.0**	74%	359.28 ± 40.5	**82%**	+
	10%	391.20 ± 65.5	70%	397.44 ± 49.0	83%	**341.00 ± 45.0**	61%	409.29 ± 48.5	**84%**	+
	20%	492.05 ± 55.0	69%	**448.85 ± 56.0**	87%	458.10 ± 95.0	59%	465.03 ± 58.5	**88%**	+
	30%	567.55 ± 60.0	68%	522.74 ± 64.0	87%	**503.64 ± 59.0**	50%	513.02 ± 62.0	**89%**	+
	40%	–	0%	–	0%	–	0%	–	0%	–

Table 4.3. Experimental Results: Convergence time (CT) and rate (CR)* for real-world problems

Problems/ % of faults		BT+ no Migration		ST (r = 0.5)+ no Migration		BT+ Migration		ST (r = 0.5)+ Migration		Test
f_{FMS}	0%	**326.12 ± 78.5**	74%	399.54 ± 81.0	**81%**	**326.12 ± 78.5**	74%	399.54 ± 81.0	**81%**	+
	10%	410.77 ± 70.0	71%	475.38 ± 89.5	**88%**	**360.04 ± 66.0**	73%	445.44 ± 81.5	**88%**	+
	20%	579.21 ± 93.5	71%	527.45 ± 116.0	87%	**449.16 ± 109.0**	73%	518.75 ± 86.0	**88%**	+
	30%	722.92 ± 92.0	69%	610.43 ± 102.5	90%	**545.70 ± 93.0**	71%	607.47 ± 122.0	**91%**	+
	40%	911.85 ± 67.0	7%	**812.47 ± 135.0**	**65%**	820.09 ± 113.0	52%	821.09 ± 102.5	62%	●
f_{SLE}	0%	**247.48 ± 28.0**	41%	404.47 ± 64.0	**63%**	**247.48 ± 28.0**	41%	404.47 ± 64.0	**63%**	+
	10%	324.96 ± 36.0	27%	499.00 ± 72.0	45%	**303.55 ± 59.0**	27%	479.46 ± 65.5	**52%**	+
	20%	603.61 ± 105.5	18%	552.86 ± 66.0	36%	**386.46 ± 61.0**	15%	557.62 ± 86.0	**37%**	+
	30%	780.50 ± 133.5	12%	712.19 ± 107.0	21%	**510.28 ± 62.0**	7%	636.40 ± 97.0	**22%**	●
	40%	1824.0 ± 0.0	1%	1513.0 ± 437.0	2%	1996.0 ± 0.0	1%	**1478.8 ± 344.0**	**4%**	●

surprising tendency to increase the search success rate as the fault rate increases. This trend occurred due to the selection rate (r = 0.5)—which was unbiased to neither good (more likely to be faulty) nor bad (more likely to be non-faulty) solutions. As the fault rate

* For more details about the performance measures, please refer to Section 2.2.3.1.

increased the selection intensity is reduced, leading to promote more exploration, which helps to escape the local minima and thus increases the hit rate.

At last, considering all fault rates, the highest search success rates were obtained by the fourth configuration (ST, r = 0.5 + *Migration*) when solving f_{SLE}. With regard to the efficiency of the algorithm, with all fault rates the minimum average generations were achieved by the third configuration (ST, r = 0.0 + *Migration*).

In general, the performance of the algorithm declined as the rate of faults increases. This decline can be observed as a drop in the search success rate and/or an increase in the number of generations required to find the desired solutions. Each fault rate represented a different level of search difficulty, worse fault distributions and patterns were formed with higher fault rates. The worst fault distribution occurred when a PE was surrounded by faulty neighbours (PEs) in all possible directions.

To evaluate and compare the different algorithm configurations, two-level ranking was performed based on two metrics: the average number of generations and the search success rate (Tables 4.4 and 4.5, respectively). In the first level, the four configurations were ranked for each problem independently. This task was accomplished by summing the positions of each algorithm configuration considering each fault rate. In the Tables, these local ranks are shown in columns 3–8. In order to obtain a global rank the second level of ranking was performed by summing the local ranks computed in the first level. In these Tables, the global rank and the summation values are shown in the first and last columns, respectively.

In summary, the third configuration (ST, r = 0.0 + *Migration*) achieved the best performance in terms of efficiency for all the considered problems (see Table 4.4). In contrast, the best efficacy was obtained by the fourth configuration (ST, r = 0.5 + *Migration*) and the second configuration (ST, r = 0.5 + *noMigration*) as the difference between the values of the sum of positions was minor (see Table 4.5). Hence, the second and fourth configurations performed similarly in terms of both performance metrics, while the first configuration (ST, r = 0.0 + *Migration*) had the worst performance.

Table 4.4. Local and global* average-generations- based ranking for stuck at '0' faults

Rank	Algorithms	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	Sum
1	ST, r = 0.0 + Migration	1	1	1	1	1	1	6
2	ST, r = 0.5 + no Migration	2	3	3	2	3	3	16
3	ST, r = 0.5 + Migration	3	2	4	4	2	2	17
4	ST, r = 0.0+ no Migration	4	3	2	3	4	3	19

* Local ranks (columns 3 to 8) are performed for each problem independently; the highest rank is assigned the lowest value. Global ranks are performed by summing the local ranks of each problem and are shown in the first column.

Table 4.5. Local and global* search-success-rate-based ranking for stuck at '0' faults

Rank	Algorithms	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	Sum
1	ST, r = 0.5 + Migration	2	3	3	1	1	1	11
2	ST, r = 0.5 + no Migration	1	3	2	2	2	2	12
3	ST, r = 0.0 + Migration	3	1	1	4	3	4	16
4	ST, r = 0.0+ no Migration	4	2	4	3	4	3	20

* Local ranks (columns 3 to 8) are performed for each problem independently; the highest rank is assigned the lowest value. Global ranks are performed by summing the local ranks of each problem and are shown in the first column.

Introducing migration adds a significant improvement when combined with ST, r = 0.0 in terms of average number of generations leading to 21–279 generations less. In addition, it leads to an increase in the search success rate in the range 2%–45%. That is, because of the high selection intensity induced—due to r = 0.0—within the fault-free neighbourhood that resulted from migration. However, the combination of migration with ST, r = 0.5 showed no improvements based on the two metrics. In contrast to ST, r = 0.0, in which the fittest individual with the minimum fitness value was favoured—bearing in mind that also faulty individuals have the minimum fitness value, ST, r = 0.5 mitigates the involvement of faulty individuals in the reproduction process. Therefore ST, r = 0.5 plays a similar role as migration; which is why combining them leads to no further improvements.

Since each of the problems considered presented a different level of complexity, the amount of selection intensity needed to effectively solve each problem was also different. Accordingly, the exploration/exploitation trade-off and the amount of diversity needed varied. Problems with high complexity, such as f_{Lang}, f_{FMS}, and f_{SLE}, required more diversity, which can be offered by tuning the rate of the selection r. The increase in the value of r would maintain worse solutions in the population for a longer time, thus more diversity is offered leading to an increase in the search success rate. However, the number of generations required to solve a given problem is increased.

4.1.3.2 Stuck at '1' Faults

The stuck at '1' fault scenario is less critical than stuck at '0' faults. However, this scenario leads to considerable impacts on the system. These impacts mainly occur as reductions in the accuracy of solutions and an increase in convergence time. Tables 4.6 and 4.7 summarise the results obtained when solving the test and the real-world problems, respectively.

With regard to f_{Ras}, the best search success rate (100%) was achieved by all algorithm configurations with up to 30% faults, except for the third configuration, as a lower rate (86%) was achieved with 30% faults. With 40% faults, a considerable drop in the search

success rate reached up to 70% is observed. However, with stuck at '1' faults the drop in the search success rate when the fault rate increases to 40% is significantly lower than the corresponding drop with stuck at '0' faults (17%), which confirms that stuck at '1' fault scenario is less destructive. Concerning the efficiency, with all fault rates the minimum average generations were obtained by the first configuration (ST, r = 0.0 + *noMigration*) with significant differences, except for 40% fault as the differences were insignificant (see test results in Table 4.6). In contrast to stuck at '0' faults, the introduction of migration results in higher number of generations reached up to 230 while maintaining almost similar efficacies.

When solving f_{Ack}, the differences between the average number of generations obtained by all configurations are insignificant (see test results in Table 4.6); generally therefore, all configurations have similar efficiencies. Concerning the efficacy of the algorithm, with all fault rates the highest search success rates were obtained by the first configuration (ST, r = 0.0 + *noMigration*). Hence, as with the previous problem the introduction of migration could not improve the performance of the algorithm. However, the efficacy obtained with migration (ST, r = 0.0 + *Migration*) was insignificantly less than those obtained without migration (ST, r = 0.0 + *noMigration*). Similar to stuck at '0' faults, with 40% faults the algorithm failed to solve the problem due to the high fault rate in combination with high problem complexity.

Table 4.6. Convergence time (CT) and rate (CR)* for test problems

Problems/ % of faults		BT + no Migration		ST, r = 0.5 + no Migration		BT + Migration		ST, r = 0.5 + Migration		Test
f_{Ras}	10%	**386.23 ± 31.0**	**100%**	541.27 ± 41.5	**100%**	417.17 ± 36.0	**100%**	507.53 ± 36.0	**100%**	+
	20%	**449.69 ± 42.0**	**100%**	616.08 ± 45.5	**100%**	583.27 ± 75.5	**100%**	616.82 ± 39.0	**100%**	+
	30%	**486.22 ± 37.5**	**100%**	667.88 ± 41.0	**100%**	717.19 ± 104	86%	673.83 ± 49.5	**100%**	+
	40%	**919.83 ± 48.0**	36%	947.90 ± 20.5	30%	943.08 ± 34.5	24%	954.55 ± 20.0	**38%**	●
f_{Ack}	10%	1848.3 ± 66.0	**77%**	1877.4 ± 57.5	40%	**1820.4 ± 76.0**	75%	1884.4 ± 39.0	59%	●
	20%	1853.1 ± 70.0	**65%**	1894.6 ± 37.0	21%	**1833.3 ± 51.0**	53%	1902.2 ± 35.0	28%	●
	30%	1906.2 ± 45.5	**48%**	1956.5 ± 15.0	15%	1886.8 ± 57.5	40%	**1817.1 ± 47.0**	18%	●
	40%	–	0%	–	0%	–	0%	–	0%	–
f_{Mic}	10%	**578.69 ± 43.5**	**100%**	769.91 ± 48.5	**100%**	615.23 ± 54.5	**100%**	773.10 ± 70.5	**100%**	+
	20%	**653.80 ± 49.5**	**100%**	863.18 ± 54.0	94%	736.67 ± 69.0	86%	844.51 ± 62.0	89%	+
	30%	**707.26 ± 71.0**	**99%**	891.19 ± 46.0	63%	852.47 ± 73.0	74%	914.89 ± 41.0	66%	+
	40%	–	0%	–	0%	–	0%	–	0%	–
f_{Lang}	10%	**309.00 ± 51.0**	58%	409.05 ± 48.0	75%	329.94 ± 48.0	67%	402.01 ± 55.0	**79%**	+
	20%	**331.02 ± 53.0**	69%	468.27 ± 64.0	77%	437.43 ± 71.0	69%	484.40 ± 53.0	**85%**	+
	30%	**402.25 ± 45.5**	74%	525.39 ± 49.5	74%	508.36 ± 99.5	58%	518.20 ± 56.5	**82%**	+
	40%	–	0%	–	0%	–	0%	**656.00 ± 0.0**	**1%**	●

* For more details about the performance measures, please refer to Section 2.2.3.1.

Table 4.7. Convergence time (CT) and rate (CR)* for real-world problems

Problems/ % of faults		BT + no Migration		ST, r = 0.5 + no Migration		BT + Migration		ST, r = 0.5 + Migration		Test
f_{FMS}	10%	**338.05 ± 73.5**	68%	472.49 ± 88.0	**87%**	348.60 ± 87.5	76%	453.56 ± 91.0	**87%**	+
	20%	**399.50 ± 79.0**	65%	489.34 ± 64.0	**87%**	465.47 ± 106.0	73%	532.77 ± 87.0	86%	+
	30%	**416.53 ± 95.0**	73%	599.85 ± 127.5	**88%**	571.92 ± 110.0	68%	613.04 ± 111.0	83%	+
	40%	**650.98 ± 163.0**	72%	806.61 ± 131.0	**77%**	809.77 ± 88.5	54%	766.16 ± 112.5	68%	+
f_{SLE}	10%	302.89 ± 44.0	29%	479.20 ± 83.0	**55%**	**282.43 ± 16.0**	23%	485.55 ± 50.0	49%	+
	20%	446.53 ± 68.5	26%	543.52 ± 59.0	**50%**	**363.47 ± 52.0**	21%	568.06 ± 100.0	45%	+
	30%	**407.76 ± 59.0**	30%	630.44 ± 87.0	**36%**	420.84 ± 41.0	13%	669.77 ± 99.0	31%	+
	40%	**639.72 ± 133.0**	11%	902.75 ± 130.0	20%	781.00 ± 117.0	11%	1042.1 ± 206.5	**26%**	+

* For more details about the performance measures, please refer to Section 2.2.3.1.

With all the considered fault rates, the highest efficacy and efficiency were significantly achieved by the first configuration (ST, r = 0.0 + *noMigration*) when solving f_{Mic}. Similarly, with 40% faults the algorithm failed to solve the problem due to high fault rate.

Solving f_{Lang}, the highest search success rates were obtained by the fourth configuration (ST, r = 0.5 + *Migration*), while the best efficiency was significantly reached by the first configuration (ST, r = 0.0 + *Migration*). f_{Lang} differed from the previously discussed problems as it is not only highly multimodal but also epistasis and asymmetric at the same time, therefore it introduces more difficulty to the search. Because of the high complexity more diversity is needed to effectively solve the problem. That is why the highest convergence rates were obtained with the configurations that introduced more diversity (i.e., the second and the fourth configurations, as they had lower selection pressure). However, more diversity increases the convergence time leading to less efficiency. Considering 40% faults, the algorithm was unable to solve the problem due to the high problem complexity and high fault rate.

f_{FMS} and f_{SLE} have similar characteristics to f_{Lang}, hence similar profiles were obtained. Generally, the best efficiency was achieved by the first configuration (ST, r = 0.0 + *noMigration*) due to the high selection pressure, while the highest search success rates were achieved by the second configuration (ST, r = 0.5 + *noMigration*) due to more diversity being offered.

As with the stuck at '0' faults, the algorithm configurations are also ranked based on the average number of generations (Table 4.8) and the search success rate (Table 4.9). In summary, the first configuration (ST, r = 0.0 + *noMigration*) obtains the best algorithm efficiency for most of the problems considered (see Table 4.8). Because of the high selection intensity as well as the less destructive scenario of faults a main profile which contradicts the

Table 4.8. Local and global* average-generations-based ranking for stuck at '1' faults

Rank	Algorithms	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	Sum
1	ST, r = 0.0 + no Migration	1	2	1	1	1	1	7
2	ST, r = 0.0 + Migration	2	1	2	2	2	1	10
3	ST, r = 0.5 + Migration	4	3	3	3	3	3	19
4	ST, r = 0.5+ no Migration	3	4	3	4	3	3	20

Table 4.9. Local and global* search-success-rate-based ranking for stuck at '1' faults

Rank	Algorithms	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	Sum
1	ST, r = 0.5 + Migration	1	3	2	1	2	2	11
2	ST, r = 0.5 + no Migration	3	4	2	2	1	1	13
2	ST, r = 0.0 + no Migration	2	1	1	3	3	3	13
4	ST, r = 0.0 + Migration	4	2	2	3	4	4	19

one observed with stuck at '0' faults is the modest increase—rather than the dramatic increase—in the number of generations against the increase in the fault rate

With regard to the efficacy, the best efficacy was obtained by the fourth and second configurations (ST, r = 0.5 + *noMigration*/*Migration*) when solving the real-world problems that are mainly characterised as epistasis and asymmetric. On the other hand, the best efficacy was obtained by the first configuration (ST, r = 0.0 + *noMigration*) when solving the test functions (see Table 4.9). As mentioned above, the more complex problems require more diversity, which in this case was offered by applying lower selection pressure (i.e., ST, r = 0.5); however more diversity leads to an increased number of generations to solve a given problem.

4.1.3.3 Study of the Failure and Expansion in Fault Rates

This section aims to firstly investigate the sharp drop in the convergence rate as well as the quick increase in the convergence time, particularly when the fault rate increases to 40%. In order to meet the first aim, the genetic diversity obtained by the different algorithm configurations of a selected problem (f_{Ras}) against the different fault rates were explored. Another aim was to explore the behaviour of the algorithm with higher fault rates (45% and 50%), in particular when solving f_{Ras}, f_{FMS}, and f_{SLE}. In addition, a fault rate of 35% is

* Local ranks (columns 3 to 8) are performed for each problem independently; the highest rank is assigned the lowest value. Global ranks are performed by summing the local ranks of each problem and are shown in the first column.

considered to investigate the sharp decline in the performance of the algorithm. The other problems were not considered in this investigation as the algorithm failed to solve these problems with 35% faults. This section illustrates the impact of faults on the genotypic diversity of the population as well as expands the results obtained in the previous sections to include 35%, 45%, and 50% faults.

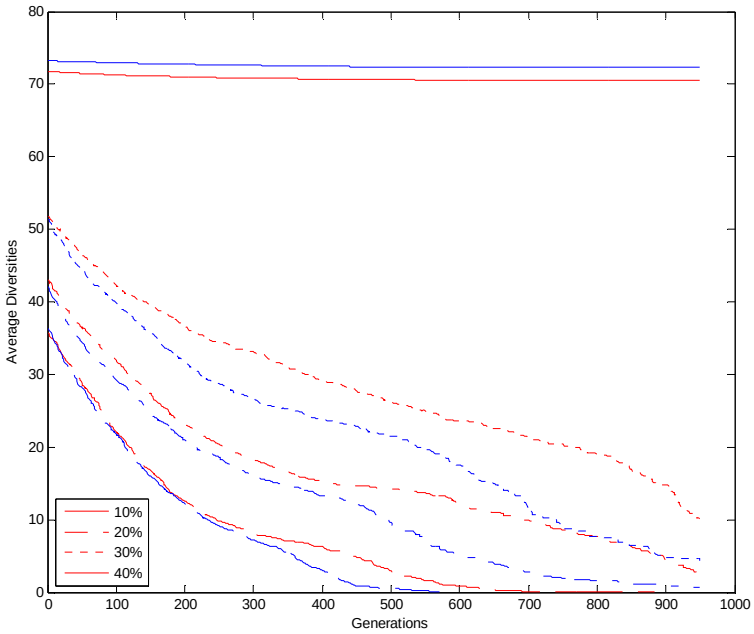
Figures 4.4(a) and (b) show the change in the genotypic diversity (computed as an average of 100 independent runs) when solving f_{Ras} with the various considered fault rates for stuck at '0' faults.

Figure 4.4(a) illustrates the changes induced by the first and third algorithm configurations (i.e., 3D-cGA with ST, r = 0.0 + *noMigration*/*Migration*, respectively), while the changes in diversity incurred by the second and the fourth algorithm configurations (i.e., 3D-cGA with ST, r = 0.5 + *noMigration*/*Migration*, respectively) are shown in Figure 4.4(b). Similarly, Figure 4.5(a) and (b) show the change in genotypic diversities with fitness score stuck at '1'.

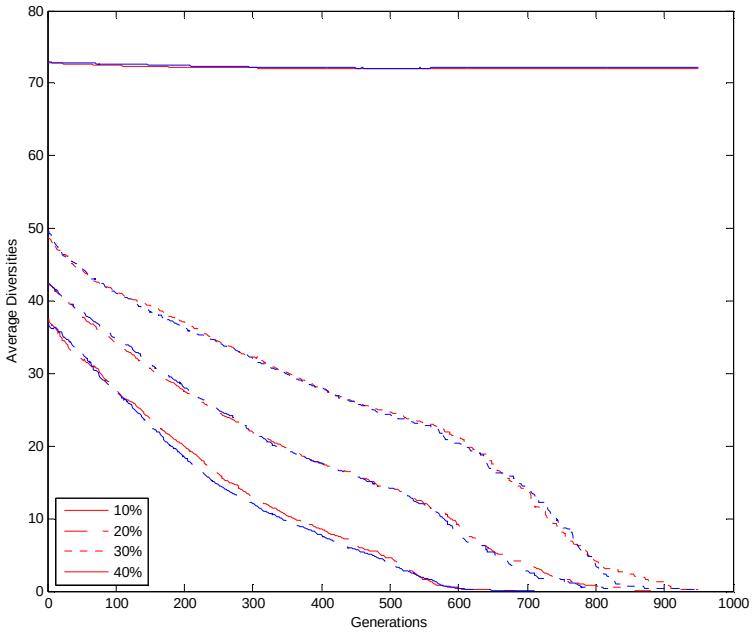
It can be clearly seen from Figure 4.4(a) that generally the time (i.e., number of generations) needed for population diversity to approach zero increases considerably as fault rate increases. This observation leads to two major conclusions. First, the algorithm is likely to fail to solve a given problem with 40% faults as the diversity level is too high and is virtually maintained. In addition, due to the major difference between diversities obtained with 30% and 40% faults, increasing the fault rate from 30% to 40% leads to a sharp drop in convergence rate as well as a major increase in the convergence time. Second, the introduction of migration (see the blue trends in Figure 4.4(a)) could slightly enhance the convergence time when considering 10%, 20%, and 30% faults. However, migration failed to add any improvement when considering 40% faults.

Similar profiles were obtained by the second and fourth algorithm configurations (see Figure 4.4(b)). However, the introduction of migration (see the blue trends in Figure 4.4(b)) added neither improvement nor deterioration to the convergence time. This finding confirms that the combination of migration and ST, r = 0.5 could not improve the performance of the algorithm.

With stuck at '1' faults (see Figure 4.5(a) and (b)) the population diversities for all considered fault rates at the beginning were reasonable and were significantly less than the corresponding ones when considering stuck at '0' faults. Therefore, it is easier and faster for the algorithm to solve, in particular minimise, a problem with stuck at '1' faults. That is to say, the stuck at '1' faults model is less critical than the stuck at '0' faults model. Moreover,

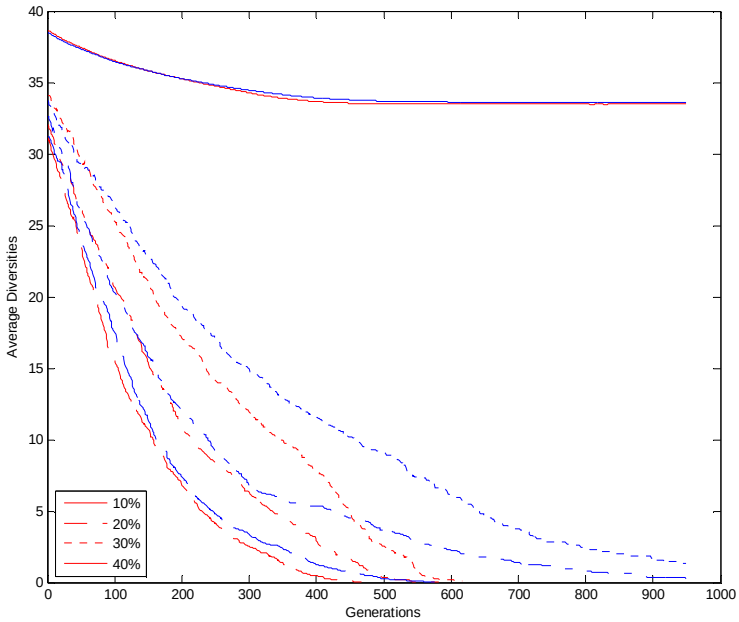


(a)

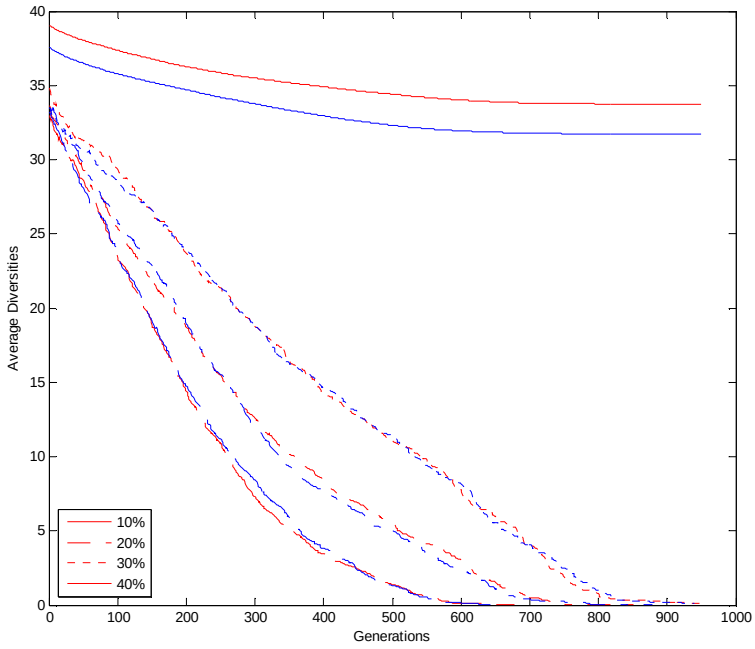


(b)

Figure 4.4. Genotypic diversities of f_{Ras} against fault rates for 'stuck at 0'. (a) 1st (red curves) and 3rd (blue curves) configurations; (b) 2nd (red curves) and 4th (blue curves) configurations.

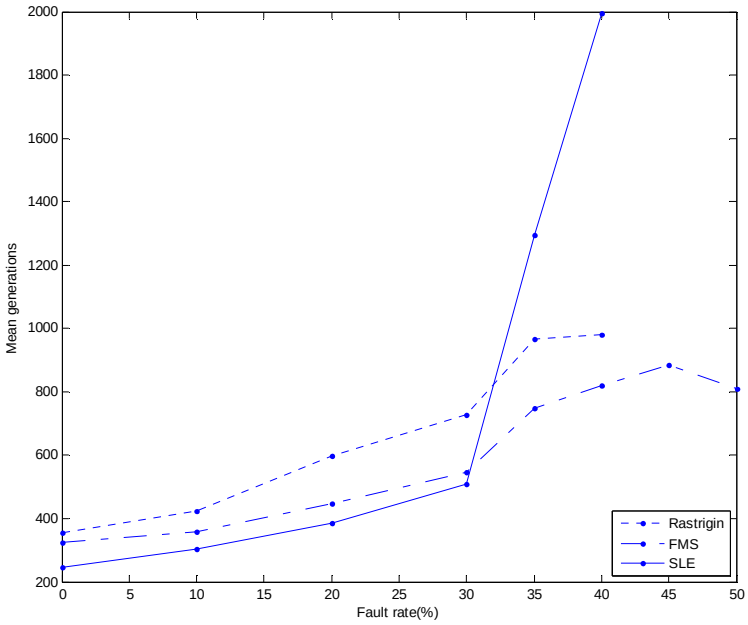


(a)

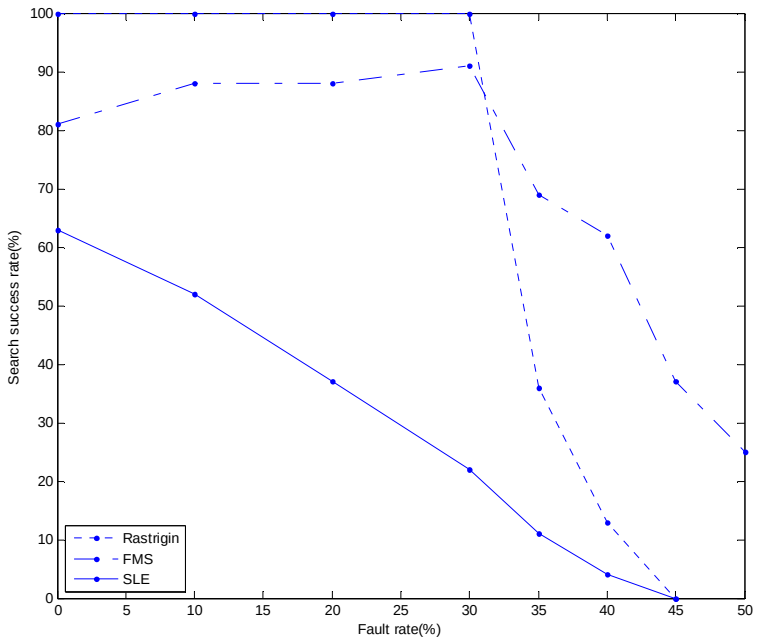


(b)

Figure 4.5. Genotypic diversities of f_{Ras} against fault rates for 'stuck at 1'. (a) 1st (red curves) and 3rd (blue curves) configurations; (b) 2nd (red curves) and 4th (blue curves) configurations.

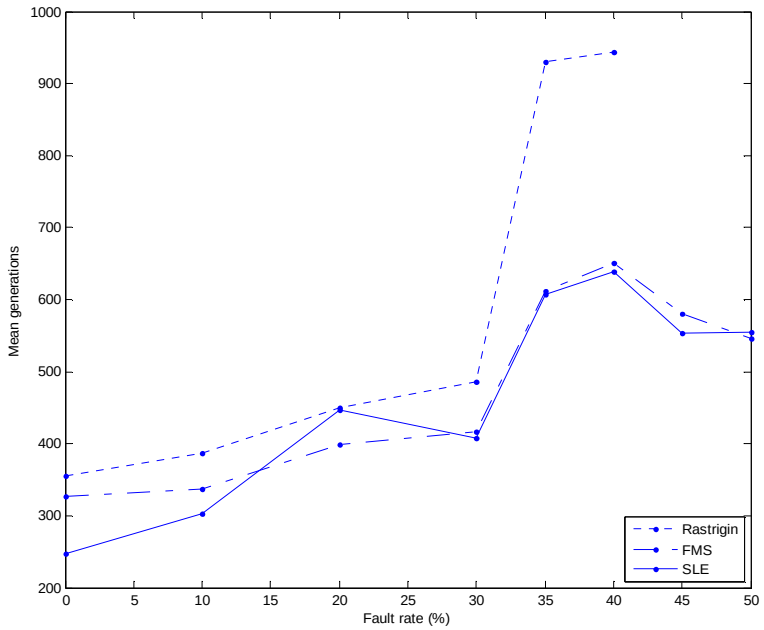


(a)

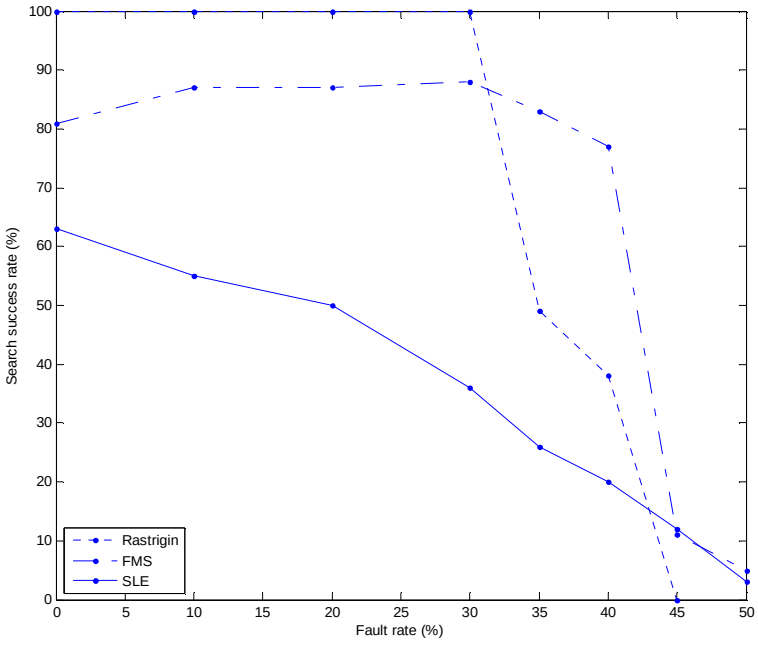


(b)

Figure 4.6. (a) Mean generations obtained by 3rd configuration for f_{Ras}, f_{FMS} and f_{SLE}; (b) search success rate obtained by 2nd configuration (for f_{Ras}) and 4th configuration (for f_{FMS} and f_{SLE}) with stuck at '0' faults.



(a)



(b)

Figure 4.7. (a) Mean generations obtained by 1st configuration for f_{Ras}, f_{FMS} and f_{SLE}; (b) search success rate obtained by 4th configuration (for f_{Ras}), and 2nd configuration (for f_{FMS} and f_{SLE}) with stuck at '1' faults.

the major difference in diversity levels obtained with 30% and 40% faults justified the sharp decline in convergence rate as well as the large increase in convergence time. Another observation is that the introduction of migration with stuck at '1' faults could not add any improvements. However, for some cases it leads to an increase in convergence time (see the blue trends in Figure 4.5(a)).

Figures 4.6 and 4.7 show the experimental results when solving f_{Ras}, f_{FMS}, and f_{SLE} with fault rates up to 50% considering the stuck at '0' and stuck at '1' faults, respectively. As mentioned previously, the other problems were not considered as the algorithm failed to solve them with faults more than 30%. Based on each performance metric, the best algorithm configuration for each problem was explored with fault rates up to 50% (the best algorithm configuration was the one with the highest rank- lowest value, refer to Tables 4.4, 4.5, 4.8, and 4.9). For example, considering f_{Ras}, the best configuration based on the efficiency is the third one for stuck at '0' faults, while the first configuration is the best one for stuck at '1' faults. The best configuration based on the efficacy is the second one for stuck at '0' faults, while the fourth configuration is the best one for stuck at '1' faults.

Considering stuck at '0' faults, solving f_{Ras} and f_{FMS} shows a gradual increase in the number of generations (see Figure 4.6(a)), while a sharp increase in convergence time was obtained when solving f_{SLE}, particularly for fault rates exceeding 30%. Conversely, a sharp drop in the search success rates was obtained when solving f_{Ras} and f_{FMS} (see Figure 4.6(b)), in particular for fault rates exceeding 30%, while a gradual drop in the convergence rate was obtained when solving f_{SLE}. The algorithm failed to solve f_{Ras} and f_{SLE} with faults more than 40%, while it converged with faults up to 50% when solving f_{FMS}. Figure 4.7(a) and (b) show the results obtained when solving the same problems with stuck at '1' faults. Generally, the algorithm failed to solve f_{Ras} with faults more than 40%; and showed a rapid increase in convergence time (see Figure 4.7(a)), as well as a rapid fall in the convergence rate (see Figure 4.7(b)) when the fault rate exceeded 30%. Solving f_{FMS} and f_{SLE}, the algorithm could converge with up to 50% faults. However, the convergence rates were very low.

In conclusion, we note that the performance of the algorithm declined significantly for fault rates of 40% or more. However, the rate of the decline differed according to the problem to solve, as each problem introduced different difficulties into the search.

4.1.4 Conclusion

This study proposed a new algorithm for tackling soft errors that target individuals' phenotypes. The algorithm is based on the canonical cGA, and is completely algorithmic (no

hardware reconfiguration). Genetic diversity is the key metric used in our approach to identify and isolate faulty cells.

This work took into consideration the most critical fault model, which is stuck at '0' faults, together with different fault rates. In general, the proposed algorithm was successful in recovering up to 40% faults with different performance rates when solving different problems. In general, the performance decayed in increments in fault rate, except when solving f_{Lang} and f_{FMS}. In order to improve the performance of the algorithm, four configurations offering different exploration/exploitation trade-offs were defined. Exploration and exploitation are two important issues in the evolution process, where the population diversity is enhanced by exploring the search space and the optimum solution can be found by exploiting the fitness information. The results show that the best efficiency is achieved by the third configuration (ST, r = 0.0 + *Migration*) for all problems. On the other hand, the best efficacy was achieved by the fourth configuration (ST, r = 0.5 + *Migration*) in 3 out of the 6 problems.

The key point underlying the introduction of migration is to cover the loss in cells due to the faults, thus enhancing the reproduction process; especially when a cell is surrounded by faulty neighbours. The results proved that migration offers a better exploration/exploitation trade-off, especially when combined with ST, r = 0.0.

4.2 Migration as a Mitigation Technique

This section introduces a number of adaptive migration schemes in order to mitigate the deterioration in the performance of Fault-Tolerant 3D-cGA, the algorithm proposed in the previous section. In addition to the migration scheme introduced in Section 4.1.2.3, the new schemes introduced in this section are tested to show superior improvements in the algorithm's performance in terms of efficiency, efficacy, and speed. In this study, several algorithm configurations related to migration are considered. The effectiveness of the algorithm is investigated using a benchmark of four test functions and three real-world problems with each presenting a different level of search difficulty. The problems are Rastrigin (f_{Ras}), Ackley (f_{Ack}), Michalewicz (f_{Mic}), Langermann (f_{Lang}), FMS (f_{FMS}), SLE (f_{SLE}) (for more details of the benchmark problems refer to Appendix A), and GPS (f_{GPS}) (which will be discussed in Section 4.2.2) problems. An investigation of the proposed migration policies that considered the problem of GPS was proposed in (Al-Naqi *et al.*, 2011a). Section 4.2.1 describes the algorithm configurations and also introduces the new adaptive migration

schemes. Section 4.2.2 provides a brief description of the problem of GPS attitude determination, while experimental results and analyses are presented in Section 4.2.3. Section 4.2.4 gives our conclusion.

4.2.1 Algorithm Configuration

The algorithm description and the pseudo-code were introduced in Section 4.1.2. As illustrated earlier, the algorithm automatically isolates the faulty individuals and consists of three stages. The first stage aims at monitoring all individuals and computing their genetic diversities by running cGA for a very short period (low number of generations). Next, the isolation stage identifies and isolates the faulty individual using feedback information from the first stage. Finally, in the third stage, another cGA is run to solve the given problem while excluding the faulty (isolated) individuals from the process (i.e., the faulty individuals are prevented from updating or communicating with other fault-free individuals when it is a part of their neighbourhood). Two fault scenarios are considered—fitness score stuck at '0' and fitness score stuck at '1'. (The fault models were previously introduced in Section 4.1.1.) The following subsection presents two new adaptive migration schemes (schemes 2 and 3) used in this work, while the first migration scheme (scheme 1) was defined in Section 4.1.2.3.

4.2.1.1 Migration Schemes

The new migration schemes defined are similar to the past migration scheme (i.e., scheme 1) in the following ways. Firstly, the migration operator frequency is set to the highest (i.e., every generation) and in each generation is applied only when there is at least a faulty individual within the current individual's neighbourhood. Secondly, the number of migrants (i.e., migration rate) is adapted and is computed whenever a migration is to be applied. This rate is equal to the number of faulty individuals, which varies from 1 to *no_of_neighbours*. Thus, all migration schemes agree in identifying which individual(s) are to be replaced. On the other hand, the three schemes differ in choosing which individual(s) are to be migrated.

In scheme 1, the migrants are chosen from the first fault-free neighbourhood found to replace the corresponding faulty individual(s) (refer to Figure 4.3). In contrast, in schemes 2 and 3, the migrants are chosen from within the current neighbourhood if at least one fault-free neighbour exists. For the case where there is no fault-free neighbour, scheme 1 is

employed. In addition, in the worst case, if there is no fault-free neighbourhood, a random neighbourhood is selected, which allows the possibility of selecting faulty individual(s).

Considering scheme 2, the best fault-free neighbour (i.e., a neighbour that has the best fitness value) is selected to replace any faulty individual within the same neighbourhood. Conversely, in scheme 3, a random fault-free neighbour is chosen as a migrant.

The aim of proposing migration schemes 2 and 3 is mainly to save time needed to search for a fault-free neighbourhood each time a fault is encountered.

4.2.2 Case study: GPS Attitude Determination

GPS technology is used in the determination of a vehicle's attitude parameters, which is achieved by calculating the correct carrier phase integer ambiguity values (Juang and Huang, 1997). One of the most efficient techniques for attitude determination using a GPS is genetic algorithm based ambiguity function search (AFGA), which was proposed by Xu *et al.* (2002). This technique is based on the observations of the GPS carrier phase and characterised as being immune to cycle slips. AFGA outperforms other techniques such as the ambiguity function method (AFM) in terms of the computational overheads incurred (Hodgart and Purivigraipong, 2000). For a full description of the technique, refer to (Xu *et al.*, 2002).

In this study, AFGA is used to determine GPS attitude. This function aims at finding the azimuth (φ) and the elevation (β) angles considering a fixed baseline (b) of 1.067 m (please refer to (Xu *et al.*, 2002) for more details) and is represented by equation (4.4):

$$AFGA(\varphi,\beta,b) = \sum_{i=1}^{m}\sum_{j=2}^{n}\cos\left(\frac{2\pi}{\lambda}\left(\frac{DD\Phi_{AB}^{1j}}{m(n-1)} - \frac{DD\Phi_{AB}^{1j}(\varphi,\beta,b)}{m(n-1)}\right)\right) \tag{4.4}$$

where $DD\Phi_{AB}^{1j}$ is the observed value of the double difference for the carrier phase, n is the number of satellites (usually 4–6 satellites), and m is the number of epochs.

The range of φ is a full 360°, while the range of β is within the interval of [–15 , +15]. This function is multimodal with strong epistasis. In addition, the optimum maximum value is very close to 1. Figure 4.8 shows the problem search space.

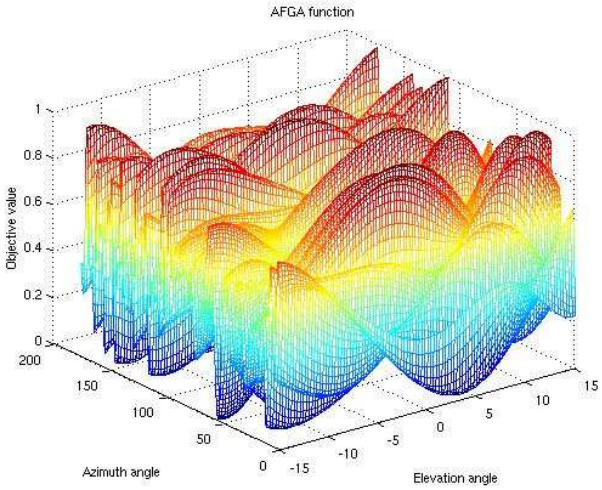


Figure 4.8. AFGA's objective function in 2D.

4.2.3 Experimental Results and Analysis

Simulation results were obtained for four configurations of the algorithm used to solve the previously mentioned problems. The first configuration applied the Fault-Tolerant 3D-cGA without migration, while the second, third, and fourth configurations combined it as follows: migration scheme 1 (first fault-free neighbourhood), scheme 2 (best fault-free neighbour), and scheme 3 (random fault-free neighbour), respectively. The parameters used in the experiments are explained and summarised in Table 4.10.

Table 4.10. Parameters used in the simulation

Population size:	343 individuals (125 for f_{GPS})
Parent selection:	Centre individual + Binary tournament
Recombination:	AX, P_c = 0.9
Mutation:	Non-uniform, P_m
Replacement:	Replace-if-better
Neighbourhood:	NEWS
Lattice:	7×7×7 (5×5×5 for f_{GPS})
Stop criterion:	Average fitness value ≤ *threshold* (≥ *threshold* for f_{GPS})

The same parameters were used for all configurations. The population size consisted of 343 individuals (125 individuals for f_{GPS}) arranged over a 7×7×7 lattice (5×5×5 for f_{GPS}). A neighbourhood was defined as seven individuals: the central individual surrounded by six

individuals (east, west, vertical north and south, and horizontal north and south). The first parent was the current individual, while the second was selected by using a binary tournament selection method. An arithmetic crossover operator with a probability of $P_c = 0.9$ was applied to generate an offspring. The offspring were mutated by applying a non-uniform mutation operator with a probability of P_m. Different mutation probabilities were defined for each problem due to their different characteristics and complexities. P_m of $1/2L$ was assigned to f_{Ras} and f_{Mic}; $1/L$ was assigned to f_{Ack}, f_{Lang}, f_{FMS}, and f_{GPS}; and $1/10L$ was assigned to f_{SLE}; where L was chromosome length. The replacement policy defined here was *replace-if-better*, during which the current individual was replaced if its competitor (offspring) was fitter. The migration parameters used in the second, third, and fourth configurations were previously described in Section 4.2.1.1.

Finally, the algorithm terminated if the *average-fitness-value* satisfied a predefined threshold. Similarly, different thresholds were defined for each problem: a value of $5e^{-5}$ was assigned to f_{Ras}, f_{Lang}, and f_{FMS}; $2e^{-4}$ was assigned to f_{Ack}; $1.89e^{-2}$ was assigned to f_{Mic}; $1e^{-1}$ was assigned to f_{SLE}; and $9.97e^{-1}$ was assigned to f_{GPS}.

During the simulations, similar ratios and fault patterns were injected for every configuration. The performance of the algorithm was measured using three metrics—the search success rate, the average number of generations, and the average run times for 100 independent runs. The defined maximum number of generations was 150 generations for f_{GPS}; 700 generations for f_{Mic} and f_{Lang}; 1000 generations for f_{Ras}, f_{FMS}, and f_{SLE}; and 2000 generations for f_{Ack}.

The experimental results are divided into two parts. The first part presents the results obtained by the algorithm for the most critical fault model (i.e., stuck at '0' faults) when solving f_{Ras}, f_{Ack}, f_{Mic}, f_{Lang}, f_{FMS}, and f_{SLE} (see Tables 4.11–4.13). The second part presents the results obtained by the algorithm when solving f_{GPS} for both defined fault models (i.e., stuck at'1' and stuck at '0') (see Tables 4.14–4.15). f_{GPS} was selected as an example to study the algorithm's behaviour and performance with more focus for both fault models. In addition, f_{GPS} differed from the other problems considered as it is a maximisation problem; resulting in the most critical fault model being stuck at '1' faults while stuck at '0' faults was the less critical model.

In Tables 4.11–4.15, the median absolute deviations *mad* (*mad* is used due to the non-normal distribution of the results obtained) are added to the results to show the robustness of the approach. In addition, significant differences are indicated by a plus sign (+), while a minus sign (−) denotes non-significant differences (details of statistical tests were provided

in Section 2.2.3.1). The best results achieved in terms of each performance metric for each fault rate is highlighted in **bold**. Furthermore, in order to provide a reference that helps in evaluating the algorithm's performance, the results obtained when there were no faults are shown for each problem; taking into account the fact that the migration operator is inactive in the absence of faults.

Although the presence and the increase in fault rate led to deterioration in the algorithm's performance by increasing the convergence time and reducing the search success rate and speed, the introduction of migration significantly reduces the convergence time and improves search success rate and speed when solving f_{Ras} (see Table 4.11).

The most significant reductions in convergence time reaching 50% were obtained with high fault rates. Furthermore, the search success rates were significantly improved when the different migration schemes were introduced. This improvement reached 100% with 40% faults. With regard to the speed, migration schemes 2 and 3 significantly reduced the running time, particularly with faults more than 10%. Generally, migration scheme 2 obtained the best algorithm performance in terms of efficiency, efficacy, and speed.

Table 4.11. Convergence time (CT), rate (CR), and speed (SP)* for the test problems

Problems/ % of faults		Without migration	Migration scheme 1 (First fault-free neighbourhood)	Migration scheme 2 (Best fault-free neighbour)	Migration scheme3 (Random fault-free neighbour)	Test
f_{Ras}	0%		266.14 ± 44.0 100% 0.71 ± 0.10 s			
	10%	372.26 ± 63.5 **100%** **0.95 ± 0.12 s**	296.94 ± 50.5 **100%** 1.68 ± 0.23 s	**294.86 ±49.0** **100%** 0.97 ±0.12 s	307.85 ± 48.5 **100%** 0.96 ± 0.14 s	+ • +
	20%	613.56 ± 93.0 93% 1.74 ± 0.39 s	352.23 ± 52.0 **100%** 2.64 ± 0.37 s	**331.10 ±57.5** **100%** **1.11 ±0.16 s**	369.17 ±60.0 **100%** 1.15 ± 0.17 s	+ + +
	30%	720.84 ± 105.0 82% 1.60 ± 0.20 s	392.39 ± 54.0 **100%** 9.18 ± 1.23 s	**351.60 ± 54.0** **100%** **1.17 ± 0.15 s**	398.86 ±77.5 **100%** 1.19 ± 0.20 s	+ + +
	40%	0%	506.17 ± 76.0 **100%** 27.38 ± 4.18 s	**401.38 ± 64.0** **100%** **1.20 ± 0.16 s**	480.29 ±65.5 **100%** 1.48 ± 0.20 s	+ + +
f_{Ack}	0%		1004.7 ± 326.0 79% 4.70 ± 1.48 s			
	10%	1247.62 ± 445.5 58% 7.07 ± 2.50 s	1361.03 ± 308.0 56% 9.30 ± 2.08 s	1221.77 ± 365.0 **62%** 5.20 ± 1.49 s	**1211.0 ± 321.0** **62%** **5.17 ± 1.33 s**	• • +
	20%	1512.96 ± 212.0 27% 7.02 ± 1.42 s	1433.83 ± 293.5 **48%** 12.52 ± 2.50 s	**1360.45 ± 376.0** **48%** **5.74 ± 1.33 s**	1435.3 ± 272.0 44% 5.90 ± 1.08 s	• + +
	30%	1481.8 ± 221.0 9% 7.07 ±1.65 s	1598.9 ± 164.0 30% 38.89 ± 4.01 s	**1399.86 ± 260.5** **46%** **5.77 ± 1.01 s**	1429.9 ± 214.0 31% 6.01 ± 0.68 s	+ + +
	40%	0%	1733.5 ± 200.5 8% 95.16 ± 9.31 s	1552.82 ± 208.0 **23%** 6.59 ± 0.67 s	**1511.6 ± 289.0** **23%** **5.96 ± 1.10 s**	• + +

* For more details about the performance measures, please refer to Section 2.2.3.1.

Table 4.12. Convergence time (CT), rate (CR), and speed (SP)* for the test problems

Problems/ % of faults		Without migration	Migration scheme 1 (First fault-free neighbourhood)	Migration scheme 2 (Best fault-free neighbour)	Migration scheme3 (Random fault-free neighbour)	Test
f_{Mic}	0%		146.55 ± 25.0 100% 1.79 ± 0.36 s			
	10%	201.22 ± 32.5 **100%** 2.33 ± 0.30 s	**169.69 ± 42.0** **100%** 1.81 ± 0.35 s	172.05 ± 27.5 **100%** **1.40 ± 0.17 s**	180.47 ± 28.5 **100%** 1.44 ± 0.18 s	+ • +
	20%	349.60 ± 58.0 98% 3.15 ± 0.71 s	197.38 ± 46.5 **100%** 2.39 ± 0.47 s	**188.96 ± 33.5** **100%** **1.39 ± 0.18 s**	233.39 ± 48.0 **100%** 1.78 ± 0.32 s	+ • +
	30%	471.62 ± 65.0 97% 34.12 ± 0.65 s	218.47 ± 41.0 **100%** 6.02 ± 1.10 s	**201.27 ± 34.5** **100%** **1.47 ± 0.19 s**	256.14 ± 42.5 **100%** 1.89 ± 0.23 s	+ • +
	40%	0%	282.07 ± 59.5 **100%** 16.46 ± 3.44 s	**226.08 ± 38.5** **100%** **1.49 ± 0.20 s**	290.63 ± 46.0 **100%** 1.94 ± 0.25 s	+ • +
f_{Lang}	0%		266.46 ± 36.0 56% 4.02 ± 0.68 s			
	10%	413.22 ± 60.0 45% 6.45 ± 1.96 s	**274.50 ± 51.0** 51% 3.78 ± 0.60 s	285.50 ± 57.0 52% 4.37 ± 0.89 s	322.29 ± 62.0 **57%** **3.66 ± 0.71 s**	+ • +
	20%	538.15 ± 57.0 39% 5.11 ± 0.51 s	332.39 ± 50.5 48% 4.91 ± 0.67 s	**311.02 ± 54.0** 49% 5.74 ± 1.54 s	371.03 ± 88.0 **52%** **3.87 ± 0.80 s**	+ • +
	30%	614.20 ± 40.0 20% 5.80 ± 0.43 s	374.40 ± 44.0 27% 11.18 ± 1.40 s	**333.22 ± 71.5** 48% 5.60 ± 1.60 s	403.05 ± 76.0 **50%** **3.73 ± 0.56 s**	+ + +
	40%	0%	438.35 ± 62.5 14% 26.40 ± 3.74 s	**354.73 ± 63.0** 41% 4.99 ± 1.18 s	425.05 ± 76.0 **51%** **3.74 ± 0.51 s**	+ + +

* For more details about the performance measures, please refer to Section 2.2.3.1.

Considering f_{Ack}, generally, the improvement in convergence time when the migration was introduced was not significant; except for 30% faults (see Test results in Table 4.11). On the other hand, the introduction of migration significantly improved search success rate, particularly for faults more than 10%. In addition, migration schemes 2 and 3 significantly reduced the running time. Overall, the best algorithm performance was achieved with migration schemes 2 and 3 when solving f_{Ack}.

Similarly, the introduction of migration significantly improved convergence time and speed when solving f_{Mic}. The improvement reached 57% for convergence time and 95% for speed (see Table 4.12). With respect to search success rate, there were no significant differences between the algorithm configurations (see Test results); however, the algorithm failed to solve the problem without migration for 40% faults. As with f_{Ras}, generally, the best algorithm performance was achieved with migration scheme 2.

Moving to f_{Lang}, in general, the introduction of migration significantly improves the algorithm's efficiency, efficacy, and speed (see Table 4.12). The best efficiency was obtained by migration scheme 2, especially for faults over 10%, while the best efficacy and

Table 4.13. Convergence time (CT), rate (CR), and speed (SP)* for real-world problems

Problems/ % of faults		Without migration	Migration scheme 1 (First fault-free neighbourhood)	Migration scheme 2 (Best fault-free neighbour)	Migration scheme3 (Random fault-free neighbour)	Test
f_{FMS}	0%		288.66 ± 84.0 63% 26.72 ± 6.45 s			
	10%	404.93 ± 91.5 **66%** 53.05 ± 10.07 s	328.90 ± 77.0 61% 29.04 ± 6.12 s	**290.69 ± 74.0** **66%** **25.87 ± 5.6 s**	344.96 ± 101.0 64% 30.18 ± 7.40 s	+ • +
	20%	520.66 ± 124.0 51% 52.07 ± 9.04 s	364.39 ± 99.5 56% **29.41 ± 6.85 s**	**327.92 ± 81.0** 57% 31.98 ± 8.15 s	378.81 ± 93.0 **59%** 29.54 ± 6.17 s	+ • +
	30%	715.88 ± 117.5 42% 48.30 ± 7.85 s	410.53 ± 91.0 47% 43.81 ± 10.34 s	**364.24 ± 85.0** **57%** **25.51 ± 5.04 s**	453.53 ± 146.5 56% 30.72 ± 9.02 s	+ + +
	40%	0%	457.14 ± 116.0 47% 55.10 ± 16.95 s	446.51 ± 126.0 47% **26.69 ± 6.46 s**	**445.82 ± 130.0** **58%** 26.82 ± 6.68 s	• • +
f_{SLE}	0%		71.31 ± 17.0 44% 0.51 ± 0.20 s			
	10%	108.79 ± 20.0 29% 0.28 ± 0.01 s	**65.40 ± 14.0** 35% 0.41 ± 0.06 s	75.93 ±12.0 33% 0.30 ± 0.03 s	80.74 ±18.0 **39%** **0.26 ± 0.03 s**	+ • +
	20%	174.21 ± 49.0 19% 0.38 ± 0.07 s	**67.23 ± 13.0** 17% 0.77 ± 0.10 s	81.10 ±13.0 19% **0.30 ± 0.04 s**	104.05 ±20.0 **20%** 0.32 ± 0.04 s	+ • +
	30%	230.77 ± 31.0 9% 0.59 ± 0.12 s	94.54 ± 28.0 11% 3.27 ± 1.0 s	**90.53 ±17.0** 13% **0.32 ± 0.03 s**	115.52 ±16.0 **19%** 0.37 ± 0.04 s	+ + +
	40%	0%	**69.00 ± 0.00** 1% 3.65 ± 0.00 s	88.16 ± 8.0 6% **0.47 ± 0.12 s**	157.44 ± 26.0 **9%** 0.48 ± 0.78 s	+ + •

* For more details about the performance measures, please refer to Section 2.2.3.1.

speed were obtained by migration scheme 3. In migration scheme 2, the fittest fault-free neighbour was selected to replace the faulty individual(s), which in turn increased the selection intensity. Consequently, the best efficiency was achieved by this scheme. On the other hand, highly complex problems such as f_{Lang} required more exploration, which was offered only by migration scheme 3. Hence, migration scheme 3 obtained the best efficacy.

Table 4.13 depicts the results obtained by all algorithm configurations when solving f_{FMS} and f_{SLE}. Significant reductions in the average number of generations was achieved when migration was introduced; with the reduction reaching 49% for f_{FMS} and 60% for f_{SLE}. Moreover, with migration, the running times were significantly reduced by up to 47% for and f_{FMS} and 45% for f_{SLE}. With regard to search success rate, generally, non-significant differences (see test results) were obtained when solving f_{FMS}, except for 30% faults. With f_{SLE}, significant differences were obtained when migration was introduced, especially for fault rates greater than 20%. Overall, the best algorithm performance was achieved by migration scheme 2 when solving f_{FMS}, while migration scheme 3 obtained the best performance when solving f_{SLE}.

Table 4.14. Convergence time (CT), rate (CR), and speed (SP)* for f_{GPS} with stuck at '1' faults

% of faults	Without migration	Migration scheme 1 (First fault-free neighbourhood)	Migration scheme 2 (Best fault-free neighbour)	Migration scheme 3 (Random fault-free neighbour)	Test
0%		23.87 ± 6.5 100% 0.09 ± 0.015 s			
10%	40.49 ± 8.0 99% 0.110 ± 0.015 s	34.00 ± 5.0 **100%** **0.110 ± 0.015 s**	**30.50 ± 5.0** **100%** **0.106 ± 0.015 s**	32.74 ± 6.5 **100%** 0.107 ± 0.015 s	+ • +
20%	67.88 ± 12.0 96% 0.137 ± 0.015 s	39.32 ± 6.0 98% 0.118 ± 0.015 s	**38.31 ± 7.0** **100%** **0.113 ± 0.015 s**	42.34 ± 6.5 **100%** 0.115 ± 0.015 s	+ + +
30%	89.43 ± 3.0 16% 0.156 ± 0.00 s	**44.36 ± 6.0** 97% 0.187 ± 0.030 s	44.80 ± 8.0 **98%** **0.118 ± 0.015 s**	54.17 ± 10.0 **98%** 0.125 ± 0.016 s	+ + +
40%	0%	**51.32** ± 10.0 94% 0.432 ± 0.078 s	53.13 ± 7.5 **98%** **0.120 ± 0.015 s**	60.04 ± 12.0 94% 0.131 ± 0.016 s	+ + +

* For more details about the performance measures, please refer to Section 2.2.3.1.

In the case of f_{GPS}, the most critical fault scenario occurred when the fitness scores of the individuals were stuck at '1'. During the update process, the local selection method selects the fittest individuals and spreads the poor solutions they provide over the population; thus, the algorithm isolates the faulty individuals. As a consequence, a smaller neighbourhood size results due to the isolation, which then deteriorates the performance of the algorithm. To maintain the population size, the migration operator is explicitly defined.

As can be seen from Table 4.14, the average number of generations increases dramatically as the faults increase, particularly when the migration operator is not used. On the other hand, the introduction of migration improves the efficiency and the robustness of the algorithm with significant differences (see Test results in Table 4.14). Migration schemes 1 and 2 achieved almost similar efficiencies, where the differences are not significant. Scheme 3 results in an almost similar performance as in the other schemes, with up to 20% faults. While for faults > 20%, the performance deteriorates significantly as compared to other two schemes.

In terms of search success rates, all three schemes performed similarly and were significantly better than the results obtained without migration. For example, observe the sharp drop in search success rate when the fault rate increased to 30% and above.

With regard to the speed, migration scheme 2 provided the best average run times, especially for high rate of faults (≥ 30%). With schemes 2 and 3, a migrant was selected from the current neighbourhood to replace the faulty individuals. Thus, the time needed to search for fault-free individuals in another neighbourhood was saved. In addition, choosing

Table 4.15. Convergence time (CT), rate (CR), and speed (SP)* for f_{GPS} with stuck at '0' faults

% of faults	Without migration	Migration scheme 1 (First fault-free neighbourhood)	Migration scheme 2 (Best fault-free neighbour)	Migration scheme 3 (Random fault-free neighbour)	Test
10%	29.51 ± 8.0	28.74 ± 5.0	**27.68 ± 5.5**	29.60 ± 7.0	•
	100%	**100%**	**100%**	**100%**	•
	0.104 ± 0.015 s	0.106 ± 0.015 s	0.105 ± 0.015 s	0.105 ± 0.015 s	•
20%	42.27 ± 8.5	33.31 ± 7.0	**29.87 ± 5.5**	36.98 ± 7.5	+
	100%	**100%**	**100%**	**100%**	•
	0.117 ± 0.015 s	0.112 ± 0.015 s	**0.105 ± 0.015 s**	0.113 ± 0.001 s	+
30%	50.93 ± 11.0	**36.35 ± 9.0**	39.25 ± 7.0	44.07 ± 8.0	+
	98%	98%	**99%**	**99%**	•
	0.121 ± 0.015 s	0.169 ± 0.0165 s	**0.116 ± 0.015 s**	0.119 ± 0.015 s	+
40%	63.44 ± 10.0	**39.75** ± 7.0	41.95 ± 8.5	51.09 ± 10.0	+
	93%	**96%**	**96%**	**96%**	•
	0.129 ± 0.015 s	0.348 ± 0.047 s	**0.116 ± 0.008 s**	0.124 ± 0.015 s	+

* For more details about the performance measures, please refer to Section 2.2.3.1.

the fittest fault-free neighbour (scheme 2) simplified the search and thus improved the efficiency of the algorithm, while selecting a random fault-free neighbour (scheme 3) reduced the efficiency. However, when all neighbours were faulty, migration scheme 1 was applied. The advantage of this scheme resides in the diversity offered as each faulty individual is replaced by a different migrant. Thus, the efficiency and the efficacy of the algorithm were improved, although the time needed increased due to the search for a fault-free neighbourhood. Thus, overall, the second scheme provided the best performance.

The other critical fault scenario occurred when the fitness scores of individuals were stuck at '0'. During the local selection, the weakest individuals are ignored leading to be implicitly isolated. Although these individuals are implicitly isolated, they may negatively affect the accuracy of solutions as they are allowed to mate with other individuals. Therefore, an explicit isolation scheme (as the when defined in this study) is required to maintain the accuracy of the results.

Considering the efficiency and speed of the algorithm (Table 4.15), similar results as that for the stuck at logic '1' fault model can be observed. On the other hand, with regard to search success rate, the differences in the results obtained with or without the use of migration were insignificant (see test results). Although this model was less critical than the first model, it was an essential issue for fault tolerant systems to handle it due to the high accuracy needed, especially for hard, real-time applications such as the problem of GPS attitude determination.

In order to provide a general conclusion, two-level ranking was performed (similar to the ranking performed in Sections 4.4.3.1 and 4.1.3.2) based on the three performance metrics independently: convergence time, convergence rate, and speed (see Tables 4.16, 4.17, and

4.18, respectively). Table 4.19 depicts a general ranks which are computed using the three metric ranks in order to find the best algorithm configurations when considering all metrics dependently. Similarly, this ranking is computed by summing the global ranks obtained based on each metric and the minimum summation result is assigned the highest rank (i.e., the lowest value).

Table 4.16. Local and global* convergence-time-based ranking

Rank	Algorithms	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	f_{GPS}	Sum
1	Migration scheme 2	1	1	1	1	1	2	1	8
2	Migration scheme 1	2	3	2	2	3	1	1	14
3	Migration scheme 3	2	1	3	3	2	3	3	17
4	Without migration	2	3	3	3	3	3	3	20

Table 4.17. Local and global* convergence-rate-based ranking

Rank	Algorithms	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	f_{GPS}	Sum
1	Migration scheme 2	1	1	1	2	1	2	1	9
1	Migration scheme 3	1	2	1	1	1	1	2	9
3	Migration scheme 1	1	3	1	2	4	2	3	16
4	Without migration	4	4	4	2	3	2	4	23

Table 4.18. Local and global* speed-based ranking

Rank	Algorithms	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	f_{GPS}	Sum
1	Migration scheme 2	1	1	1	2	1	1	1	8
2	Migration scheme 3	3	1	2	1	3	2	3	15
3	Migration scheme 1	3	3	2	2	2	3	2	17
4	Without migration	2	3	2	2	3	3	3	18

Table 4.19. Convergence-time (CT), rate (CR), and speed (SP) based-ranking†

Rank	Algorithms	Convergence-time	Convergence-rate	Speed	Sum
1	Migration scheme 2	1	1	1	3
2	Migration scheme 3	3	1	2	6
3	Migration scheme 1	2	3	3	8
4	Without migration	4	4	4	12

In all the rankings, we considered only the worst-case fault model, which was stuck at '1' faults for f_{GPS} and stuck at '0' for all other problems.

* Local ranks (columns 3 to 8) are performed for each problem independently; the highest rank is assigned the lowest value. Global ranks are performed by summing the local ranks of each problem and are shown in the first column.

† The rankings are performed by summing the global ranks computed based on each performance metric; summation values are shown in the last column while the ranks are shown in the first column.

In summary, the introduction of the migration operator added a significant advantage as the performance of the algorithm improved considerably, especially for high rates of faults. In particular, migration scheme 2 achieved the best performance overall (see Table 4.19). As mentioned previously, migration scheme 2 replaces faulty individual(s) with the fittest fault-free one within the same neighbourhood. Therefore, this scheme saves time needed to search other neighbourhoods as well as provides high selection intensity leading to reduce the number of generations required to find desired solutions. However, problems with higher complexities such as f_{Lang} and f_{SLE} need more diversity which could be offered by reducing the selection intensity. Therefore, due to the random selection of a fault-free neighbour to replace faulty individual(s) migration scheme 3 is preferred for those problems in order to improve the search success rate.

4.2.4 Conclusion

In this study, we proposed two new adaptive migration schemes in order to improve the performance of the algorithm. Simulation results demonstrate that the new migration schemes excelled in improving the efficiency, efficacy, and speed considerably, in particular migration scheme 2, thereby enhancing the reliability of the algorithm, especially for high rates of faults.

Besides being a mitigation technique, the integration of migration has played an important role in controlling the exploration/exploitation trade-off. Exploration and exploitation are the two main issues in enhancing the performance of evolutionary algorithms. Population diversity is improved by exploring the search space, while the optimum solution can be found by exploiting the fitness information. In this work, the best overall performance in terms of efficiency, efficacy, and speed was achieved with migration scheme 2 due to its effect in enhancing the local selection intensity and population diversity.

The grid topology (i.e., 3D grid) has also contributed to the effectiveness of the algorithm. The vertical expansion of cells leads to shorter diameter and denser neighbourhood compared to 2D grids with similar neighbourhood topology and equal population size. Therefore, it can be concluded that for problems of high degree of complexity, higher cellular dimensions could be beneficial.

4.3 Dynamic Fault Tolerant 3D-cGA

This section presents a new Dynamic Fault-Tolerant 3D-cGA (Dynamic FT 3D-cGA) that is based on the canonical cGA search model discussed earlier. In this study, the proposed algorithm is a modified version of Fault-Tolerant 3D-cGA—the algorithm previously proposed in Section 4.1. In order to improve the performance and reliability of Fault-Tolerant 3D-cGA, new adaptive migration schemes were introduced in Section 4.2, while this section introduces dynamic adaptation schemes to achieve further improvement. The same test bench suite is used to test the performance of the Dynamic FT 3D-cGA. The suite includes the Rastrigin (f_{Ras}), Ackley (f_{Ack}), Michalewicz (f_{Mic}), Langermann (f_{Lang}), FMS (f_{FMS}), SLE (f_{SLE}), and GPS (f_{GPS}) problems (f_{GPS} was discussed in Section 4.2.2, while the details of the remaining benchmark problems are provided in Appendix A). Furthermore, different algorithm configurations are defined considering the introduction of migration and two different dynamic adaptation schemes. The description of the algorithm, configurations, and dynamic adaptation schemes are presented in Section 4.3.1. Section 4.3.2 discusses and analyses the simulation results obtained with the various algorithm configurations. Concluding remarks are given in Section 4.3.3.

4.3.1 Algorithm Configuration

As the proposed algorithm (i.e., Dynamic FT 3D-cGA) is an improved version of the FT 3D-cGA, this section starts with a brief description of FT 3D-cGA (for more details refer to Section 4.1). Next, a description of the dynamic features added is given. FT 3D-cGA automatically isolates the faulty individuals and consists of three phases. In the first phase, the changes in the genetic diversity of each individual is observed independently and computed by running a cGA for a few generations. In the second phase (i.e., the isolation phase), the faulty individuals are determined and isolated using genetic information from the first phase. Finally, another cGA is run, until the termination criterion is satisfied, to solve the given problem while excluding the faulty individuals from the process. In this study, the Dynamic FT 3D-cGA follows the same first and second phases as in the FT 3D-cGA. However, the difference resides in the third phase as it starts with a new derived value of maximum number of generations (*MaxGens*) based on the ratio of faults encountered in the preceding phase. The dynamic setting of *MaxGens* aims at balancing the number of

evaluations due to the reduction in the number of individuals alive, and will be introduced later in this section.

In Section 4.2 it has been concluded that the migration scheme 2 was the best. However, in this section, the migration scheme 1 (was proposed in Section 4.1.2.3) is employed as the main concern of this study is the influence of the dynamic mechanism on FT 3D-cGA. In addition, the fault models considered and isolation criterion were proposed in Sections 4.1.1 and 4.1.2.2, respectively.

The local selection method defined in this study is the binary tournament selection (BT), in which two random individuals are selected and the fittest individual wins the tournament. The crucial role of the local selection method comes from its direct effect on the global selection pressure. The selection pressure determines the convergence speed as well as population diversity (Simoncini *et al.*, 2007), and can be evaluated by monitoring the growth of the best individual (by letting the selection be the only active operator) (Goldberg and Deb, 1991). Figure 4.9 shows an average of 100 independent runs for the growth of the best individual over a cubic grid against different fault rates, where the population size is 343 (7 × 7 × 7).

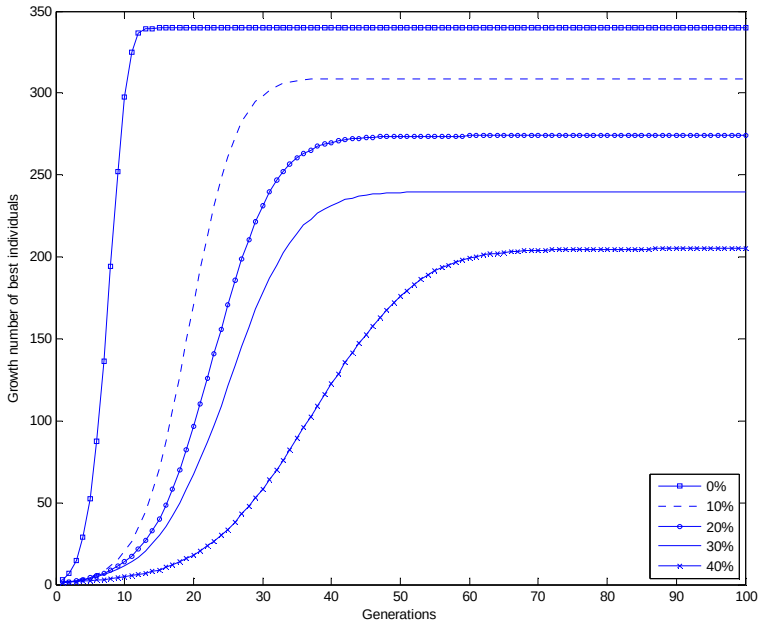


Figure 4.9. Growth curves of the best individual for various fault ratios using BT.

As shown in Figure 4.9, the growth of the best individual without faults (0%) sharply increases to conquer the whole population, thus promoting exploitation by increasing the global selection intensity on the population. Although this exploitative behaviour may improve the efficiency of the algorithm, a premature convergence may occur and negatively affect the algorithm's efficacy. Conversely, with faults, the behaviour tends to be more explorative (see Figure 4.9). However, an explicit migration operator is defined in this research not only to mitigate the impact of faults that occurred, but also to enhance the exploration/exploitation trade-off, and thus improve the performance of the algorithm (Al-Naqi *et al.*, 2011a).

4.3.1.1 Dynamic Adaptation Schemes

This section presents two dynamic adaptation schemes: $MaxGens_1$ and $MaxGens_2$. The basic idea is to adapt the value of *MaxGens* based on the fault ratio identified in the isolation phase. In other words, the value of *MaxGens* is dynamically tuned at the start of the final phase, and the initial value of *MaxGens* is manually set.

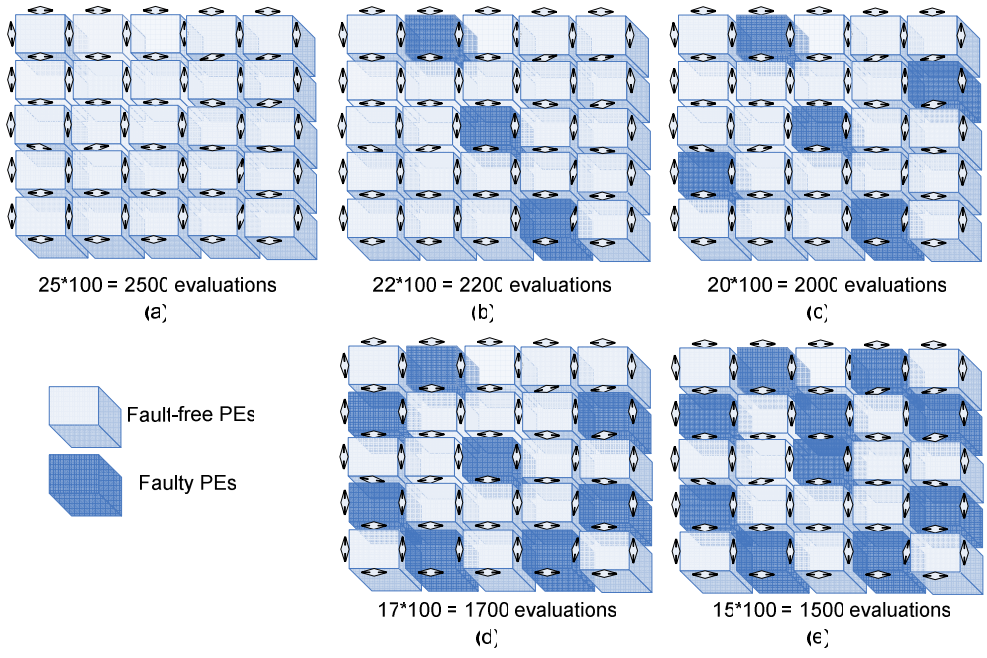
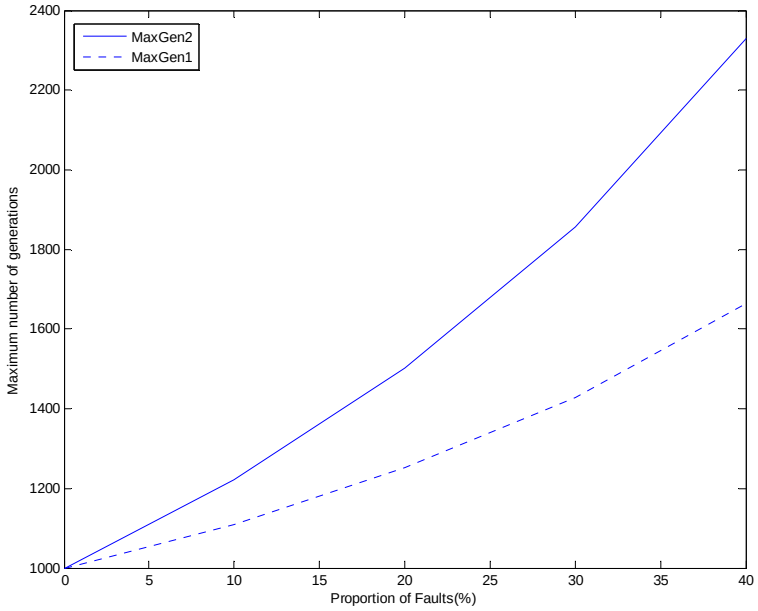
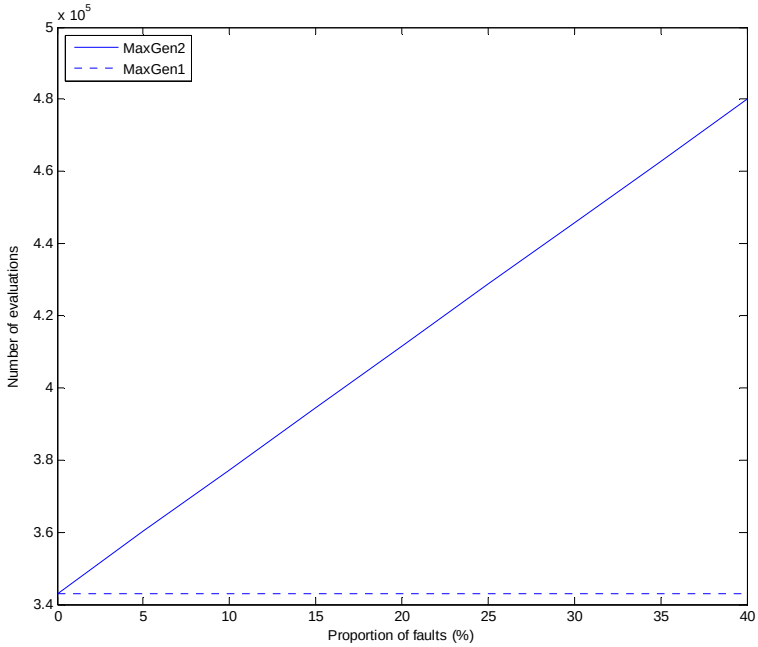


Figure 4.10. Fitness evaluations for various fault ratios. (a) 0% faults, (b) 10% faults, (c) 20% faults, (d) 30% faults, (e) 40% faults.



(a) Maximum number of generations (*MaxGens*) versus proportion of faults.



(b) Number of fitness evaluations versus proportion of faults.

Figure 4.11. *MaxGens* and fitness evaluations as a function of fault ratio for a population size of 343 individuals.

Isolating faulty individuals reduces the number living, which reduces the virtual size of the population. This thus minimises the chance of finding the optimum solutions, and deteriorates the performance of the algorithm, especially when solving complex problems (Cantu-Paz, 1995). For example, by assuming a population of 25 individuals and *MaxGens* of 100, the number of fitness evolutions is 2500 for 0% faults, 2200 for 10% faults, and so on (see Figure 4.10). Therefore, the value of *MaxGens* dynamically increases as a function of fault ratio to offer a similar number of evaluations as with 0% faults (e.g., 125 generations are needed for 20% faults (see Figure 4.10)).

The computation of the new *MaxGens* is shown in (4.5). The actual population size is denoted by *popSize*, while the number of living individuals is indicated by *aliveSize*. The initially defined maximum number of generations is indicated by *gens*.

$$MaxGens_1 = \frac{(popSize * gens)}{aliveSize}. \tag{4.5}$$

In addition, we define a further increase in *MaxGens* to tackle added difficulties caused by faults, although this leads to increased computation cost. Equation 4.6 describes this situation, where *Pfaults* indicates the ratio of the faults.

$$MaxGens_2 = \frac{(popSize * gens)}{aliveSize} * (1 + Pfaults). \tag{4.6}$$

Locally, the selection of a second parent is limited by the number of fault-free neighbours of the current individual. The solutions provided by individuals surrounded by a considerable number of faulty neighbours are less likely to be optimised, which introduces more difficulty in search, particularly with high fault ratios.

Figure 4.11(a) shows the increment of *MaxGens* as a function of the fault ratio. For example, if *gens* is 1000, *MaxGens* will be 1110, based on equation (4.5) and 1221, based on equation (4.6) for 10% faults, and so on. From Figure 4.11(b), we see that the number of evaluations is maintained for all fault ratios based on equation (4.5); thus, there is no added computation cost. Based on equation (4.6), the number of evaluations increases in proportion to the fault ratio (e.g., 10% for 10% faults, and so on).

4.3.2 Experimental Results and Analysis

This section first introduces the parameters and performance metrics used in experiments. Next, the results obtained for the FT 3D-cGA on the test suite described earlier are presented and analysed. Then, the results obtained when solving all the problems using the Dynamic FT 3D-cGA with both adaptation schemes are presented and analysed. Finally, a comparison between the FT 3D-cGA and the Dynamic FT 3D-cGA with the best adaptation scheme is provided.

Table 4.20 shows the parameters that were used in the experiment. For all the problems, the same parameters were employed in order to achieve a fair comparison. A population size of 343 individuals was used. These were arranged over a 7×7×7 lattice. As an exception, for $f_{GPS,}$ a population of 125 individuals organised over a 5×5×5 lattice was used due to its low-dimensional space (n = 3 for f_{GPS} versus n = 10 for all other problems; except for f_{FMS} as n = 6). The local neighbourhood contained seven individuals, which were positioned to the east, west, vertical north and south, and horizontal north and south, plus the central one. The first parent was the current individual, while the second parent was selected by using binary tournament selection. An arithmetic crossover operator with a rate of P_c = 0.9 was applied to generate an offspring. The offspring was mutated by a non-uniform mutation operator, with rate P_m.

Table 4.20. Parameters used in the experiments

Population size:	343 individuals, 125 individuals for f_{GPS}
Parent selection:	Centre +BT
Recombination:	AX, P_c = 0.9
Mutation:	Non-uniform, P_m
Replacement:	Replace-if-better
Neighbourhood:	NEWS
Lattice:	7×7×7 (5×5×5 for f_{GPS})
Termination criterion:	Average fitness value ≤ *threshold* (≥ *threshold* for f_{GPS})

Table 4.21. Convergence time (CT), rate (CR), and speed (SP)* for benchmark problems when there were no faults

Performance metrics	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	f_{GPS}
Average no. of generations	266.29 ± 47.00	914.94 ± 428.5	153.78 ± 34.00	262.42 ± 36.00	294.65 ± 74.0	59.83 ± 9.5	23.87 ± 6.5
Search success rate	100%	78%	100%	57%	63%	54%	100%
Average run times (seconds)	0.71	4.71	1.34	4.80	49.27	0.81	0.09

* For more details about the performance measures, please refer to Section 2.2.3.1.

A different value of P_m was assigned for each problem due to their different complexities: $P_m = 1/2L$ for f_{Ras} and f_{Mic}, $P_m = 1/10L$ for f_{SLE}, and $P_m = 1/L$ for other problems; L is the length of the chromosome. The replacement policy defined was *replace-if-better*, during which the current individual was replaced if its competitor (offspring) was fitter. The migration parameters used were previously described in Section 4.1.2.3. Finally, the algorithm terminated if the *average-fitness-value* satisfied a predefined threshold. A different threshold was defined for each problem: ≥ 0.997 for f_{GPS}, $\leq 1e^{-4}$ for f_{SLE}, $\leq 2e^{-4}$ for f_{Ack}, $\leq 1.89e^{-2}$ for f_{Mic}, and $\leq 5e^{-5}$ for other problems[*].

In addition, for each problem, the assigned value of the initial maximum number of generations was 150 generations for f_{GPS}, 700 generations for f_{Lang} and f_{Mic}, 1000 generations for f_{Ras}, f_{FMS}, and f_{SLE}, and 2000 generations for f_{Ack}[†].

Similar ratios and fault patterns were injected for each algorithm and problem. The performance of the algorithms was measured using three metrics: the search success rate (i.e., the efficacy), the average number of generations (i.e., the efficiency), and the average execution times of 100 independent runs.

Tables 4.22–4.27 present the results obtained for the FT 3D-cGA and the Dynamic FT 3D-cGA. Each algorithm was tested with and without the employment of the migration technique introduced earlier to investigate the effectiveness of the migration. Furthermore, to show the robustness of the algorithms, the median absolute deviations, *mad*, was added to the results obtained (*mad* is used due to the non-normal distribution of the results obtained). The best results achieved for each fault ratio are marked in **bold**. Significant improvement is indicated by a plus sign (+), while a non-significant difference is denoted by a dot (•) (details about the statistical tests were provided in Section 2.2.3.1). Furthermore, the results obtained when there were no faults are shown in Table 4.21, taking into account the fact that the migration operator and the adaptation scheme are inactive in the absence of faults.

4.3.2.1 Fault-Tolerant 3D-cGA

This subsection discusses and compares the results obtained for the FT 3D-cGA with and without migration when solving the problems of the test suite.

Tables 4.22–4.24 depict the results obtained. In general, FT 3D-cGA showed its ability and successfully solved different problems with up to 40% faults, especially when it was

[*] These thresholds were obtained by carrying out preliminary experiments which aimed to identify a single threshold for each problem that results in the most desirable overall performance.

[†] Similarly, the number of generations for each problem was chosen to provide the most appropriate trade-off between algorithm performance and time constraints.

combined with the migration technique. Further, the introduction of migration provided significant improvements in terms of all performance metrics considered.

As can be seen from Table 4.22, the efficiency of the algorithm significantly improved with the migration to reach up to 74% for all real-world problems and most of the test functions (see test results), except for f_{Ack}. Complex problems require high diversity levels to achieve reliability.

Table 4.22. Convergence time (CT)* obtained for FT 3D-cGA with/without migration

Algorithms	Problem/ Fault ratio	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	f_{GPS}
Fault-tolerant 3D-cGA *Without migration*	10%	366.9 ± 49.5	**1246.4** ± 322.0	215.96 ± 34.0	399.60 ± 53.5	348.10 ± 71.5	89.24 ± 10.0	40.49 ± 8.0
	20%	591.66 ± 78.0	**1343.0** ± 303.0	363.86 ± 62.0	540.79 ± 85.0	636.82 ± 137.0	187.15 ± 35.0	67.88 ± 12.0
	30%	738.5 ± 95.0	**1412.3** ± 183.5	466.41 ± 56.0	621.25 ± 41.5	723.30 ± 122.0	308.73 ± 88.0	89.43 ± 3.0
	40%	- ± 0.00	- ± 0.00	- ± 0.00	- ± 0.00	- ± 0.00	- ± 0.00	- ± 0.00
Fault-tolerant 3D-cGA *With migration*	10%	**304.58** ± 51.0	1269.0 ± 341.0	**175.39** ± 35.5	**304.51** ± 63.0	**319.80** ± 94.5	**64.75** ± 12.0	**29.80** ± 6.0
	20%	**353.98** ± 48.5	1426.6 ± 234.0	**202.8** ± 44.0	**340.78** ± 66.0	**352.89** ± 85.0	**74.89** ± 19.0	**40.48** ± 8.0
	30%	**424.26** ± 49.0	1576.7 ± 170.0	**222.08** ± 46.5	**419.89** ± 64.0	**359.66** ± 114.0	**79.91** ± 9.0	**47.45** ± 9.0
	40%	**490.89** ± 55.5	**1642.8** ± 166.5	**289.98** ± 59.0	**435.79** ± 49.0	**492.30** ± 185.0	**152.60** ± 31.0	**48.78** ± 9.5
Tests		+	•	+	+	+	+	+

Table 4.23. Convergence rate–CR* (%) obtained for FT 3D-cGA with/without migration

Algorithms	Problem/ Fault ratio	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	f_{GPS}
Fault-tolerant 3D-cGA *Without migration*	10%	**100**	60	**100**	**46**	58	**33**	99
	20%	90	23	99	**34**	51	**26**	96
	30%	83	8	95	12	47	**15**	16
	40%	0	0	0	0	0	0	0
Fault-tolerant 3D-cGA *With migration*	10%	**100**(.)	**66**(.)	**100**(.)	45(.)	**62**(.)	24(.)	**100**(.)
	20%	**100**(+)	**45**(+)	**100**(.)	33(.)	**56**(.)	19(.)	**99**(.)
	30%	**100**(+)	**33**(+)	**100**(+)	**29**(+)	**53**(.)	12(.)	**96**(+)
	40%	**100**(+)	**12**(+)	**100**(+)	**15**(+)	**43**(+)	**5**(+)	**94**(+)

Table 4.24. Speed–SP* (seconds) obtained for FT 3D-cGA with/without migration

Algorithms	Problem/ Fault ratio	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	f_{GPS}
Fault-tolerant 3D-cGA *Without migration*	10%	0.88	**6.35**	1.56	5.42	50.59	**1.09**	0.110
	20%	1.36	**6.45**	2.12	6.00	58.11	**1.20**	0.137
	30%	1.52	**6.41**	2.43	5.52	75.4	**1.31**	0.156
	40%	-	-	-	-	-	-	-
Fault-tolerant 3D-cGA *With Migration*	10%	**0.86**	**6.35**	**1.44**	**5.20**	**44.53**	1.48	**0.104**
	20%	**1.14**	7.50	**1.55**	**5.80**	**51.81**	2.20	**0.114**
	30%	**1.32**	7.46	**1.58**	**5.40**	**59.21**	2.04	**0.120**
	40%	**1.50**	**7.81**	**1.85**	**5.40**	**61.05**	**2.26**	**0.125**

* For more details about the performance measures, please refer to Section 2.2.3.1.

The increase in fault ratio promotes exploitation by lowering the diversity level due to the isolated individuals; thus, the reliability of the algorithm is deteriorated. For example, the search success rates obtained without migration when solving f_{Ack} were very low, particularly for 20% faults and above. On the other hand, the introduction of migration enhanced diversity as the migrant individuals replaced the isolated (faulty) ones, which significantly increased the search success rates. The significant differences between the search success rates obtained with and without migration justify the lower number of generations achieved without migration when solving f_{Ack}. Overall, the migration assists in obtaining robustness of the algorithm (see *mad* values), especially for 20% faults and above; with the exceptions being as a result of the very low search success rates obtained when the migration was not employed.

Table 4.23 shows the search success rates obtained. In general, it can be seen that FT 3D-cGA with migration obtained higher search success rates for all test functions and most real-world problems, reaching up to 100%. However, an exception was for f_{SLE}, where a decline in the search success rates is observed for up to 30% faults. Nevertheless, this deterioration is not significant (see test results). For 40% faults, a significant improvement in the search success rate was achieved, although the rate obtained was very low (5%).

The average execution times are shown in Table 4.24. For most of the problems, employment of the migration technique led to speeding up of the execution time, reaching up to 35%. However, the most important exception was encountered when solving f_{SLE} due to low search success rate obtained. f_{SLE} is a rather difficult problem and was extensively affected by the faults that occurred. This fact induces a negative effect on the performance and reliability of the algorithm.

In summary, the use of migration as a mitigation technique to achieve fault tolerance added considerable improvements in terms of efficiency, efficacy, speed, and reliability of the algorithm, especially for the high ratio of faults.

4.3.2.2 Dynamic Fault-Tolerant 3D-cGA

This subsection presents and analyses the results obtained for the Dynamic FT 3D-cGA when solving the problems of the test suite. The proposed dynamic mechanism adapts the permitted maximum number of generations to solve a given problem based on the number of faulty individuals observed. The proposed algorithm was tested with and without migration, as well as for each of the two adaptation schemes defined earlier, $MaxGens_1$ and $MaxGens_2$, to explore the influence of migration and the increment in the number of fitness evaluations

on the performance of the algorithm.

Dynamic FT 3D-cGA with MaxGens$_1$

Tables 4.25–4.27 exhibit the results obtained. Generally, for all the problems, the employment of the migration technique resulted in better efficiency with significant differences (Table 4.25), except for $f_{Ack,}$ as the improvement was not significant (see Test results). The robustness and improvement rate of the efficiency increased together with the increment in the fault ratio (see *mad* values). These improvements reached up to 42% reduction in the number of generations, and most importantly, when similar search success rates were obtained by both algorithm configurations. For example, when solving f_{Mic} for 20% faults, the average number of generations obtained was 340.92 without migration and 198.39 with migration, whereas a search success rate of 100% was obtained for both.

Considering the search success rate, with the use of migration, the algorithm was found to obtain higher efficacy for most of the problems, except for f_{SLE} (see Table 4.26). Although migration enhanced the population diversity, the algorithm showed an exploitative behaviour when solving f_{SLE} as the search success rate deteriorated. Owing to the diverse characteristics and complexities, different problems need different exploration/exportation degrees; f_{SLE} is a very complex problem. In addition, although the performance of the algorithm could be improved by tuning the parameters to suit a particular problem, it is not our concern in this study.

Table 4.25. Convergence time (CT)* obtained for Dynamic FT 3D-cGA with *MaxGens$_1$*

Algorithms	Problem/ Fault ratio	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	f_{GPS}
Dynamic Fault-tolerant 3D-cGA *Without migration*	10%	379.08 ± 64.00	1468.32 ± 413.0	202.89 ± 28.50	364.43 ± 45.00	409.15 ± 86.00	104.90 ± 25.00	38.82 ± 7.00
	20%	631.63 ± 97.00	1787.13 ± 359.0	340.92 ± 45.00	567.23 ± 83.00	606.58 ± 138.0	176.38 ± 40.00	58.39 ± 6.00
	30%	812.32 ± 104.5	2180.14 ± 302.5	476.36 ± 66.00	718.17 ± 82.00	777.31 ± 159.0	249.6 ± 35.50	83.26 ± 5.00
	40%	-	-	1053.0 ± 00.00	679.00 ± 00.00	1289.42 ± 146.5	332.00 ± 00.00	-
Dynamic Fault-tolerant 3D-cGA *With migration*	10%	**317.00** ± 59.00	**1406.98** ± 278.0	**166.30** ± 33.00	**299.80** ± 65.00	**339.79** ± 101.0	**78.51** ± 18.00	**30.10** ± 6.00
	20%	**340.29** ± 53.00	**1720.50** ± 384.0	**198.39** ± 37.50	**341.04** ± 58.00	**318.52** ± 81.00	**71.68** ± 14.00	**38.81** ± 6.00
	30%	**415.33** ± 64.50	**2013.64** ± 429.0	**246.30** ± 42.00	**387.26** ± 76.50	**429.0** ± 132.0	**83.00** ± 13.00	**43.88** ± 7.00
	40%	**502.82** ± 66.00	**2518.65** ± 345.5	**284.80** ± 67.50	**470.12** ± 66.50	**502.76** ± 151.0	**138.00** ± 00.00	**49.70** ± 11.50
Tests		+	•	+	+	+	+	+

* For more details about the performance measures, please refer to Section 2.2.3.1.

Table 4.26. Convergence rate–CR* (%) obtained for Dynamic FT 3D-cGA with *MaxGens*$_1$

Algorithms	Problem/ Fault ratio	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	f_{GPS}
Dynamic Fault-tolerant 3D-cGA *Without migration*	10%	**100**	55	**100**	48	58	**33**	89
	20%	93	43	**100**	39	51	**21**	66
	30%	92	20	98	29	55	**14**	19
	40%	0	0	1	1	18	**1**	0
Dynamic Fault-tolerant 3D-cGA *With migration*	10%	**100**(.)	**70**(+)	**100**(.)	**51**(.)	**64**	25(.)	**95**(.)
	20%	**100**(+)	**64**(+)	**100**(.)	**41**(.)	**53**	19(.)	**95**(+)
	30%	**100**(+)	**65**(+)	**100**(.)	**30**(.)	**56**	11(.)	**95**(+)
	40%	**100**(+)	**52**(+)	**100**(+)	**16**(+)	**43**	1(.)	**94**(+)

Table 4.27. Speed–SP* (seconds) obtained for Dynamic FT 3D-cGA with *MaxGens*$_1$

Algorithms	Problem/ Fault ratio	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	f_{GPS}
Dynamic Fault-tolerant 3D-cGA *Without migration*	10%	0.91	6.90	1.52	5.33	54.68	**1.16**	0.115
	20%	1.49	**7.75**	2.21	6.49	**59.90**	**1.46**	0.133
	30%	1.69	**8.73**	2.53	6.81	65.55	**1.71**	**0.151**
	40%	-	-	5.07	7.55	81.42	**2.33**	-
Dynamic Fault-tolerant 3D-cGA *With migration*	10%	**0.89**	**6.85**	**1.40**	**5.16**	**51.68**	1.58	**0.106**
	20%	**1.02**	8.22	**1.55**	**5.86**	60.29	2.12	**0.118**
	30%	**1.28**	9.35	**1.71**	**5.95**	**64.78**	2.79	0.188
	40%	**1.52**	**10.90**	**1.87**	7.71	**63.35**	3.92	**0.421**

Table 4.27 shows the average execution times. In general, for most of the problems, a faster speed was achieved with migration, except for f_{Ack} and f_{SLE}. Although the migration reduced the average number of generations, the average execution times needed by the algorithm to solve f_{Ack} was slightly increased to reach up to 6%. Furthermore, the average execution times needed to solve f_{SLE} was significantly increased to reach up to 40%. This behaviour is due to the difficult search incurred when solving complex problems, especially for high fault ratios.

Dynamic FT 3D-cGA with MaxGens$_2$

Tables 4.28–4.30 show the results obtained. It should be noted that the algorithm shows similar behaviour as with *MaxGens*$_1$ in terms of efficiency, efficacy, and speed. In summary, with migration, the algorithm shows significantly better efficiency and stronger robustness for all problems (Table 4.28), as well as higher search success rates (Table 4.29). Additionally, considerable improvement in terms of speed is achieved due to the migration (Table 4.30).

A difference in the algorithm's behaviour was observed when solving $f_{Ack,}$ as the search success rate obtained increased marginally when the fault ratio also increased. This odd

* For more details about the performance measures, please refer to Section 2.2.3.1.

behaviour is due to the increase in the number of fitness evaluations, i.e., the initial maximum number of generations defined is not the best to solve this problem. Thus, the new $MaxGens_2$ calculated improves the search success rates by offering more fitness evaluations. This confirms that f_{Ack} needs more exploration, which can be promoted by offering more generations. However, we continued with the initial maximum number of generations defined (2000 generations for f_{Ack}) to test the algorithm, as time was a critical factor.

Table 4.28. Convergence time (CT)* obtained for Dynamic FT 3D-cGA with $MaxGens_2$

Algorithms	Problem/ Fault ratio	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	f_{GPS}
Dynamic Fault-tolerant 3D-cGA *Without migration*	10%	397.14 ± 66.0	**1328.7** ± 456.5	220.55 ± 32.5	380.19 ± 73.0	382.14 ± 72.0	96.62 ± 22.0	40.02 ± 7.0
	20%	694.17 ± 148.0	1933.2 ± 495.0	340.98 ± 57.0	625.93 ± 110.5	581.73 ± 123.0	181.83 ± 33.0	69.54 ± 9.0
	30%	898.92 ± 153.5	2523.9 ± 907.0	468.95 ± 70.5	764.27 ± 140.5	852.33 ± 169.5	219.07 ± 37.0	111.52 ± 13.0
	40%	977.00 ± 0.0	- ± 0.0	1264.4 ± 115.0	- ± 0.0	1670.7 ± 282.0	1649.0 ± 0.0	162.19 ± 13.0
Dynamic Fault-tolerant 3D-cGA *With migration*	10%	**294.17** ± 37.5	1430.2 ± 431.5	**175.63** ± 40.0	**301.15** ± 51.0	**309.50** ± 21.00	**69.87** ± 13.0	**31.24** ± 6.5
	20%	**339.45** ± 52.5	**1914.4** ± 440.0	**191.36** ± 35.0	**326.20** ± 60.5	**427.40** ± 139.0	**66.31** ± 10.0	**39.38** ± 7.0
	30%	**414.00** ± 62.5	**2348.2** ± 460.5	**227.75** ± 42.5	**404.47** ± 74.5	**447.27** ± 118.5	**99.33** ± 16.0	**41.36** ± 7.0
	40%	**499.11** ± 37.5	**3028.6** ± 448.5	**294.82** ± 56.0	**485.95** ± 55.5	**646.04** ± 193.0	**162.00** ± 0.0	**52.04** ± 10.0
Tests		+	•	+	+	+	+	+

Table 4.29. Convergence rate–CR* (%) obtained for Dynamic FT 3D-cGA with $MaxGens_2$

Algorithms	Problem/ Fault ratio	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	f_{GPS}
Dynamic Fault-tolerant 3D-cGA *Without migration*	10%	**100**	64	**100**	**51**	63	**45**	91
	20%	97	57	**100**	46	**60**	**25**	87
	30%	95	44	**100**	36	54	**13**	71
	40%	1	0	33	0	38	**1**	56
Dynamic Fault-tolerant 3D-cGA *With migration*	10%	**100**(.)	**78**(+)	**100**(.)	**51**(.)	**65**(.)	31(–)	**100**(+)
	20%	**100**(.)	**79**(+)	**100**(.)	**48**(.)	55(.)	19(.)	**100**(+)
	30%	**100**(+)	**82**(+)	**100**(.)	**40**(.)	**55**(.)	6(.)	**99**(+)
	40%	**100**(+)	**84**(+)	**100**(+)	**20**(+)	**46**(.)	**1**(.)	**99**(+)

Table 4.30. Speed–SP* (seconds) obtained for Dynamic FT 3D-cGA with $MaxGens_2$

Algorithms	Problem/ Fault ratio	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	f_{GPS}
Dynamic Fault-tolerant 3D-cGA *Without migration*	10%	0.97	**6.50**	1.58	**5.63**	**51.51**	**1.06**	0.11
	20%	1.58	**8.23**	2.05	7.23	**63.09**	**1.65**	0.14
	30%	1.91	**10.08**	2.41	8.07	79.30	**2.25**	0.17
	40%	3.93	-	6.10	-	105.4	**2.94**	0.20
Dynamic Fault-tolerant 3D-cGA *With migration*	10%	**0.89**	6.73	**1.43**	5.77	68.60	1.46	**0.105**
	20%	**1.08**	8.94	**1.59**	**6.40**	87.02	2.64	**0.113**
	30%	**1.31**	10.72	**1.60**	**7.73**	**62.46**	3.93	**0.114**
	40%	**1.51**	**12.94**	**1.85**	**10.51**	**77.56**	5.14	**0.126**

* For more details about the performance measures, please refer to Section 2.2.3.1.

Table 4.31. Comparison of $MaxGens_2$ versus $MaxGens_1$ in terms of convergence time (CT) and rate (CR)*

Problem	Without migration	With migration
f_{Ras}	•, •	•, •
f_{Ack}	•, +	•, +
f_{Mic}	•, •	•, •
f_{Lang}	•, •	•, •
f_{FMS}	•, •	•, •
f_{SLE}	•, •	•, •
f_{GPS}	•, +	•, +

* For more details about the performance measures, please refer to Section 2.2.3.1.

Let us now proceed to compare the two adaptation schemes discussed earlier (see Table 4.31). The aim of this comparison is to show the influence of increasing the number of fitness evaluations, when the number of faults increases, on the performance of the algorithm, in particular, the efficiency and efficacy. The speed follows similar behaviour pattern as the efficiency.

Although the initial expectation is that better efficiency and speed is achieved with $MaxGens_1$, this is not always the case. The main cause of this surprising behaviour is the existence of faults, which added more difficulty to the search, and thus more generations and time were needed to determine the desired solutions. From the above Tables, generally better efficiency and speed were achieved with $MaxGens_1$ for most of the problems; however, the differences were not significant (non-significant differences are indicated by the symbol '•' in Table 4.31). Higher search success rates were obtained with $MaxGens_2$, with differences that were not significant for most of the problems. However, for f_{Ack} and f_{GPS}, the improvement in the efficacies was significant, with and without the use of migration (significant differences are indicated by the symbol '+' in Table 4.31).

In summary, the Dynamic FT 3D-cGA showed its ability to solve different problems for up to 40% faults, and significant improvements in the performance of the algorithm were achieved, especially with migration. The computational cost of increasing the number of fitness evaluations was not significant, whereas significant improvements in the efficacy were achieved for some of the problems. At this point, we can state that the best performance was obtained with $MaxGens_2$, and accordingly, we continued our analysis based on this scheme.

4.3.2.3 Dynamic FT 3D-cGA vs. FT 3D-cGA

This subsection compares Dynamic FT 3D-cGA based on $MaxGens_2$ with FT 3D-cGA. The subsequent paragraphs discuss the behaviour in terms of genetic diversity for both the algorithms.

To simplify the comparison and to reach an accurate conclusion, the two algorithms were compared according to statistically significant differences, and the rankings of the algorithms in terms of efficiency, efficacy, and speed. Table 4.32 illustrates the statistically significant differences between Dynamic FT 3D-cGA based on $MaxGens_2$ and FT 3D-cGA in terms of efficiency and efficacy. For example, solving f_{Ras} without migration by both algorithms shows no significant difference in terms of average number of generations (indicated by the symbol '•' in the middle Column of Table 4.32), while a significant difference is obtained when considering the search success rate (indicated by the symbol '+' in the middle Column of Table 4.32). Tables 4.33–4.35 show the rankings of the algorithms in terms of the average number of generations needed to find the solutions, search success rates, and average execution times, respectively. Each problem is independently ranked, and these local rankings are shown in Columns 2–8. The global ranking is shown in the last column, which is determined based on the summation of the local rankings (Column 9) to identify the best algorithm for all the problems in terms of each performance metric. For each fault ratio, the local ranking is determined by adding the positions of the algorithms according to the results obtained based on each performance metric, and the highest rank (lowest value) is assigned to the one with the minimum summation value. For example, when solving $f_{Ras,}$ the best efficiency is achieved by the Dynamic Fault-Tolerant 3D-cGA with migration, while the best speed is achieved by the FT 3D-cGA with migration; and the best efficacy is achieved by both algorithms, because they have similar ranks. In this work, the details of the local ranking are omitted and only the final ranks are shown.

Although the numbers of fitness evaluations were dissimilar for both the algorithms, we continued to compare them based on the average number of generations. The aim behind this consideration was to show how the increment in the maximum number of generations would influence the efficiency of the algorithm, and its effect when the migration technique is introduced. As can be seen from Table 4.33, in general, the best efficiency was achieved by FT 3D-cGA with migration, while Dynamic FT 3D-cGA with migration achieved the second best efficiency. However, the differences were not significant, except for f_{Ack} (see Table 4.32). The best efficacy was achieved by the Dynamic Fault-Tolerant 3D-cGA with

migration for most of the problems, except for f_{SLE}. The second best efficacy was achieved by the Fault-Tolerant 3D-cGA with migration (see Table 4.34), and the differences were not significant. Hence, we can confirm the effectiveness of migration in improving algorithm performance regardless of the difference in the number of fitness evaluations.

The integration between the Dynamic Fault-Tolerant 3D-cGA and the migration technique significantly improved the performance of the algorithm as it offered better exploration/exploitation trade-off. A further analysis of the behaviour of the algorithms is provided later in this section.

With regard to the execution time, the Fault-Tolerant 3D-cGA, especially with migration, significantly surpassed the Dynamic Fault-Tolerant 3D-cGA with/without migration (see Table 4.35). Although the obvious reason for the deterioration in the speed of the algorithm may have been thought to be the increase in the number of fitness evaluations, the distinct search success rates obtained were the main reason. We note that integration of the migration technique into the Dynamic Fault-Tolerant 3D-cGA failed to provide significant improvements in the speed, when compared with the Fault-Tolerant 3D-cGA.

Table 4.32. Comparison of Dynamic FT 3D-cGA versus FT 3D cGA in terms of convergence time (CT) and rate (CR)*

Problem	Without migration	With migration
f_{Ras}	•, +	•, •
f_{Ack}	+, +	+, +
f_{Mic}	•, +	•, •
f_{Lang}	•, +	•, •
f_{FMS}	•, +	•, •
f_{SLE}	•, •	•, •
f_{GPS}	•, +	•, •

Table 4.33. Ranking of the algorithms based on efficiency (CT)*

Problem / Algorithm	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	f_{GPS}	Sum	Rank
FT 3D-cGA	3	1	3	3	3	4	3	20	3
FT 3D-cGA+ mig.	2	2	1	2	1	1	1	10	1
DFT 3D-cGA	4	4	4	4	3	3	3	25	4
DFT 3D-cGA+ mig.	1	3	2	1	2	2	1	12	2

* For more details about the performance measures, please refer to Section 2.2.3.1.

Table 4.34. Ranking of the algorithms based on efficacy (CR)*

Problem / Algorithm	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	f_{GPS}	Sum	Rank
FT 3D-cGA	4	4	4	2	4	2	3	23	4
FT 3D-cGA+ mig.	1	2	1	2	3	3	2	14	2
DFT 3D-cGA	3	2	3	2	2	1	3	16	3
DFT 3D-cGA+ mig.	1	1	1	1	1	4	1	10	1

1 Efficacy is measured as the search rate of successful experiments (Convergence rate) out of 100 independent runs.

Table 4.35. Ranking of the algorithms based on speed (SP)*

Problem / Algorithm	f_{Ras}	f_{Ack}	f_{Mic}	f_{Lang}	f_{FMS}	f_{SLE}	f_{GPS}	Sum	Rank
FT 3D-cGA	3	1	3	2	2	1	3	15	2
FT 3D-cGA+ mig.	1	1	1	1	1	3	1	9	1
DFT 3D-cGA	4	3	4	4	4	1	4	24	4
DFT 3D-cGA+ mig.	2	4	2	3	3	4	1	19	3

To better demonstrate the behaviour of the algorithms and the effect on performance of increasing the fault ratio, we focused on one problem from the test suite. The problem used was f_{Lang}. When solving f_{Lang} for 10% faults, the results obtained for the Fault-Tolerant 3D-cGA were 399.6 average generations, 46% search success rate, and 5.42 seconds average execution times. For 20% faults, the results obtained were 540.79 average generations, 34% search success rate, and 6.0 seconds average execution times. These showed a significant deterioration in the performance of the algorithm due to the increase in the fault ratio. However, when solving f_{Lang} for 10% faults using the Dynamic FT 3D-cGA, the average number of generations was reduced to 380.19, the search success rate increased to 51%, and the average execution times increased to 5.63 seconds. Consequently, we can confirm that the increment in the number of generations can alleviate the search difficulty, despite the increase in the time needed for the algorithm to converge (from 5.42s to 5.63s). The increase in execution time normally results from the increase in the number of generations; however in this case it may also refer to the variations in hit rate obtained (46% vs. 51%). For 20% faults, the average number of generations increased to 625.93, despite the improvement in the search success rate obtained (46%). The decline in the efficiency was resolved by integrating the migration technique into the Dynamic Fault-Tolerant 3D-cGA; e.g., the average number of generations decreased by 48% to reach 326.2 generations.

To summarise, in general, the Dynamic FT 3D-cGA improved algorithm reliability, especially when it was combined with the migration technique, despite an increase in the computation cost. Increasing the maximum number of generations was a critical factor in increasing the probability of finding the desired solutions. However, offering more

* For more details about the performance measures, please refer to Section 2.2.3.1.

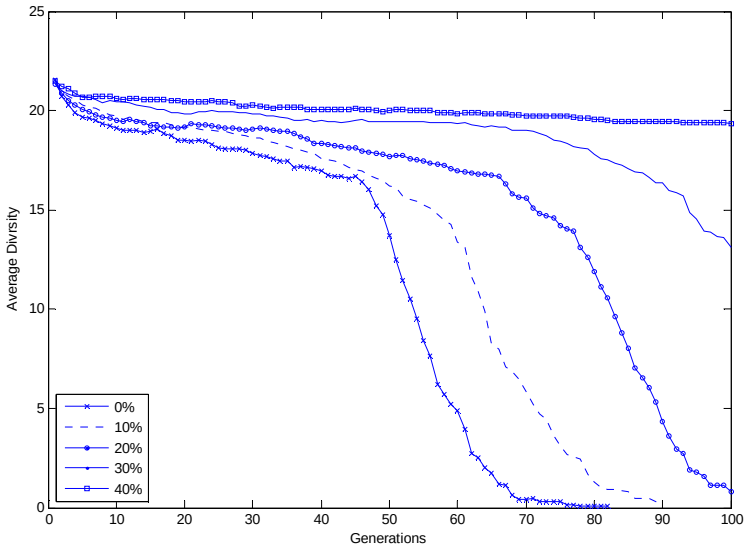
generations alone was not enough due to the following reasons. First, more faults leading to less individuals alive deteriorated the genetic diversity. Second, worse faults distribution occurred for high fault ratios, and the worst case happened when a fault-free individual was surrounded by faulty neighbours. Therefore, local enhancing was required, and the best and simplest way that we came up with is to control the global selection pressure by employing the migration technique.

f_{GPS} was used as an example to show and understand the behaviour of the different approaches and the influence of the migration technique by computing and plotting the population's diversity (genotypic entropy) as a function of generations. Figure 4.12(a) and (b) show the average genotypic diversities obtained by the FT 3D-cGA without and with migration, respectively; while the average genotypic diversities obtained by the Dynamic FT 3D-cGA without and with migration are shown in Figure 4.13(a) and (b), respectively. As can be seen from Figure 4.12(a), the population diversity increased significantly as the fault ratio increased, and this behaviour shows how the search difficulty dramatically increased leading to an increase in the number of generations, reduction in search success rate, and increase in execution time. It can be noted that for 40% faults, the population diversity trend was almost steady over all the allowed number of generations (see Fig. 4.12(a)) due to the difficulty in the convergence ability for high fault ratio. This tendency led to the sharp drop in the search success rate. For instance, for 40% faults, the search success rate obtained when solving not only $f_{GPS,}$ but also all the other problems, was 0%.

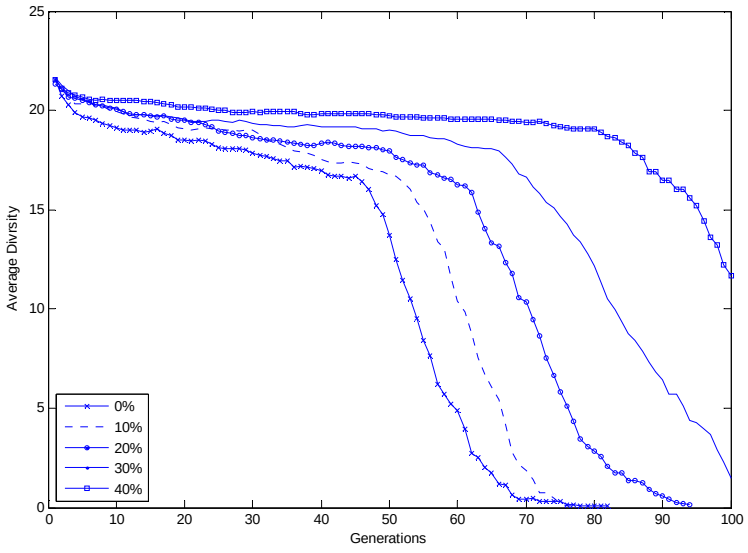
Figure 4.12(b) shows the effect of introducing the migration on the population's diversity and thus the performance of the algorithm. The migration technique significantly enhanced the ability of the algorithm to converge, leading to a considerable reduction in the number of generations. Nevertheless, a main observation is the ability of the algorithm to converge for 40% faults; e.g., for $f_{GPS,}$ the search success rate obtained increased significantly to 94%. However, the efficiency deteriorated as the algorithm started to converge at a late stage. For example, the algorithm convergence began at generation 80 (see the diversity trend in Fig. 4.12(b)).

As mentioned earlier, the main reason for increasing the number of generations is to increase the reliability of the algorithm. Figure 4.12(b) provides a clearer illustration in this regard, specifically the diversity trend for 40% faults. It can be seen that the diversity level at the last generation is still too high, despite the ability of the algorithm to converge. Hence, the increase in the maximum number of generations, in-line with fault ratio, is intended to deal with this issue.

The diversity obtained for the Dynamic FT 3D-cGA without migration is shown in Figure 4.13(a). As can be seen, the diversity trends could approach almost zero for all fault ratios

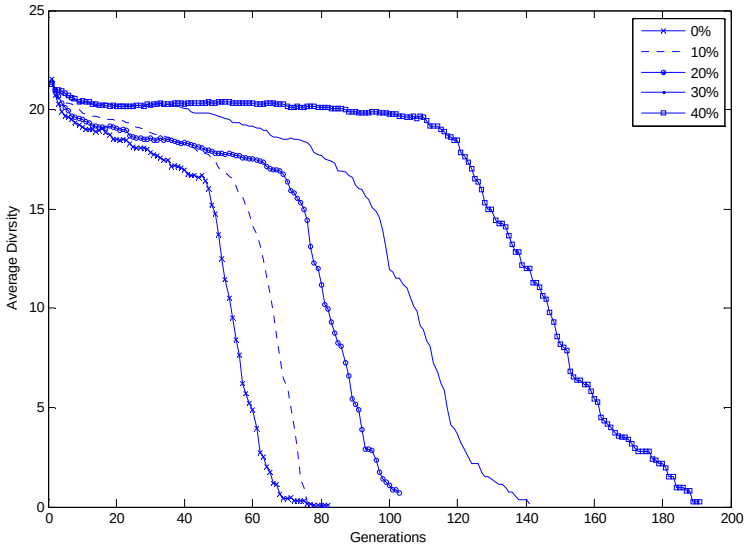


(a) Without migration.

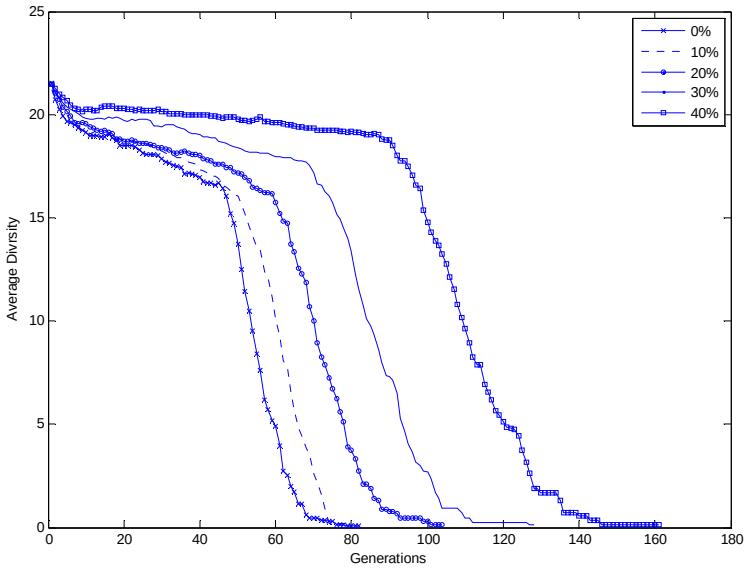


(b) With migration.

Figure 4.12. The average genotypic diversities obtained by FT 3D-cGA when solving f_{GPS} for each fault ratio.



(a) Without migration



(b) With migration

Figure 4.13. The average genotypic diversities obtained by Dynamic FT 3D-cGA when solving f_{GPS} for each fault ratio.

considered; however, the efficiency of the algorithm deteriorated slightly due to the increased number of generations offered. As expected, although this approach significantly outperformed the FT 3D-cGA without migration, it was still worse than the FT 3D-cGA with migration (e.g., the results obtained by the dynamic algorithm were 162.19 generations and 56% search success rate, while those obtained by the latter were 48.78 average generations and 94% search success rate). The increase only in the number of generations promoted

more exploration, which in turn affected the quality of the solutions. Consequently, it could not greatly add benefits due to the search difficulty induced by the faults. This observation confirmed the need and the importance of employing a mitigation technique, especially in the presence of faults.

The influence of combining the migration technique with the Dynamic FT 3D-cGA on the population's diversity is demonstrated in Figure 4.13(b). Commonly, the migration enhances diversity by promoting more exploration; however, in this study, the migration promotes the exploitation because it aims at enhancing the local selection intensity through substituting fault-free individuals for the isolated ones. In other words, the size of the neighbourhood is preserved leading to maintain the local selection intensity. Consequently, this combination shows a balance between the exploration offered by increasing the number of generations and the exploitation offered through migration. The effect of this balance can be seen by comparing Figure 4.13(a) with 4.13(b) as the number of generations needed by the algorithm to converge was significantly reduced while alleviating the premature convergence. For instance, for 40% faults, the search success rate obtained was 99% within an average of 52.04 generations and 0.126 seconds, while a search success rate of 56% within an average of 162.19 generations and 0.2 seconds was obtained without migration.

4.3.3 Conclusion

This study proposed a new algorithm, the Dynamic FT 3D-cGA, for handling failures that occurred at individuals' phenotypes due to SEUs in particular. The algorithm is based on the canonical model of cGAs and is a modified version of the past approach (FT 3D-cGA) that uses genetic diversity to identify and isolate faulty individuals. The most critical fault model was tackled in conjunction with different fault ratios.

Our main motivation for this study was to improve the reliability and performance of the FT 3D-cGA through dynamic control of the exploration/exploitation trade-off. The dynamic calculation of *MaxGens* based on fault ratio encountered helped to enhance the exploration. On the other hand, the exploitation was enhanced through the use of the proposed migration technique.

To illustrate the improvements achieved, the Dynamic FT 3D-cGA was compared with the FT 3D-cGA in terms of efficiency, efficacy, and speed. Both the algorithms demonstrated successful recovery of up to 40% faults, especially when the migration technique was employed. Thus, we can confirm that the use of migration as a mitigation technique to fault tolerance offers considerable improvements in the efficiency, efficacy,

speed, and reliability of the algorithms, especially for the high ratio of faults.

Besides being a mitigation technique, the integration of migration into both algorithms plays an important role in controlling exploration/exploitation trade-off. Exploration and exploitation are the two main issues that determine the performance of EAs. The population diversity is improved by exploring the search space, while the optimum solution could be found by exploiting the fitness information. In this work, the best overall performance in terms of efficiency, efficacy, and speed was achieved with the use of the migration technique owing to its effect in enhancing the local selection intensity and diversity in proportion.

In conclusion, we note that the FT 3D-cGA and Dynamic FT 3D-cGA with migration showed the best performance, and the differences between the results obtained by both algorithms were not significant. An exception was for f_{Ack}, in which case the Dynamic FT 3D-cGA with migration significantly outperformed the FT 3D-cGA with migration mainly in terms of efficacy and reliability. The best efficiency (or the minimum number of generations) was achieved by the FT 3D-cGA with migration; however, the lower number of generations was found to be due to the significant difference in the obtained search success rate. For example, solving f_{Ack} by FT 3D-cGA with migration resulted in average number of generations and search success rate as follows: 1269 (66%), 1426.6 (45%), 1576.7 (33%), 1642.8 (12%) for 10%, 20%, 30%, and 40% faults, respectively (refer to Tables 4.22 and 4.23). In contrast, solving the same problem by the Dynamic FT 3D-cGA with migration resulted in 1460.2 (78%), 1914.4 (79%), 2348.2 (82%), and 3028.6 (84%) for 10%, 20%, 30%, and 40% faults, respectively (refer to Tables 4.28 and 4.29). From the previous example it can be noticed that for all fault rates the number of generations obtained by FT 3D-cGA with migration were significantly lower than those obtained by the Dynamic FT 3D-cGA with migration. Conversely, when observing the hit rates obtained by the dynamic algorithm, they found to be higher than those obtained by the static version, which explains the difference in the obtained number of generations.

4.4 Summary and Contribution to Knowledge

This chapter aimed to propose a highly reliable cGA that is tolerant to failures, for SEUs in particular. This research targeted fitness score registers due to the importance of the fitness information in guiding the search process. Two critical fault models were considered—stuck at ‘0’ and stuck at ‘1’ faults. The main objective was to propose an algorithm-based fault tolerant algorithm using the inherent features of cGAs in order to deal with SEUs. Another

objective was to improve the performance of the algorithm in order to effectively deal with high fault ratios. The following points summarise what this study has contributed to knowledge.

- The proposed FT 3D-cGA showed its ability to automatically identify and isolate faulty cells based on genetic information such as genetic diversity. In addition, the algorithm was successful in recovering up to 40% faults taking into consideration the two most critical fault models (i.e., stuck at '0' and stuck at '1' faults). However, the performance of the algorithm varied according to the fault model and the problem to be solved.

- Different selection intensities were defined and assessed in order to improve the performance of the algorithm. The different intensities came about by controlling the selection rate *r* of the local selection, which is ST. The different selection pressures showed different exploration/exploitation trade-offs, which in turn showed different rates of improvements.

- An explicit migration technique was proposed and shown to not only mitigate the impact of faults but also to improve the performance of the algorithm. The technique's main aim was to replace faulty individuals by fault-free ones, thereby reducing the impact of faults. In addition, through migration, the genetic diversity was enhanced, leading to improved algorithm performance.

- Several algorithm configurations concerning migration and selection intensity were assessed. The best efficiency was achieved by the third configuration (ST, $r = 0.0$ + *Migration*) for stuck at '0' faults, while for stuck at '1' faults the first configuration (ST, $r = 0.0$ + *noMigration*) achieved the best efficiency. The best efficacy was obtained by the fourth configuration (ST, $r = 0.5$ + *Migration*) for both fault models, mainly due to the selection intensity provided with ST, $r = 0.5$. A rate of 0.5 was selected as a way to enhance the genetic diversity and therefore promote more exploration leading to improve the efficacy; however, the efficiency of the algorithm was deteriorated.

- Considering the most critical fault model (i.e., stuck at '1' for f_{GPS} and stuck at '0' for the other problems), the combination of ST, r = 0.0 and migration showed significant improvement mainly in the efficiency of the algorithm reaching to 35.9%. The introduction of migration covered the loss of cells and therefore enhanced the genetic diversity, while at the same time ST, r = 0.0 offered high selection pressure leading to a reduction in the number of generations required to solve a problem. Thus, this combination offered a better exploration/exploitation trade-off.

- Different migration schemes were proposed and measured to further improve the performance of the algorithm, in particular for high fault ratios. The proposed migration schemes were similar in their frequency and rates. However, the difference resided in the source and/or the fitness of the migrants. Migration scheme 2, which used the fittest migrants within the current neighbourhood to replace the faulty individuals, showed its ability to enhance the local selection intensity and diversity in the population. Therefore, it achieved the best overall algorithm performance in terms of efficiency, efficacy, and speed for both fault models.

- A dynamic fault tolerant approach (Dynamic FT 3D-cGA) was proposed and it showed further improvements in the performance and the reliability of the algorithm. Two dynamic adaptation schemes were introduced, the first scheme ($MaxGens_1$) aimed to balance the number of fitness evaluations due to the reduction in the number of individuals alive. Therefore, this scheme used the number of faulty individuals to recalculate the number of evaluations needed to solve a problem effectively. The second scheme ($MaxGens_2$) is similar to the first; however, this scheme considered the impact of faults. Therefore, a further increase in the number of evaluations was offered to tackle the added difficulty caused by faults.

- Several algorithm configurations concerning migration and dynamic adaptation were defined and assessed. The dynamic calculation of the number of fitness evaluations enhanced the exploration, while exploitation was enhanced by introducing migration. The introduction of migration resulted in significant improvements, up to 66.7% in efficiency with $MaxGens_1$ and 62% with $MaxGens_2$, 100% in efficacy with $MaxGens_1$ and 99% with $MaxGens_2$, and 32.4% in speed with $MaxGens_1$ and 33.6% with $MaxGens_2$.

- The proposed FT 3D-cGA was compared to the proposed Dynamic FT 3D-cGA with and without migration. With migration, both approaches showed the best overall performance with non-significant differences in the results obtained when solving most of the problems. An exception was for f_{Ack}, as Dynamic FT 3D-cGA with migration significantly outperformed FT 3D-cGA with migration in terms of efficacy and reliability, while the latter achieved the best efficiency. However, this less number of generations was found to be due to a significant difference in the search success rate obtained.

Chapter 5

Dynamic-Adaptive cGAs

Genetic search occupies an important position in evolutionary computation. The most important issues in the evolution process of genetic search are exploration and exploitation (Oei, Goldberg, and Chang, 1991). The aim of this chapter is to investigate the inherent ability of cGAs in controlling the exploration/exploitation trade-off. Exploring the search space enhances population diversity and helps with escaping local optima; which is provided by the existence of overlapped neighbourhoods. At the same time, exploitation reduces diversity by focusing on the fitter individuals inside each neighbourhood, which in turn improves the quality of the solution. Improper balance between exploration and exploitation leads to ineffective EA. Hence, proposing a new approach that dynamically balances between exploration and exploitation is another aim of this chapter. The concepts of exploration and exploitation are strongly related as an increase in one results in a proportional decrease in the other. For example, increasing exploration (or genetic diversity) decreases exploitation, and vice versa.

In addition, the balance between exploration and exploitation is the key to determining an algorithm's behaviour and performance (Herrera and Lozano, 2000). Several studies have been carried out to investigate and dynamically control this trade-off. One way of doing this is to tune the relationship between the shape and/or size of the neighbourhood and the grid (*NGR*) (Alba and Troya, 2000; Giacobini *et al.*, 2005). Another way is through the use of probabilistic selection mechanisms such as anisotropic, stochastic, and centric selections (Simoncini *et al.*, 2006; Simoncini *et al.*, 2009). All the techniques cited are aimed at controlling the global selection pressure as high selection pressure supports exploitation while low selection pressure favours exploration (Sarma and De Jong, 1996).

The selection pressure has a huge impact on the exploration/exploitation trade-off and therefore algorithm performance. With high selection pressure, only the fittest individuals survive and conquer the entire population, leading to reduction in convergence time. However the quick convergence may lead to the algorithm becoming stuck in local optima. On the other hand, low selection pressure weakens the influence of the fittest individuals on the population, leading to algorithm divergence. A study showing the influence of the selection pressure on the performance of cGAs was presented by Simoncini *et al.* (2007).

Ursem (2002) presented a diversity-guided approach (DGEA) to dynamically alternate between exploration (mutation) and exploitation (recombination and selection). The diversity measure used in this work is the distance-to-average-point. The DGEA was compared to different evolutionary search models and showed outstanding improvement not only in accuracy but also in algorithm efficiency.

Alba and Dorronsoro (2005), in their research proposed an adaptive cGA that controls the exploration/exploitation trade-off through the interchange between three grid topologies: square, rectangular, and narrow. These topologies were selected to present different ratios (*NGR*) and thus different selection pressures. The convergence speed was used as feedback to alternate between the exploration and exploitation phases. A shift to 'explore' mode occurred if the convergence speed was too high. Conversely, a shift to 'exploit' mode occurred if the convergence speed was too slow. The proposed algorithm outperformed the other studied algorithms such as static and pre-programmed cGAs. In addition, it has been concluded that narrow grids are well suited for multimodal and complex problems, while wider grids are more appropriate for simple problems.

In a later study, Maeda and Li (2007) proposed a fuzzy adaptive approach that uses a diversity measure to tune the genetic parameters of the island search model (dGA). Simulation results showed the efficiency of the proposed algorithm.

From all the previous studies, it is believed to be more efficient to induce different levels of exploration/exploitation trade-off in different timings of the search process. Therefore, studies concerning the dynamic control of exploration/exploitation trade-off are increasingly being conducted.

In this chapter, the main motivation is to design an effective algorithm that can dynamically adapt to changes in the convergence speed through appropriate balancing between exploring the search space and exploiting the good solutions. Two new adaptive 3D-cGAs that dynamically control exploitation/exploration trade-off are proposed (Al-Naqi *et al.*, 2010b; 2012). The first approach uses a probabilistic selection mechanism and

gradually tunes the selection probability based on a population diversity measure (which will be discussed in Section 5.2). In the second approach, the same metric used in the study of Alba and Dorronsoro (2005), (population entropy) is used to guide the search process (this will be discussed in Section 5.3). In addition, in order to validate and provide a thorough study of the performance of the proposed algorithms, a comparison between the proposed algorithms and other static and dynamic algorithms is provided in Section 5.4.

5.1 Study of Selection Pressure

Selection pressure is a critical factor that differentiates between the different EA search models. Different parameters such as topology and/or size of the grid, shape and/or size of the local neighbourhood, and the parameters of the genetic operators have an impact on the global selection pressure. The global selection pressure determines the ability of the good solutions to survive in the population. Therefore, the appropriate selection pressure should be applied on the population in order to offer the best balance between exploring the search space and exploiting good solutions. As mentioned earlier, one way to control the selection pressure is through the use of appropriate selection parameters. Simoncini *et al.* (2007, 2009) proposed new selection techniques—anisotropic and centric selections—to appropriately control the selection pressure. Moreover, in the former study, a stochastic binary tournament (ST) selection was tested to show its ability to appropriately control the selection pressure. In both studies, it was proven that the global selection pressure could be monitored by using adequate selection parameters. In this work, the ST operator is selected as a local selection method. Similar to the binary tournament (BT) selection, two individuals are randomly selected and the best individual is assigned a probability of $(1- r)$, while the worst one is assigned a probability of r; where $r \in [0 , 1]$. ST is equivalent to BT when $r = 0$ as the best solution is always favoured.

The selective pressure defines the convergence speed as well as population diversity, and can be measured using growth curves and takeover time models. Takeover time is defined as the time needed for the best solution to conquer the entire population. In other words, the takeover time is reached when the growth number of the best individual is equal to the population size (Simoncini *et al.*, 2006). This technique is used to study the induced selection pressure; therefore, to be used effectively the selection should be the only active genetic operator (Goldberg and Deb, 1991). Figures 5.1 and 5.2 show an average of 100

independent runs for the growth number of the best individual and takeover time, respectively, for a cubic grid with a population size of 216 individuals arranged as 6×6×6.

In Figure 5.1, it can be observed that the increase in the selection rate *r* leads to slow growth in the best individual. In other words, the opportunity for worse solutions to be maintained in the population increases, offering more diversity and promoting more exploration. As a result, it leads to weaker selection pressure and a longer takeover time (see Figure 5.2). On the other hand, the global selection pressure is strengthened through the decrease in the *r* value, promoting more exploitation.

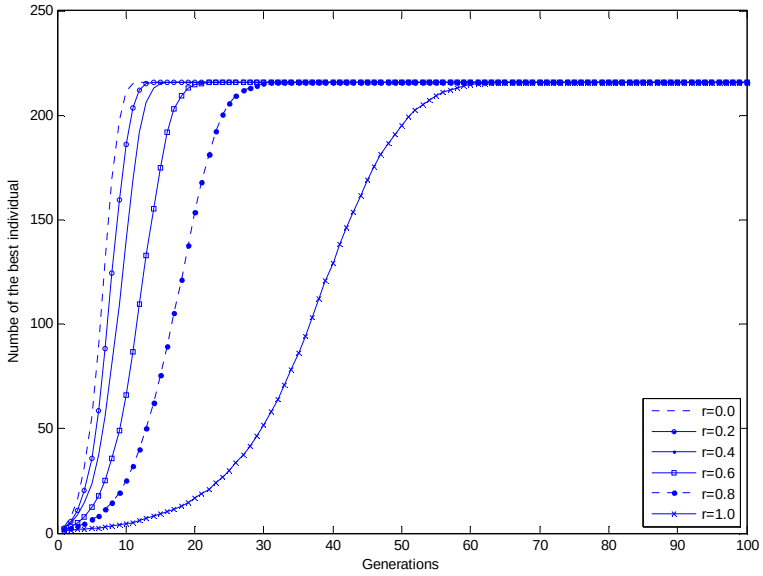


Figure 5.1. The growth number of the best individual for different selection rates *r*.

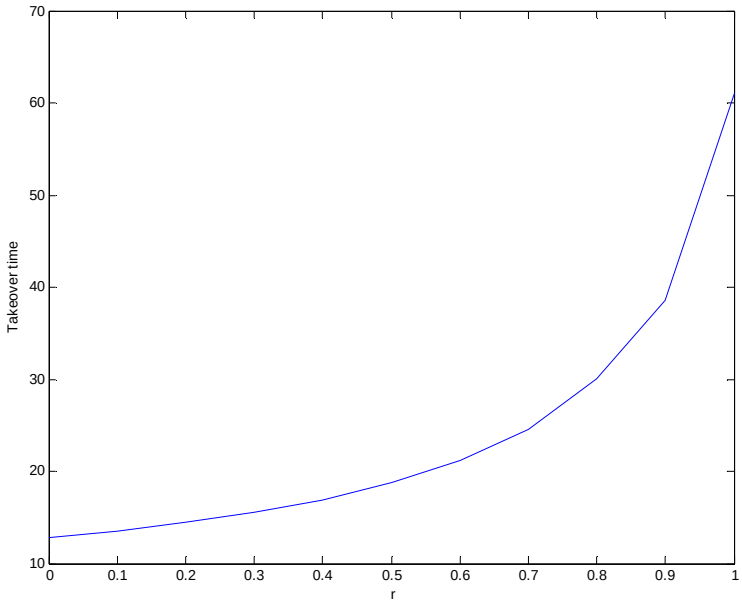


Figure 5.2. The takeover time for different selection rates *r*.

5.2 Diversity-Guided 3D-cGA

This section presents a new adaptive gradual algorithm that is based on the 3D-cGA. The main motivation for the proposed approach is to appropriately control the balance between exploring the search space and exploiting the best solution. Based on diversity measure, the proposed algorithm gradually tunes the selection pressure by modifying the genetic parameters, specifically the selection rate *r*. The exploration/exploitation trade-off is a direct effect of the selection pressure; in which there is no one appropriate pressure for all problems. This approach will be compared to three algorithms with static genetic parameters in turn to show and confirm the cases in which the adaptive approach surpasses the other static ones. In order to reach valid conclusions, the algorithm is assessed using a benchmark of six test functions and two real world problems that present variable complexities. They are: Rastrigin (f_{Ras}), Schwefel (f_{Sch}), Rosenbrock (f_{Ros}), Ackley (f_{Ack}), Michalewicz (f_{Mic}), Langermann (f_{Lang}), FMS (f_{FMS}), and SLE (f_{SLE}) problems (refer to Appendix A for details of the benchmark problems). The algorithm description is presented in Section 5.2.1. Section 5.2.2 discusses and analyses the simulation results obtained by the Diversity-Guided 3D-cGA and the three defined static algorithms. Concluding remarks are given in Section 5.2.3.

5.2.1 Algorithm Configuration

This section introduces the Diversity-Guided 3D-cGA. As mentioned previously, the approach proposed here tunes the selection rate *r* based on population diversity. In this study, the employed grid topology is fixed in order to reduce computation overhead. This is unlike the approach proposed in (Alba and Dorronsoro, 2005), in which the grid topology has to be changed, leading to misshapen neighbourhood relations and therefore requiring the computation of positions of new neighbours.

Before introducing the adaptive model, an explanation of some facts that lead to the selection of the adaptive criterion is presented as follows. First, as the search process progresses the population diversity decays to reach almost zero, in particular when a good solution conquers the entire population. However, the diversity of the population could be lost too quickly, leading the algorithm to get trapped into local optima. Second, although the ‘explore’ mode allows the algorithm to escape local optima, improvements in the solutions only occur during the ‘exploit’ mode (Ursem, 2002). As a result, the proposed adaptive criterion aims to reduce the convergence speed by gradually reducing the selection pressure

as the search process progresses. The selection pressure is reduced by decreasing the selection rate *r*, particularly when the computed average population diversities during the last ten generations fall below a specified threshold. In order to calculate the population diversity, the well-known 'distance-to-average-point' measure is employed (Ursem, 2002).

Algorithm 5.1 illustrates the adaptive model. The adaptive criterion is $\overline{d} < \gamma$, where $\overline{d}$ is the average population diversities of the last ten generations, and γ is the threshold.

Algorithm 5.1 Adaptive model of Diversity-Guided 3D-cGA

```
1. if d̄ < γ then
2.    if r ≠ 0 then
3.       Offer more exploitation[1]
4.    end if;
5. else
6.    Do not change
7. end if;
```

1 The exploitation is offered by decreasing the section rate *r* which in turn reduces the global selection pressure. Note that the algorithm starts with *r* = 1.0.

The idea behind the introduced adaptive criterion is to offer adequate time to explore the search space, which contributes to the discovery of promising areas and avoids local optima. This is followed by gradual exploitation, which contributes to enhancing the solutions. This gradual alteration reduces the possibilities of premature convergence. In the final stage, the algorithm proceeds with the highest degree of exploitation (r = 0.0). Strong exploitation makes the genetic search more effective, especially when solutions are near optimum. To achieve the above objectives, the algorithm starts with the highest possible selection rate, which is r = 1.0 ('explore' mode). This rate is then lowered when the adaptive criterion is fulfilled. The selection rate is then decreased regularly in order to gradually introduce exploitation until the lowest bound is reached, which is r = 0.0 ('exploit' mode).

5.2.2 Experimental Results and Analysis

In this section the results obtained by Diversity-Guided 3D-cGA and three configurations of 3D-cGA, each with different static selection rate, are presented and analysed. The same parameters were used for all the considered problems (see Table 5.1). The population size used here was 343 individuals arranged into a 7×7×7 lattice. The defined neighbourhood contained seven individuals (east, west, vertical north and south, and horizontal north and south, plus the one under consideration). The first parent was the current individual while the second one was selected by ST with rate r. An arithmetic crossover operator with P_c = 0.9

was applied to generate an offspring. The offspring was then mutated by a non-uniform mutation operator, the best for real optimisation (Back, 1996), with a probability of $P_m = 1/L$, where L is the dimension of a problem (i.e., the length of the chromosome). Although the dimension of f_{FMS} is six, the same mutation probability was used as with all the other problems. A *replace-if-better* was used as a replacement policy, during which the current individual was replaced if it competed with a better offspring. Finally, the algorithm ended when the difference between the average fitness values (*avgf*) and the optimum fitness value (*optf*) satisfied a specified threshold ($\pm\varepsilon$).

Table 5.1. Parameterization used in the algorithms

Population size:	343 individuals
Parent selection:	Current + ST, r
Recombination:	AX, $P_c = 0.9$
Mutation:	Non-uniform, $P_m = 1/L$ (L = individual length)
Replacement:	Replace-if-better
Neighbourhood:	NEWS
Lattice:	7×7×7
Stop criterion:	$\|avgf - optf\| \leq \varepsilon$

Since the complexity of the considered problems varied, different values of ε were used: 0.05 for f_{Ros}, 0.005 for f_{Lang}, 0.01 for f_{FMS}, and 0.3 for f_{SLE}; while a more precise ε value (0.001) was applied for the remaining problems. Similarly, different numbers of maximum generations were used: 1000 generations for f_{Ras}, f_{Mic}, f_{Lang}, and f_{SLE}; 1500 generations for f_{Sch}; and 2000 generations for f_{Ack}, f_{Ros}, and f_{FMS}.

The performance of the algorithm was measured using two metrics, the search success rate, or the efficacy, and the average number of generations, or the efficiency, of 100 independent experiments. Furthermore, in order to determine the significance level of the differences in efficiencies obtained by the Diversity-Guided 3D-cGA and the static algorithms, statistically significant tests, with 95% confidence level were applied (details about the statistical tests were provided in Section 2.2.3.1).

The results are presented in Table 5.2, where the average number of generations and the percentage of successful runs are shown for every problem and the best values are in **bold**. The symbol '+' in the Table indicates that the efficiency obtained by Diversity-Guided 3D-cGA was significantly better than the one obtained by the corresponding algorithm, while worse efficiency is indicated by the symbol '–'. The symbol '•' denotes non-significant

differences between the efficiencies obtained by the compared algorithms. Furthermore, median absolute deviations (*mad*) are included after the symbol '±'.

The value of γ was selected based on preliminary experiments in which different γ values (0.3, 0.35, 0.4, and 0.45) were tested. A value of $\gamma = 0.4$ was selected as the best one in terms of efficiency and efficacy for most of the studied problems (in order to avoid reader distraction the details are provided in Appendix B.2, Tables B.3 and B.4). Certainly, there is no one best γ value for all problems, as pointed out by Alba and Dorronsoro (2005), who also indicated that there is no global best algorithm for all problems.

In order to evaluate the Diversity-Guided 3D-cGA, it was compared to three 3D-cGAs with static *r*. The first 3D-cGA used the lowest *r* bound ($r = 0.0$); while the third one used the highest *r* bound ($r = 1.0$). The second static 3D-cGA used the mean value of $r = 0.5$.

As can be seen from Table 5.2, the best efficiencies to solve f_{Ras}, f_{Sch}, f_{Mic}, and f_{Ack} were obtained by the 3D-cGA with $r = 0.0$ (with significant differences only for f_{Sch} and f_{Mic}—see test results in Table 5.2). However, concerning similar efficacies the Diversity-Guided 3D-cGA is more robust as it obtains smaller median absolute deviations; the exceptions are due to the significant differences in the search success rates. In addition, using higher selection rates to alleviate the exploitative behaviour of the 3D-cGA deteriorates the efficiency as well as reduces the search success rates. The worst performance was achieved by the 3D-cGA with $r = 1.0$ as it showed more explorative behaviour, which lacked the power to improve the quality of the solutions.

Table 5.2. Convergence time (CT) and rate (CR)* obtained by the Diversity-Guided 3D-cGA and 3D-cGAs with static *r* values

Problem	3D-cGA			Diversity-Guided 3D-cGA
	$r = 0.0$	$r = 0.5$	$r = 1.0$	
f_{Ras}	**611.14** ± 62.50 (•) **100%**	717.61 ± 37.50 (+) **100%**	866.31 ± 83.00 (+) 94%	641.72 ± 55.0 **100%**
f_{Sch}	**1003.5** ± 181.5 (–) **100%**	1089.8 ± 176.0 (–) **100%**	1498.7 ± 1.000 (•) 3%	1209.9 ± 116.0 **100%**
f_{Ack}	**1848.1** ± 72.00 (•) **83%**	1856.2 ± 62.00 (•) 54%	- 0%	1897.4 ± 52.0 **83%**
f_{Ros}	1518.3 ± 124.5 (+) 10%	1763.1 ± 99.00 (+) 14%	1303.7 ± 309.0 (+) 15%	**881.7** ± 636.5 **50%**
f_{Mic}	**512.45** ± 46.50 (–) **100%**	671.28 ± 70.00 (+) **100%**	966.33 ± 3.000 (+) 3%	628.30 ± 43.0 **100%**
f_{Lang}	**231.67** ± 20.50 (–) 70%	331.12 ± 39.00 (•) 83%	707.91 ± 67.00 (+) 95%	308.45 ± 14.5 **96%**
f_{FMS}	1317.7 ± 259.5 (•) 74%	1360.5 ± 316.0 (•) 89%	**1220.9** ± 261.0 (•) 47%	1294.7 ± 381.5 **100%**
f_{SLE}	**330.86** ± 51.00 (•) 15%	471.63 ± 68.00 (+) 33%	917.33 ± 35.50 (+) 12%	341.48 ± 35.0 **39%**

* For more details about the performance measures, please refer to Section 2.2.3.1.

Regarding f_{Ros} and f_{FMS}, the Diversity-Guided 3D-cGA outperformed the other static algorithms based on the two metrics, with a significant difference in the efficiency for f_{Ros} and a non-significant difference for f_{FMS}. Concerning the static algorithms, although higher values of r (i.e., r = 0.5 and r = 1.0) should result in a higher average number of generations, f_{Ros} and f_{FMS} showed exceptions. This was due to the exploration/exploitation trade-off offered by the algorithm, as well as the problems geometry. More exploration improves both the efficiency and efficacy of the algorithm when solving f_{Ros}, while only the algorithm efficiency is improved when solving f_{FMS} (remember that the best exploration/exploitation trade-off is problem dependant).

Finally, concerning f_{Lang} and f_{SLE}, the Diversity-Guided 3D-cGA significantly outperformed the 3D-cGA with r = 0.5 and r = 1.0 in terms of both metrics, while it considerably outperformed the static algorithms with r = 0.0 in terms of efficacy. This latter improvement reached 24% when solving f_{SLE} and 26% when solving f_{Lang}. Furthermore, the Diversity-Guided 3D-cGA showed more robust behaviour although the difference in efficiencies comparing to 3D-cGA with r = 0.0 is non-significant (see test results in Table 5.2).

Figure 5.3 shows the average genotypic diversity trends of 100 runs, which were obtained by the Diversity-Guided 3D-cGA for the considered problems. From the figure, it is clearly observed that the speed of the population diversity loss is differed between the considered problems. These differences confirm that each problem introduces a different level of difficulty to the search, which therefore requires different exploration/exploitation tradeoffs. In addition, it can be seen from Figure 5.3 that problems with higher complexity level such as f_{Lang} and f_{FMS}, show two distinctive trends as the diversity level started with an increase rather than a reduction. The difference between these trends is the speed of the diversity loss, which started at later stages; for f_{Lang} the diversity level steeply decreases, while this reduction is gentler for f_{FMS}.

In general, 3D-cGA with r = 0.0 achieved the best efficiency (in 6 out of 8 problems), while the Diversity-Guided 3D-cGA showed more robust behaviour. In addition, the statistically significant results assert that the Diversity-Guided 3D-cGA is favoured for most the problems. With respect to the efficacy of the algorithm, the Diversity-Guided 3D-cGA obtained the best search success rates for all the studied problems. Hence, it can be concluded that the Diversity-Guided 3D-cGA provides the best efficacy with adequate computation cost (i.e., number of generations).

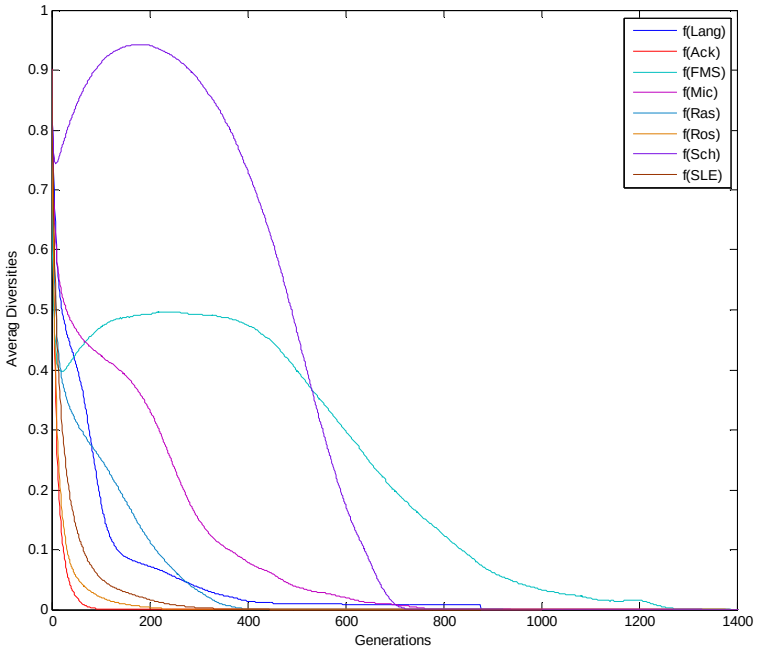


Figure 5.3. Average genotypic diversities based on 'Distance-to-average-point' measure for the Diversity-Guided 3D-cGA.

Furthermore, the Diversity-Guided 3D-cGA confirms the common belief referred to in (Alba and Dorronsoro, 2005), which states that "a good optimisation algorithm must initially seek promising regions, and then gradually search in neighbourhood of the best so far points". That is because the Diversity-Guided 3D-cGA starts with an 'explore' mode (i.e., low selection pressure) and then shifts to 'exploit' mode in an adaptive and gradual manner.

5.2.3 Conclusion

This section presented a new dynamic 3D-cGA, Diversity-Guided 3D-cGA, which uses diversity measure to control the exploration/exploitation trade-off. The main idea behind the adaptivity is to control and provide an appropriate balance between the exploration and the exploitation for an algorithm. This goal is achieved by tuning a genetic parameter, which is the rate of the local selection mechanism.

The Diversity-Guided 3D-cGA showed superior performance, mainly in terms of the efficacy of the algorithm. In addition, the dynamic algorithm outperformed the static approaches for most of the problems studied. The exceptions either did not have statistically significant differences or showed more erratic behaviour (see *mad* values). Hence, in

general, the proposed adaptive model could achieve a suitable balance between enhancing population diversity (to escape local optima—efficacy) and tuning good solutions (to improve solution quality—accuracy).

5.3 Convergence-Speed-Guided 3D-cGA

This section proposes another adaptive algorithm that aims to control the exploration/exploitation trade-off dynamically. The algorithm is designed based on 3D-cGAs because of their high performance features. In this section, the methodology is based on the change in the global selection pressure induced by dynamic tuning of the local selection rate. The parameter tuning of the local selection method is a way to define the global selection pressure. A diversity speed measure is used to guide the algorithm. This measure is adapted from (Alba and Dorronsoro, 2005). A benchmark of well-known test functions and real world problems was selected to investigate the effectiveness of the algorithm proposed. They are: Rastrigin (f_{Ras}), Rosenbrock (f_{Ros}), Ackley (f_{Ack}), FMS (f_{FMS}), SLE (f_{SLE}), and GPS (f_{GPS}) problems (details about f_{GPS} are provided in Section 4.2.2, while details about the other problems are provided in Appendix A). In addition, in this Section a comparison between the proposed algorithm and other static and dynamic algorithms are provided in order to study the different effects on the performance of the algorithms.

Section 5.3.1 describes the configuration of the proposed algorithm as well as other static and dynamic approaches, which are used in the comparison. The experimental parameters and results are provided in Section 5.3.2, while Section 5.3.3 gives the conclusion.

5.3.1 Algorithm Configuration

In this section, three different static 3D-cGAs are first discussed; then they are evaluated against the static algorithms proposed in (Alba and Dorronsoro, 2005). Following that, the Convergence-Speed-Guided 3D-cGA is introduced and the similarities and differences between the proposed algorithm and the dynamic-adaptive algorithm proposed in (Alba and Dorronsoro, 2005) are outlined.

In previous discussion, the influence of using different selection rates on the behaviour of the algorithm was observed (refer to Section 5.1). Furthermore, in order to investigate these effects on the performance of the algorithm, first two groups of static algorithms consisting of three distinct 3D-cGAs are described. The algorithms in the first group use different static

selection rates while the algorithms in the second group use different static *NGR*s, in particular, different grid shapes (Alba and Dorronsoro, 2005).

The local selection method used in the first algorithmic group is ST with $r = 0.0$, $r = 0.7$, and $r = 1.0$; while the same grid and neighbourhood topologies are defined for all algorithms. In contrast, for the second group, Alba and Dorronsoro (2005) defined three 2D-cGAs that use different static *NGR*s, while the same selection method (BT) is used in all algorithms. In order to carry out a fair comparison, these algorithms are implemented over 3D grid topologies. The first algorithm works over a cubic grid topology arranged as $6 \times 6 \times 6$ with an *NGR* of 0.313. The second algorithm employs a rectangular cuboid arranged as $3 \times 24 \times 3$ with an *NGR* of 0.129. Finally, a narrow cuboid grid arranged as $2 \times 54 \times 2$ is used by the third algorithm with an *NGR* of 0.059. The grid dimensions were chosen based on two reasons: the first is to produce an equivalent population size of 216 for the different shapes (i.e., narrow, rectangle, and square), and the second is to produce selection pressures similar to those obtained by the algorithms in the first group. More discussion about the latter issue is provided below.

The choice of the selection rates and *NGR*s above are made to offer the closest selection pressure between the compared algorithms. As can be seen from Figure 5.4, the same growth curve is obtained by the first algorithms in each group as they have similar parameters, most importantly the cubic grid and selection intensity (as illustrated earlier, ST with $r = 0.0$ is

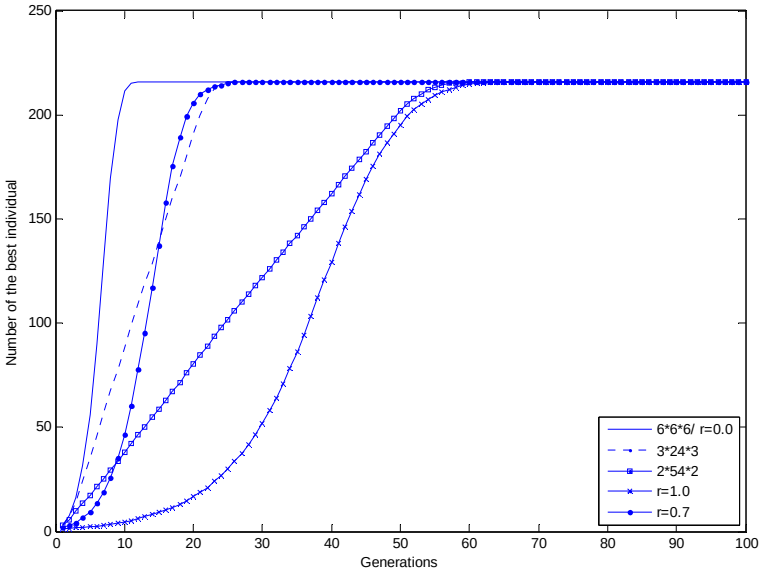


Figure 5.4. Growth number of the best individual with different grid shapes ($6 \times 6 \times 6$, $3 \times 24 \times 3$, and $2 \times 54 \times 2$) and selection rates ($r = 0.0$, $r = 0.7$, and $r = 1.0$).

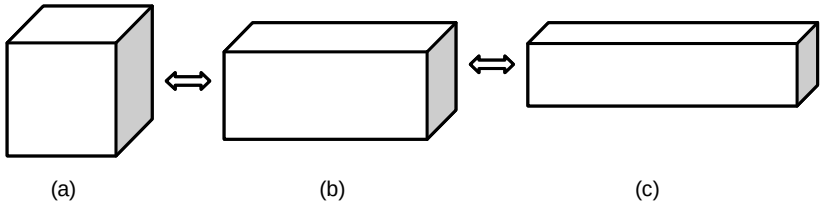


Figure 5.5. Alternation between different ratios: (a) cubic (*NGR* = 0.313), (b) Rectangular cuboid (*NGR* = 0.129), (c) narrow cuboid (*NGR* = 0.059).

equivalent to BT). In contrast, the growth curves obtained by the second and third algorithms in each group are slightly different. For example, the takeover time (i.e., the point where both curves started to stabilise) reached with different *NGR*s was two generations prior to the algorithms with different selection rates. However, those curves are significantly different in the way they change. For instance, the growth curves obtained with different topologies show almost linear trends while the curves obtained with different selection rates are nonlinear.

As pointed out earlier, increasing r results in more exploration while more exploitation is observed when r is decreased (refer to Figure 5.1). Hence, the proposed adaptive algorithm, the Convergence-Speed-Guided 3D-cGA, tunes the value of r for a specific convergence speed in order to control the exploration/exploitation trade-off.

A similar approach in determining the convergence speed is followed and the same adaptive pattern is used as in (Alba and Dorronsoro, 2005). The only difference between the dynamic-adaptive algorithm proposed in (Alba and Dorronsoro, 2005) and the proposed Convergence-Speed-Guided 3D-cGA is in the way the exploration/exploitation trade-off is controlled. Alba and Dorronsoro (2005) defined three different grid shapes—square, rectangular, and narrow—in order to alternate between the exploration and exploitation modes on the basis of the convergence speed (remember that in this study these are implemented over 3D grid topologies). Similar grid shapes are defined as in the static algorithms discussed previously; the cubic grid is used to promote more exploitation while the narrow cuboid grid is used to offer more exploration. A middle point between exploration and exploitation is provided by the use of the rectangular cuboid grid (see Figure 5.5).

In contrast, the Convergence-Speed-Guided 3D-cGA alternates between different selection rates, which are similar to the ones defined for the static algorithms. The exploitation is promoted through r = 0.0, r = 0.7 presents the middle point, and r = 1.0 promotes the exploration.

The convergence speed is measured through the calculation of genotypic diversity, in particular, the population entropy (H_t). Besides being an inexpensive metric, it efficiently represents the state of the search (Alba and Dorronsoro, 2005). H_t is calculated as the average values of the entropy of each gene in the population. Hence, the convergence speed is determined by the difference in the population entropies of two successive generations ($\Delta H_t - \Delta H_{t-1}$, $\Delta H_t = H_t - H_{t-1}$). If the difference decreases by a factor of ε, then the convergence speed is fast; otherwise, the convergence speed is slow when the difference increases by $(1 - \varepsilon)$ (refer to (Alba and Dorronsoro, 2005) for more details).

The adaptive pattern defined is summarised in Algorithm 5.2 (Alba and Dorronsoro, 2005). According to the convergence speed, in order to promote more exploitation, the proposed algorithm changes to the next lower *r* value (in (Alba and Dorronsoro, 2005), next wider grid shape) while it changes to the next higher *r* value (in (Alba and Dorronsoro, 2005), next narrower grid shape) to promote more exploration.

Algorithm 5.2 Dynamic adaptive pattern

```
1. if C1 then
2.     Promote more exploitation;        // change r to lower value
3. else if C2 then
4.     Promote more exploration;         //change r to higher value
5. else
6.     No change;
7. end if;
```

C_1 and C_2 are the convergence speed measures such that C_1 is satisfied when the convergence speed is fast and C_2 is satisfied when the convergence speed is slow. C_1 and C_2 are defined as follows (refer to (Alba and Dorronsoro, 2005) for more details):

$$\begin{aligned} C_1 &\equiv \Delta H_t \prec (1+\varepsilon)\cdot \Delta H_{t-1}, \\ C_2 &\equiv \Delta H_t \succ (2-\varepsilon)\cdot \Delta H_{t-1}. \end{aligned} \tag{5.1}$$

5.3.2 Experimental Results and Analysis

In this section, first the parameters and performance metrics used in the experiments are presented. Next, the results obtained for the static and the dynamic 3D-cGAs proposed in the previous section are presented and analysed. Finally, a comparison between the

Convergence-Speed-Guided 3D-cGA and the other static and dynamic 3D-cGAs are provided.

The same parameters are used during the experiment in order to arrive at a fair comparison. Table 5.3 summarises these parameters. For all problems, a population size of 216 individuals is used. These are arranged over a 6×6×6 lattice. An exception is made for f_{GPS}, as a population size of 64 individuals organised over a 4×4×4 lattice is used because of its lower complexity compared to the other problems.

The local neighbourhood defined contains seven individuals, which are positioned on the east, west, vertical north and south, horizontal north and south, and the centre. The first parent was the current individual while the second parent was selected by using ST with rate *r*. An arithmetic crossover operator with probability P_c = 0.9 was applied to generate an offspring. The offspring was mutated by applying a non-uniform mutation operator, with probability P_c = 0.1. The replacement policy defined here was *replace-if-better*, during which the current individual was replaced if its competitor (offspring) was fitter. Finally, the algorithm terminated if the difference between the average fitness values *avgf* and the optimum fitness value *optf* satisfied a specified threshold. Because of the different characteristics, we used different thresholds for each problem: 0.003 for f_{GPS}, 0.3 for f_{SLE}, 0.05 for f_{FMS}, 0.1 for f_{Ros}, and 0.005 for the other two problems. Similarly, the maximum number of generations assigned was 150 generations for f_{GPS}; 1000 generations for f_{SLE}, f_{Ras}, and f_{Ros}; and 2000 generations for f_{FMS} and f_{Ack}.

The performance of the algorithms was measured using three metrics: the search success rate (efficacy), the average number of generations (efficiency), and the average execution times (speed) of 100 independent runs.

Preliminary experiments were carried out taking into consideration the proposed algorithm (i.e., The Convergence-Speed-Guided 3D-cGA) and the dynamic 3D-cGA based on (Alba and Dorronsoro, 2005), in which different ε values (0.05, 0.15, 0.25, and 0.3) were tested. Based on these tests, ε = 0.05 was selected for both algorithms as the best one in terms of efficiency, efficacy, and speed for most the problems considered (to avoid reader distraction the details are provided in Appendix B.3, Tables B.6–B.9).

Table 5.4 presents the results obtained for all the algorithms compared. For each algorithm and problem, the average number of generations, the search success rate, and the average run times are illustrated. In addition, in order to show the robustness of the algorithms, the median absolute deviations *mad* are added to the results obtained (*mad* is used because of the non-normal distribution of the results obtained). The best results achieved for each problem are

marked in **bold**. The symbol '+' in Table 5.4 indicates that generally there are significant differences between all the compared algorithms in terms of all performance metrics (details about the statistical tests were provided in Section 2.2.3.1).

Table 5.3. Parameters used in the experiments

Population size:	216 individuals (64 individuals for f_{GPS})
Parent selection:	Current + ST, *r* (BT for the algorithms in (Alba & Dorronsoro, 2005))
Recombination:	AX, $P_c = 0.9$
Mutation:	Non-uniform, $P_m = 0.1$
Replacement:	Replace-if-better
Neighbourhood:	NEWS
Lattice:	Cubic: 6×6×6 (4×4×4 for f_{GPS}) Rectangular cuboid: 3×24×3 (2×8×4 for f_{GPS}) Narrow cuboid: 2× 54× 2 (2×16×2 for f_{GPS})
Termination criterion:	$\|avgf - optf\| \leq threshold$

Table 5.4. Experimental Results: Convergence time (CT), rate (CR), and speed (SP)* obtained by different dynamic and static 3D-cGAs

Problem	**Dynamic 3D-cGAs**		**Static 3D-cGAs**					Test
	The convergence -Speed-Guided	The approach in (Alba & Dorronsoro, 2005)	*r* = 0.0/ cubic	*r* = 0.7	*r* = 1.0	Rectangular cuboid (Alba & Dorronsoro, 2005)	Narrow cuboid (Alba & Dorronsoro, 2005)	
f_{Ras}	752.89 ± 65.5 **100%** 51.25 ± 4.61	541.43 ± 59.5 **100%** 37.68 ± 4.10	561.19 ± 41.0 **100%** 41.17 ± 2.89	781.32 ± 68.5 **100%** 58.33 ± 5.69	949.64 ± 27.0 57% 46.48 ± 1.68	**521.68** ± 47.5 **100%** **34.15** ± 3.19	635.06 ± 27.5 100% 44.74 ± 3.47	+
f_{Ack}	1598.1 ± 143.0 99% 117.7 ± 10.3	1337.5 ± 189.5 **100%** 100.54 ± 13.76	1256.8 ± 203.5 **100%** 97.36 ± 15.9	1592.2 ± 151.5 **100%** 116.57 ± 10.94	1991.0 ± 0.00 1% 128.3 ± 0.00	1224.4 ± 255.0 **100%** 91.98 ± 18.8	**1168.2** ± 201.0 **100%** **88.71** ± 15.5	+
f_{Ros}	661.68 ± 192.0 **22%** 45.39 ± 13.05	0%	**610.09** ± 125.0 11% **44.72** ± 9.70	728.41 ± 173.5 12% 53.23 ± 12.2	679.71 ± 177.0 14% 46.24 ± 11.8	953.0 ± 8.0 3% 60.44 ± 0.50	869.66 ± 10.0 6% 57.7 ± 0.78	+
f_{FMS}	981.41 ± 342.0 72% 94.41 ± 32.5	**944.53** ± 380.0 54% **90.65** ± 35.3	1039.1 ± 252.0 58% 114.13 ± 27.9	1127.5 ± 284.0 69% 105.9 ± 27.7	1533.1 ± 315.0 61% 143.3 ± 31.2	1022.3 ± 383.0 64% 113.1 ± 41.3	1317.0 ± 254.0 **81%** 144.3 ± 29.2	+
f_{SLE}	535.46 ± 99.0 26% 37.91 ± 7.02	**228.2** ± 3.00 5% **16.27** ± 0.28	278.0 ± 0.00 1% 19.5 ± 0.00	565.71 ± 67.5 **28%** 45.5 ± 5.20	866.62 ± 40.5 8% 59.27 ± 2.75	632.00 ± 0.00 1% 41.42 ± 0.00	0%	+
f_{GPS}	93.02 ± 9.00 **100%** 1.76 ± 0.16	**70.57** ± 9.00 97% **1.38** ± 0.15	71.52 ± 8.5 96% 1.44 ± 0.17	113.44 ± 11.0 97% 2.36 ± 0.25	137.63 ± 5.00 11% 2.88 ± 0.10	74.48 ± 7.5 **100%** 1.57 ± 0.15	91.65 ± 7.0 **100%** 1.95 ± 0.28	+

* For more details about the performance measures, please refer to Section 2.2.3.1.

In general, the dynamic algorithm based on (Alba and Dorronsoro, 2005) achieved the best performance in terms of efficiency and speed when solving most problems concerned, while the Convergence-Speed-Guided 3D-cGA achieved the best performance in terms of efficacy.

Complex problems need a high level of diversity to converge to the global optimum. Changing the grid shapes requires a recalculation of the positions of the individuals, which introduces a kind of migration. This migration offers more diversity; however, it is limited by good solutions because of BT. The combination of different selection intensities induced by the alternation between different grid shapes and the more diversity induced by the migration leads to significant reduction in convergence time (i.e., number of generations), and thus the run time. For example, the improvement in the efficiency and speed reached up to 28% and 26%, respectively, for f_{Ras} when compared to the Convergence-Speed-Guided 3D-cGA. However, this approach failed to solve more complex problems such as f_{Ros}, and achieved low search success rates when solving real world problems, in particular f_{SLE}, in which the inter-parameter linkage is very strong.

In contrast, the Convergence-Speed-Guided 3D-cGA controls the selection intensity and the level of diversity by allowing worse solutions to be involved in the update process, which induces a positive effect on problems with high degrees of complexity. Looking back at Figure 5.4, it can be seen that there is a difference between the trends obtained with $r > 0.0$ and those obtained with the cuboid shapes. The trends obtained with $r > 0.0$ show more gradual growth in the number of the best individuals that leads to a better explorative/exploitative behaviour. Good exploration is essential especially at initial stages in order to discover promising areas, while gradual offering of exploitation is crucial at later stages in order to improve the quality of solutions (Alba and Dorronsoro, 2005). Thus, the behaviour observed helps to raise the search success rate; however, it increases the convergence time.

Regarding static algorithms, in general, the worst performance is achieved with $r = 1.0$ in most problems because poor solutions are always favoured, which leads to weak exploitation and premature convergence. In contrast, the best performance is achieved with the different cuboid shapes in most problems. Exceptions are f_{Ros} and f_{SLE}, as the best efficacy is achieved with $r = 1.0$ and $r = 0.7$, respectively, because of these problems' higher complexities.

The proposed Convergence-Speed-Guided 3D-cGA was compared with all other dynamic and static algorithms considered. The results are shown in Tables 5.5 and 5.6, in which the symbol '+' indicates that the proposed Convergence-Speed-Guided 3D-cGA is significantly better than its counterpart, the symbol '•' denotes no statistical difference and the symbol '–'

indicates that the proposed Convergence-Speed-Guided 3D-cGA did worse than its counterpart.

Table 5.5 compares the Convergence-Speed-Guided 3D-cGA to all other algorithms in terms of average number of generations and average run times, as similar results are obtained with both metrics. Concerning f_{SLE}, f_{Ros}, and f_{FMS}, the efficiency and speed obtained by the proposed algorithm were either significantly better or had no significant statistical differences to those compared. An exception is for f_{SLE}, as the dynamic 3D-cGA based on (Alba and Dorronsoro, 2005) outperformed the proposed Convergence-Speed-Guided 3D-cGA. With regard to other problems, the efficiency and speed achieved by the proposed algorithm were worse than those achieved by other algorithms, except for static algorithms with $r = 0.7$ and $r = 1.0$. An exception is for f_{GPS}, as the efficiency and speed obtained by the proposed algorithm were statistically insignificant compared to the ones obtained by the static algorithm with narrow cuboid. Based on the problems' characteristics, f_{Ras} and f_{Ack} are considered to be less complex than other problems concerned. Thus, the level of diversity needed to solve the two problems efficiently is less than the one needed to solve the other problems; however, f_{Ack} requires more diversity than f_{Ras}. The high diversity provided by the proposed algorithm (refer to Figure 5.4) is the main cause that leads to additional cost in terms of convergence time and speed. Another additional cost in efficiency and speed were observed for f_{GPS}; although f_{GPS} is of high complexity, the problem's dimension is considerably lower than other problems. Hence, the efficiency and speed obtained by the proposed algorithm are either significantly better or have insignificant differences, especially when solving problems of high complexity.

Table 5.6 evaluates the Convergence-Speed-Guided 3D-cGA in terms of search success rate. The proposed algorithm achieves superior efficacy for most problems; the improvements are either significantly better or similar to the other algorithms compared.

To summarise, we note that for most cases the proposed Convergence-Speed-Guided 3D-cGA does either significantly better or similar to the other algorithms in terms of all performance metrics; the exceptions are mainly for f_{Ras} and f_{Ack}. Thus, it can be concluded that the proposed Convergence-Speed-Guided 3D-cGA has the most desirable behaviour among all the compared algorithms.

5.3.3 Conclusion

This study analysed the behaviour of a 3D-cGA against different grid shapes and selection rates over several problems with variable difficulties to investigate their influence on the

performance of the algorithm. Next, a new dynamic-adaptive 3D-cGA, the Convergence-Speed-Guided 3D-cGA was proposed, which aims to dynamically balance the exploration/exploitation trade-off. The proposed algorithm is compared to the first dynamic-adaptive cGA reported in (Alba and Dorronsoro, 2005).

The proposed Convergence-Speed-Guided 3D-cGA provides higher search success rates than all the other algorithms compared. In addition, it provides adequate efficiency, particularly when solving problems of high complexity. Thus, in general, it can be stated that the Convergence-Speed-Guided 3D-cGA could successfully achieve an appropriate balance between the exploration and exploitation.

Table 5.5. Comparison of the Convergence-Speed-Guided 3D-cGA versus other dynamic and static 3D-cGAs in terms of convergence time (CT) and speed (SP)*

Problem	The approach in (Alba & Dorronsoro , 2005)	r = 0.0/ cubic	r = 0.7	r = 1.0	Rectangular cuboid (Alba & Dorronsoro, 2005)	Narrow cuboid (Alba & Dorronsoro , 2005)
f_{Ras}	–	–	•	+	–	–
f_{Ack}	–	–	•	•	–	–
f_{Ros}	•	•	•	•	•	•
f_{FMS}	•	•	•	+	•	+
f_{SLE}	–	•	•	+	•	+
f_{GPS}	–	–	+	+	–	•

Note that the comparison results based on the two metrics (CT and SP) are merged as the results obtained were similar.

Table 5.6. Comparison of the Convergence-Speed-Guided 3D-cGA versus other dynamic and static 3D-cGAs in terms of convergence rate (CR)*

Problem	The approach in (Alba & Dorronsoro , 2005)	r = 0.0/ cubic	r = 0.7	r = 1.0	Rectangular cuboid (Alba & Dorronsoro, 2005)	Narrow cuboid (Alba & Dorronsoro , 2005)
f_{Ras}	•	•	•	+	•	•
f_{Ack}	•	•	•	+	•	•
f_{Ros}	+	+	•	•	+	+
f_{FMS}	+	+	•	•	•	•
f_{SLE}	+	+	•	+	+	+
f_{GPS}	•	+	•	+	•	•

* For more details about the performance measures, please refer to Section 2.2.3.1.

5.4 Comparison of Diversity-Guided versus Other Dynamic and Static 3D-cGAs

This section compares the Diversity-Guided 3D-cGA proposed in Section 5.2 to other dynamic and static 3D-cGAs that were discussed in Section 5.3. They are as follows: Convergence-Speed-Guided 3D-cGA, Dynamic and static 3D-cGAs based on (Alba and Dorronsoro, 2005), and 3D-cGAs with static selection rate (r = 0.0, r = 0.7, and r = 1.0). In order to obtain a fair comparison, in this section the Diversity-Guided 3D-cGA are re-evaluated such that similar test suite and experimental parameters to those defined in Section 5.3 are used. The aim of the comparison is to study the behaviour of the different algorithms by exploring the influence of the exploration/exploitation trade-off on the search.

The benchmark chosen for evaluating the compared algorithms consisted of the following test and real-world problems: Rastrigin (f_{Ras}), Rosenbrock (f_{Ros}), Ackley (f_{Ack}), FMS (f_{FMS}), SLE (f_{SLE}), and GPS (f_{GPS}) problems (details about f_{GPS} are provided in Section 4.2.2, while details about the other problems are provided in Appendix A).

The experimental parameters defined for the Diversity-Guided 3D-cGA were similar to the parameters illustrated in Table 5.1, the only difference being the population size as in this section a population of 216 individuals arranged as 6×6×6 was used to provide similar number of individuals to those offered by the other compared algorithms. Preliminary experiments were performed in order to select the best value of γ (recall that in this section the population size is smaller than the one defined in Section 5.2, therefore another set of preliminary experiments were needed to select the best γ); the best chosen γ value based on the convergence time, rate, and speed is also 0.4 (refer to Appendix B.2, Tables B.4 and B.5 for more details). The parameters defined for the Convergence-Speed-Guided and other dynamic and static 3D-cGAs are summarised in Table 5.3.

The comparison was performed in terms of the following performance metrics: convergence time, rate, and speed. Statistically significant tests were used as approaches to compare the different algorithms. These tests determined the significance level of the differences between the compared algorithms (details about the statistical tests were provided in Section 2.2.3.1). The significance levels are indicated using the following symbols. A plus sign '+' denotes that the Diversity-Guided 3D-cGA significantly outperformed its counterpart, while a non-significant difference is denoted by the symbol '•'. The symbol '–' indicates that the Diversity-Guided 3D-cGA did worse than its counterpart (see Tables 5.8–5.10).

Table 5.7 presents the results obtained for the Diversity-Guided 3D-cGA with 6×6×6 grid topology rather than the 7×7×7 grid used in Section 5.2. The comparison results based on each metric: convergence time, rate, and speed are illustrated in Tables 5.8–5.10, respectively.

Table 5.7. Experimental Results: Convergence time (CT), rate (CR), and speed (SP)[*] obtained by the Diversity-Guided 3D-cGA with 6×6×6 grid

Problem	$\gamma = 0.4$
f_{Ras}	$570.83_{\pm 49.5}$ 100% $36.80_{\pm 3.23}$
f_{Ack}	$1399.0_{\pm 190.0}$ 100% $89.58_{\pm 12.34}$
f_{Ros}	$445.87_{\pm 256.0}$ 55% $27.83_{\pm 15.9}$
f_{FMS}	$981.65_{\pm 336.5}$ 52% $89.77_{\pm 31.91}$
f_{SLE}	$300.80_{\pm 36.0}$ 10% $20.28_{\pm 2.56}$
f_{GPS}	$96.26_{\pm 6.5}$ 100% $1.88_{\pm 0.11}$

Table 5.8. Comparison of the Diversity-Guided 3D-cGA versus other dynamic and static 3D-cGAs in terms of convergence time (CT)[*]

Problem	The Convergence-Speed-Guided	The approach in (Alba & Dorronsoro, 2005)	$r = 0.0$/ cubic	$r = 0.7$	$r = 1.0$	Rectangular cuboid (Alba & Dorronsoro, 2005)	Narrow cuboid (Alba & Dorronsoro, 2005)
f_{Ras}	+	•	•	+	+	–	+
f_{Ack}	+	•	•	+	•	–	–
f_{Ros}	•	+	•	+	•	+	+
f_{FMS}	•	•	•	•	+	•	+
f_{SLE}	+	•	•	+	+	+	+
f_{GPS}	•	–	–	+	+	–	•

[*] For more details about the performance measures, please refer to Section 2.2.3.1.

Looking at Table 5.8, in general, it can be seen that in most cases the Diversity-Guided 3D-cGA outperforms the other compared algorithms in terms of efficiency (see the '+' sign), in particular the ones that show more explorative behaviour such as 3D-cGA with r = 0.7, r = 1.0, and narrow cuboid; while there are only few cases in which the Diversity-Guided 3D-cGA does worse than the algorithms compared (indicated by the symbol '–'). The remaining cases show non-significant differences between the compared algorithms (see the symbol '•'). The reduced selection pressure induced by 3D-cGA with r = 0.7, r = 1.0, and a narrow cuboid assists the exploration leading to increase in the convergence time; however the dynamic control of the selection pressure overcomes this issue.

Table 5.9. Comparison of the Diversity-Guided 3D-cGA versus other dynamic and static 3D-cGAs in terms of convergence rate (CR)*

Problem	The Convergence-Speed-Guided	The approach in (Alba & Dorronsoro, 2005)	r = 0.0/ cubic	r = 0.7	r = 1.0	Rectangular cuboid (Alba & Dorronsoro, 2005)	Narrow cuboid (Alba & Dorronsoro, 2005)
f_{Ras}	•	•	•	•	+	•	•
f_{Ack}	•	•	•	•	+	•	•
f_{Ros}	+	+	+	+	+	+	+
f_{FMS}	–	•	•	•	•	•	+
f_{SLE}	–	•	+	–	•	+	+
f_{GPS}	•	•	+	•	+	•	•

Table 5.10. Comparison of the Diversity-Guided 3D-cGA versus other dynamic and static 3D-cGAs in terms of convergence speed (SP)*

Problem	The Convergence-Speed-Guided	The approach in (Alba & Dorronsoro, 2005)	r = 0.0/ cubic	r = 0.7	r = 1.0	Rectangular cuboid (Alba & Dorronsoro, 2005)	Narrow cuboid (Alba & Dorronsoro, 2005)
f_{Ras}	+	•	+	+	+	•	+
f_{Ack}	+	+	•	+	•	•	•
f_{Ros}	+	+	•	+	•	•	+
f_{FMS}	•	•	•	•	+	•	+
f_{SLE}	+	•	•	+	+	•	+
f_{GPS}	•	–	–	+	+	–	+

* For more details about the performance measures, please refer to Section 2.2.3.1.

With regard to the efficacy, by inspecting Table 5.9, generally, it can be observed that in most cases the differences between the compared algorithms are insignificant (observe the '•' symbol), while very few cases show the deterioration of the Diversity-Guided 3D-cGA (observe the '–' symbol). The significant success of the Diversity-Guided 3D-cGA (observe the '+' symbol) is mainly noticed when compared to the algorithms that strongly support the 'explore' mode (i.e., have weak selection pressure) such as the ones with r = 1.0 and narrow cuboid, or the ones that support the 'exploit' mode (i.e., have strong selection pressure) such as 3D-cGA with r = 0.0/cubic. As mentioned earlier, the exploration may lead to reduction in solutions accuracy, while the exploitation may lead to premature convergence, with both situations the algorithm would fail to find the best solutions leading to divergence and hence reduction in the search success rate.

Table 5.10 compares the different algorithms in terms of the execution time (speed). Overall, for most cases, the Diversity-Guided 3D-cGA outperformed the other compared algorithms (observe the '+' sign). Very few cases show a decline in the speed obtained by the Diversity-Guided 3D-cGA (observe the '–' symbol), while the rest of the cases show non-significant differences (see the symbol '•'). As with the case for the algorithm's efficiency, most cases that show the superior improvements of the Diversity-Guided 3D-cGA are acquired by the algorithms with more explorative behaviour (i.e., 3D-cGA with r = 0.7, r = 1.0, and narrow cuboid). Hence, a relation between the efficiency and the speed of the algorithm could be determined.

As each of the problems considered possessed different characteristics, which presented different levels of difficulty, there is no one globally best algorithm for all problems. Hence, different exploration/exploitation tradeoffs are needed to effectively solve a given problem. More complex problems require more diversity and hence more exploration, however too much exploration leads to a reduction in the quality of the solutions. That is why the algorithms with dynamic balancing between exploration and exploitation are favoured. A general conclusion that was drawn from the previous sections stated that the dynamic algorithms showed superior improvement in terms of all performance metrics comparing to the static algorithms; this conclusion also conforms to that of (Alba and Dorronsosro, 2005).

The above discussion has provided a general indication about the benefits gained by the dynamic algorithms as these show the best performance. Now, in order to provide a deep insight into the behaviours of the different algorithms, the problem of Rastrigin (f_{Ras}) is selected for use in a case study of the behaviour of the algorithms by inspecting the change in the population diversities (remember that the diversity loss trends are diverse among the

different problems—refer to Figure 5.3). Figures 5.6 and 5.7 show trends in the average genotypic diversities obtained by the Diversity-Guided 3D-cGA and the other compared algorithms, respectively.

Solving f_{Ras}, the lowest numbers of generations were obtained by the algorithms based on (Alba and Dorronsoro, 2005), particularly the dynamic and static 3D-cGAs with cubic and rectangular cuboids (refer to Table 5.4). Although the measure of diversity used was different, the Diversity-Guided 3D-cGA obtained the next better number of generations (see Table 5.7). These achievements could be justified by looking at Figures 5.6 and 5.7; the diversity obtained by the Diversity-guided 3D-cGA starts reaching almost zero at generation 400 (see Figure 5.6), while for the dynamic and static 3D-cGAs with cubic and rectangular cuboids (based on (Alba and Dorronsoro, 2005)) it starts to reach zero slightly before generation 400 (see Figure 5.7). On the other hand, the diversity reaches zero at extremely later stages with the other compared algorithms (i.e., the Convergence-Speed-Guided 3D-cGA and the static 3D-cGAs with r = 0.7, r = 1.0, and narrow cuboid); the worst efficiency was obtained by 3D-cGA with r = 1.0 as it shows the most explorative behaviour. For that reason, 3D-cGA with r = 1.0 also obtained the worst search success rate. As is well known, the diversity is reaches almost zero when the best-found solution conquers the entire population.

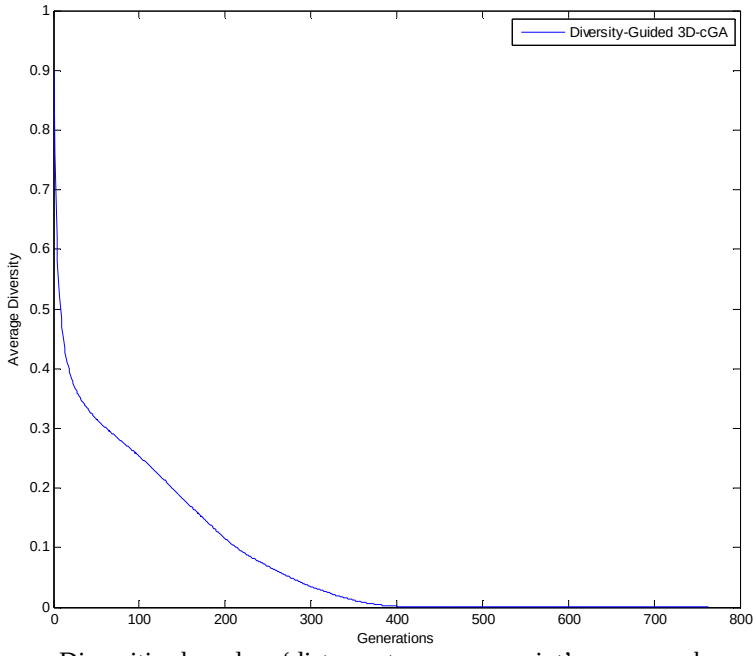


Figure 5.6. Average Diversities based on 'distance-to-average-point' measure when solving f_{Ras} by Diversity-Guided 3D-cGA.

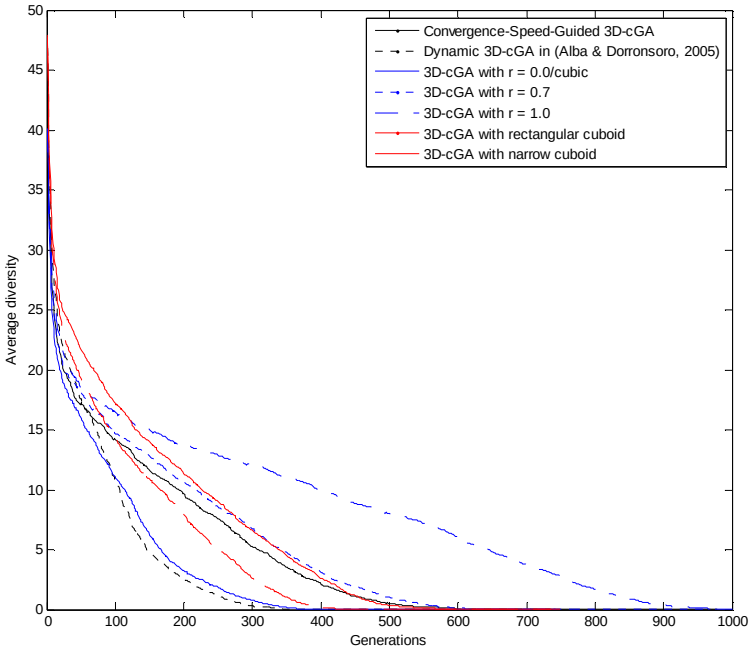


Figure 5.7. Average Diversities based on 'genotypic entropy' when solving f_{Ras} by the dynamic and static 3D-cGAs under study.

5.4.1 Conclusion

This section analysed and compared several dynamic and static algorithms based on canonical cGA while maintaining similar parameters and test suite. The main motivation for this comparison was to study the influences of introducing different exploration/exploitation tradeoffs on the performance of the algorithms. Furthermore, the comparison provided has been validated through the use of statistical significance tests.

Theoretically, although there is no one adaptive criterion which is best and appropriate for all problems, in general, the Diversity-Guided 3D-cGA achieves the most desirable performance for the most considered problems (has been confirmed by the statistical significance tests). In addition, the Diversity-Guided 3D-cGA not only improves the existing performance, but also incurs no implementation costs (no grid shape change is needed). Hence, it can be concluded that the Diversity-Guided 3D-cGA is an effective algorithm that balances between the exploration and the exploitation in a dynamic and continuous manner.

5.5 Summary and Contribution to Knowledge

This chapter emphasised the field of dynamic-adaptation in structured EAs, specifically cGAs. The class of the adaptation considered for this work was the adaptive-dynamic in which the change occurs according to feedback information from the algorithm. The importance of dynamic cGAs is growing due to their capability for self-adapting their exploration/exploitation trade-off. In the literature, several ways have been investigated in order to enhance the population diversity and accordingly the global selection pressure. A simple way was to rearrange the locations of the individuals through the change in the grid shape. Another way is through the control of genetic parameters such as the selection rate (which is the method used in this work). The main motivation for this work was to introduce new and effective algorithmic variants with low computation costs that contribute to the field of dynamic adaptation in EAs. Two new dynamic algorithms have been proposed, namely: the Diversity-Guided and Convergence-Speed-Guided 3D-cGA, which are mainly differing in the adaptive criterion used. These algorithms were compared with other dynamic and static algorithms from the literature. The Diversity-Guided 3D-cGA achieved the most desirable performance over the other compared algorithms for problems considered. The following points sum-up the main contributions of this study to existing knowledge.

- The stochastic binary selection operator is used as a mechanism to dynamically balance between the exploration and the exploitation. The selection operator guides the search towards exploration by increasing the rate of the selection (i.e., offer more chances for even worse solutions to survive), or guides the search towards exploitation by reducing the rate of selection (i.e., focus on fitter solutions).

- The Diversity-Guided 3D-cGA showed superior improvement in terms of efficacy and reached up to 35% compared with static 3D-cGAs for the most studied problems, and in specific problems with higher complexity. This improvement varied due to different problem complexities. With regard to algorithm efficiency, the Diversity-Guided 3D-cGA showed the most robust behaviour although the best efficiencies were achieved by 3D-cGA with $r = 0.0$. However, the differences in the efficiencies obtained were not significant for the

most studied problems; the exceptions could be referred to the differences in the search success rate obtained.

- The Diversity-Guided 3D-cGA demonstrated its capability to offer the most suitable balance between enhancing population diversity (exploration) and tuning good solutions (exploitation) for the most studied problems. The following example confirms the conclusion stated above. For f_{Lang} and f_{FMS}, the algorithm tended to promote exploration after the initial stage (refer to Figure 5.3) instead of starting by introducing the exploitation, as is the case with the other considered problems (less complex).

- The Convergence-Speed-Guided 3D-cGA used similar adaptive criterion compared to the dynamic algorithm in (Alba and Dorronsoro, 2005). The mechanisms used to swap between 'explore' and 'exploit' modes for the former was the change in the selection rate, while the change in the grid shapes was used for the latter. The change in the grid shape leads to a rearrangement of the positions of individuals, which therefore induces a kind of individual migration that contributes to improvement. In contrast, the proposed algorithm was successful in obtaining an appropriate balance between exploration and exploitation without affecting other genetic operations.

- The proposed algorithms (i.e., the Diversity-Guided and the Convergence-Speed-Guided 3D-cGA) showed their capability in balancing exploration and exploitation. Improvements in the performance presented as a reduction in the convergence time and an increase in the convergence rate were achieved. However, the rates of the improvements varied mainly due to the different problems' characteristics. In both algorithms the adjacency of the individuals were maintained, which awards any improvement achieved to the change in selection rates.

- The comparative analysis of the proposed algorithms (the Diversity-Guided and the Convergence-Speed-Guided 3D-cGA) and other dynamic and static algorithms, showed that the most desirable performance for the most studied problems was achieved by the Diversity-Guided 3D-cGA. Therefore, the start

with ‘explore’ mode following a gradual introduction of the exploitation resulted in the best balance between exploration and exploitation (recall that the other dynamic algorithms alternated between the two modes).

- In the Diversity-Guided 3D-cGA, the gradual introduction of the ‘exploit’ mode was carried out by a reduction in the selection rate. This reduction only occurred when the adaptive criterion was satisfied. This mechanism conforms to cGAs inherent features as cGAs starts with an exploration of promising areas followed by an exploitation of good solutions. The dynamic control of the move towards exploitation added a significant advantage to cGAs.

Chapter 6

Thesis Summary, Conclusion, and Future Work

This thesis aimed to utilise the unique embedded features of cGAs in order to further improve their performance, particularly when tackling hard real-world optimisation problems. As a result, the structural characteristics of cGAs, genetic operations, and critical fault scenarios were investigated form static and dynamic perspectives. From the structural point of view, the topology of the grid on which a cGA should be implemented was targeted as one way to improve the performance of cellular optimisation engines. From a fault tolerance point of view, genetic characteristics such as diversity were investigated to cope with faults encountered. Critical fault scenarios and mitigation techniques to tackle these scenarios were targeted through the utilisation of the genetic operations. In addition, the genetic operations were investigated from a dynamic point of view in order to obtain further improvements. The changes that occur in the genetic diversity as the search process progress was used as a guide and a key factor to induce a dynamic alternation between exploration and exploitation modes.

In this chapter the works presented in this thesis are summarised (Section 6.1). Section 6.2 draws overall and study-specific conclusions. Finally, guidelines for future works are presented in Section 6.3.

6.1 Summary

This thesis demonstrated the effectiveness of cellular optimisation engines in tackling problems of diverse complexities such as highly multimodal, epistasis, asymmetry problems. The most well known standard GAs, ssGAs and genGAs, were compared to three-

dimensional cGAs—the basis for this research (refer to Section 2.2.5.1). Similar algorithmic parameters and benchmark problems were used to achieve a fair comparison. The comparative results indicated the advantage of cGAs in that higher efficacy was achieved while maintaining desirable efficiency. Further, cGAs proved their ability to solve problems of different characteristics, while standard GAs failed to solve some of these problems.

Chapter 3 analysed the performance of cellular GAs implemented on grids with different cellular dimensions. The expectation that cGAs with higher cellular dimensions may offer advantages over lower cellular dimensions was the main motivation. This study is a continuation of a preliminary study that was carried out with other members in the System Level Integration research group. In this research, an experimental study was carried out by considering an extended test bench including test functions of higher dimensions and real-world problems to compare the performance of cGAs when implemented on 3D and 2D grid structures. In addition to the cellular dimensions, the experimental settings included different population and neighbourhood sizes.

In summary, the various configurations of the 3D-cGA have proven to be more efficient than the 2D-cGAs in terms of convergence time when tackling all the considered problems. With respect to the efficacy, both cellular structures showed similar success rates. However, the 3D-cGA showed improvement over the 2D-cGAs when a smaller local neighbourhood radius was applied. A 3D grid provides a larger neighbourhood size than a 2D grid considering similar population sizes. This is a consequence of the cell arrangement as it consists of several 2D-layers. Interconnections between the cells result in vertical expansion, instead of horizontal expansion as in a 2D grid. Although this interconnection causes the algorithm to be more exploitative, the balance between exploitation and exploration is kept by choosing an appropriate neighbourhood radius with respect to the grid's topology. Therefore, if the selection pressure is controlled by these parameters, higher search success rates and better convergence time are reached.

If the benefits of the performance results obtained are merged with the advantages that 3D technology brings, the resulting architecture offers significant advantages in terms of the following: routing length decrease, interconnection delay reduction, and logic and memory density increase. As a result, in the future, it will be possible to improve the performance of today's optimisation engines at both software and hardware levels.

Chapter 4 has targeted the area of fault tolerance. The fault looked at in this research was SEU and the phenotypes were the data targeted by faults. This study focused on faults that targeted phenotypes due to their significant role in guiding the search process. If SEU affects

essential system data, the system will fail. Accordingly, isolation approaches and several mitigation techniques were introduced. These techniques were assessed against a benchmark suite of well-known test and real-world problems. These problems were selected to include diverse characteristics, which presented different difficulties to the search. Two fault scenarios were considered in this research. These scenarios were defined as being the most critical. This chapter was divided into three parts, with each part introducing and adding new mechanisms in order to increase the reliability of a system and to improve its performance.

In the first part (Section 4.1), a new algorithmic approach that tackled SEU errors targeting individuals' phenotypes was proposed. The proposed approach, Fault-Tolerant 3D-cGA, is based on the canonical cGA, and genetic diversity is the key metric used to identify and isolate faulty cells (individuals). For both fault scenarios, different fault ratios were considered; the ratio of the faults varied from 0% to 40% of the population. The use of genetic diversity demonstrated success in identifying and therefore isolating faulty cells. In addition, the integration of an explicit migration operation played a significant role in mitigating the impact of faults. The proposed migration operation in this research was designed to adapt to fault ratio encountered and showed significant improvement in the performance of a system. Another operation that was used to mitigate faults was the selection operation. In this study a stochastic binary tournament selection was used, two selection rates that have different effects on the exploration and the exploitation were assessed. These rates were selected to provide lower opportunity for faulty individuals to be selected and involved while updating a fault-free cell. Hence, different algorithmic configurations offering different exploration/exploitation tradeoffs were evaluated. These configurations mainly differed in the defined selection rate and the use of the migration operation. Overall, the proposed algorithm demonstrated success in recovering up to 40% of faults. However, the level of improvement in performance varied according to the type of problem and declined following the increment in fault rates. For all problems, the best efficiency was achieved by the configuration that employed the highest selection pressure with migration. Conversely, the best efficacy was achieved by the configurations that used a lower selection pressure, in this case the integration of the migration operation showed no significant improvement.

In the second part of Chapter 4, two new migration schemes were proposed in order to further improve the performance of the algorithm proposed in the previous section. The only difference between the newly introduced schemes and the one proposed in the first part was the source of the migrants. Using the first defined migration policy, the migrants were

selected from the first fault-free neighbourhood identified, while using the new policies the migrants were selected from the current neighbourhood (the one for the currently updated individual). However, the new schemes differed as one selected the fittest fault-free individual while the other selected a random fault-free individual from the current neighbourhood. In this study, the different migration policies were compared for similar fault scenarios and ratios. Simulation results demonstrated the approach's success in recovering up to 40% of faults. In addition, the use of migration as a mitigation technique for fault tolerance offered considerable improvements in the efficiency, efficacy, speed, and reliability of the algorithm, especially for a high ratio of faults. In addition to being a mitigation technique, the integration of migration played an important role in controlling the exploration/exploitation trade-off. Exploration and exploitation are the two main issues in enhancing the performance of evolutionary algorithms. Overall, the best performance in terms of efficiency, efficacy, and speed was achieved with the migration operation that selected the fittest neighbour from the current neighbourhood due to its effect in enhancing the local selection intensity and diversity in the population.

The last part of Chapter 4 proposed a new algorithm, the Dynamic FT 3D-cGA, for handling failures that occurred at individuals' phenotypes, in particular, due to SEUs. Similarly, the approach is based on the canonical model of cGAs and is a modified version of the past approach (FT 3D-cGA) that used genetic diversity to identify and isolate faulty individuals. The most critical fault models were tackled in conjunction with different fault ratios. The main motivation for this study was to improve the reliability and performance of the FT 3D-cGA through dynamic control of exploration/exploitation trade-off. The dynamic calculation of the maximum allowed number of generations based on fault ratio encountered helped in enhancing the exploration. On the other hand, the exploitation was enhanced through the use of the proposed migration technique. In this study, several configurations concerning dynamic adaptation and migration were defined and evaluated. In addition, to illustrate the improvements achieved, the Dynamic FT 3D-cGA was compared to the FT 3D-cGA in terms of efficiency, efficacy, and speed. The results indicated that both algorithms demonstrate successful recovery of up to 40% of faults, especially when the migration technique was employed. Thus, it was confirmed that the use of migration as a mitigation technique to fault tolerance offers considerable improvements in the efficiency, efficacy, speed, and reliability of the algorithms, especially for the high ratio of faults. Overall, the best performance in terms of efficiency, efficacy, and speed was achieved with the use of the migration technique owing to its effect in enhancing the local selection intensity and

diversity in proportion. The FT 3D-cGA and the Dynamic FT 3D-cGA both with migration showed the best performance. The differences between the results obtained by the compared algorithms were not significant. An exception was for Ackley's problem as the Dynamic FT 3D-cGA with migration significantly outperformed the FT 3D-cGA with migration mainly in terms of efficacy and reliability. The best efficiency was achieved by the FT 3D-cGA with migration. However, this lower number of generations was found to be due to the significant difference in the obtained search success rate.

Chapter 5 emphasised the area of dynamic adaptation. The main idea behind the adaptivity was to dynamically control and provide an appropriate balance between exploration and exploitation for an algorithm. Exploration and exploitation are vital issues in improving the effectiveness and the performance of evolutionary algorithms. Population diversity is improved by exploring the search space, while the optimum solution can be found by exploiting the fitness information. Inappropriate balance between exploration and exploitation leads to inefficient search. This chapter was mainly divided into three parts.

The first part of Chapter 5 discussed the concept of selection pressure. In this part, the selection operation, the stochastic binary tournament selection, was used to induce different selection pressures through the use of different selection rates. An experimental setup was carried out to demonstrate the affect of only the selection operation on the selection intensity and the takeover time. The selection rates that were evaluated varied between 0 and 1. The results showed an indirect proportion between selection pressure and selection rate. In other words, the selection pressure decreased as the selection rate increased. This section established the basis for the subsequent parts.

The second part of Chapter 5 presented a new dynamic 3D-cGA that used genetic diversity measure to activate the control of the exploration/exploitation trade-off (Diversity-guided 3D-cGA). In this study, tuning the genetic operator parameters, specifically the selection rate, is the way to dynamically control the exploration/exploitation trade-off. A set of diverse characteristic problems was used to assess the performance of the algorithm. The dynamic algorithm was also compared to three static versions, each using a constant selection rate. These selection rates were selected to offer strongest, moderate, and weakest selection pressures. Simulation results showed that the dynamic algorithm outperformed the static ones with significant improvement for most of the problems studied. The exceptions either did not have statistical differences or showed more erratic behaviour. In general, the proposed adaptive criteria showed the ability to achieve a suitable balance between

enhancing population diversity (to escape local optima—efficacy) and tuning solutions (to improve solution quality—accuracy).

The last part of Chapter 5 analysed the behaviour of a 3D-cGA against various grid shapes and selection rates over several problems with variable difficulty to investigate their influence on the performance of the algorithm. Next, a new dynamic-adaptive 3D-cGA (Convergence-speed-guided 3D-cGA) that aimed at dynamically balancing the exploration/exploitation trade-off was presented. The proposed algorithm used convergence speed to activate the dynamic control, the measure of the convergence speed and the adaptive criteria used in this study were adopted from the work of Alba and Dorronsoro (2005). In their work, the alternation between shapes of grid structure was the way to dynamically tune the exploration/exploitation trade-off. Three different shapes were defined—square, narrow, and rectangular grids—to promote more exploration (change to next narrower shape) or more exploitation (change to next wider shape), while the proposed algorithm in this study alternated between three selection rates to tune the exploration/exploitation trade-off. These selection rates were selected to induce similar effect to that of the one induced by the alternation between grid shapes. The proposed algorithm was assessed against a benchmark of tests and real-world problems and was compared to the static and dynamic-adaptive cellular algorithm that were reported in (Alba and Dorronsoro, 2005), and the static algorithms from the previous section. Simulation results showed that the proposed adaptive algorithm provided higher search success rates than all other compared algorithms, as well as providing adequate efficiency, particularly when solving problems of high complexity. Generally, it can be stated that the proposed adaptive algorithm successfully achieved a suitable balance between exploration and exploitation.

In addition, the two proposed dynamic algorithms (Diversity-guided 3D-cGA and Convergence-speed-guided 3D-cGA) were compared. Similar parameters and test suites were compared to enable fair comparison. The motivation for this comparison was to study the different effects of introducing different exploration/exploitation tradeoffs on the performance of the algorithms. This comparison was validated through the use of statistical significance tests. In general, comparative results showed that Diversity-Guided 3D-cGA achieved the most desirable performance for most of the problems considered. In addition, the Diversity-Guided 3D-cGA incurred no implementation costs (no grid shape change was needed). Hence, it can be stated that the Diversity-Guided 3D-cGA is an effective algorithm that balances between exploration and exploitation in a dynamic and continuous manner.

6.2 Conclusion

The overall aim of this thesis was to investigate the inherent characteristics and the ability of cellular genetic algorithms to improve their performance and reliability when tackling hard optimisation problems. New techniques that added the features of fault tolerance and dynamic adaptation to the algorithms were introduced. Structural characteristics, decentralised population, the shape and the size of the population and neighbourhood topologies, implicit and explicit migration operations, genetic diversity, selection operation, and selection pressure were all utilised to achieve the aim of this research. This research was carried out in three main stages.

The first stage explored the cellular dimensionality and their implications on the performance of the algorithms. Several problems from the real world and test functions were tackled. These problems have diverse characteristics and thus introduced different complexity to the search. As the topology of the grid plays a significant role in determining the performance of EAs, a comparative analysis between cGAs with two-dimensional grid (the most common grid topology) and three-dimensional grid (rarely investigated) was developed. cGAs are commonly implemented on 1D or 2D toroidal grid structures. The comparison between 2D-cGA and 3D-cGA showed that the 3D-cGA is more efficient in terms of convergence time than 2D-cGA for all the problems considered, while both algorithms achieved similar efficacies. Due to the vertical expansion, the 3D structure provided a larger neighbourhood size than the 2D structure with similar distance steps. This led 3D-cGA to show more exploitative behaviour; however, a balance between exploitation and exploration was maintained by selecting an appropriate neighbourhood radius with respect to the grid topology. In conclusion, the control of the selection intensity through the size of the neighbourhood led to the attainment of higher search success rate and less convergence time. The findings will add significant benefits for future optimisation engines. Achieving better algorithmic performance with 3D-cGA creates a promising opportunity to combine the algorithmic benefits with the benefits of advanced custom silicon chip technology, 3D-IC.

The second stage was concerned with improving the effectiveness as well as the reliability of cellular genetic engines. Due to the significant miniaturisation of systems' electronics and its operation in hostile environments, systems are subjected to different kind of failures. Hence, in this stage fault-tolerant approaches and mitigation techniques were proposed. The first approach utilised cGAs' inherent features such as genetic diversity and

the selection operation. An algorithmic-based approach for tolerating SEE errors as well as an explicit migration operation were developed. A set of diverse-characteristic problems were tackled and critical fault models together with different fault rates were considered. Results showed that the algorithm was successful in isolating faults and showed the ability of the algorithm to converge with up to 40% faults. The best performance was achieved when an explicit migration operation was integrated into the algorithm. The migration aimed at covering the loss in cells due to the faults, which enhanced the reproduction process. In conclusion, the explicit migration operation played a vital role in mitigating faults and offered a better exploration/exploitation trade-off.

Subsequently, two more migration operations were proposed with the aim of further improving the performance. The best overall performance was achieved when the migration scheme that selected fault-free and fittest migrants from the current neighbourhood was utilised. A final improvement of the proposed fault tolerant 3D-cGA was carried out by introducing a dynamic adaptation technique as a mitigation measure. Several algorithm configurations were defined and assessed which also concerned the integration of migration. Results confirmed the previous findings, especially the vital role of the migration operation.

During the final stage of this research, adaptive-dynamic 3D-cGAs were developed in order to obtain an appropriate balance between exploration and exploitation. A first approach was introduced by utilising the genetic diversity. The dynamic search was guided by the genetic diversity and the selection rate was dynamically tuned according to the degree of diversity. The other proposed approach was guided by the convergence speed and accordingly the selection rate was tuned. The two proposed approaches were evaluated and compared with other static and dynamic 3D-cGAs. Results demonstrated the high performance of the first proposed approach with respect to other compared algorithms. The achievement of appropriate exploration and exploitation balance while maintaining algorithms' performance will positively contribute to the field of dynamic adaptation.

6.3 Future Work

This thesis focused on the inherent features of cGAs and their ability to improve cGAs performance. Three main aspects were explored: the structural characteristics including the cellular dimension and the topologies, size, and shape of the population and local neighbourhoods, the area of fault tolerance, and the dynamic adaptation. Although this

research thoroughly explored cGAs from various aspects, several aspects are still available for research.

This thesis has explored the effectiveness of cGAs when implemented on 3D cubic grid, while a little attention was paid to other 3D grid shapes. Therefore, the main opportunity to work in the future is to investigate the performance of cGAs when implemented over other 3D gird shapes such as narrow and rectangular cuboids. Different grid shapes offers different exploration/exploitation tradeoffs and therefore the use of a particular grid shape may allow more efficient optimisation for a specific type of problems. Previous researchers have showed the influence of using various grid shapes on the performance of cGAs when solving problems of various complexities; however, these studies concerned 2D grid topology. Thus, there is a need to extend previous studies to 3D grids. In addition, this thesis has showed that the use of higher cellular dimensions (i.e., 3D) offers promising results, in particular when solving problems of high complexity (i.e., real-world problems). This finding encourages the investigation of use even higher cellular dimensions such as 4D topology. Increasing the cellular dimensionality would result in more interconnections between cells producing a denser neighbourhood and faster spreading of individuals. Such configurations may offer advantageous for even harder problems. However, a careful selection of the genetic operations and other parameters should b made; these issues open a wide research area that is worth studying.

The next opportunities for further investigations concern the area of fault tolerance. The proposed fault tolerant approach focused on SEEs when targeting only the phenotypic space. This approach can be further extended to tackle errors targeting the genotypic space in conjunction with the phenotypic space. The changes that occur in one space are clearly reflected in the other. Hence, further investigation is needed to develop isolation criteria that tackle faults in both spaces. Another opportunity is to investigate other fault scenarios as the research on fault tolerance only considered the two most critical fault scenarios. Moreover, besides the phenotypic and genotypic spaces, the Finite State Machine is a potential structure of cGAs for faults to occur. Therefore, fault tolerant technique can be further investigated to consider other critical internal cGA structures.

Other opportunities can also focus on the area of dynamic adaptation. This research focused on the diversity and convergence speed genotypic-based measures. Therefore, further investigations are needed to evaluate phenotypic- or hybrid-based measures for diversity and convergence speed. In addition, the approaches proposed to dynamically balance between exploration and exploitation used the selection operation as a way to

achieve the dynamic control. Other genetic operations such as crossover and mutation can play an important role in the dynamic adaptation. Exploring the ability of these operations to dynamically tune the exploration/exploitation trade-off may result in less computation time.

Appendix A

Description of the Benchmark Problems

The algorithms, which were proposed and investigated in this research, were evaluated with a careful selection of performance benchmark problems in order to avoid an ad-hoc conclusion. The problems considered are selected as they possess diverse characteristics such as multimodality, epistasis, regularity, and asymmetry, introducing different levels of difficulty into the search (GEATbx, 2005; Alba and Dorronsoro, 2008). The details of these problems were omitted from the previous chapters so readers can pay more attention without any distraction. This Appendix presents a brief description of all the problems used in this research. All of the problems studied were belong to the field of continuous optimisation due to their complex features are commonly acquired by real-world problems. Some of the problems selected were from well-known academic test functions, while others were obtained from the real world.

Details about the test functions selected are presented in Section A.1, while Section A.2 gives the details of the real-world problems.

A.1 Test Functions

In this section seven benchmark test functions from well-known continuous minimisation functions are illustrated. They are: Rastrigin, Schwefel, Griewangk, Ackley, Michalewicz, Rosenbrock, and Langermann problems—the details are provided below.

Rastrigin's problem (f_{Ras}):

f_{Ras} is non-linear, multimodal, separable, and symmetric function. Multimodality—a

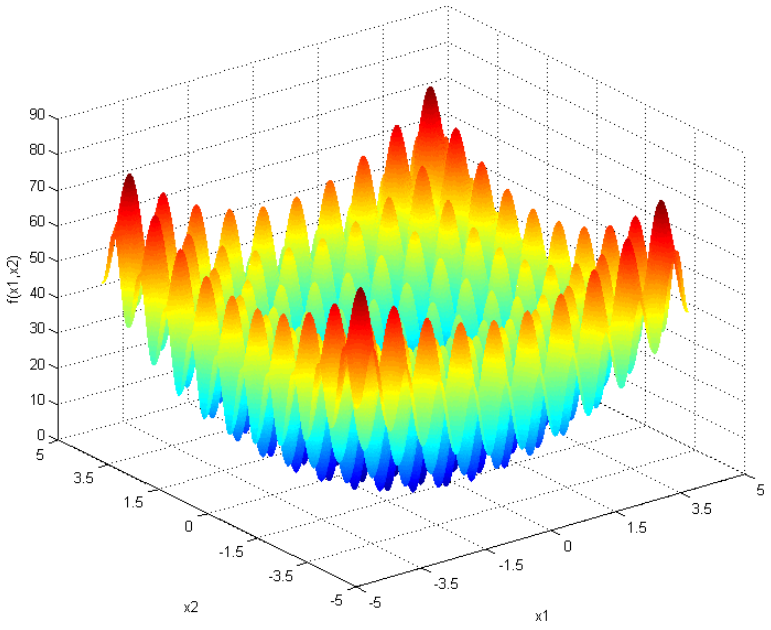


Figure A.1. Search space of Rastrigin function of two variables.

large number of local optima—incurs more complexity into search process. That is because, during the search process, an algorithm tries to escape local optima to avoid stagnation. Separability indicates the inter-dependency of genes; therefore, a separable function has no epistatic interactions between its decision variables. Consequently, an algorithm tackles each variable independently. The objective function of Rastrigin's problem is provided in (A.1).

$$f_{Ras}(\vec{x}) = 10n + \sum_{i=1}^{n} (x_i^{\,2} - 10\cos(2\pi x_i)). \tag{A.1}$$

where n is the number of variables (i.e., problem dimension) and $\vec{x}$ represents the encoded variables with each variable x_i ranges within the interval of $[-5.12,+5.12]$. The global minimum value is located at $x_{\min} = (0,...,0)$ such that $f_{Ras}(x_{\min}) = 0$.

This function is comparatively difficult because of its large search space and its large number of local minima; however these local minima are regularly distributed. Figure A.1 illustrates the search space of a two-variable f_{Ras}.

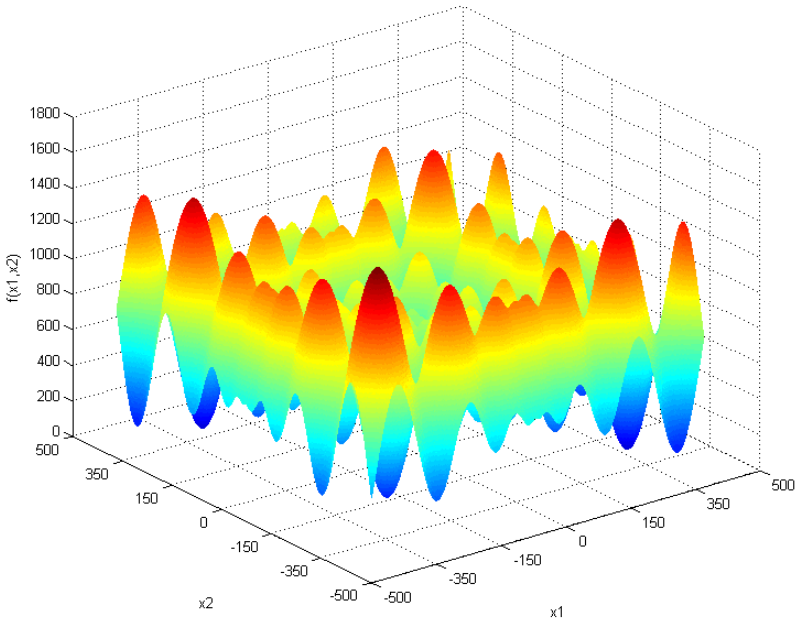


Figure A.2. Search space of Schwefel function of two variables.

Schwefel's problem (f_{Sch}):

f_{Sch} is also highly multimodal, regular, and separable function. It is characterised by its global minimum is geometrically far from the next best local minima. Consequently, it is catalogued as a difficult test function for most optimisation techniques as these may trapped in a local minimum region. Equation (A.2) illustrates the objective function of f_{Sch}.

$$f_{Sch}(\vec{x}) = 418.9829.n + \sum_{i=1}^{n} x_i.\sin(\sqrt{|x_i|}). \qquad \text{(A.2)}$$

where n is the dimension of the function. The variables $\vec{x}$ are delimited within a range of $[-500,+500]$. This function has its global minimum located at $x_{\min} = (420.9687,...,420.9687)$ and has a value $f_{Sch}(x_{\min}) = 0$. Figure A.2 shows the search space of two variables f_{Sch}.

Griewangk's problem (f_{Gri}):

f_{Gri} is a highly multimodal—has many local minima, regular, and non-separable function. Non-separable functions are highly epistatic—strong interactions between genes—

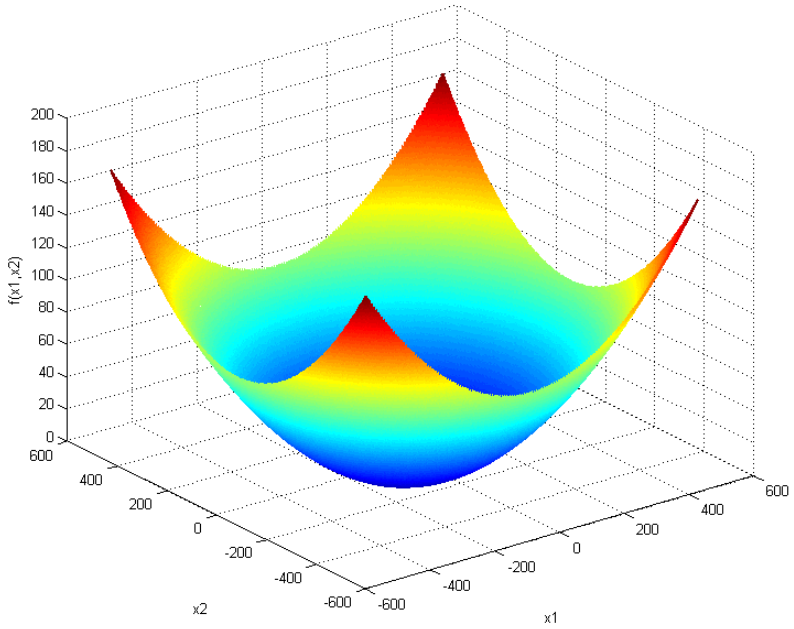


Figure A.3. Search space of Griewangk function of two variables.

and has the ability of modifying one gene by the joined effect of one or more genes. Therefore, this kind of functions is more difficult to optimise since moving from one point to another in the search space highly depends on the joint action of two or more genes. Thus, the phenotype of an individual is affected by one or more genes. The objective function of f_{Gri} is defined in (A.3).

$$f_{Gri}(\vec{x}) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right). \tag{A.3}$$

where n is the dimension of the function. The search space is delimited within a range of $[-600,+600]$ units per variable x_i. The global minimum is located at $x_{\min} = (0,...,0)$ with a value $f_{Gri}(x_{\min}) = 0$. The number of local minima grows exponentially when the number of variables increases. Figure A.3 shows f_{Gri} search space of two variables.

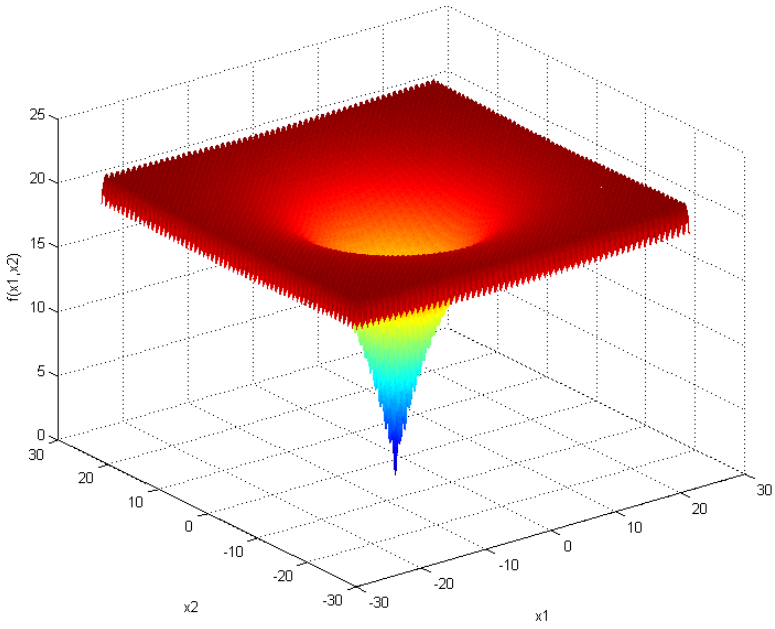


Figure A.4. Search space of Ackley function of two variables.

Ackley's problem (f_{Ack}):

f_{Ack} possesses similar characteristics as those of f_{Gri}: multimodal, regular, symmetric, and non-separable. This function has moderate complexity; however algorithms that only use the gradient steepest descent are likely be trapped in local minima areas. Therefore, the best algorithms to solve this problem should efficiently balance between the exploration and the exploitation. To solve this problem the objective function shown in (A.4) has to be minimised.

$$f_{Ack}(\vec{x}) = 20 + e - 20 \cdot \exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)). \qquad \text{(A.4)}$$

where n is the dimension of the function. The search space is delimited within the range of $[-30,+30]$ units per variable x_i, and the global minimum is located at $x_{\min} = (0,...,0)$ with a value $f_{Ack}(x_{\min}) = 0$. The search space of two variables f_{Ack} is shown in Figure A.4.

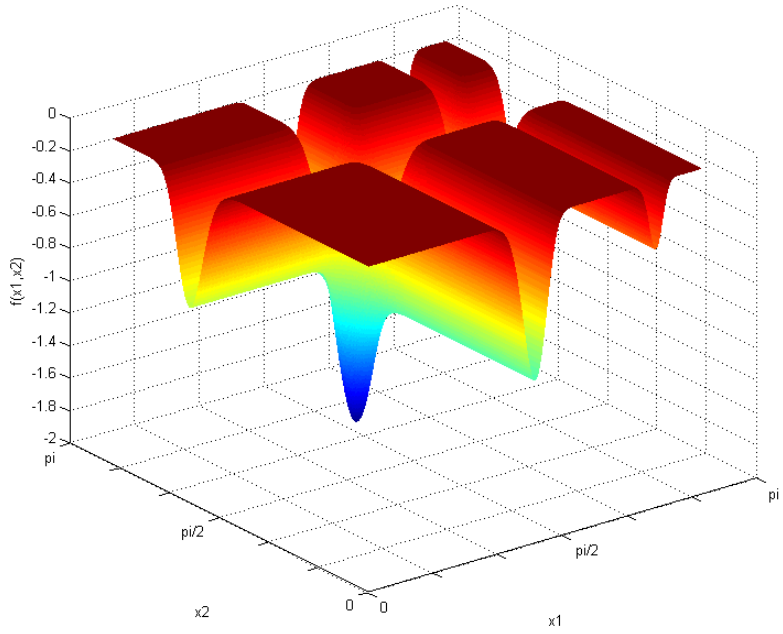


Figure A.5. Search space of Michalewicz function of two variables.

Michalewicz's problem (f_{Mic}):

The main characteristic of f_{Mic} is its asymmetry. Consequently, this function is added to the test suite in order to avoid the exploitation of the symmetry possessed by the above problems. In addition, f_{Mic} is a separable and multimodal function. The number of local minima grows, in a factorial manner, as the dimension n of the problem increases, leading to a total of $n!$ local minima. The objective function is provided in (A.5).

$$f_{Mich}(\vec{x}) = -\sum_{i=1}^{n} \sin(x_i) \cdot \left(\sin\left(\frac{i \cdot x_i^2}{\pi} \right) \right)^{2 \cdot m} . \tag{A.5}$$

where n is the dimension of the function and the parameter m defines the sharpness of valleys. Larger value of m introduces more difficulty into search process; in this research a value of $m = 10$ is used. The global minimum value is $f_{Mic}(x_{\min}) = -9.66$ for a problem's dimension $n = 10$. (The location and value of the global minimum vary according to the dimension of the problem.) The search space is delimited within the range of $[0, \pi]$ units per variable x_i. The search space of two-dimensional f_{Mic} is illustrated in Figure A.5.

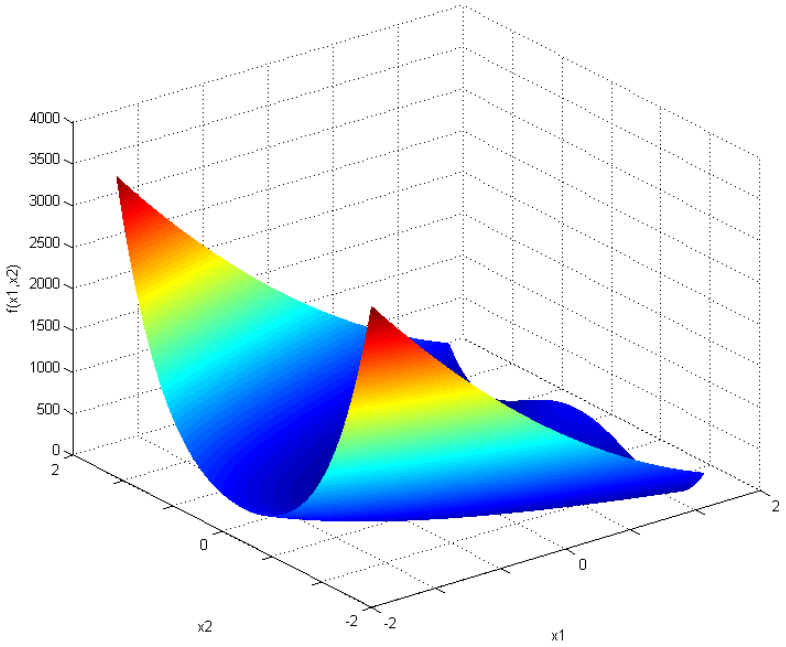


Figure A.6. Search space of Rosenbrock function of two variables.

Rosenbrock's problem (f_{Ros}):

f_{Ros} is a multimodal and non-separable test function. It is characterised as its global minimum is located inside a narrow, long, and flat valley. Although most optimisation techniques can easily locate this valley, the global minimum is difficult to reach. The objective function is shown in (A.6).

$$f_{Ros}(\vec{x}) = \sum_{i=1}^{n-1} 100 \cdot (x_{i+1} - x_i^2)^2 + (1 - x_i)^2. \qquad \text{(A.6)}$$

where n is the dimension of the function, and the global minimum is located at $x_{\min} = (1,...,1)$ with a value $f_{Ros}(x_{\min}) = 0$. The variables $\vec{x}$ range in the interval of $[-5,10]$. The visualization of the non-convex search space of Rosenbrock function is illustrated in Figure A.6. The plot focuses on the area around the global minimum for two variables.

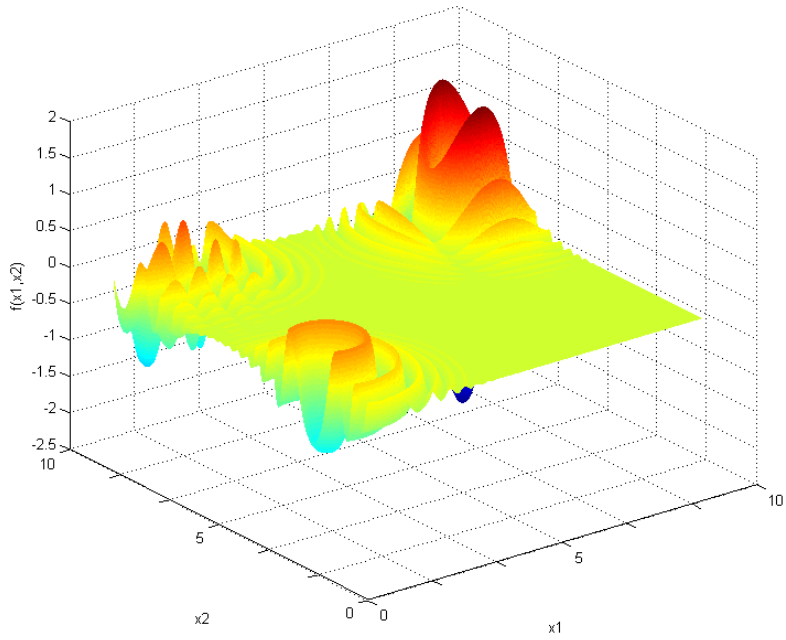


Figure A.7. Search space of Langermann function of two variables.

Langermann's problem (f_{Lang}):

f_{Lang} is also a multimodal and non-separable function; its local minima are irregularly distributed. Equation (A.7) demonstrates the objective function.

$$f_{Lang}(\vec{x}) = -\sum_{i=1}^{m} c_i e^{-\frac{1}{\pi}\sum_{j=1}^{n}(x_i - a_{ij}^2)} \cos\left(\pi \sum_{j=1}^{n}(x_j - a_{ij})^2\right). \tag{A.7}$$

where n is the dimension of the function. The values of vector C (c_i; $i = 1,...,m$) and matrix A (a_{ij}; $j = 1,...,n$; $i = 1,...,m$) are randomly generated in order to obtain a random distribution of the minima. However, in this research these values were constant numbers fixed in advance from (Bersini *et al.*, 1996) for $m = 5$. The variables $\vec{x}$ range in the interval of $[0,10]$. The global minimum value varies and depends on vector C and matrix A. In this research, the global minimum value is $f_{Lang}(x_{\min}) = -1.49$. Figure A.7 shows the search space of the f_{Lang} in 2D.

A.2 Real-World Problems

This section describes the benchmark problems selected from the real world: *frequency modulation sound parameter identification* (f_{FMS}) and *systems of linear equations* (f_{SLE}) (Alba and Dorronsoro, 2008). The third real-world problem considered in this research (*GPS attitude determination*— f_{GPS}) was introduced in Chapter 4, Section 4.2.2.

Frequency modulation sound parameter identification (f_{FMS}):

In this problem, six parameters must be determined $\vec{x} = (a_1, \omega_1, a_2, \omega_2, a_3, \omega_3)$ of the frequency modulation sound model represented by (A.8) in order to approximate it to the sound wave represented by (A.9) with $\theta = 2 \cdot \pi / 100$. The parameters range within the interval of $[-6.4, +6.35]$.

$$y(t) = a_1 \cdot \sin(\omega_1 \cdot t \cdot \theta + a_2 \cdot \sin(\omega_2 \cdot t \cdot \theta + a_3 \cdot \sin(\omega_3 \cdot t \cdot \theta))). \tag{A.8}$$

$$y_0(t) = 1.0 \cdot \sin(5.0 \cdot t \cdot \theta - 1.5 \cdot \sin(4.8 \cdot t \cdot \theta + 2.0 \cdot \sin(4.9 \cdot t \cdot \theta))). \tag{A.9}$$

The fitness function is defined as the summation of square errors represented by (A.10). This problem is a highly complex multimodal one, non-symmetric, and with strong epistasis. The optimum minimum value is $f_{FMS}(x_{\min}) = 0.0$.

$$f_{FMS}(\vec{x}) = \sum_{t=0}^{100} \left(y(t) - y_0(t)\right)^2. \tag{A.10}$$

Systems of linear equations (f_{SLE}):

In this problem, ten parameters of a vector $\vec{x}$ are to be determined such that $A \cdot \vec{x} = \vec{b}$, in order to minimise the objective function represented by (A.11). The global minimum value is $f_{SLE}(x_{\min}) = 0.0$. The matrix A and the vector $\vec{b}$ are given by (A.12), and the ten parameters are in the range $[-9.0, +11.0]$.

$$f_{SLE}(\vec{x}) = \left| \sum_{i=1}^{n} \sum_{j=1}^{n} (a_{ij} \cdot x_j) - b_i \right|. \tag{A.11}$$

$$A = \begin{pmatrix} 5,4,5,2,9,5,4,2,3,1 \\ 9,7,1,1,7,2,2,6,6,9 \\ 3,1,8,6,9,7,4,2,1,6 \\ 8,3,7,3,7,5,3,9,9,5 \\ 9,5,1,6,3,4,2,3,3,9 \\ 1,2,3,1,7,6,6,3,3,3 \\ 1,5,7,8,1,4,7,8,4,8 \\ 9,3,8,6,3,4,7,1,8,1 \\ 8,2,8,5,3,8,7,2,7,5 \\ 2,1,2,2,9,8,7,4,4,1 \end{pmatrix}; \quad \vec{b} = \begin{pmatrix} 40 \\ 50 \\ 47 \\ 59 \\ 45 \\ 35 \\ 53 \\ 50 \\ 55 \\ 40 \end{pmatrix}. \tag{A.12}$$

f_{SLE} is a complex and quite difficult real-world problem, with inter-parameter linkage (i.e., non-separable). This problem is unlikely to be used to assess the performance of GAs as these techniques are not the most suitable to solve this problem. However, some authors still believe in using this problem to evaluate GAs (Herrera and Lozano, 2000; Alba and Dorronsoro, 2008; El–Emary and El–Kareem, 2008).

Appendix B

Extended Experimental Results

In order to make this research comprehensive, this appendix is added to provide extended and preliminary results that may be useful. These results were omitted from main chapters in order to avoid reader distraction. Section B.1 gives the entire results from the experiment in Chapter 3; some of these results were omitted as either were not desirable (i.e., had very low search success rate) or had no significant influences on the analysis. This section (B.1) aims to support the decision made about what to consider or to disregard. The remaining sections (B.2 and B.3) give the results from preliminary experiments that are related to the selection of a single threshold for adaptive algorithms proposed in Chapter 5. These sections aim to give detailed justification on how to choose a single value for all problems under consideration (recall that there is no one best value for all problems). Section B.2 provides the results and criteria on which the selection is made for Diversity-Guided 3D-cGA (refer to Section 5.2), while those for Convergence-Speed-Guided 3D-cGA and Dynamic 3D-cGA based on (Alba and Dorronsoro, 2005) are given in Section B.3 (refer to Section 5.3).

B.1 Comparison of 3D and 2D cGAs

In Chapter 3, 3D-cGAs were compared to 2D-cGAs; part of the results were omitted such as search success rates for f_{Ras}, f_{Sch}, and f_{Ack} as they were either similar or only differed slightly. Other results omitted from Chapter 3 relate to those obtained for f_{SLE} as they had undesirable performance. In this section, these results are included in order to show the slight improvements achieved. Table B.1 summarises all results.

Various configurations concerning population and neighbourhood sizes were defined for both algorithms (i.e., 2D and 3D cGAs); the parameters used were summarised in Table 3.1. Various population sizes were selected in order to introduce similar number of individuals for both grid topologies (i.e., 2D and 3D grids). For more details, refer to Chapter 3.

Table B.1. Convergence time (CT) and rate (CR)* obtained by 3D-cGA and 2D-cGA for various population sizes and step distances (r)

Population size/ Problem	5×5 3×3×3	8×8 4×4×4		11×11 5×5×5		15×15 6×6×6		19×19 7×7×7	
	r =1	r =1	r =3	r =1	r =3	r =1	r =3	r =1	r =3
f_{Ras}	492.61 ± 39.5 96%	416.24 ± 26.0 100%	381.93 ± 35.5 100%	398.31 ± 22.0 100%	358.85 ± 29.5 100%	401.52 ± 19.5 100%	354.38 ± 30.5 100%	406.57 ± 18.5 100%	360.42 ± 20.0 100%
	685.22 ± 102.5 100%	450.46 ± 46.0 100%	–	377.12 ± 35.0 100%	356.21 ± 35.5 100%	366.02 ± 27.5 100%	350.72 ± 20.0 100%	362.12 ± 27.0 100%	345.17 ± 29.5 100%
f_{Sch}	302.89 ± 32.0 79%	245.01 ± 21.0 99%	215.15 ± 18.0 99%	234.02 ± 16.0 100%	203.03 ± 16.0 100%	226.77 ± 13.5 100%	193.00 ± 13.5 100%	231.13 ± 10.5 100%	190.99 ± 12.5 100%
	258.46 ± 33.5 84%	227.97 ± 21.0 99%	–	212.69 ± 17.5 100%	200.53 ± 16.5 100%	205.28 ± 13.5 100%	189.09 ± 15.5 100%	199.87 ± 12.5 100%	181.9 ± 14.5 100%
f_{Grie}	448.00 ± 0.00 1%	511.11 ± 48.0 9%	431.90 ± 2.50 10%	408.63 ± 82.0 11%	324.50 ± 39.0 18%	604.87 ± 116.0 31%	366.92 ± 23.0 39%	489.78 ± 53.0 42%	353.35 ± 30.0 53%
	386.50 ± 2.00 4%	475.15 ± 75.0 13%	–	345.36 ± 49.0 19%	300.40 ± 28.5 22%	426.47 ± 38.0 34%	331.86 ± 33.5 44%	303.54 ± 29.0 51%	298.52 ± 22.0 57%
f_{Ack}	343.21 ± 23.0 100%	263.41 ± 5.0 100%	216.96 ± 5.0 100%	261.78 ± 4.0 100%	210.79 ± 3.5 100%	259.01 ± 3.5 100%	207.63 ± 2.0 100%	258.83 ± 2.5 100%	209.01 ± 2.0 100%
	275.82 ± 18.0 100%	230.56 ± 5.5 100%	–	225.07 ± 3.0 100%	211.36 ± 3.0 100%	223.61 ± 2.0 100%	204.59 ± 2.0 100%	222.01 ± 2.0 100%	200.17 ± 2.0 100%
f_{Mic}	445.62 ± 51.0 16%	481.00 ± 66.0 33%	444.17 ± 48.0 41%	488.79 ± 51.0 43%	446.15 ± 51.5 60%	548.60 ± 58.0 75%	442.92 ± 45.0 81%	560.44 ± 43.0 89%	461.78 ± 43.0 84%
	419.31 ± 90.0 19%	435.05 ± 46.0 35%	–	453.83 ± 55.5 54%	437.85 ± 47.0 56%	457.67 ± 44.0 65%	433.05 ± 41.0 72%	467.40 ± 46.0 80%	433.13 ± 50.5 82%
f_{Lang}	260.50 ± 47.0 4%	292.57 ± 49.0 28%	193.80 ± 24.0 35%	358.21 ± 33.5 65%	221.68 ± 25.5 64%	361.56 ± 60.0 93%	213.16 ± 23.0 89%	327.44 ± 26.0 97%	205.47 ± 16.0 96%
	205.87 ± 29.5 8%	226.33 ± 30.5 30%	–	251.94 ± 25.0 76%	218.30 ± 20.5 62%	228.19 ± 22.0 91%	204.01 ± 25.0 85%	239.83 ± 22.0 99%	186.83 ± 15.0 99%
f_{FMS}	602.40 ± 198.0 25%	264.38 ± 57.5 44%	196.14 ± 35.5 42%	258.64 ± 37.0 62%	170.58 ± 19.0 65%	270.62 ± 32.0 87%	190.07 ± 18.0 77%	296.11 ± 25.0 96%	209.97 ± 22.0 89%
	442.93 ± 202.0 33%	208.95 ± 37.0 49%	–	207.69 ± 29.0 63%	186.75 ± 23.0 68%	202.41 ± 17.0 81%	184.06 ± 21.5 72%	220.63 ± 22.0 92%	197.77 ± 21.5 80%
f_{SLE}	0%	0%	0%	0%	382.00 ± 62.0 2%	662.00 ± 0.00 1%	657.50 ± 106.5 2%	823.00 ± 78.0 2%	625.50 ± 89.5 6%
	0%	0%	–	891.00 ± 0.00 1%	556.50 ± 173.5 2%	744.00 ± 0.00 1%	668.85 ± 167.0 7%	752.16 ± 14.0 6%	567.66 ± 37.0 6%

* For more details about the performance measures, please refer to Section 2.2.3.1.

Note: For each problem, the results obtained by 2D-cGA are shown above the results obtained by 3D-cGA. The symbol '–' means that the corresponding algorithm configuration has not been evaluated.

For each problem studied, convergence time and convergence rate obtained for 2D-cGAs and 3D-cGAs of various configurations are given in Table B.1. The discussion and analysis of results were provided in Chapter 3.

B.2. Selection of γ Diversity-Guided 3D-cGA

This section studies the behaviour of the adaptive criterion under different γ values (refer to Chapter 5, Section 5.2). Four γ values: 0.3, 0.35, 0.4, and 0.45, which represented high to low restrictive conditions to reduce the selection rate were tested. The purpose is to select single γ value for all of the problems considered.

Table B.2. Experimental Results: Convergence time (CT) and rate (CR)[*] obtained by Diversity-Guided 3D-cGAs for various thresholds (γ), and $7\times7\times7$ grid

Problem	γ = 0.3	γ = 0.35	γ = 0.4	γ = 0.45
f_{Ras}	678.22 ± 65.5 **100%**	683.94 ± 54.5 **100%**	**641.72 ± 55.0** **100%**	675.61 ± 48.5 **100%**
f_{Sch}	1259.2 ± 97.5 **100%**	1247.3 ± 100.0 **100%**	1209.9 ± 116.0 **100%**	**1207.3 ± 126.0** **100%**
f_{Ack}	1854.9 ± 70.0 70%	**1850.4 ± 72.00** 76%	1897.4 ± 52.0 **83%**	1872.2 ± 50.0 79%
f_{Ros}	908.47 ± 534.0 42%	**861.21 ± 603.0** 41%	881.7 ± 636.5 **50%**	1010.7 ± 470.5 44%
f_{Mic}	712.63 ± 32.5 **100%**	664.62 ± 40.5 **100%**	628.30 ± 43.0 **100%**	**599.69 ± 47.5** **100%**
f_{Lang}	340.57 ± 25.5 84%	346.01 ± 16.0 94%	**308.45 ± 14.5** **96%**	320.13 ± 23.0 92%
f_{FMS}	1386.1 ± 278.0 68%	1396.7 ± 250.0 61%	**1294.7 ± 381.5** **100%**	1337.6 ± 267.5 70%
f_{SLE}	345.91 ± 32.0 37%	**320.97 ± 33.5** 34%	341.48 ± 35.0 **39%**	323.33 ± 31.0 21%

Table B.3. Local and global[†] ranking of γ values based on two performance metrics

Convergence time	Convergence rate	γ	Sum	Rank
1. 0.35	**1.** 0.40	0.30	6	4
1. 0.40	**2.** 0.30	0.35	3	2
3. 0.45	**2.** 0.35	0.40	2	1
4. 0.30	**2.** 0.45	0.45	5	3

[*] For more details about the performance measures, please refer to Section 2.2.3.1.

[†] Local ranking is performed for each performance metric independently. Local ranks are marked in **bold** in the first two columns. Global ranking is performed for all performance metrics and is shown in the last column. It is computed by summing the local ranks for each γ. Sum of local ranking values are shown in the column prior to the last.

Table B.2 shows the results obtained. For each problem, the best results are marked in **bold**. In order to select one γ value two-level ranking was performed (see Table B.3). In the first level—local ranking, γ values were ranked based on convergence time and rate independently. The value of γ that resulted in the lowest convergence time for most cases was assigned the highest rank (i.e., the smallest number) (column 1), and so forth. Similarly, γ value that resulted in the highest convergence rate for most cases was assigned the highest rank (column 2) , and so forth. In the second level—global ranking, γ values were ranked based on their local ranks; the value of γ that resulted in the minimum sum of local ranks was assigned the highest rank (the last column). Consequently, the best γ value was 0.4.

Table B.4. Experimental Results: Convergence time (CT), rate (CR), and speed (SP)* obtained by Diversity-Guided 3D-cGAs for various thresholds (γ), and $6\times6\times6$ grid

Problem	γ = 0.05	γ = 0.15	γ = 0.25	γ = 0.30	γ = 0.40
f_{Ras}	576.32 ± 63.5 **100%** **36.34 ± 4.01**	581.6 ± 54.5 **100%** 37.31 ± 3.5	586.86 ± 51.0 **100%** 38.01 ± 3.12	576.91 ± 52.5 **100%** 37.49 ± 3.74	**570.83 ± 49.5** **100%** 36.80 ± 3.23
f_{Ack}	1430.4 ± 173.5 **100%** 90.93 ± 10.96	1406.7 ± 188.5 **100%** 90.53 ± 12.23	1411.8 ± 163.5 **100%** 90.29 ± 10.5	1457.5 ± 180.0 **100%** 95.69 ± 12.6	**1399.0 ± 190.0** **100%** **89.58 ± 12.34**
f_{Ros}	632.45 ± 159.0 42% 40.87 ± 10.2	510.66 ± 281.0 48% 33.25 ± 18.03	452.34 ± 254.5 46% 29.18 ± 16.13	497.74 ± 325.5 50% 31.57 ± 20.64	**445.87 ± 256** **55%** **27.83 ± 15.9**
f_{FMS}	**872.55 ± 314.0** **58%** **80.02 ± 28.81**	899.31 ± 261.0 48% 82.85 ± 24.35	1001.6 ± 248.5 52% 91.54 ± 23.41	979.00 ± 213.0 49% 89.95 ± 19.57	981.65 ± 336.5 52% 89.77 ± 31.91
f_{SLE}	365.8 ± 42.0 15% 24.34 ± 2.80	321.07 ± 33.0 13% 21.74 ± 2.56	338.18 ± 51.0 **16%** 22.75 ± 3.64	**291.40 ± 14.5** 10% **19.61 ± 0.96**	300.80 ± 36.0 10% 20.28 ± 2.56
f_{GPS}	125.58 ± 6.00 99% 2.45 ± 0.109	102.40 ± 7.00 99% 2.00 ± 0.14	96.45 ± 6.00 99% 1.91 ± 0.109	**96.06 ± 7.00** 99% 1.89 ± 0.15	96.26 ± 6.5 **100%** **1.88 ± 0.11**

Table B.5. Local and global† ranking of γ values based on three performance metrics

Convergence time	Convergence Rate	Convergence speed	γ	Sum	Rank
1. 0.40	**1.** 0.40	**1.** 0.40	0.05	7	2
2. 0.30	**2.** 0.25	**2.** 0.05	0.15	12	5
3. 0.05	**2.** 0.05	**3.** 0.30	0.25	10	4
4. 0.15	**4.** 0.15	**4.** 0.15	0.30	9	3
4. 0.25	**4.** 0.30	**4.** 0.25	0.40	3	1

* For more details about the performance measures, please refer to Section 2.2.3.1.

† Local ranking is performed for each performance metric independently. Local ranks are marked in **bold** in the first three columns. Global ranking is performed for all performance metrics and is shown in the last column. It is computed by summing the local ranks for each γ. Sum of local ranking values are shown in the column prior to the last.

In Chapter 5, Diversity-Guided 3D-cGA was compared with other algorithms. To perform fairness comparison, similar parameters for algorithms compared were used; a size of 216 individuals arranged as $6\times6\times6$ was used. Consequently, experiments to select single γ value were repeated for the reduced size of population. Population size can significantly influence algorithm performance; larger populations offer more diversity and therefore promote more exploration.

Table B.4 shows the results. Values of γ included: 0.05, 0.15, 0.25, 0.3, and 0.4. The best results achieved for each problem are marked in **bold**. Two-level ranking was performed. Locally, γ values that achieved the lowest convergence time, the highest rate, and the fastest convergence in parallel was assigned the highest rank (Table B.5, columns 1, 2, and 3, respectively). Globally, the highest rank was assigned to the γ that resulted in the best overall performance (the last column). Consequently, the best γ value is 0.4.

B.3. Selection of ε for Convergence-Speed-Guided 3D-cGA

This section examines the performance of the adaptive criteria defined for the Convergence-Speed-Guided 3D-cGA and the Dynamic 3D-cGA based on (Alba and Dorronsoro, 2005) against different ε values to facilitate the selection of single ε value for all the considered problems (refer to Chapter 5, Section 5.3).

Table B.6. Experimental Results: Convergence time (CT), rate (CR), and speed (SP)* obtained by Convergence-Speed-Guided 3D-cGAs for various thresholds (ε)

Problems	ε = 0.3	ε = 0.25	ε = 0.15	ε = 0.05
f_{Ras}	755.55 ±46.5 **100%** 53.11 ±3.53	777.21 ±70.0 99% 51.61 ±4.45	**735.97 ±61.0** **100%** **50.86 ±3.67**	752.89 ±65.5 **100%** 51.25 ±4.61
f_{Ack}	1609.0 ±153.5 **100%** 106.25 ±9.44	**1519.6 ±144.5** **100%** 106.83 ±9.69	1566.7 ±141.0 99% **105.29 ±9.76**	1598.1 ±143.0 99% 117.7 ±10.3
f_{Ros}	**599.70 ±235.0** 10% **40.15 ±15.46**	686.75 ±135.0 16% 46.54 ±9.41	692.69 ±203.0 13% 46.73 ±11.82	661.68 ±192.0 **22%** 45.39 ±13.05
f_{FMS}	1161.8 ±308.0 **69%** 108.86 ±29.92	1167.3 ±272.0 64% 109.63 ±25.46	1127.7 ±300.0 68% 105.93 ±28.35	**944.53 ±380.0** 54% **90.65 ±35.3**
f_{SLE}	515.08 ±50.0 24% 34.33 ± 3.35	522.95 ±66.0 23% 36.33 ±4.54	**498.95 ±67.0** 20% **33.80 ±4.44**	535.46 ±99.0 **26%** 37.91 ±7.02
f_{GPS}	106.70 ±14.0 99% 2.02 ±0.281	95.77 ±10.5 96% 1.82 ±0.203	102.15 ±13.0 98% 1.84 ±0.226	**93.02 ±9.00** **100%** **1.76 ±0.16**

* For more details about the performance measures, please refer to Section 2.2.3.1.

Table B.7. Local and global* ranking of ε values based on three performance metrics

Convergence time	Convergence rate	Convergence speed	ε	Sum	Rank
1. 0.05	**1.** 0.05	**1.** 0.15	0.05	4	1
1. 0.15	**2.** 0.30	**2.** 0.05	0.15	5	2
3. 0.25	**3.** 0.15	**3.** 0.30	0.25	10	4
3. 0.30	**3.** 0.25	**4.** 0.25	0.30	8	3

Four ε values—0.3, 0.25, 0.15, and 0.05—were assessed, representing low to high restrictive adaptive conditions. Table B.6 shows the results obtained. The best results for each problem are marked in **bold**. To select single ε value, two-level ranking was performed based on convergence time, rate, and speed. Locally, the highest rank was assigned to ε value that achieved the lowest convergence time, the highest rate, and fastest convergence in parallel (Table B.7, columns 1, 2, and 3, respectively). Globally, the highest rank was assigned to ε value that achieved the best overall performance (Table B.7, the last column). Consequently, the best ε value is 0.05.

Similarly, the same ε values were tested for Dynamic 3D-cGA based on (Alba and Dorronsoro, 2005). Table B.8 shows the results obtained. For each problem considered, the best results achieved are marked in **bold**.

Table B.8. Experimental Results: Convergence time (CT), rate (CR), and speed (SP)† obtained by Dynamic 3D-cGAs based on (Alba and Dorronsoro, 2005) for various thresholds (ε)

Problems	ε = 0.3	ε = 0.25	ε = 0.15	ε = 0.05
f_{Ras}	547.27 ±61.0 **100%** 37.51 ±3.91	564.64 ±59.5 **100%** 40.05 ±4.32	544.21 ±51.5 **100%** **36.07 ±3.19**	**541.43 ±99.5** **100%** 37.68 ±4.10
f_{Ack}	**1298.4 ±200.0** **100%** 88.27 ±13.5	1399.4 ±203.0 **100%** 93.92 ±13.18	1301.6 ±195.0 **100%** **87.97 ±12.94**	1337.5 ±189.5 **100%** 100.54 ±13.76
f_{Ros}	998.0 ±0.0 1% **66.25 ±0.0**	**995.00 ±2.0** **2%** 66.82 ±0.23	0%	0%
f_{FMS}	905.38 ±340.0 50% 86.25 ±31.94	880.08 ±261.0 50% 83.35 ±24.76	**839.90 ±273.0** 54% **79.34 ±26.63**	981.41 ±342.0 **72%** 94.41 ±32.5
f_{SLE}	**209.33 ±42.0** 3% **14.75 ±1.68**	219.3 ±23.5 10% 15.43 ±1.46	216.81 ±43.0 **11%** 15.14 ±3.0	228.2 ±3.0 5% 16.27 ±0.28
f_{GPS}	**70.80 ±7.0** 97% **1.37 ±0.125**	72.65 ±9.0 **100%** 1.42 ±0.156	72.17 ±7.0 98% 1.42 ±0.12	71.79 ±8.0 **100%** 1.40 ±0.15

* Local ranking is performed for each performance metric independently (marked in **bold**). Global ranking is performed for all performance metrics (last column). It is computed by summing the local ranks for each ε.

† For more details about the performance measures, please refer to Section 2.2.3.1.

Table B.9. Local and global* rankings of ε values based on three performance metrics

Convergence time	Convergence rate	Convergence speed	ε	Sum	Rank
2. 0.05	**3.** 0.15	**1.** 0.30	0.05	6	1
2. 0.15	**1.** 0.25	**1.** 0.15	0.15	6	1
2. 0.25	**4.** 0.30	**3.** 0.05	0.25	6	1
1. 0.30	**1.** 0.05	**3.** 0.25	0.30	6	1

* Local ranking is performed for each performance metric independently (marked in **bold**). Global ranking is performed for all performance metrics (last column). It is computed by summing the local ranks for each ε.

To choose single ε value, two-level ranking was performed based on performance metrics: convergence time, rate, and speed. Locally, the highest rank was allocated to ε value that resulted in: the lowest convergence time, the highest convergence rate, and the fastest convergence in parallel (Table B.9, columns 1, 2, and 3, respectively). Globally, the highest (or final) rank was assigned to ε value that obtained the best overall performance (Table B.9, the last column). From Table B.9, it can be seen that all of the ε values tested obtained similar global ranks. A value of $\varepsilon = 0.05$ that represented the most restrictive condition was selected for Convergence-Speed-Guided 3D-cGA (refer to Table B.9). In addition, for Dynamic 3D-cGA based on (Alba and Dorronsoro, 2005), a similar ε value—0.05—was selected.

References

Alba, E. (2002). Parallel Evolutionary Algorithms Can Achieve Super-Linear Performance. *In Information Processing Letters*, 82(1), pp. 7–13. Elsevier.

Alba, E. (2005). *Parallel Metaheuristics a New Class of Algorithms*. John Wiley & Sons Publication. 978-0-471-67806-9.

Alba, E. and Cotta, C. (2006). Evolutionary Algorithms. *In Handbook of Bioinspired Algorithms and Applications*, Olariu, S. and Zomaya, A. Y., eds. pp. 3–19. Chapman & Hall/CRC.

Alba, E., Cotte, C., and Troya, J. M. (1999a). Numerical and Real Time Analysis of Parallel Distributed GAs with Structured and Panmictic Populations. *In IEEE Transactions on Evolutionary Computation*, pp. 1019–1026. IEEE.

Alba, E., Cotte, C., and Troya, J. M. (1999b). Entropic and Real-Time Analysis of the Search with Pancmictic, Structured, and Parallel Distributed Genetic Algorithms. *In LCC Technical Report ITI 99-7*, pp. 1–9. Universidad de Malaga.

Alba, E. and Dorronsoro, B. (2005). The Exploration/Exploitation Tradeoff in Dynamic Cellular Genetic Algorithms. *IEEE Transaction on Evolutionary Computation*, 9(2), pp. 126–142. IEEE.

Alba, E. and Dorronsoro, B. (2008). *Cellular Genetic Algorithms*. New York: Springer Sciences + Business Media. 978-0-387-77609-5.

Alba, E., Giacobini, M., Tomassini, M., and Romero, S. (2002). Comparing Synchronous and Asynchronous Cellular Genetic Algorithms. *In Proceedings of the Parallel Problem*

Solving from Nature–PPSN VII, Granada, Spain, LNCS 2439, pp. 601–610. Springer-Verlag.

Alba, E., Luna, F., and Nebro, A. (2004). Advances in Parallel Heterogeneous Genetic Algorithms for Continues Optimization. *International Journal of Applied Mathematics and Computer Science*, 14(3), pp. 101–117. AMCS.

Alba, E., Nebro, A., and Troya, J. (2002). Heterogeneous Computing and Parallel Genetic Algorithms. *Journal of Parallel and Distributed Computing*, 62(9), pp. 1362–1385. Elsevier.

Alba, E. and Tomassini, M. (2002). Parallelism and Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5), pp.443–462. IEEE.

Alba, E. and Troya, J. M. (1999a). A Survey of Parallel Distributed Genetic Algorithms. *Complexity*, 4(4), pp. 31–52. John Wiley & Sons.

Alba, E. and Troya, J. M. (1999b). An Analysis of Synchronous and Asynchronous Parallel Distributed Genetic Algorithms with Structured and Panmictic Islands. *In Proceedings of the 11th IPPS/SPDP99 Workshops*, pp. 248–256.

Alba E. and Troya, J. M. (2000). Cellular Evolutionary Algorithms: Evaluating the Influence of Ratio. *In proceedings of the 6th International Conference on Parallel Problem Solving from Nature–PPSN VI*, pp. 29–38. Springer-Verlag.

Alba, E. and Troya, J. M. (2001). Analyzing Synchronous and Asynchronous Parallel Distributed Genetic Algorithms. *Future Generation Computer Systems*, 17(2001), pp. 451–465. Elsevier.

Alba, E. and Troya, J. M. (2002). Improving Flexibility and Efficiency by Adding Parallelism to Genetic Algorithms. *Statistics and Computing*, 12(2), pp. 91–114. Kluwer Academic Publishers.

Al-Naqi, A., Erdogan, A.T., and Arslan, T. (2010a). Fault Tolerance through Automatic Cell Isolation Using Three-Dimensional Cellular Genetic Algorithms. *In Proceedings of the IEEE Congress on Evolutionary Computation (CEC '10)*, Barcelona, Spain, pp. 1–8. IEEE.

Al-Naqi, A., Erdogan, A.T., and Arslan, T. (2010b). Balancing Exploration and Exploitation in Adaptive Three-Dimensional Cellular Genetic Algorithm via Probabilistic Selection Operator. *In Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS '10)*, Anaheim, California, USA, pp. 258–264. IEEE.

Al-Naqi, A., Erdogan, A.T., and Arslan, T. (2011a). Fault Tolerant Three-Dimensional Cellular Genetic Algorithms with Adaptive Migration Schemes. *In Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS '11)*, San Diego, California, USA, pp. 352–359. IEEE.

Al-Naqi, A., Erdogan, A.T., and Arslan, T. (2012). Adaptive Three-Dimensional Cellular Genetic Algorithm for Fine Balancing Exploration and Exploitation Processes. *Special Issue on Bio-inspired Algorithms with Structured Populations–Soft Computing Journal* (submitted).

Asenek, A.V., Underwood, C.I., and Oldffield, M.K. (1997). Predicting the Rate and Effects of Single Event Upsets on Satellite Application Software using a Microprocessor Simulator. *BMUS*, The Microsatellite Baumanetz, [online] Available at: <http://microsat.sm.bmstu.ru/e-library/ccdh/eccdh.htmv>.

Avizienis, A. (1971). Fault Tolerant Computing: An Overview. *IEEE Transactions of computer*, 4(1), pp. 5–8. IEEE.

Back, T. (1996). *Evolutionary Algorithms in Theory and Practice*. Oxford University Press. 978-0-19-509971-3.

Back, Th. And Breukelaar, R. (2005). Using Genetic Algorithms to Evolve Behaviour in Cellular Automata. *In Proceedings of the 4th International Conference on Unconventional Computation*, LNCS 3699, pp. 1–10. Springer-Verlag.

Baluja, S. (1993). Structure and Performance of Fine-Grain Parallelism in Genetic Search. *In Proceedings of 5th International Conference on Genetic Algorithms–ICGA93*, pp. 155–162. Morgan Kaufmann.

Borkar, S. (2011). 3D Integration for Energy Efficient System Design. *In Proceedings of Design Automated Conference–DAC'11*, San Diego, California, USA, 5–10 June, pp. 214–219. IEEE.

Breukelaar, R. and Back, Th. (2005). Using a Genetic Algorithm to Evolve Behaviour in Multi Dimensional Cellular Automata. *In Proceedings of GECCO'05*, Washington, DC, USA, 25–29 June 2005, pp. 107–114. ACM.

Cantu-Paz, E. (1995). A Summary of Research on Parallel Genetic Algorithms. *In IlliGAL report no. 95007*, pp. 1–17. University of Illinois at Urbana-Champaign.

Cantu-Paz, E. (1999a). Topologies, Migration Rates, and Multi-Population Parallel Genetic Algorithms. *In proceedings of the 1999 Genetic and Evolutionary Computation Conference*. San Mateo, California, USA, 13–17 July 1999, pp. 91–98. Morgan Kaufmann.

Cantu-Paz, E. (1999b). Migration Policies and Takeover Times in Parallel Genetic Algorithms. *In IlliGAL report no. 99008*, pp. 1–11. University of Illinois at Urbana-Champaign.

Cantu-Paz, E. (2000). *Efficient and Accurate Parallel Genetic Algorithms*. Boston/Dordrecht/London: Kluwer Academic Publisher. 978-0-7923-7221-9.

Cantu-Paz, E. (2001). Migration Policies, Selection Pressure, and Parallel Evolutionary Algorithms. *In Journal of Heuristics*, 7(4), pp. 311–334. Kluwer Academic Publisher.

Cantu-Paz, E. and Goldberg, D. E. (1999). On the Scalability of Parallel Genetic Algorithms. *In Evolutionary Computation*, 7(4), pp. 429– 449. MIT Press.

Capcarrere, M., Tomassini, M., Tettamanzi, A., and Sipper, M. (1999). A Statistical Study of a Class of Cellular Evolutionary Algorithms. *Evolutionary Computation*, 7(3), pp. 255–274. MIT Press.

Culler, D. E., Singh, J. P., and Gupta, A. (1998). *Parallel Computer Architecture: A Hardware/Software Approach*. USA: Morgan Kaufmann. 978-1558603433.

Das, S., Chandrakasan, A., and Reif, R. (2003). Three-Dimensional Integrated Circuits: Performance, Design Methodology, and CAD Tools. *In Proceedings of IEEE Computer Society Annual Symposium on VLSI*, Tampa, Florida, USA, 20–21 Feb., pp. 13–18. IEEE.

De Jong, K. and Sarma, J. (1995). On Decentralizing Selection Algorithms. *In Proceedings of the Sixth International Conference on Genetic Algorithms*, Pittsburgh, PA, pp. 17–23. Morgan Kaufmann.

Dorronsoro, B. and Alba, E. (2006). A Simple Cellular Genetic Algorithm for Continuous Optimization. *In Proceedings of Congress on Evolutionary Computation –CEC'06*, 16-21 July 2006, pp. 2838–2844. IEEE.

Dorronsoro, B., Alba, E., Giacobini, M., and Tomassini, M. (2004). The Influence of Grid Shape and Asynchronicity on Cellular Evolutionary Algorithms. *In Proceedings of Congress on Evolutionary Computation –CEC'04*, 19-23 June 2004, pp. 2152–2158. IEEE.

Eklund, S. E. (2004). A Massively Parallel Architecture for Distributed Genetic Algorithms. *Parallel Computing*, 30(2004), pp. 647–676. Elsevier.

El-Emary, I. M. M. and Abd El-Kareem, M. M. (2008). Towards Using Genetic Algorithm for Solving Nonlinear Equation Systems. *World Applied Sciences Journal*, 5(3), pp. 282–289. IDOSI Publications.

GEATbx (2005). GEATbx Examples: Examples of Objective Functions. [online] *Genetic and Evolutionary Algorithm Toolbox for use with Matlab*, November 2005, Available at: < http://www.geatbx.com >.

Giacobini, M., Alba, E., and Tomassini, M. (2003). Selection Intensity in Asynchronous Cellular Evolutionary Algorithms. *In Proceedings of the Genetic and Evolutionary Computation Conference –GECCO'03*, pp. 955–966. Springer-Verlag.

Giacobini, M., Tomassini, M., Tettamanzi, A. G. B., and Alba, E. (2005). Selection Intensity in Cellular Evolutionary Algorithms for Regular Lattices. *IEEE Transactions on Evolutionary Computation*, 9(5), pp. 489–505. IEEE.

Goldberg, E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishers. 978-0-20-115767-3.

Goldberg, E. and Deb, K. (1991). A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. *In Foundations of Genetic Algorithms*, 1(90007), pp. 69–93. Morgan Kaufmann.

Gong, R., Chen, W., Liu, F., Dai, K. and Wang, Z. (2008). A New Approach to Single Event Effect Tolerance Based on Asynchronous Circuit Technique. *Journal of Electronic Testing: Theory and Applications*, 24(1–3) pp. 57–65. Springer-Verlag.

Greenwood, G. (2005). On the Practicality of Using Intrinsic Reconfiguration for Fault Recovery. *In IEEE Transactions on Evolutionary Computation*, 9(4), pp. 398–405. IEEE.

Greenwood, G. (2008). Attaining Fault Tolerance through Self-Adaptation: The Strengths and Weaknesses of Evolvable Hardware Approaches. *In World Congress on Computational Intelligence*, Plenary Invited Lecture, J. Z. *et al.*, Ed. LNCS, pp. 368–387. Springer-Verlag.

Herrera, F. and Lozano, M. (2000). Gradual Distributed Real-Coded Genetic Algorithms. *IEEE Transaction on Evolutionary Computation*, 4(1), pp. 43–63. IEEE.

Herrera, F., Lozano, M., Verdegay, J. L. (1998). Tackling Real-Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis. *Artificial Intelligence Review*, 12, pp. 265–319. Kluwer Academic Publishers.

Hodgart, M. and Purivigraipong, S. (2000). New Approach to Resolving Instantaneous Integer Ambiguity Resolution for Spacecraft Attitude Determination using GPS Signals. *In IEEE Position*, Location and Navigation Symposium, pp. 132–139. IEEE.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. USA: MIT Press. 0262082136.

Hounsell, B. I. and Arslan, T. (2001). Evolutionary Design and Adaptation of Digital Filters Within an Embedded Fault Tolerant Hardware Platform. *In 3rd NASA/DoD Workshop on Evolvable Hardware*, Long Beach, CA, 12–14 July 2001, pp. 127–135. IEEE.

Jacob, C. (2001). *Illustrating Evolutionary Computation with Mathematica*. USA: Morgan Kaufmann. 978-1-55860-637-1.

Juang, J. and Huang, G. (1997). Development of GPS-based Attitude Determination Algorithms. *In IEEE Transactions on Aerospace and Electronics Systems*, 33(3), pp. 968–976. IEEE.

Kirley, M., Li, X., and Green, D. G. (1999). Investigation of a Cellular Genetic Algorithm that Mimics Landscape Ecology. *In Lecture Notes in Computer Sciences*, 1585, pp. 90–97. Springer-Verlag.

Koza, J. (1992). *Genetic Programming- on the Programming of Computers by Means of Natural Selection*. England: MIT Press. 0-262-11170-5.

Label, K.A. (1996). Single Event Effect Criticality Analysis. *In NASA HQ/code QW*. 16 Feb. 1996. NASA.

Lala, P.K. (1985). *Fault Tolerant and Fault Testable Hardware Design*. Prentice Hall International. 0-13-308248-2.

Lee, C.-H., Park, S.-H., and Kim, J.-H. (2000). Topology and Migration Policy of Fine-Grained Parallel Evolutionary Algorithms for Numerical Optimization. *In Proceedings of the Congress on Evolutionary Computation*, La Jolla, CA, USA, 16–19 July 2000, pp. 70–76. IEEE.

Li, X. and Kirley, M. (2002). The Effects of Varying Density in a Fine-Grained Parallel Genetic Algorithm. *In Proceedings of the Congress on Evolutionary Computation*, Honolulu, HI, USA, 12–17 May 2002, pp. 1709–1714. IEEE.

Lyons, R. and Vanderkulk, W. (1962). The Use of Triple-Modular Redundancy to Improve Computer Reliability. *IBM Journal of Research and Development*, 6(2), pp. 200–209. IBM.

Mastipuram, R. and Wee, E.C. (2004). Soft Errors' Impact on System Reliability. [online] *EDN*, 30 September 2004, Available at: < http://www.edn.com/article/CA454636 >.

Matsumura, T., Nakamura, M., Miyazato, D., Onaga, K., and Okech, J. (1997). Effects of Chromosome Migration on Parallel and Distributed Genetic Algorithm. *In Proceedings of the 3rd International Symposium on Parallel Architectures, Algorithms, and Networks*, Taipei, Taiwan, 18–20 December 1997, pp. 357–361. IEEE.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Series: Artificial Intelligence. 3540606769.

Morales-Reyes, A. (2010). Fault Tolerant and Dynamic Evolutionary Optimization Engines. *In PhD Thesis*. The University of Edinburgh.

Morales-Reyes, A., Al-Naqi, A., Erdogan, A. T., and Arslan, T. (2009). Towards 3D Architectures: A Comparative Study on Cellular GAs Dimensionality. *In Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems*, San Francisco, California, USA, 29 July – 1 August 2009, pp. 223–229. IEEE.

Morales-Reyes, A., Stefatos, E., Erdogan, A. T., and Arslan, T. (2008a). Fault Tolerant Cellular Genetic Algorithm. *In IEEE Congress on Evolutionary Computation*, Hong Kong, 1–6 June 2008, pp. 2676–2682. IEEE.

Morales-Reyes, A., Erdogan, A. T., Arslan, T., and Stefatos, E. (2008b). Towards Fault Tolerant Systems Based on Adaptive Cellular Genetic Algorithms. *In Proceedings of IEEE NASA/ESA Conference on Adaptive and Hardware and Systems*, Noordwijk, the Netherlands, 22–25 June 2008, pp. 398–405. IEEE.

Morales-Reyes, A., Haridas, N., Erdogan, A. T., and Arslan, T. (2009). Fault Tolerant and Adaptive GPS Attitude Determination System. *In IEEE Aerospace conference*, Big Sky, MT, 7–14 March 2009, pp. 1–8. IEEE.

Muhammad, A., Bargiela, A., and King, G. (1997). Fine-Grained Parallel Genetic Algorithm: A Stochastic Optimisation Method. *In Proceedings of the 1st World Congress on System Simulations*, Singapore, September 1997, pp. 199–203.

Muhammad, A., Bargiela, A., and King, G. (1999). Fine-Grained Parallel Genetic Algorithm: A Global Convergence Criterion. *In International Journal in Computer Mathematics*, pp. 139–155. Gordon and Breach Science.

Normand, E. (1996). Single Event Upset at Ground Level. *IEEE Transaction Nuclear Science*, 43(6), pp.2742–2750. IEEE.

Nowostawski, M. and Poli, R. (1999). Parallel Genetic Algorithm Taxonomy. *In the Proceedings of the 3RD International Conference on Knowledge-Based Intelligent Information Engineering Systems*, Adelaide, Australia, 31 August – 1 September 1999, pp. 1–5.

Oei, C., Goldberg, D., and Chang, S.-J. (1991). Tournament Selection, Niching and the Preservation of Diversity. *In IlliGAL Report No. 91011*, pp. 1–11. University of Illinois.

Olariu, S. and Zomaya, A. Y. (2006). *Handbook of Bioinspired Algorithms and Applications*. Chapman and Hall/CRC. 9781584884750.

Pant, D. and Joshi, K.C. (2007). Software Fault Tolerant computing: Needs and Prospects. *In Ubiquity*, 8(16). ACM.

Pickle, J.C. (1996). Single-Event Effects Rate Prediction. *IEEE Transactions on Nuclear Science*, 43(2), pp. 483–495. IEEE.

Rahman, A., Das, S., Chandrakasan, A. P., and Reif, R. (2003). Wiring Requirement and Three-Dimensional Integration Technology for Field Programmable Gate Arrays. *IEEE Transactions on Very large Scale Integration Systems*, 11(1), pp.44–54. IEEE.

Rebaudengo, M. and Sonza Reorda, M. (1993). An Experimental Analysis of the Effects of Migration in Parallel Genetic Algorithms. *In proceedings of Euromicro Workshop on Parallel and Distributed Processing*, Gran Canaria, Spain, 27–29 January, pp. 232–238. IEEE.

Roosta, S. H. (1999). *Parallel Processing and Parallel Algorithms: Theory and Computation*. USA: Springer-Verlag. 978-0387987163.

Rothlauf, F. (2006). *Representations for Genetic and Evolutionary Algorithms*. 2nd ed. Berlin Heidelberg: Springer-Verlag. 978-3-540-25059-3.

Rudolph, G. (2000). On Takeover Times in Spatially Structured Populations: Array and Ring. *In Proceedings of the 2nd Asia-Pacific Conference on Genetic Algorithms and Applications*, Hong Kong, pp. 144–151. Global-Link Publishing Company.

Sarma, J. and De Jong, K. (1996). An Analysis of the Effects of Neighbourhood Size and Shape on Local Selection Algorithms. *In Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, LNCS 1141, pp. 236–244. Springer-Verlag.

Schofisch, B. and de Roos, A. (1999). Synchronous and Asynchronous Updating in Cellular Automata. *BioSystems*, 51(1999), pp. 123–143. Elsevier.

Simoncini, D., Collard, P., Verel, S., and Clergue, M. (2006b). From Cells to Islands: An Unified Model of Cellular Parallel Genetic Algorithms. *In Proceedings of the 7th International Conference on Cellular Automata for Research and Industry*, pp. 248–257. Springer-Verlag.

Simoncini, D., Collard, P., Verel, S., and Clergue, M. (2007). On the Influence of Selection Operators on Performances in Cellular Genetic Algorithms. *In IEEE Congress on Evolutionary Computation–CEC'07*, Singapore, 25–28 Sept. 2007, pp. 4706–4713. IEEE.

Simoncini, D., Verel, S., Collard, P., and Clergue, M. (2006a). Anisotropic Selection in Cellular Genetic Algorithms. *In Proceedings of the Genetic and Evolutionary Computation Conference–GECCO'06*, Seattle, Washington, USA, 8–12 July 2006, pp. 559–566. ACM.

Simoncini, D., Verel, S., Collard, P., and Clergue, M. (2009). Centric Selection: A Way to Tune the Exploration/Exploitation Trade-Off. *In Proceedings of the Genetic and Evolutionary Computation Conference–GECCO'09*, Montréal Québec, Canada, 8–12 July 2009, pp. 1–20. ACM.

Singh, K., Agbaria, A., Kang, D.-I., and French, M. (2006). Tolerating SEU Faults in the Raw Architecture. *In Proceedings of the 3rd International Workshop on Dependable Embedded Systems*, Leeds, UK, October 2006, pp. 35–40.

Sipper, M., Tomassini, M., and Capcarrere, M. S. (1997). Evolving Asynchronous and Scalable Non-Uniform Cellular Automata. *In Proceedings of International Conference on Artificial Neural Networks and Genetic Algorithms–ICANNGA97*, pp. 67–71. Springer-Verlag.

Spiessens, P. and Manderick, B. (1991). A Massively Parallel Genetic Algorithm: Implementation and First Analysis. *In Proceedings of the 4th International Conference on Genetic Algorithms*, San Diego, CA, July 1991, pp. 279–286. Morgan Kaufmann.

Stefatos E. and Arslan, T. (2004a). High-performance Adaptive GPS Attitude Determination VLSI Architecture. *In IEEE Workshop on Signal Processing Systems*, Austin, Texas, USA, 13–15 Oct. 2004, pp. 233–238. IEEE.

Stefatos, E. and Arslan, T. (2004b). An Efficient Fault-Tolerant VLSI Architecture Using Parallel Evolvable Hardware Technology. *In NASA/DoD Conference on Evolvable Hardware*, Seattle, 24–26 June 2004, pp. 97–103. IEEE.

Stefatos, E., Arslan, T., and Hamilton, A. (2008). Enhanced Evolutionary Techniques for Precise and Real-Time Implementation of Low-Power FIR Filters. *In IEEE Congress on Evolutionary Computation–CEC'08*, pp. 2701–2708. IEEE.

Su, S. Y. H. and Spillman, R. J. (1977). An Overview of Fault-Tolerant Digital System Architecture. *In Proceedings of National Computer Conference*, 13–16 June 1977, pp. 19–26. ACM.

Thomson, R. and Arslan, T. (2002). An Evolutionary Algorithm for the Multi-Objective Optimization of VLSI Primitive Operator Filters. *In Congress on Evolutionary Computation–CEC'02*, May 2002, pp. 37–42. IEEE.

Thomson, R. and Arslan, T. (2003). On the Impact of Modelling, Robustness, and Diversity to the Performance of a Multi-Objective Evolutionary Algorithm for Digital VLSI System Design. *In Congress on Evolutionary Computation–CEC'03*, December 2003, pp. 382–389. IEEE.

Thomson, R. and Arslan, T. (2005). Techniques for the Evolution of Pipelined Linear Transforms. *In IEEE Congress on Evolutionary Computation–CEC'05*, September 2005, pp. 2476–2482. IEEE.

Tomassini, M. (1999). Parallel and Distributed Evolutionary Algorithms: A. In: Miettinen, K., Neittaanmaki, P., Makela, M. M., and Periaux, J., eds. (1999). *Evolutionary Algorithms in Engineering and Computer Science Recent Advances in Genetic Algorithms Evolution Strategies Evolutionary Programming Genetic Programming and Industrial Applications*. England: John Wiley & Sons. Ch. 7. 978-0471999027.

Tomassini, M. (2005). *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time*. Germany: Springer-Verlag. 978-3540241935.

Topol, A. W., La Tulipe, D. C., Shi, L., Frank, D. J., Bernstein, K., Steen, S. E., Kumar, A., Singco, G. U., Young, A. M., Guarini, K. W., and Leong, M. (2006). Three-Dimensional Integrated Circuits. *IBM Journal of Research and Development*, 50(4/5), pp. 491–506. IBM

Turton, B. C. H. and Arslan, T. (1995a). A Parallel Genetic VLSI Architecture for Combinatorial Real-Time Applications - Disc Scheduling. *In the 1st IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications –GALESIA '95*, 12–14 September 1995, Sheffield, UK, pp. 493–498. IEEE.

Turton, B.C.H. and Arslan, T. (1995b). An Architecture for Enhancing Image Processing via Parallel Genetic Algorithms and Data Compression. *In IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications –GALESIA '95*, 12–14 September 1995, Sheffield, UK, pp. 337–342. IEEE.

Turton, B. C. H., Arslan, T., and Horrocks, D. H. (1994). A Hardware Architecture for a Parallel Genetic Algorithm for Image Registration. *In Digest of IEE Colloquium on Genetic Algorithms in Image Processing and Vision*, 20 October 1994, London, UK, pp. 1–6. IEEE.

Ursem, R. K. (2002). Diversity-Guided Evolutionary Algorithms. *In Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, 2439, pp. 462–471, Springer-Verlag.

Velazco, R., Ecoffet, R., and Faure, F. (2005). How to Characterise the Problem of SEU in Processors & Representative Errors Observed on Flight. *In the 11th IEEE International On-Line Testing Symposium–IOLTS'05*, 6–8 July 2005, pp. 303–308. IEEE.

Xie, Y. and Ma, Y. (2008). Design Space Exploration for 3D Integrated Circuits. *In the 9th International Conference on Solid-State and Integrated-Circuit Technology*, Beijing, 20–23 Oct. 2008, pp. 2317–2320. IEEE.

Xu, J., Arslan, T., Wan, D., and Wang, Q. (2002a). GPS Attitude Determination using a Genetic Algorithm. *In Congress on Evolutionary Computation–CEC'02*, Honolulu, HI, USA, 12–17 May 2002, pp. 998–1002. IEEE.

Xu, J., Arslan, T., Wang, Q., and Wan, D. (2002b). An EHW Architecture for Real-Time GPS Attitude Determination based on Parallel Genetic Algorithm. *In Proceedings of Conference on Evolvable Hardware– NASA/DoD'02*, Washington DC, USA, 15–18 July 2002, pp. 133–141. IEEE.

Yarema, R. (2006). Fermilab Initiatives in 3D Integrated Circuits and SOI Design for HEP. *In ILC Vertex Workshop*, Ringberg, Germany, 29–31 May 2006, pp. 1–38.

Zhong, J., Hu, X., Gu, M., and Zhang, J. (2005). Comparison of Performance between Different Selection Strategies on Simple Genetic Algorithms. *In Proceedings of IEEE International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, Vienna, Austria, 28–30 November 2005, pp. 1115–1121. IEEE.