Compact Dynamic Optimisation Algorithm

Chigozirim Justice Uzor

A thesis submitted in partial fulfilment for the

degree of Doctor of Philosophy

at

De Montfort University

October 2015

Abstract

In recent years, the field of evolutionary dynamic optimisation has seen significant increase in scientific developments and contributions. This is as a result of its relevance in solving academic and real-world problems. Several techniques such as hyper-mutation, hyper-learning, hyper-selection, change detection and many more have been developed specifically for solving dynamic optimisation problems. However, the complex structure of algorithms employing these techniques make them unsuitable for real-world, real-time dynamic optimisation problem using embedded systems with limited memory.

The work presented in this thesis focuses on a compact approach as an alternative to population based optimisation algorithm, suitable for solving real-time dynamic optimisation problems. Specifically, a novel compact dynamic optimisation algorithm suitable for embedded systems with limited memory is presented. Three novel dynamic approaches that augment and enhance the evolving properties of the compact genetic algorithm in dynamic environments are introduced. These are: 1) change detection scheme that measures the degree of dynamic change 2) mutation schemes whereby the mutation rate is directly linked to the detected degree of change and 3) change trend scheme that monitors change pattern exhibited by the system.

The novel compact dynamic optimization algorithm outlined was applied to two differing dynamic optimization problems. This work evaluates the algorithm in the context of tuning a controller for a physical target system in a dynamic environment and solving a dynamic optimization problem using an artificial dynamic environment generator. The novel compact dynamic optimisation algorithm was compared to some existing dynamic optimisation techniques. Through a series of experiments, it was shown that maintaining diversity at a population level is more efficient than diversity at an individual level. Among the five variants of the novel compact dynamic optimization algorithm, the third variant showed the best performance in terms of response to dynamic changes and solution quality. Furthermore, it was demonstrated that information transfer based on dynamic change patterns can effectively minimize the exploration/exploitation dilemma in a dynamic environment.

Acknowledgements

First, my deepest appreciation, thanks and praise goes to God Almighty for granting me the wisdom, ability and divine guidance throughout this research. Furthermore, a warm thanks to my parents for their immense support throughout the program.

I would like to express my sincere gratitude to my supervisors Dr. Mario A. Gongora, Dr. Benjamin N. Passow and Dr. Simon Coupland for their invaluable encouragement, guidance, insightful comments and support throughout the duration of this research.

I wish to take this opportunity to thank my friends and fellow colleagues within Centre for Computational Intelligence (CCI) for all the brainstorming, discussions, laughs and words of encouragement, making the whole experience enjoyable.

Finally, I am grateful to my wife Tia-Monique, whose support, encouragement and love enabled me to complete this thesis. Last but not least I would like to thank my daughter Aletheia Ifunanya for giving me a reason to conclude this research.

Contents

List of Figures

List of Tables

List of Abbreviations

DO	Dynamic optimisation
EA	Evolutionary algorithm
DOP	Dynamic optimisation problem
cOA	Compact optimisation algorithm
SA	Simulated annealing
GA	Genetic algorithm
ACO	Ant colony optimisation
DE	Differential evolution
ROOT	Robust optimisation over time
PID	Proportional Integral Differential controller
DOA	Dynamic optimisation algorithm
EDA	Estimation of distribution algorithm
cGA	Compact genetic algorithm
pe-cGA	Persistent compact genetic algorithm
ne-cGA	Non-persistent compact genetic algorithm
rcGA	Real-valued compact genetic algorithm
cDE	Compact differential evolution
PBIL	Population-based incremental learning
BB	Building block
mcGA	Compact genetic algorithm with hypermutation
amcGA	Adaptive-mutation compact genetic algorithm
DUF	Decomposable unitation-based function
DDUF	Dynamic decomposable unitation-based function
DKP	Dynamic knapsack problem
GAm	Genetic algorithm with hypermutation
PBILm	Population-based incremental learning with hypermutation
TMSDS	Torsional mass spring damper system
PID	Proportional integral derivative controller
RMSE	Root mean square error

r-cDE	Compact differential evolution with restart scheme
r-rcGA	Real-valued compact genetic algorithm with restart scheme

Chapter 1

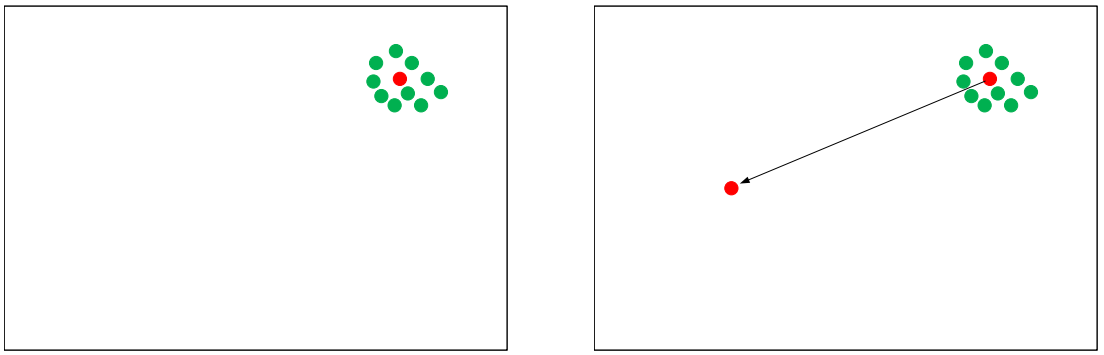
Introduction

The application of evolutionary algorithms to dynamic optimisation problems has been an active area of research for the past decade. This research field, although relatively young, has increasingly attracted interest from the evolutionary computation community. As pointed out by Jin and Branke (2005), the earliest application of evolutionary computation to dynamic optimisation dates back to 1966 (Fogel et al., 1966). Although this research field is relatively advanced, there are still open research questions. One important question is how well do current dynamic optimisation algorithms integrate properly in the context of real-time dynamic optimisation using embedded systems with limited computational resources.

1.1 Overview

Optimisation in a sense has existed since the beginning of civilization. Today optimisation problems can be found everywhere in science, technology and even our daily life activities e.g. planning (Bui et al., 2012), vehicle routing (Dantzig and Ramser, 1959; Mavrovouniotis and Yang, 2015) and tuning of controllers (Gongora et al., 2009; Zhang et al., 2009). Most real-world optimisation problems are often influenced by uncertain and dynamic factors, as such it is unlikely that a solution found for a particular problem would remain valid for a long period of time. In order to counter these dynamic changes, appropriate mechanisms are required to adapt the current solution. This type of problem can be referred to as a dynamic optimisation problem (DOP) (Jin and Branke, 2005).

In DOPs, values of the optima change with time, thus rendering the problem of optimum finding to optimum tracking. This means the fitness landscape[1] of a given problem is dynamic with both the search space and fitness being time dependent. When solving DOPs, evolutionary algorithms (EAs) are considered a good choice because they are inspired from the principles of biological evolution, which takes place in dynamic environment (Goldberg, 1989).



(a) Search space with popultation converging at time t

(b) Optimum change position at time t^{+1}

FIGURE 1.1: Typical behaviour of an optimisation algorithm in a dynamic environment (green circles represents candidate solutions and red circle represent optimum solution)

The interest in improving the performance of EAs in dynamic environments continues to increase so as to identify promising techniques capable of addressing more complex dynamic optimisation problems. Many studies have demonstrated that the conventional EA is suitable for finding the optimum of complex problems when the promising region of the search space remains constant during an optimisation process (Battiti and Passerini, 2010; Panda, 2011; Nelson et al., 2009; Gongora et al., 2009). However, when solving dynamic optimisation problems the conventional EA is not suitable because the algorithm is expected to not only find the optimum but also track the optimum with respect to time (Jin and Branke, 2005; Nguyen et al., 2012).

The nature of DOPs presents challenges to traditional optimisation algorithms because these problems usually require the tracking of the changing environment

[1] A fitness landscape represents the search space of an optimisation problem that exposes the difference in fitness of the solution such that those with good fitness are higher. This means optimal solutions are the maxima of the landscape. This idea has been extended to dynamic fitness seascape. This dynamic fitness seascape represents a quantitative measure of adaptation that counts the excess of good genomic change over disruptive genomic change (see Mustonen and Lässig (2009, 2010) for more details).

with respect to time. In general, addressing DOPs using EAs can be grouped into four classes of techniques Nguyen et al. (2012):

- Using implicitly or explicitly defined memory to store and reuse useful information so as to adapt the EA whenever a change occurs (Yang and Yao, 2008).
- Creating multi-populations to distribute the search force into the search space (Branke et al., 2000).
- Promoting diversity by inserting random immigrants back into the population (Tinós and Yang, 2007b; Mavrovouniotis and Yang, 2013a).
- Adjusting genetic operators adaptively (Eiben et al., 1999, 2006).

Most of the dynamic EAs are a combination of one or two of the above techniques, which increases the computational complexity of the respective algorithm (Nguyen and Yao, 2009a; Yang, 2008; Turky and Abdullah, 2014).

Evolutionary optimisation algorithms have been successfully applied to solving DOPs. However, their application to real-world problems using small embedded systems with resource constraints is limited due to the complex nature of the algorithm. As a result, a compact dynamic optimisation approach is investigated as an efficient alternative since it requires less computational resources.

The next section introduces the research problem followed by the research approach, research objectives and the outline of this thesis.

1.2 Research problem

While research in dynamic optimisation is relatively advanced, most of the research is based on artificial benchmark problems (Li et al., 2011; Halder et al., 2013) where the degree of dynamism and complexity of the problem is controlled. These benchmark problems are in the form of a dynamic optimisation problem generator that is used to generate a predefined landscape and change dynamics (e.g Li et al. (2008); Li and Yang (2008)). There have been some applications based on real-world problems either using real-world data or solving problems originating from real world scenarios (e.g Kanoh (2007); Atkin et al. (2008)).

Numerous dynamic techniques have been proposed to tackle dynamic problems more effectively. However, the complex structure of existing population-based dynamic optimisation algorithms makes them unsuitable for solving real-time DOPs on-board an embedded system with limited memory. Therefore a memory saving dynamic approach can be considered appropriate for such DOPs. In a nutshell, this thesis seeks to answer the following research question:

Is it possible to solve dynamic optimisation problems using a compact optimisation algorithm, especially real-time dynamic optimisation on-board a small-scale embedded system with limited memory?

The main reason for investigating such problems is that, despite the ever growing availability of enhanced computational devices, there are some applications that are based on systems with limited computational resources, specifically limited amount of available memory. Some of these systems are required to execute complex, specific operations such as online training procedures, generating solutions for an optimisation problem. A good example of such application is the optimisation of control systems' parameters in real-time. The optimisation process must be carried out on board a micro-controller as quickly as possible, in order to leave a larger time slot for real-time communication with sensors and actuators.

In this situation, the structure of dynamic optimisation algorithms plays a crucial role if a high efficiency of operation is desired. Therefore many complex dynamic optimisation algorithms e.g. population-based algorithms have been developed as they often allow the detection of promising solutions. Due to hardware limitations, these types of dynamic optimisation algorithms can be inappropriate and unacceptable. This is because of the complex structure of the algorithms, as they employ different modifications which increase the computational resources required such as memory space and execution time (e.g Yang et al. (2013); Li et al. (2012); Nguyen et al. (2013)). As stated earlier most dynamic optimisation (population-based) algorithms have a long execution time which makes them unsuitable for most real-time control optimisation problems. This is because most control problems requires a fast response and algorithms in this category mainly rely on the concurrent evaluation of multiple solutions (Iacca et al., 2012).

For the purpose of this thesis, hardware limitations and small scale embedded systems refer to devices and applications with limited computational resources specifically limited in the amount of available memory.

1.3 Research approach

The complex structure of most dynamic optimisation algorithms hinders their application using a small-scale embedded system. Using classic search and optimisation methods, it is not possible to achieve optimal performance in a dynamic environment. This is because dynamic optimisation problems are time dependent and real-world problems experience dynamic changes that are unpredictable.

This research develops a novel compact optimisation algorithm, suitable for solving DOPs affected by hardware limitation i.e. using small scale embedded systems with limited computational resource. More specifically, a novel variant of the compact genetic algorithm is developed without compromising on computational requirements. The standard compact genetic algorithm (Harik et al., 1999) has been applied to static optimisation problems (Gallagher et al., 2004; Phiromlap and Rimcharoen, 2013a).

However, optimal performance of the compact genetic algorithm is limited when solving dynamic optimisation problems. As a result, modifications to the standard compact genetic algorithm are investigated, so as to improve the performance of the algorithm for such problems.

The realisation of such compact dynamic optimization algorithm is of high relevance, especially in the design of robust controllers for control applications in dynamic environments. Rather than testing candidate solutions on a synthetic benchmark simulation, the individuals can be evaluated on a real system, thus reducing the existing gap between academic research and real world applications. An example of a possible application is the tuning of control parameters for unmanned aerial vehicles in a dynamic environment. Other areas of application could be the design of active automobile suspension systems, evolving parameter for eye tracking systems, forecasting of real time data and applications with hardware limitations and time constraints.

1.4 Research objectives

The primary objective of this research is to develop a novel compact optimisation approach as an alternative to complex population based algorithms suitable for solving real-time DOPs on-board embedded systems with limited memory. Specifically the following schemes are investigated in this research:

- A novel change detection scheme that measures the degree of change in a dynamic environment without a significant increase in computational requirements. This scheme regulates how and when the compact algorithm reacts to a dynamic change.
- A novel adaptive mutation scheme that depends on the degree of change in a dynamic environment such that the probability of mutation is directly proportional to the degree of change. This scheme focuses on the maintenance of diversity to preserve the ability of the compact algorithm to adapt quickly in a dynamic environment.
- A novel change trend scheme that monitors the change pattern in a dynamic environment such that the algorithm learns and adapts quickly. This scheme in combination with the change detection and adaptive-mutation scheme is used to enhance the performance of the algorithm.

The schemes above are designed to allow direct implementation on embedded hardware systems. This research examines the schemes mentioned above in the context of tuning a controller for a physical target system, a torsional mass spring system in a dynamic environment. In addition, these schemes are evaluated using an existing dynamic benchmark generator.

1.5 Structure of thesis

The remainder of this thesis is organised as follows:

- Chapter 2 provides background information and a literature survey on optimisation techniques, evolutionary dynamic optimisation, evolutionary algorithms in control systems and compact optimisation algorithms. A detailed survey on recent advances coupled with analysis and discussion is presented.
- Chapter 3 describes a novel compact optimisation algorithm for solving dynamic optimisation problems. An adaptive-mutation compact genetic algorithm suitable for solving real-world dynamic optimisation problem, specifically embedded hardware systems with limited computational resources is presented.
- Chapter 4 describes the setup scene for all experiments. This includes an artificial dynamic benchmark generator and a dynamic optimisation problem

using a physical system. Rather than testing candidate solutions of competing algorithms in an artificial benchmark simulation only, solutions are evaluated on the real-physical system.

- Chapter 5 presents experimental test results and analysis to support the research hypothesis. In this chapter, competing algorithms are evaluated in different environment dynamics using the dynamic benchmark generator and the physical system described in chapter 4. The chapter is concluded by a summary of the findings and discussion of the results.

- Chapter 6 concludes this thesis with discussion on achievement and respective impacts. Finally, opportunities for further research on the theory and application of the adaptive-mutation compact genetic algorithm are discussed.

Chapter 2

Literature Review

A literature review is presented in this chapter to provide the required background knowledge. This review sets the context of the field to enhance the accessibility for both the evolutionary dynamic optimisation and compact optimisation algorithm (cOA) communities. Key DOP techniques and cOAs are reviewed to understand previous work and the relevance of this thesis.

This chapter is organised as follows; Section 2.1, introduces optimisation algorithms. Specifically, a review of deterministic and heuristic optimisation algorithms is presented. Section 2.2 provides background knowledge about DOPs. Section 2.3 reviews evolutionary algorithms used in control system design in static and dynamic environment. Section 2.4 reviews compact optimisation algorithms. Finally, section 2.5 summarizes this Chapter.

2.1 Optimisation Algorithms

Generally optimisation algorithms are iterative. They start with an initial random solution or guess of a defined variable, then generate a sequence of estimates before converging to an optimum (local or global). Algorithms in this category differ in the strategies used to move from one generation to another. Most optimisation algorithms make use of a target function to measure the fitness of solutions every generation (Rahmati and Mallakzadeh, 2011), some make use of the gradient of a given problem (Koussa et al., 2012) while others solve for the first and second derivative (Pazderin and Yuferev, 2009). There are optimisation algorithms that pile up information every generation (Bao et al., 2009) and some that improve

the quality of solutions sampled by using the local information discovered at each generation (Pham et al., 2011).

Regardless of the description above, a good optimisation algorithm should be efficient, accurate and robust, identify good solutions without being affected by approximation error and noise. Some of these properties are likely to infringe with one another. For example an optimisation algorithm that exhibits fast convergence, may converge to a local optimum. On the other hand a robust optimisation algorithm can be slow and computationally intensive. Therefore, when developing or selecting an optimisation algorithm the trade-off between the rate of convergence, robustness and memory requirements must be put into consideration in order to achieve the desired performance (Črepinšek et al., 2013).

The field of optimisation algorithms has seen improvements over the past decades, with researchers comparing, evaluating and examining different algorithms. Recent developments of optimisation algorithms can be grouped into heuristic and deterministic optimisation approach. This section gives an explanation of these methods.

2.1.1 Deterministic algorithms

Optimisation algorithms in this category take advantage of the analytical attributes of a problem to generate chronological sequence of points before converging to an optimum. For some deterministic optimisation algorithms, calculus serves as a tool for the optimum of a given problem (Haupt and Haupt, 2004). For instance, when solving for the extrema of function with multiple variables, an approach will be to set the gradient of the function to zero and solve for the roots. This means that the roots of such function are minima if the gradient is greater than zero. However, deterministic optimisation algorithms do not necessarily reveal a global minimum. Therefore, the gradient of a given function in most cases points to the steepest downhill.

This method is suitable for problems where the minimum is close to the initial random guess but inefficient when the gradient of a function can not be determined. Some examples of algorithms in this category are: Steepest decent (Cartis et al., 2010; Meza, 2010), Conjugate gradient (Dai, 2011; Narushima et al., 2011), Newton-raphson method (Zhu, 2014; Polyak, 2007) and Quasi-Newton method (Fletcher, 1987; Heath, 1998; Lewis and Overton, 2013).

These algorithms have been used to solve optimisation problems (Zanella et al., 2012). They have a rigorous guarantee for finding at least one solution. However, the computational complexity of deterministic algorithms can be excessive for problems characterised by large dimensions. In addition, most combinatorial and continuous problems are NP-Hard, thus increasing the computational resource would not resolve the associated complexity. Others have compared various deterministic optimisation algorithms in order to identify which is more efficient, accurate and robust.

A review of the application of deterministic optimisation algorithms to management and engineering problems was presented by Lin et al. (2012). Although this paper highlighted successful applications based on the deterministic algorithms, they pointed out that for convex problems, there are other efficient numerical techniques suitable for such problems. However, real-world optimisation problems are (in most cases) non-convex with large dimensions (see Chiang (2008)). This implies that the use of deterministic optimisation technique becomes inappropriate because of the associated computational complexity.

A gradient-based optimisation algorithm was compared to a genetic and hybrid optimisation algorithm in Chaparro et al. (2008) for the task of determining the parameters of a constituent model. The results shown prove that the deterministic optimisation algorithm exhibits fast convergence to a local minimum unlike the genetic algorithm which converged to a global minimum. The hybrid algorithm presented in the paper takes the advantage of both optimisation algorithms by locating a point close to the minima using the genetic algorithm before using the gradient-based algorithm as a local search algorithm to achieve the minimum.

Colaço and Dulikravich (2009) investigated the application of heuristic and deterministic optimisation algorithms to an optimisation problem. The paper confirmed that most deterministic optimisation algorithms converge to a local optimum unless modified using an adaptive search step size technique. They pointed out that most optimisation problems are non-convex. This implies that solving for the second derivative of such a problem is not possible. Therefore, second derivative and gradient-based optimisation methods are not suitable and would result in robustness and reliability problems.

Also, it is important to state at this point that to the author's best knowledge, there have been no algorithms employing only the deterministic optimisation approach for solving dynamic optimisation problems due to the computational complexity associated with this method.

2.1.2 Heuristic algorithms

Unlike deterministic techniques, heuristic optimisation techniques take a different approach. These are algorithms that search for solutions to optimisation problems in conditions where the complex nature of the problem or the permitted computing time available do not allow an accurate solution. This means heuristic optimisation algorithms do not guarantee finding the optimal solutions. These algorithms are designed to search for solutions near the optimum in a reasonable time and are studied for both discrete and continuous optimisation problems.

Heuristic optimisation algorithms can be grouped into population-based and single solution algorithms. The single solution approach builds up a solution step by step, always picking the next step that offers the most obvious benefits. Some examples of algorithms following this approach are Tabu search (Gonzalez-Sieira et al., 2013) and Simulated annealing (SA) (Kirkpatrick et al., 1983). The population-based approach simultaneously updates a set of solutions and permits the transfer of a worse solution from one generation to another so as to avoid getting stuck in the local minimum (i.e. promoting diversity), e.g. Genetic algorithm (GA) (Patel and Raghuwanshi, 2010), Ant colony optimisation (ACO) (Dorigo et al., 2006) and Differential evolution (DE) (Neri and Tirronen, 2010).

There are several algorithms that have employed the heuristic approach in solving complex, challenging optimisation problems. Ying et al. (2009) proposed a modified GA for the design of a PID controller. Due to the issue of premature convergence associated with a simple GA, the author describes new crossover and mutation functions. The modified GA was used to tune the gains of a PID controller for an experimental fermentation device. Performance of the algorithm was compared to the classic Ziegler-Nichols and Smith techniques. From results shown, the proposed modified GA performed better than the classic method.

A hybrid genetic algorithm which is based on particle swarm optimisation and interval algorithm was proposed in Xiao et al. (2010). They applied particle swarm optimisation to the mutation operation in the GA and an interval algorithm in the initialization of population of the GA. The hybrid approach was used in optimising parameters of a PID controller. Their results showed the proposed algorithm prevented the issue of premature convergence, improved stability of the algorithm and achieved a good step response of the PID controller.

These algorithms adopt different techniques to avoid getting stuck in a local optimum. So exploration and exploitation properties are considered important for

finding the global optimum of a given problem. Usually a heuristic algorithm explores a solution space during the initial generation, with big and/or random steps. Then in subsequent generations the algorithm explores the solution space with small steps in order to find the best acceptable solution. It is important to state that while the heuristic approach offers more flexibility than the deterministic approach, some heuristic optimisation algorithms are known to be computationally expensive (require long execution time and memory).

2.1.3 Discussion

This section discussed two optimisation techniques. The deterministic optimisation approach takes advantage of the analytical properties of a given problem to generate a sequence of points that converge to a minimum. The nature of DOPs presents challenges to the deterministic approach as this type of problem requires tracking of the changing environment with respect to time. The performance of a deterministic optimisation algorithm cannot be guaranteed when solving DOPs since this type of problem takes into account other factors such as noisy fitness functions and approximation error (Nguyen et al., 2012).

The deterministic approach works well in situations where the model of a problem is well-defined with low dimensions such that the gradient, first and second derivative can be solved. It is also important to state that the computational effort of algorithms employing the deterministic approach increases with the problem size and in the case of DOPs becomes too complex to tackle (Colaço and Dulikravich, 2009).

On the other hand, the heuristic approach offers more flexibility and efficiency for DOPs. According to Zanakis and Evans (1981), the heuristic optimisation approach is considered advantageous when a simplified model of a problem which can be an inaccurate representation of the real-world problem is used. In such cases a near optimal solution is considered instead of searching for the exact solution to an inexact problem. The performance of the heuristic optimisation method is independent of the initial solution and is derivative-free. It overcomes the main limitations of the deterministic approach getting stuck in local minima. In addition to this, the characteristics of an objective function are inconsequential for the success of algorithms employing the heuristic optimisation approach.

The work in this thesis follows the heuristic optimisation approach based on the evidence that it is easier than others to implement and treats the objective function as a black box, a simple connection between inputs and outputs with no derivative information required. The heuristic approach relies on initial randomization associated with logical patterns as well as different constraint handling methods. This approach is suitable for non-linear and non-convex problems with many decision variables which are features of most DOPs (Hatzakis and Wallace, 2006; Mavrovouniotis and Yang, 2013b).

2.2 Dynamic Optimization Problems (DOPs)

Optimisation problems exist in all areas of industry, research and management. For more than 20 years static optimisation has been an active area of research. However, the inclusion of time dependency by Goldberg and Smith (1987) created a distinct degree of difficulty. Many static problems can be modelled as dynamic optimisation problems in which some parameters change during an optimisation process.

A DOP can be defined mathematically as follows:

$$DOP = \begin{cases} optimise & f(x,t) \quad (2.1a) \\ s.t. & \text{x} \in F(t) \subseteq S(t) \in T \quad (2.1b) \end{cases}$$

where S is the problem search space, $f : S \times T \rightarrow R$ is a target function that assigns a fitness value $f(x,t)$ at time t to candidate solutions $x \in S$. $F(t)$ represents a set of viable solutions $x \in F(t) \subseteq S$.

From Eq. 2.1, a DOP can defined as a succession of static optimisation problems that are linked under some dynamic rules. The dynamic nature of such a problem comes from the magnitude and frequency of change in an environment.

Dynamic changes in an environment can be dimensional or non-dimensional changes. Problems that involve the addition or removal of design variables during an optimisation process are referred to as dimensional changes. In this type of problem, a change in environment affects the representation of solutions and in some cases transforms viable solutions to non-viable solutions. The non-dimensional change corresponds to dynamic changes that affect the values of design variables. This type of change can be easily handled by an efficient dynamic algorithm.

The next section discusses different dynamic optimisation techniques, performance measure and change detection techniques.

2.2.1 Enhancement Techniques to EAs for Solving DOPs

The nature of dynamic optimisation problems (DOPs) presents challenges to traditional optimisation algorithms because these problems require the changing environment to be tracked/monitored with respect to time. EAs can be considered a good option for solving DOPs. In DOPs, values of an optimum changes with time, thus changing the problem of optimum finding to optimum tracking. This means the fitness landscape of a given problem is dynamic with both the fitness and search space being time dependent. Below are some enhancement techniques to EAs for solving DOPs:

2.2.1.1 Memory

The addition of memory schemes to EAs in order to enhance performance in a dynamic environment has been implemented by researchers (Branke, 1999; Barlow and Smith, 2008), especially when changes in the environment are periodical or recurrent. The basic principle of a memory scheme is to store good solutions (with any other information) and reuse them when a new environment is detected. Information can be stored in memory in two ways: either integrated implicitly as redundant representation or explicitly as a separate memory component.

Implicit memory schemes make use of redundant representations to store useful information for EAs to exploit during an optimisation process. There exist different redundant representations of memory for EAs such as diploid, haploid[1] and multiploid [2] (Uyar and Harmanci, 2005; Yang, 2006). Redundant representations using diploid genomes are the most common implicit memory scheme. A diploid EA is usually an algorithm whose chromosomes contain two alleles at each locus (Yoshida and Adachi, 1994). According to Lewis et al. (1998) and Yang (2007b), for tackling dynamic optimisation problems, the diploid and multiploid representation are highly suitable.

[1] A haploid or single-stranded chromosome contains half the cell of a diploid genome.

[2] A multiploid consists of a number of chromosome and a dominance mechanism underlying it interpretation Collingwood et al. (1996).

Lewis et al. (1998) demonstrated that the redundant coding scheme does not ensure enough diversity in dynamic optimisation problems that experience random or oscillatory changes (for example situations where one or more target variables change randomly during a search). Also in implicit coding schemes, the redundant representation may become too large and this affects the performance of the optimisation algorithm (i.e. increased computational burden). In order to recover knowledge about the previous state of the environment, the implicit coding scheme needs to encode the information about its current state into the representation. The number of changing states of a given dynamic optimisation problem is directly proportional to the redundant code needed for representing the changing states.

The explicit mechanism makes use of a precise representation (in the form of a defined memory size) to store a good solution which will be used later in a new environment(or in the event of an environment change). For the explicit memory scheme, certain factors need to be carefully considered: what to store in memory, how to represent the environment, how to update and organise solutions in memory and how to use solution (and associated information) stored in memory. A common practice with regards to the first factor is to store best performing solutions and use them later when an environment change occurs. This is known as direct memory scheme (Simões and Costa, 2008; Yang and Yao, 2005). For some dynamic optimisation problems, the most diversified solutions (in terms of fitness value) are stored in memory. Instead of storing only a good solution, it is also good to store any relevant information associated with the best performing solutions in memory. Then in the event of an environment change, the good solutions and associated information are reused as a similarity measure to compare the new environment with good memory solutions. This method is known as an associative memory scheme (Richter, 2010).

One of the early implementations of the memory scheme was by Ramsey and Grefenstette (1993) for a robot control problem. The robot stores good solutions in a permanent memory with relevant information about the robot's current environment. Memory size or space are usually limited, as a result the allocated memory needs to be used efficiently. This leads to the second factor; how to organize and update memory. This can be achieved by storing the best found solution of the current generation (or final generation) in memory and the update mechanism can be implemented in two ways: by replacing old solutions in memory (individual oriented) (Simoes and Costa, 2007) or by replacing the solution with the least contribution to diversity (population oriented) (Yang, 2005a). And for the last factor on how to use solutions and information stored in memory, common

practice is to retrieve the best solution in memory at every generation or after a defined number of iterations or only when the environment changes.

In general, memory techniques are most useful in cyclic dynamic environments (environments that reappear), otherwise they are not very effective (Branke, 1999).

2.2.1.2 Diversity

Promoting and maintaining diversity during an optimisation process can be achieved in many ways, for example by increasing the mutation rate (Cobb, 1990), by inserting random individuals back into the population (Yang and Yao, 2008), by moving individuals from one sub-population to another and by keeping the sub-population away from one another (Zhu et al., 2006).

In Yang and Tinos (2008), a hyper-selection scheme for GAs was presented to tackle DOPs. This scheme increased the selection pressure whenever an environment change occurred. In conventional GAs, individuals in the population converge to an optimal solution in a static environment as a result of the selection pressure. However, in a dynamic environment, converging to an optimum becomes a problem for the conventional GA since it does not encourage sufficient diversity, thus making it hard to adapt to a changed environment.

In Cobb (1990); Cobb and Grefenstette (1993), a mutation scheme was proposed to introduce diversity into an EA in a dynamic environment. The algorithm starts with a small mutation rate that is applied to the EA population, but this mutation rate is not fixed. When there is a change in the fitness landscape, the mutation rate is increased using a mutation factor (which is user defined) so as to encourage diversity in the population. But if the mutation factor is too high then the effect of the mutation scheme becomes equivalent to a restart scheme (i.e. restarting the optimisation process).

According to Morrison and De Jong (2000), a triggering mechanism is employed to activate the mutation scheme whenever a change occurs. This was achieved by monitoring the running average of the best individuals of the population over a defined number of iterations. When there is a decrease in the running average of the best individuals of the population, the mutation rate is raised. This mutation scheme is referred to as the hyper-mutation. Several authors have combined the hyper-mutation scheme with other methods to improve the performance of EAs in dynamic environments (Cheng and Yang, 2012; Cheng, 2012). Vavak et al. (1996,

1997) introduced the variable local search in which the probability of mutation is increased gradually.

The hyper-mutation scheme creates an adaptive EA with small incremental and computational cost, but requires that the mutation factor be picked a priori. While several articles have adopted the hyper-mutation method, none has considered an adaptive method for controlling the hyper-mutation factor such that the probability of mutation is directly proportional to the diversity scheme used.

Apart from the hyper-mutation and random immigrant techniques, the co-evolutionary technique has been used to encourage population diversity e.g. fitness sharing and crowding (Cedeno and Vemuri, 1997; Manner et al., 1992).

2.2.1.3 Multi-population

Another approach which resembles the diversity approach is the multi-population scheme. Algorithms in this category maintain several sub-populations in parallel so that the search force is distributed. Each sub-population tracks a different area in the search space. One sub-population will focus on locating the global optimum, while another sub-population monitors the environment for any possible change.

Algorithms employing this technique must consider two important factors for an efficient application:

- Assigning different tasks to the sub-population i.e. one sub-population to search and another sub-population to track the global optimum.
- The sub-population should be divided appropriately so as to make sure that sub-populations are not overlapped to have best diversity and to avoid situations where many sub-populations locate the same peak.

In Branke et al. (2000) a Self-organizing Scout was presented, where the main population of the algorithm explored the search space to locate an optimum. When an optimum was found, a new sub-population was created and used to track changes in the optimum. Nguyen and Yao (2009a) presented a Repair-GA for solving dynamic constrained problems. This method makes use of a large sub-population which mainly explores the search space and a smaller sub-population which tracks the moving feasible region in the search space.

In Ursem (2000), a multi-national GA (which grants the search and track ability to the sub-population) was presented. Each sub-population can search for new optimums and track changes. This way, whenever a sub-population locate a new optimum, it splits into two sub-populations to ensure that each sub-population only tracks one optimum at a time.

Algorithms following the multi-population approach have the ability to generate and maintain diversity, and are a good candidate for tackling problems with competing peaks (Li et al., 2015; Turky and Abdullah, 2014). This technique can be used to simulate the memory technique (i.e. used to recall information from previous generation) since one sub-population is dedicated for tracking and retaining previous solutions. This technique is able to search and track the movement of multiple optima in the search space.

However, too many sub-populations can slow down an optimisation process due to a large number of parallel locations in the search space. Also, the multi-population approach is not suitable for environments that experience cyclic or recurrent dynamic change as it requires the calculation of distance or similarity metrics to separate sub-populations which might affect performance. In fact, the multi-population approach is not suitable for memory constrained dynamic optimisation problems e.g. embedded system or direct hardware implementation.

There are other DO techniques which aim to improve the performance of conventional EA when tackling DOPs such as the Prediction approach which learn dynamic change patterns from previous environments and try to predict changes in future environments (Hatzakis and Wallace, 2006; Rossi et al., 2008; Nguyen and Yao, 2009b; Simões and Costa, 2009a,b) and the Self-adaptive mechanism for EAs (Woldesenbet and Yen, 2009; Cobb, 1990; Yang and Yao, 2005).

Yu et al. (2010) proposed a new concept for solving DOPs known as robust optimisation over time (ROOT). They pointed out that the idea of tracking moving optimum of a dynamic optimisation problem has some practical limitations;

- In order to keep track of optimum in dynamic optimisation problems, detection of environmental changes is important.
- Since environmental change is unpredictable, the idea of using knowledge from the past is not reliable as there are situations where the environment could change even before the optimum for the current state is found. This can lead to restarting the search process multiple times, which is time-consuming and inefficient.

- Due to limited computational resource such as computing time, an optimum solution may not be used (even if found suitable for real world application).
- In real world applications a decision made to improve performance at present may become void or reduce performance in future environments.

According to the authors, a solution is robust over a defined time interval if its fitness value remains insensitive to dynamic changes. Therefore, the main task is to find a sequence of robust solutions over a defined time interval. Fu et al. (2012) developed and discussed measures that can be used to classify and analyse dynamic changes for the purpose of tackling ROOT problems. The main objective of this paper was to identify what aspect of the fitness landscape is affected by an environment change. Some of the measures developed by the authors are:

- Optimum degradation, which measures the rate at which the performance of a previous optimum degrades for a particular environment change.
- Estimating optimum degradation, which is used to estimate the rate of optimum degradation without knowledge of the exact optimum.
- Optimum survival length, this measures how long the performance of previous optimum can remain valid in consecutive environment change.
- Estimated optimum survival length, similar to the idea of estimating optimum degradation, but estimates the optimum survival length in consecutive environment changes.
- Survival rate, this measure is used to estimate how long good solution from the last environment change remains valid in a later environment change.
- Fitness correlation, measures the correlation coefficients of fitness of before and after a dynamic change.

In order to analyse these measures, three problem instances of the moving peak problem were used. Simulation results shown indicated that these measures permit evaluation of various environmental changes. For a more detailed review of DO techniques please refer to Nguyen et al. (2012).

Apart from the approaches mentioned above, for an EA to function properly the genetic operators need to be tuned properly (as it affects performance and is problem-dependent) and this can be achieved in three ways: 1) Deterministic

method which involves adjusting the value of the strategy parameter using a deterministic rule which is fixed. 2) Adaptive method that makes use of feedback from the optimisation process to determine when to change the strategy parameter, which can be in form of an IF-THEN rule and may involve a credit assignment which defines the quality of the solution discovered. 3) Self-adaptive method where the mechanism for updating the strategy parameter is implicitly defined i.e. within the EA being used (Eiben et al., 1999; Eiben and Smit, 2012).

2.2.1.4 Change detection

Detecting changes in a fitness landscape has been an active research area with many methods being proposed but the task itself is not easy. In change detection, several important factors need to be taken into consideration such as the severity of change, change predictability, frequency of change, and cycle accuracy/length (Richter, 2009).

Some of the existing dynamic optimisation algorithms employ an explicit action to respond to dynamic changes. Some of these algorithms assume that any change in the environment is made known to the algorithm or the algorithm has to detect the change.

Re-evaluating dedicated solutions

Algorithms in this category regularly re-evaluate dedicated detectors in order to detect environmental changes. The detectors are usually part of a population of solutions e.g. elite solutions (Altin and Topcuoglu, 2014; Hu and Eberhart, 2002; Kramer and Gallagher, 2003) or can be a separate sub-population (Nguyen and Yao, 2010; Li et al., 2006; Zou et al., 2004).

However, these detectors do not always guarantee constant change detection since the dynamic change may not affect the location of the detectors in the search space. Several algorithms such as (Yang, 2007a), have adopted this method in detecting environmental change by using solutions stored in memory as dedicated detectors.

It was demonstrated by Richter (2009) that this change detection technique favours situations where environmental changes are difficult to detect. A common limitation of this method is when noise exists in the objective function, the detectors

may not detect a dynamic change. Another concern is the additional cost of re-evaluating the detectors every generation.

Monitoring algorithms behaviour

Some of the existing algorithms employing this method monitor the performance of the best solutions in the population. This means a dynamic change is detected by monitoring a drop in the value of the best solution over a number of defined generations (Cobb, 1990). While other algorithms constantly monitor for a drop in diversity. Morrison (2004) studied the possibility of detecting change based on population diversity, relationship between the diversity of a fitness value and success rate of the change detection. Richter (2009) presented a change detection technique based on a statistical hypothesis test, so as to find the difference between the distribution of population for two generations.

Algorithms that follow this approach, do not require any additional function evaluation. However, since there are no dedicated detectors, there is no guarantee that a dynamic change will be detected.

2.2.2 Performance measure

When tackling DOPs, it is important to assess the quality of solutions so as to compare the performance against other dynamic optimisation algorithms. In the context of a static environment, during an optimisation process it is often enough to report the best solution found by an algorithm at the end of the process (or end of each iteration/generation). In some cases the memory used, execution time and some error measures are considered. However, when dealing with DOPs reporting just the above mentioned information is not enough because there is other information about the optimisation process that needs to be taken into account. This information includes the algorithm's ability to detect and respond to change, the algorithm's ability to differentiate between noise and change, the ability to switch region of concentration in the search space as the change occurs, and most importantly how well the algorithm tracks the moving optimum.

An in-depth analysis of performance measures for optimisation algorithms in dynamic environments was presented in Weicker (2002). Three characteristics were considered important for evaluating an optimisation algorithm in a dynamic environment:

- How accurately the algorithm is able to track the optimum at a given time.
- The stability of an algorithm: if change in the environment does not affect the optimisation accuracy severely, then the algorithm is considered stable. Even in extreme changes an optimisation algorithm should limit the drop in fitness.
- How fast the algorithm reacts to changes during the optimisation process.

In the field of dynamic optimisation, there are many measures for evaluating the performance of dynamic algorithms. These performance measures can be grouped into two categories namely Fitness-based and Behaviour-based performance measures (Nguyen et al., 2012). The common measures within these categories are discussed below:

2.2.2.1 Fitness-based measure

These are performance measures that evaluate the ability of an algorithm to find an optimum solution or optimum solutions that are closest to global optimum. This section will give a brief review of common fitness-based measures for DOPs:

- **Best-of-generation:** This is one of the most commonly used performance measures (Alba and Sarasola, 2010; Chen et al., 2011; Mavrovouniotis and Yang, 2013b). This measure is calculated as the average for a defined number of runs of the best fitness values at each generation:

$$\overline{F}_{BOG} = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{1}{N}\sum_{j=1}^{N}F_{BOG_{ij}}\right) \tag{2.2}$$

  where $F_{BOG_{ij}}$ expresses the fitness value of the best solution at generation i of run j, G is the total number of generations for a run and N is the total number of runs. This measure can be used to compare algorithms quantitatively. However, due to the difference in the fitness landscape and different change periods, this measure can be considered to be biased as it is not normalised.

- **Modified Offline error:** This measure calculates the average error of the best solution since the last change in environment (Branke, 2001). This

measure returns a value $x \geq 0$, where a *0* signifies best performance. The modified offline error is defined as follows:

$$F_{oe} = \frac{1}{G} \sum_{i=1}^{G} f_e(i) \tag{2.3}$$

where G is the number of generations so far and $f_e(i)$ is the best error since the last dynamic change found by an algorithm at generation i. This measure is commonly used in evaluating the performance of dynamic optimisation algorithms. However, it requires that the moving optimum be made known for every period, which is not true in real-world dynamic problem.

- **Modified Offline performance:** Similar to the modified offline error, the modified offline performance evaluates the performance of a dynamic optimisation algorithm in situations where the global optimum is unknown (Mavrovouniotis and Yang, 2014b,a) and is defined as follows:

$$F_{op} = \frac{1}{G} \sum_{i=1}^{G} f_p(i) \tag{2.4}$$

where G is the number of generations so far and $f_p(i)$ is the best error since the last dynamic change found by an algorithm at generation i. The advantages and disadvantages of F_{op} are similar with the $\overline{F}_{BOG}$. However, it has an additional disadvantage because it requires that the period of a dynamic change is known.

- **Accuracy:** Also known as relative error, measures the accuracy of an optimisation process assuming that the best and worst values in the search space are known a priori (Weicker, 2002). The accuracy of a dynamic optimisation algorithm is defined as follows:

$$acc = \frac{F_{BOG}(t) - f_{min}(t)}{f_{max}(t) - f_{min}(t)} \tag{2.5}$$

where F_{BOG} expresses the fitness value of the best solution, $f_{max}(t)$ is the max value in a search space and $f_{min}(t)$ is the min value in a search space at generation t respectively. The value returned is between *0* and *1*, where *1* is the best possible performance.

This measure does not depend on the fitness rescaling and is less biased to the time of change, where the fitness difference becomes large. Weicker (2002) pointed out that this measure is only effective if the entire search

space is not a plateau at any iteration otherwise the denominator of Eq.(2.5) at generation t would be *0* (i.e. resulting to a division by *zero error*).

2.2.2.2 Behaviour-based measure

The behaviour-based performance measure is used to evaluate whether a dynamic optimisation algorithm exhibits certain behaviour in a dynamic environment. This measure takes many forms such as monitoring and maintaining diversity throughout an optimisation process, and avoiding a significant fitness drop when a change occurs. Some of the behaviour based measures are discussed below:

- **Total diversity:** The total-diversity T_{div} measures the level of diversity within a population. This also depends on the encoding scheme used, as there are different metrics for different coding scheme. However, only the binary encoding scheme is shown here (interested readers please refer to Nguyen et al. (2012)). Usually, on binary-encoded problems, the hamming-distance is used as the diversity measure(Rand and Riolo, 2005; Yu et al., 2009). T_{div} at generation i can be defined as follows:

$$T_{div} = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{1}{N}\sum_{j=1}^{N} div_{ij}\right) \tag{2.6}$$

  where in the case of binary encoded representation, div_{ij} is given as:

$$div_{ij} = \frac{1}{l\mu\left(\mu-1\right)}\sum_{a=1}^{\mu}\sum_{b\neq a}^{\mu} hd\left(a,b\right) \tag{2.7}$$

  where μ is the population size, l is the encoding length and $hd(a,b)$ is the hamming distance between solutions a and b.

  T_{div} can be used in tuning an optimisation algorithm for a balance between exploration and exploitation. This means a high diversity score signifies high exploration which is equivalent to a random walk.

- **Stability:** This performance measure is used to evaluate the increase of fitness after a dynamic change occurs (Weicker, 2002). In general an algorithm is considered *stable* if the algorithms accuracy is not significantly affected by a dynamic change.

$$stab_{(t)} = max\Big\{0, \left(acc_{(t)} - acc_{(t+1)}\right)\Big\} \tag{2.8}$$

where $acc_{(t)}$ and $acc_{(t+1)}$ is optimisation accuracy (as defined in equation 2.5) before and when a dynamic change occurs respectively. The values returned from this performance measure is between *0* and *1*, where *0* is the best possible value.

- **Reactivity (convergence speed after change):** This performance measure evaluates how fast a dynamic optimisation algorithm reacts to dynamic changes. The reactivity of a dynamic algorithm at time t is defined as follows:

$$react_{(t)} = min\left\{t^{'} - t \mid t<t^{'} \leq G, t^{'} \in \mathbb{N}, \frac{Acc_{(t')}}{Acc_{(t)}} \geq (1-\epsilon)\right\} \cup (G-t) \quad (2.9)$$

  where G is the total number of generations. This particular performance measure suffers from a particular drawback: it is only useful if there exists an actual drop in performance whenever a change in environment occurs. If not, there is no information about how well a dynamic algorithm responds to dynamic changes.

Apart from these measures other authors have reviewed different performance measures e.g. Ben-Romdhane et al. (2013); Nguyen et al. (2012). Having looked at some of these measures, a conclusion can drawn that there exists no universal performance measure for comparing algorithms, rather performance measures are problem dependent or user defined.

2.2.3 Discussion

Dynamic optimisation problems present challenges to the conventional evolutionary algorithm. As stated earlier, these problems are influenced by uncertain and dynamic factors. In order to counter these factors an adaptive mechanism is required to introduce changes to the optimisation process. When solving such problems the main objective is to minimize cost over a period of time. There is literature addressing DOPs, but none has considered a memory efficient compact approach as an alternative to population-based algorithms. In DOPs, fitness values of the optima change with time. Consequently, the fitness landscape of such problem is dynamic with the possibility of both the fitness and search space being time dependent.

While the research in DOPs is progressing, most of the current research is based on artificial benchmark problems (Li et al., 2011, 2008; Mavrovouniotis et al., 2013)

where the level of dynamism and complexity of a problem are controlled. These artificial benchmark problems are in the form of a DOP generator that generates a predefined landscape and permits the control of how the change occurs, and the type of change. One of the commonly used artificial benchmark problem is the Moving Peak Benchmark (MPB) (Branke, 1999; Mavrovouniotis et al., 2013). This benchmark consists of a multi-dimensional landscape with several peaks, where the position, width and height of each peak are modified once every defined iteration to simulate a change in environment. The level of complexity of this problem can be augmented further by increasing the problem dimension, number of peaks and by introducing noise into the whole landscape.

There are other artificial benchmark problems (Li and Yang, 2008; Yang and Yao, 2005; Rohlfshagen and Yao, 2009) that follow the same idea but what remains unclear is what feature of a real-world problem do these artificial benchmark problems model. Since real-world problems experience dynamic changes that are unpredictable and uncontrollable, how do these benchmark problems take into account these unpredictable changes?

Interesting real-world dynamic optimisation problems have began to appear in the last few years. However, these real-world problems make use of real-world data to do simulations (e.g. Atkin et al. (2008)) before exporting solutions to the actual real-world problems (Mills-Tettey et al., 2008). For example; in Dam et al. (2007), an online data mining task in a changing environment was solved using a classifier system known as a genetic-based learning classifier system (GBLCS). The evaluation function of a multi-rover was evolved in an uncertain and noisy environment using an EA. A pollution control system, car distribution system for off-lease vehicles and path planning of ships was solved in Michalewicz et al. (2007) using Adaptive Business Intelligence (ABI).

There have been several other applications to real-world problems with positive results. However a major concern is the complex structure of the algorithms used, as they employ so many modifications to the conventional algorithm which increases the computational resources needed to execute them. This hinders their applications to real-world optimisation problems (using memory constrained embedded systems). Most of these algorithms have long execution time and this makes them unsuitable for most real-world problems, since most real-world optimisation requires a fast response.

2.3 Evolutionary Algorithms in Control Systems

The field of control systems has been an active area of research for many decades. A control system can be found in cars, industrial equipment, communication systems, medical equipment and many more. For a system to function properly a control system is required. A control system plays a key role in the design of experimental equipment used in scientific research. This implies that the field of control systems will continue to grow as long as new technologies are developed. The main aim of control system design is to find a controller that makes it possible to control complex systems effectively.

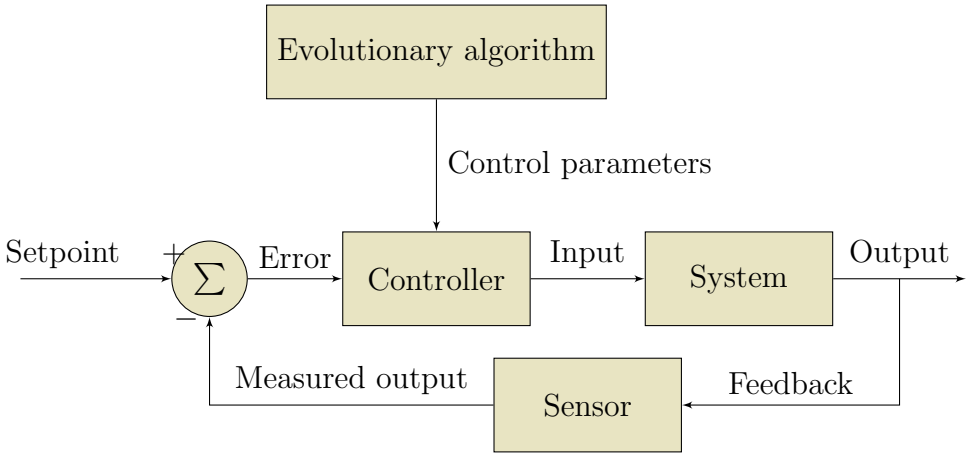


FIGURE 2.1: Overview of a typical evolutionary feedback controller loop

2.3.1 Control design techniques

For an effective control of a dynamic plant (system), it is important to have a firm understanding of the plant's behaviour i.e input-output relationship (see Fig. 2.1). This behaviour can be explicitly defined using a model of the plant or implicitly defined using a numerical model of the plant. This can be in form of experimental data, control laws that have been tuned based on experiments.

2.3.1.1 Model-based

Most control algorithms rely on models of a plant to be controlled (Karsai et al., 2003). These models are constructed from physical observations and considerations of a defined plant (i.e. mathematical behaviour of the plant). These models are accurate to an acceptable level in order to achieve dynamic performance of the

feedback control (Jensen et al., 2011). This method is known as Model-based design. These models are usually built for a particular plant (i.e. plant specific) and cannot be transferred to another model (e.g by using a helicopter model to control a cart pole system model), therefore restricting its application to a particular type of plant.

The ability of a model to act as a physical system must be reviewed by comparing model behaviour with the actual system behaviour. This means any mathematical model of a plant must be validated, as wrong and erroneous conclusions can be drawn from results obtained while simulating inadequate plant models. In order to avoid such mistake, a control/physical law (which can be in the form of Newton's law of motion) is used to model the behaviour and parameters of a plant (Franceschini and Macchietto, 2008).

2.3.1.2 Model-free

Model-free control method implies the absence of a detailed analytical (software) model of the physical system, derived by a control designer, engineer or technician. This model consists of very few parameters that are estimated on-line while the plant is active (Fliess and Join, 2009). There are situations where a numerical model of a plant is considered valid in understanding the behaviour of a plant. These are situations where a plant is too complex and expensive to model. This includes time varying systems and plant models with parameters that needs to be defined experimentally.

Model-free control involves trying to estimate what can be modified through the measurement of input and output of a system in order to achieve good output, and this means constructing a numerical model of the system (Gédouin et al., 2011). The main idea behind model-free control is to identify the input-output behaviour of a controlled system so as to modify the behaviour of the system using a defined control law. This is achieved within a defined time and for each sample time, the estimation is updated for good performance.

Close coupling of system identification and the control design steps is important features of model-free control design technique. This makes the design process easy so that if the plant controller is simplified, the simplification process is carried out with respect to the controllers input-output relationship, unlike model based design which follows the plant input-output relationship (Woodley et al., 2001; Lauwerys et al., 2005).

2.3.2 Evolutionary optimisation of controllers

The global optimisation field has been an active area of research, giving rise to different optimisation algorithms for problems in both continuous and combinatorial domain. Heuristic and stochastic algorithms have been developed as efficient tools for global optimisation and have been applied to various optimisation problems in diverse field.

Evolutionary optimisation techniques have been applied in the design of control systems (hui Chang et al., 2011; Gongora et al., 2009; Passow et al., 2008; Smit and Eiben, 2009). Control system design based on optimisation can be carried out in order to obtain optimum control parameters (which can be offline or online). Optimisation based on EAs evaluates over many generations population of individuals (solutions) usually specified by artificial genotypes. In some cases this process follows the Darwinian principle of evolution, the fittest individuals proceed to next generation to produce new offspring.

An important decision in evolving control systems is whether or not to use simulation at the evaluation stage, then transfer the end result to the real-world problem. Since a conventional evolutionary approach potentially evaluates a population of solutions over many generations, a natural approach would be to carry out simulations so as to speed up the process, making it more feasible.

A control system evolved using a carefully constructed simulation environment would generate almost an identical behaviour in the actual real system. However, this is only true for simple control problems and simple environments. If there exist any form of change in the behaviour of a system, the results obtained from simulation becomes invalid. In such situations, the simulation becomes time consuming to construct and computational intensive and expensive.

2.3.2.1 Static optimisation

The majority of control system optimisation approaches have been in a static environment, where the fitness landscape, objective function and control parameters remain constant. However this is not always true in a dynamic environment.

A new design method was presented in hui Chang et al. (2011) to determine optimal parameters for a PID controller using the adaptive fast evolutionary algorithm

(AFEP). The AFEP was used in combination with a PID control to solve the position control problem of a DC motor. The method was tested on the position control system of a SYL-5 PM DC motor and compared with conventional EP and the Z-N (Ziegler and Nichols) tuning method. Results obtained showed that the proposed method can improve dynamic performance of the system in a more efficient way.

A neural adaptive control system based on the accelerated evolutionary programming was proposed in (Wang et al., 2006). The AEP was used to train an ANN so as to obtain optimal performance. The ANN of the control system, models complex processes with uncertainty which then returns the signals for on-line tuning of the controller.

In Makaitis (2003), evolutionary programming was used as an iterative learning process to evolve rules of a Mamdani fuzzy controller. This controller was applied to an elevator control system problem. The fuzzy controller consists of two inputs (position and velocity) and a single output (acceleration). A total of 17 linguistic terms were defined for both the inputs and output of the controller. The evolutionary algorithm was also used for modifying the rules in the knowledge base, so as to permit concurrent adjustment of both the linguistic terms and the fuzzy logic rules.

In Chiang (2010) an intelligent control system using genetic programming was proposed. The control system generates control rules automatically even in uncertain or unknown environments. The controller consists of a symbolic rule controller, percepter and an adaptor. The control system is made up of two stages, the learning stage - which uses flexible genetic programming to build initial control rules and the adaptive stage - which is for controlling the plant. The control system was applied to a robot path planning problem and result obtained, showed that control performs well but requires more cpu time.

A PID control optimisation technique using a GA was presented by Passow and Gongora (2008). The optimisation process was implemented on a real flying robot. This optimisation technique was compared with hand tuning method and it was observed that the real-time tuning technique using GA granted more robust performance.

Some control optimisation process run on a host computer while potential solutions are sent from the computer to a target system through a communication link (please see Passow and Gongora (2008); Caraffini et al. (2013); Gongora et al.

(2009)). These solutions are evaluated using the target system where control performance is recorded and sent back to the host computer for fitness evaluation. Although this method has proven effective, the environment upon which these controls are evaluated experience some form of uncertain and dynamic changes which are unpredictable. In many control systems these uncertain and dynamic factors come from the errors of the real-world physical equipments and disturbances.

2.3.2.2 Dynamic optimisation

In general, the gap between academic benchmarks and real-world applications (using physical systems) in control optimisation exist not only in the way academic benchmark environments are designed but also in the way the associated optimisation algorithms are designed. Optimisation algorithms often exploit simulation discrepancies in an opportunistic manner to achieve high fitness values with unrealistic behaviours.

On the other hand, in DOPs most optimisation algorithms are also designed to work well in benchmark simulations. However, for a real-world problem some algorithms may not work well to satisfy other optimisation goals. This is because environmental changes in real-world scenarios are unpredictable. As a result the best solution achieved by a dynamic algorithm after a change might be notably different from the solution before a change (i.e. promising region in the search space experience significant displacement).

In Isaacs et al. (2008), a memetic algorithm was presented for solving dynamic multi-objective optimisation problems. The learning error of a neural network was minimized using a memetic algorithm. The optimised neural network was then used to control a dynamic model of an Unmanned Aerial Vehicle (UAV). The neural network controller was trained online due to the significance of noise which could render offline training ineffective.

Mitra and Venayagamoorthy (2008) presented an adaptive control strategy for a DSTATCOM controller in an electric ship power supply based on Artificial Immune System (AIS). The main objective of their paper was to maintain the regulation of a reference value for a bus voltage in a ship power supply unit (i.e. using a simplified hardware set-up of the ship system) in real-time. According to this paper, a PI controller was first tuned using PSO so as to obtain optimum parameter. Then AIS was used to adjust the solution online to minimize disturbance. Although this

strategy is limited by its computing time and complex structure of the algorithms, the results were promising.

An optimal visual proportional differential controller was presented by Wang et al. (2008). The PD controller was used to improve the performance of a visual servoing system. Using the visual servoing system experiments were carried out to minimize robot trajectory from a reference trajectory. According to the paper, experiments were carried out using a physical system. However, a mathematical model of the system was defined offline and optimisation was also carried out offline.

There are other applications of DOAs based on models of real-world problems or using data obtained from real-world scenarios (Alvarez-Gallegos et al., 2005; Tawdross et al., 2006; Jatmiko et al., 2008; Haugwitz et al., 2009). Although the dynamic algorithms used in these papers show good performance, the complex structure of these algorithms limit their direct implementation on embedded hardware with limited memory. This is because these algorithm employ more than one dynamic optimisation techniques which in turn increases the memory requirements needed to run such algorithm. In addition, the model used may not relate directly to the actual system. Some important features such as noise and uncertainties may have been neglected.

2.3.3 Discussion

From the reviews carried out, it was observed that there exists a great gap between academic research theory and practice. From a pragmatic perspective, some researchers have developed algorithms that successfully solve optimisation problems using real-world scenarios or real-world data, while other researchers have evaluated dynamic algorithms using artificial benchmark problems. A good observation is the complex structure of dynamic optimisation algorithms being proposed, they employ different modifications to improve performance in dynamic environments. However, these modifications increase the computational complexity of the algorithm making them unsuitable for small scale embedded micro-controllers.

Real-time control applications are typically implemented on embedded micro-controllers with limited computational power. This makes them a good candidate for developing dynamic optimisation algorithms suitable for solving real-time, real-world problems notwithstanding resource constraints. Available computational resource is an important factor which is often neglected. In embedded micro-controllers, the memory is often restricted and has to be used efficiently.

However, the need to solve an optimisation problem at a specific rate, imposes a very large computational demand on the device implementing the control operations. So far control optimisation has been widely adopted in situations that permit long execution and evaluation times. The bulk of research on the development of dynamic optimisation algorithms focuses on computation counts or fitness evaluations and implementation, and can afford long computing times.

Therefore, it is essential to develop algorithms that can be directly applied to real-world dynamic problems, specifically real-world dynamic problems using an embedded system (with limited limited memory) so as to reduce the gap between academic research and real-world applications.

2.4 Compact Optimization Approach (cOA)

The availability of powerful and complex computational devices continues to increase constantly. However, there are some applications that make use of intelligent systems that are affected by hardware limitation to perform some specific operations in real-time. This situation is typical in an embedded systems. These impose the use of simple hardware structures (due to cost and physical space requirements) that are affordable for real-time control system optimisation, where all control, evaluation and optimisation operations are carried out on-board the micro-controller.

In such situations, the structure of an optimisation algorithm plays an important role in the performance of the algorithm if high efficiency is desired. The research on complex population-based algorithms has been increasing constantly because of their ability to detect a solution in the promising region of a search space. However, direct implementation of these algorithms on an embedded system is not possible due to hardware limitations and the complex structure of these algorithms.

This section gives an overview of compact optimisation algorithms (cOA), which are optimisation algorithms that belong to a class of estimation of distribution algorithms (EDAs) (Larraanaga and Lozano, 2001; Pelikan et al., 2000). Applications to DOPs and limitations of existing cOAs are also highlighted.

2.4.1 Compact genetic algorithm (cGA)

The compact genetic algorithm was proposed by Harik et al. (1999) as an estimation of distribution algorithm that creates offspring populations based on an estimated probability model of the parent population. The cGA uses a probability vector ($\overrightarrow{P}$) to model the distribution of the population and represent the bit probability of 0 or 1.

$$\overrightarrow{P} = \{P_1, \ldots, P_l\} \tag{2.10}$$

where l is the binary string length and $P_i \in \{0, 1\}$, ($i = 1,\ldots,l$). The probability vector is initially assigned 0.5 to represent a randomly generated population. Two candidate solutions are sampled from $\overrightarrow{P}$ and their respective fitness calculated. The solution with the best fitness value biases the $\overrightarrow{P}$ based on the population size N_p. The probabilities P_i in $\overrightarrow{P}$ increases by $\frac{1}{N_p}$ if the ith gene of the best solution sampled displays a 1 and deceases by $\frac{1}{N_p}$ if the ith gene of the best solution displays a 0. On the contrary, if the ith gene of both the best and worst solution sampled are the same, P_i is not modified.

$$P_i' = \begin{cases} P_i + \dfrac{1}{N_p} & \text{if } best_i = 1, \quad (2.11a) \\ P_i - \dfrac{1}{N_p} & \text{if } best_i = 0. \quad (2.11b) \end{cases}$$

One of the advantages of the cGA is that the population size does not mean a real population but a simulated population and based on this a large population can be effectively exploited without compromising the memory requirement of any application. The cGA can simulate high selection pressure by incorporating the elitism method. This provides a means for reducing genetic drift by ensuring that the best performing solution is passed onto the next generation. The pseudo-code describing cGA is shown in Table 2.4.1.

Other variants of the cGAs have been developed to enhance the speed and accuracy of convergence of the original cGA. Some of the cGA variants are discussed in the following section:

Algorithm 2.4.1: *cGA* (l, Np)

$l \leftarrow$ *string length*, $Np \leftarrow$ *population size*
Step 1 : *Initialize probability vector*(P)
for $i \leftarrow 1$ **to** l
  do $P[i] \leftarrow 0.5$
while t<*max_generation*
  then
    Step 2 : *Sample solutions from* P
    *solution*1 $\leftarrow$ *generate*(P)
    *solution*2 $\leftarrow$ *generate*(P)
    Step 3 : *Let solutions compete*
    {*winner*, *loser* $\leftarrow$ *compete*(*solution*1, *solution*2)
    Step 4 : *Update working* P
    for $i \leftarrow 1$ **to** l
      do **if** *winner*$[i] \neq$ *loser*$[i]$
        then **if** *winner*$[i] \leftarrow 1$
          then $\{P[i] \leftarrow P[i] + \frac{1}{Np}$
          else $\{P[i] \leftarrow P[i] - \frac{1}{Np}$
    $t \leftarrow t + 1$

TABLE 2.1: Pseudo-code of a conventional compact genetic algorithm (cGA)

Ahn and Ramakrishna (2003) presented two novel variants of the cGA. These algorithms are known as the persistent cGA (denoted as pe-cGA) and the non-persistent cGA (denoted as ne-cGA). Both algorithms share the same idea as the conventional cGA. However, pe-cGA and ne-cGA achieved better fitness when compared to the conventional cGA. During the initialisation stage, two solutions are randomly generated. Then in subsequent generations, only one solution (usually the best in terms of fitness value) is retained as an elite and compared to another randomly generated solution. The elite solution biases the $\overrightarrow{P}$ only if it outperforms the randomly generated solution. However, if the randomly solution outperforms the elite solution, it replaces the elite and $\overrightarrow{P}$ is updated using the new elite. The pe-cGA differs from the ne-cGA in that the elite replacement only occurs if a random solution outperforms the elite solution. While in ne-cGA, the elite is replaced not only if outperformed by a randomly generated solutions but also after every n generations regardless of the fitness value of the elite.

In Mininno et al. (2008), a real-valued compact Genetic Algorithm (rcGA) was proposed. The rcGA exports the idea of the compact logic of the cGA to the real-valued domain without a substantial increase in memory requirements. The

Algorithm 2.4.2: *nonpersistent rcGA* ($d,\ \eta,\ Np,\ \theta,\ \lambda$)

$d \leftarrow$ *problem dimension*
$Np \leftarrow$ *population size*
$\lambda \leftarrow$ *initial standard deviation*
$\theta \leftarrow$ *the present length of inheritance*
$\eta \leftarrow$ *the allowable length of inheritance*

Step 1 : *Initialize mean* (μ), *and variance* (σ) *vectors of* P
for $i \leftarrow 1$ **to** d
do $\mu_j \leftarrow 0$, $\sigma_j \leftarrow \lambda$
while t<*max_generation*
then
Step 2 : *Sample one solution from* P
if *first generation*
then $\theta \leftarrow 0$, *elite solution* $\leftarrow$ *generate*(P)
random solution $\leftarrow$ *generate*(P)
Step 3 : *Let solutions compete*
winner, *loser* $\leftarrow$ *compete*(*elite solution*, *random solution*)
if (θ<η) *and* (*winner* == *elite solution*)
then $\theta \leftarrow \theta + 1$
else if (*winner* == *random solution*)
then *elite solution* $\leftarrow$ *winner*, $\theta \leftarrow$ *winner*
else *elite solution* $\leftarrow$ *generate*(P), $\theta \leftarrow 0$
Step 4 : *Update mean* (μ), *and variance* (σ) *of* P
for $i \leftarrow 1$ **to** d
do
$\mu'[i] \leftarrow \mu[i] + \frac{1}{Np}\left(winner[i] - loser[i]\right)$
$\left(\sigma'[i]\right)^2 \leftarrow max\{0, v_i'\}$
$v_i' \leftarrow (\sigma[i])^2 + (\mu[i])^2 - (\mu'[i])^2 + \frac{1}{Np}\left(winner[i]^2 - loser[i]^2\right)$
$t \leftarrow t + 1$

TABLE 2.2: Pseudo-code of a non-persistent real-valued compact genetic algorithm (rcGA)

rcGA makes use of a $n \times 2$ matrix:

$$P^t = \left[\mu^t, \sigma^t\right] \tag{2.12}$$

where μ and σ are vectors that represent the mean and standard deviation of solutions sampled using a Gaussian probability distribution function truncated within the interval [-1, 1]. At the initial stage of an optimisation, for each variable i, $\mu^l[i] = 0$ and $\sigma^l[i] = \lambda$ where λ is positive value.

Similar to the pe-cGA and ne-cGA, two solutions are sampled from P^t with one as an elite. The mechanism for sampling a design variable $x[i]$ is defined as follows:

$$PDF(truncNorm(x)) = \frac{e^{-\frac{(x-\mu[i])^2}{2\sigma[i]^2}}\sqrt{\frac{2}{\pi}}}{\sigma[i]\left(erf\left(\frac{\mu[i]+1}{\sqrt{2}\sigma[i]}\right) - erf\left(\frac{\mu[i]-1}{\sqrt{2}\sigma[i]}\right)\right)} \tag{2.13}$$

where erf represents an error function (please refer to Abramowitz and Stegun (1972)). After the evaluation stage, the elite/winner solution biases P and the update rule for μ and σ is defined as follows:

$$\mu^{'}[i] = \mu[i] + \frac{1}{N_p}\left(winner[i] - loser[i]\right), \tag{2.14}$$

$$\left(\sigma^{'}[i]\right)^2 = (\sigma[i])^2 + (\mu[i])^2 - (\mu^{'}[i])^2 + \frac{1}{N_p}\left(winner[i]^2 - loser[i]^2\right) \tag{2.15}$$

where N_p is the simulated population size. The pseudo-code describing rcGA is shown in Table 2.4.2. Two variants of the rcGA were studied (i.e. pe-rcGA and ne-rcGA) in Mininno et al. (2008) with results and it was observed that the level on the elitism is problem dependent.

2.4.2 Other cOA

Apart from the cGA and its variants, other cOA have been proposed, suitable for solving memory constrained optimisation problems. Mininno et al. (2011) introduced the compact Differential Evolution (cDE). The cDE modifies the rcGA by sampling more solutions in accordance to the mutation rule of the conventional Differential Evolution (DE) (Das and Suganthan, 2011). A memetic variant of the cDE was presented in (Baraglia et al., 2001), which is composed of a cDE and a local search. Other similar related cOA research is presented and discussed in Neri et al. (2010); Lanzi et al. (2008); Fossati et al. (2007); Nakata et al. (2013) and Tominaga et al. (2013).

2.4.3 Addressing DOPs

Although optimisation in dynamic environments has been an active field of EA research for the past two decades, only recently the DOP has started to raise interest among Estimation of Distribution Algorithms (EDAs) researchers.

One of the first EDAs applied to a DOP was the population-based incremental learning algorithm (PBIL) (Baluja, 1994), which was used in Yang and Yao (2008, 2003, 2005) to solve DOPs created by a benchmark generator. The first cGA applied to DOP was presented in Harik (1999). The approach used was based on random restart mechanism i.e. at each environment change the population undergoes a random restart so that diversity in the population can be increased at the beginning of the new environment. The paper also demonstrated the use of learned structural information about the problem to accelerate the growth of highly fit substructures so that the algorithm can adapt to the new environment. This method was later extended in Sastry et al. (2005) to include sub-structural niching (i.e. in sub-structural niching, niches are defined within linkage group rather at individual level). After computing the average fitness of each substructure, the sampling probabilities are changed based on the associated fitness.

An approach for maintaining diversity throughout an optimisation process using EcGA was presented in Chuang and Smith (2013). A restricted tournament replacement strategy was used to demonstrate that the problem structure should be considered when maintaining diversity. The main aim of their paper was to develop an algorithm that responds to change without the additional means of detecting an alteration in the fitness landscape. In addition, They explained that diversity should be maintained at a sub-structural level when solving DOPs.

2.4.4 Discussion

Although some of these algorithms have been successful in tackling DOPs, from reviews, it was observed that some of the applications of variants of the cGA to DOPs did not explicitly state how changes in the environment was determined or sensed. There have been no mechanisms that permit knowledge and information transfer from one environment to another. To the best of the author's knowledge none of these applications to DOPs using cGA has applied a mechanism to determine the degree of change in the dynamic environment while maintaining diversity. The observed questions are addressed in this thesis.

2.5 Summary

This chapter introduced the existing literature on the methods and concepts discussed within the thesis. Dynamic optimisation techniques and performance measures were presented and discussed. In addition, some limitations of the existing dynamic approach were discussed. Compact optimisation algorithms were presented in this chapter, as well as associated progress and limitations. This chapter can be considered as the base upon which this work is built.

The next chapter presents and discusses in detail a novel compact optimisation algorithm that is suitable for real-world (real-time) dynamic optimisation problems and more specifically dynamic problems using memory constrained embedded systems.

Chapter 3

Compact Approach to Dynamic Optimisation

In order to successfully solve dynamic optimisation problems using an evolutionary algorithm, there is a need for additions to the conventional algorithmic structure. Most of the dynamic optimisation techniques discussed in section 2.2 can be triggered and hence made to work properly if dynamic changes are made known to the algorithm (visible and detectable). This means that the issue of change detection is of high practical relevance in solving DOPs. Apart from detecting change in a dynamic environment, there are other important questions that need answers such as 1.) "how does the change detection improve the performance of an algorithm?" and 2.) "does the environment exhibit change patterns that can be learnt by a dynamic optimisation algorithm?" This chapter seeks to provide answers to these questions.

This chapter presents and discusses a novel variant of the cGA known as ***adaptive-mutation compact genetic algorithm*** (denoted as amcGA). Specifically three novel contributions are presented in this chapter that make up the amcGA structure: a novel change detection scheme 3.1.1, novel adaptive mutation schemes 3.1.2 and a novel change trend scheme 3.1.3. Based on these schemes, five variants of the amcGA are presented in this chapter and are denoted as amcGA1, amcGA2, amcGA3, amcGA4 and amcGA5 (please see Eq. 3.5, 3.6, 3.7, 3.10 and 3.11 respectively).

3.1 Adaptive-mutation Compact Genetic Algorithm (amcGA)

In static environments, the goal of an optimisation algorithm (in most cases) is to find the global optimum. However, in a dynamic environment where the considered problem is time varying, the goal of an optimisation algorithm is to find and track the changing optimum over time.

The cGA is of particular interest in this research, not only for the reduction of memory requirements but also because of its compact structure, the absence of random selection, recombination and the lack of population. As discussed in section 2.4.1, the cGA represents its population as a probability vector over a set of solutions. This implies that the cGA only models the existence of its population, similar to the sGA with uniform crossover using a small amount of memory (Goldberg, 1989; Harik et al., 1999). This makes the cGA suitable for small scale embedded hardware systems with limited memory and processing capabilities since population-based optimisation algorithms are computationally more expensive, and thus unsuitable for such application.

More specifically, the potential of the cGA is particularly promising because embedded systems are an important part of modern technology in various fields, such as evolvable hardware (Gallagher et al., 2004; Kramer and Gallagher, 2003; Aporntewan and Chongstitvatana, 2001). Although, the conventional cGA is computationally efficient (in terms of memory requirements), it can not solve difficult optimisation problems such as deceptive functions. This is due to the fact that it lacks the memory required to retain information (e.g. linkage of genes) about such problems (Harik et al., 1999).

Previous studies have shown limitations of the conventional cGA (Chuang and Smith, 2013; Phiromlap and Rimcharoen, 2013b). It lacks the search power required to solve some optimisation problems such as dynamic optimisation, multi-objective and highly dimensional problems. Several techniques have been proposed aiming to improve the performance of the conventional cGA. Most of these authors (e.g Ahn and Ramakrishna (2003); Silva et al. (2007, 2008)) focus on introducing techniques to overcome reduction in performance when tackling problems with high order building blocks (BBs). Another contribution is the addition of the elitism scheme by Ahn and Ramakrishna (2003). The elitism scheme has been

used to demonstrate the best balance between performance and demand of resources (see Ahn and Ramakrishna (2003)). Although, the elitism approach does not encourage genetic drift, the level of elitism should be controlled appropriately because high selection pressure may lead to premature convergence.

Despite the performance exhibited by the cGA when tackling problems of low order BBs, its performance is limited when high order BBs are inherent in a problem. This implies that even with the compact nature of the cGA, the information about high order interaction between solution structure (chromosome) does not last over generations. However, the performance of the cGA improves when the selection pressure is increased. The selection pressure increases the probability of high order building blocks to last over generations. Some authors have modified the conventional cGA to enhance performance by regulating selection pressure e.g. pe-cGA and ne-cGA (Ahn and Ramakrishna, 2003), mcGA (Zhou et al., 2002; Gallagher et al., 2004) and mcGA with elitism (Silva et al., 2007, 2008). Among all modified variants, the mcGA with elitism showed best performance in terms of trade off between solution quality and convergence speed (Silva et al., 2008).

The performance of the conventional cGA in a dynamic environment is limited as once the probability vector converges it is unable to adapt to the changed environment. In addition to this the conventional cGA has a limited ability of finding the global optimum in highly dimensional search space. As a result, novel modifications to the original algorithm are required so as to boost its performance when tackling DOPs. The novel modifications discussed in this section are designed to augment the abilities of the conventional cGA by modifying the standard evaluation cycle. These novel modifications allow the algorithm to retain the small footprint of the conventional cGA and greatly increase its search efficiency in dynamic environments (see Chapter 5).

3.1.1 Change detection

The first novel contribution is a change detection scheme that detects and determines the degree of change in a dynamic environment. Usually, for an algorithm to address DOPs efficiently, it is important to detect changes in a dynamic environment (Richter, 2009). Section 2.2.1.4 in chapter 2 described different change detection schemes (highlighting advantages and limitations of each approach). This section introduces a novel change detection scheme that measure the degree of change c_d in a dynamic environment based on the fitness of an elite. In order to

achieve this, a Gaussian function is employed because this function is suitable for problems that require continuously differentiable curves and smooth transitions (Hameed, 2011). This function is defined as follows:

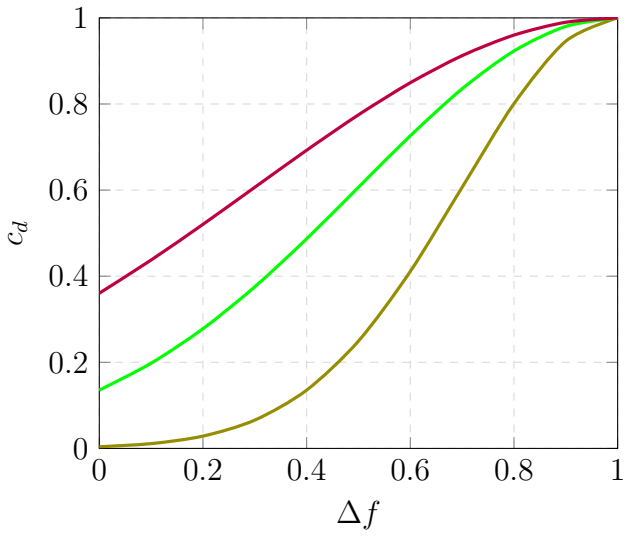
$$c_d = e^{-a} \tag{3.1}$$

where:

$$a = \frac{(\Delta f - c)^2}{2\sigma^2} \tag{3.2}$$

and $c = 1$ is the mean (i.e. maximum change), σ represents standard deviation and Δf is the change in fitness or fitness difference between the elite solution at generation $t-1$ and the same elite solution re-evaluated at generation t:

$$\Delta f = f(e, t-1) - f(e, t) \tag{3.3}$$



The curves in the plot represents different variance for the change detection scheme (i.e. σ). This parameter also defines how sensitive the ***amcGA*** *is to dynamic changes i.e. when to output a new c_d which is used to calculate probability of mutation.*

FIGURE 3.1: Diagram demonstrating the change detection scheme with *x-axis* as *input* and *y-axis* as *output*

It is important to state that decline in fitness of the elite solution is considered in this research as a sign of change in the environment i.e. the algorithm monitors the performance of the elite solution (see Fig. 3.1). The algorithm employs the elitism approach, where the best solution from a previous generation is transferred and evaluated in subsequent generations.

3.1.2 Adaptive mutation

The second novel contribution is the addition of a diversity scheme to help maintain diversity during an optimisation process. This is accomplished by modifying the conventional cGA cycle so that an improved mutation scheme is used to generate a mutated version of an elite solution for tournament selection.

In this section, it is hypothesized that:
"using an adaptive mutation scheme is better than a mutation scheme with two fixed mutation rates (base mutation and high mutation) which are controlled by a mutation factor that must be picked a priori"
i.e. the *hypermutation scheme* (Cobb, 1990; Morrison and De Jong, 2000).

If the mutation factor in the hypermutation scheme is not picked carefully, it would result in insufficient variation in the population or would clearly disrupt the overall performance of an EA. The adaptive mutation scheme discussed here makes use of the degree of change c_d in an environment to regulate the probability of mutation p_m (see Eq. 3.4) such that the degree of change in a dynamic environment is directly proportional to the probability of mutation.

Adaptation of genetic operators (in some cases) depends on how operators are updated. In general, adaptation of genetic operators can be grouped into two main categories i.e. *population-level* and *individual-level* adaptation (Angeline, 1995). At an *individual-level* adaptation, the properties of an individual in a population are modified, whereas in *population-level* adaptation, parameters are globally adapted by using the feedback information from the population or feedback during an optimisation process. The majority of existing literature follow the individual-level adaptation e.g. (Cheng and Yang, 2012; Kramer and Gallagher, 2003; Morrison and De Jong, 2000) and many more. However, the individual-level adaptation suffers the drawback of requiring more time to evaluate or calculate new genetic parameters for each gene locus at different iterations (please see Korejo et al. (2009)).

Unlike the mutation scheme adopted by most EAs where mutation is applied directly to candidate solutions (individual-level mutation) to create another solution for selection, the mutation scheme discussed in this section is applied directly to the probability vector that generated the best solution (i.e. elite) since the probability vector represents a distribution of the population.

In order to adaptively control the mutation rate (probability of mutation) p_m, c_d is converted to the mutation rate such that a high degree of change results in a high mutation rate and a low degree of change results in a low mutation rate. However, when no change occurs the algorithm proceeds as a conventional cGA. The probability of mutation p_m is defined as:

$$p_m = m_l + (c_d - d_l) \cdot \left(\frac{m_h - m_l}{d_h - d_l} \right), p_m \left[0.01, 0.5 \right] \tag{3.4}$$

where $m_l = 0.01$ is low probability of mutation, $m_h = 0.5$ is high probability of mutation, $d_l = 0.0$ is low degree of change and $d_h = 1.0$ is high degree of change. These parameters have been set to these values so as to control the probability of mutation and prevent *"random-walk"* in a search space.

Suppose at generation $t-1$ an elite solution e with fitness $f(e, t-1)$ was obtained, the $\overrightarrow{P}$ that generated the solution (i.e. associated $\overrightarrow{P}$) is held in a temporary memory $\overrightarrow{mP}$. At generation t, e is re-evaluated and a new fitness value is obtained i.e. $f(e, t)$. If the fitness difference Δf is greater than a defined threshold $cThres$ (i.e. sensitivity level), then a change is said to have occurred which triggers the mutation scheme. The mutation scheme is applied directly to $\overrightarrow{mP}$ to generate a mutated version of the elite solution e_m to compete with e.

The conventional cGA makes use of a real-valued probability vector which generates two solutions when sampled. In order to apply the mutation scheme to $\overrightarrow{mP}$, a random number $r = rand\,(0.0, 1.0)$ is generated and compared with p_m, $\overrightarrow{mP}$ is mutated as follows:

- *amcGA1*:

$$mP_i^{'} = \begin{cases} rand\,(c_d, mP_i) & \text{if } r < p_m, \quad (3.5a) \\ mP_i & \text{if } r > p_m \quad (3.5b) \end{cases}$$

- *amcGA2*:

$$mP_i^{'} = \begin{cases} |mP_i + (r - p_m)| & \text{if } r < p_m, \quad (3.6a) \\ mP_i & \text{if } r > p_m \quad (3.6b) \end{cases}$$

- *amcGA3*:

$$mP_i^{'} = \begin{cases} mP_i + \left(r - \frac{p_m}{2} \right) & \text{if } r < p_m, \quad (3.7a) \\ mP_i - \left(r - \frac{p_m}{2} \right) & \text{if } r > p_m \quad (3.7b) \end{cases}$$

Algorithm 3.1.1: *amcGA1→3*(l, Np, $cThres$)

$l \leftarrow$ *string length*
$Np \leftarrow$ *population size*
$cThres \leftarrow$ *change threshold*
$t_m \leftarrow$ *mutation count*
$M_{max} \leftarrow$ *maximum mutation*
Step 1 : *Initialize probability vector*(P)
for $i \leftarrow 1$ **to** l
  do $P[i] \leftarrow 0.5$
while t<*max generation*
then
  Step 2 : *Sample solutions from* P
  if $t \leftarrow 1$
    then {*elite solution* ← *generate*(P)
    else if $t > 1$ *and envChanged* ← *true*
      then {*mutated elite solution* ← *generate*(mP)
        random solution ← *mutated elite solution*
    else {*random solution* ← *generate*(P)
  Step 3 : *Let solutions compete*
  winner, *loser* ← *compete*(*elite solution*, *random solution*)
  if *new elite found* ← *true and* $t > 1$
    then {$mP \leftarrow P$
      $newEliteFound \leftarrow false$
  Step 4 : *Detect change*
    compute c_d
    if $t>1$ *and* $c_d >$ *cThres*
      then {*envChanged* ← *true*
  elite solution ← *winner*
  Step 5 : *Apply mutation scheme*
  if *envChanged* ← *true*
    then *amcGA1 or amcGA2 or amcGA3*
      (*please see eqns* 3.5, 3.6 & 3.7 *respectively*)
      $t_m \leftarrow t_m + 1$
      if $t_m \geq M_{max}$
        then {$t_m \leftarrow 0$
          $envChanged \leftarrow false$
  Step 6 : *Update working* P
  if *envChanged* ← $false$
    then **for** $i \leftarrow 1$ **to** l
      do **if** $winner[i] \neq loser[i]$
        then **if** $winner[i] \leftarrow 1$
          then $\{P[i] \leftarrow P[i] + \frac{1}{NP}$
          else $\{P[i] \leftarrow P[i] - \frac{1}{NP}$
  $t \leftarrow t + 1$

TABLE 3.1: Pseudo-code of amcGA1, amcGA2 and amcGA3

The amcGA2 and amcGA3, make use of the scaled mutation rate p_m (i.e. based on the degree of change) to regulate the amount a mutation operation alters the probability vector within the algorithm. From equations 3.6 and 3.7, it can be

noted that amcGA3 increases and decreases $\overrightarrow{mP'}$ at a reduced scale $\left(r - \frac{p_m}{2}\right)$. The amcGA2 alters $\overrightarrow{mP'}$ only if $r<p_m$, while amcGA1 is a randomised mutation based on the current values of elements of $\overrightarrow{mP'}$ and the degree of change c_d. The pseudo-code describing amcGA1, amcGA2 and amcGA3 is shown in Table 3.1.1.

3.1.3 Change trend

Sometimes, changes in dynamic environments may exhibit some trends. In such cases, it might be beneficial to try to use these change trends to improve the algorithm's response to subsequent changes in such dynamic environments. Some studies have been made following this idea by exploiting the predictability of dynamic environments (Simões and Costa, 2009b,a).

Memory approaches (Branke, 1999; Yu and Suganthan, 2009), which were originally proposed to deal with periodical changes, can also be considered as a type of prediction method. Algorithms following the prediction approach make use of a memory scheme to cope with various types of changes (e.g. cyclic, noisy and random). However they require the use of accurate training data and dedicated memory allocation, which makes the respective algorithm computationally more expensive.

In this thesis, a novel change trend $\overrightarrow{t_{chg}}$ is employed to enhance the performance of the amcGA in a dynamic environment. This is achieved by applying $\overrightarrow{t_{chg}}$ to $\overrightarrow{mP}$ so that $\overrightarrow{mP}$ learns from past dynamic changes and adapts to subsequent dynamic changes, instead of explicitly using stored training data. The change trend $\overrightarrow{t_{chg}}$ can be implemented in two ways: a) binary-encoded problems (bt_{chg}) and b) real-valued problems (rt_{chg}).

- **Trend scheme for binary-encoded problems** bt_{chg}:

  When solving binary-encoded problems, a straight forward approach is to continuously calculate the difference between the probability vector held in temporary memory $\overrightarrow{mP}$ and the current working probability vector $\overrightarrow{P'}$ before and after an environment change occurs. This way the behaviour exhibited by the $\overrightarrow{P'}$ is monitored (direction and rate of convergence).

  From the bt_{chg} pseudo-code, it is clear that the t_{chg} is obtained based on how much elements of $\overrightarrow{P'}$ increase or decrease with regards to $\overrightarrow{mP}$. Obtaining $\overrightarrow{t_{chg}}$ for binary-encoded problems based on solutions sampled can be difficult since

the difference between solutions will be made up of "1s" or "0s" depending on preference. This implies that for binary-encoded problems, $\overrightarrow{t_{chg}}$ should be monitored at population-level rather than individual-level (see Table 3.1.2).

Algorithm 3.1.2: $bt_{chg}(l)$

$l \leftarrow$ *binary string length*
if $eliteFound \leftarrow true$ & $t > 1$
then { **for** $i \leftarrow 1$ **to** l
do $\{bt_{chg}[i] \leftarrow mP[i] - P'[i]$

TABLE 3.2: Pseudo-code of binary change trend scheme

- **Trend scheme for real-valued problems** rt_{chg}:

  In order to calculate t_{chg} of n variables, $\overrightarrow{P}$ and $\overrightarrow{mP}$ are defined as an array that holds $l \times n$ probabilities (where l is the binary string length) i.e.:

$$\overrightarrow{P} = \{P_i, ..., P_{(l \times n)}\} \quad P \in \{0, 1\},\ i = \{1, ..., (l \times n)\} \tag{3.8}$$

$$\overrightarrow{mP} = \{mP_i, ..., mP_{(l \times n)}\} \quad mP \in \{0, 1\},\ i = \{1, ..., (l \times n)\} \tag{3.9}$$

  From the $\overrightarrow{rt_{chg}}$ pseudo-code, it can be observed that for real-valued problems, the change trend obtained is based on the difference between an old elite solution and a newly discovered elite solution. This is because the change trend comes from the variation in solutions sampled from the current working $\overrightarrow{P'}$. The means this scheme updates automatically based on how often a new elite is obtained (see Table 3.1.3).

  Similar to the pe-cGA (Ahn and Ramakrishna, 2003), two solutions ($s1 = [l \times n]$ and $s2 = [l \times n]$) are sampled from $\overrightarrow{P}$. Then in subsequent generations, only one solution (usually the best in terms of fitness value) is retained as an elite and compared to another randomly generated solution. If another new elite (i.e. e^{new}) is discovered before or after an environment change occurs then $\overrightarrow{t_{chg}}$ is calculated as the difference between the new elite and previous elite (i.e. e^{old}).

Algorithm 3.1.3: $rt_{chg}(n)$

$n \leftarrow$ *problem dimension*
if $eliteFound \leftarrow true$ & $t > 1$
then { **for** $i \leftarrow 1$ **to** n
do $\{rt_{chg}[i] \leftarrow elite^{new}[i] - elite^{old}[i]$

TABLE 3.3: Pseudo-code of real-valued change detection scheme

3.1.4 Adaptive-mutation with change trend

The change trend scheme is integrated into the amcGA such that the amcGA makes use of change patterns exhibited by the current working probability vector $\overrightarrow{P}$ to mutate the probability vector held in memory $\overrightarrow{mP}$, so as to boost the algorithm's response to dynamic change i.e:

- *amcGA4*:

$$mP_i' = \begin{cases} \left| mP_i + \left(rand(0, p_m) - \frac{t_{chg}}{l} \right) \right| & \text{if } r < p_m, \quad (3.10a) \\ \left| mP_i - \left(rand(0, p_m) - \frac{t_{chg}}{l} \right) \right| & \text{if } r > p_m \quad (3.10b) \end{cases}$$

- *amcGA5*:

$$mP_i' = \begin{cases} \left| mP_i + \left(rand(0, p_m) - \frac{p_m}{2} \right) - \frac{t_{chg}}{l} \right| & \text{if } r < p_m, \quad (3.11a) \\ mP_i & \text{if } r > p_m \quad (3.11b) \end{cases}$$

  where l is the binary string length.

From Eq. 3.10 and 3.11, it can be observed that the change trend scheme was applied to amcGA2 and amcGA3 which yields amcGA5 and amcGA4 respectively to study the effect of t_{chg} on the performance of the algorithm (see Table 3.1.4). This way, the mutation strategy updates itself based on the change pattern exhibited by the probability vector. Also t_{chg} controls the amount (and direction) a mutation operation changes the value of each element in $\overrightarrow{mP}$.

After the mutation operation, a mutated version elite solution e_m is sampled from $\overrightarrow{mP'}$ to compete with e. if e_m outperforms e, it replaces e and $\overrightarrow{mP'}$ replaces the

Algorithm 3.1.4: *amcGA4 & 5*$(l,\ n,\ Np,\ cThres)$

$l \leftarrow$ *string length*
$n \leftarrow$ *problem dimension*
$Np \leftarrow$ *population size*
$cThres \leftarrow$ *change threshold*
$t_m \leftarrow$ *mutation count*
$M_{max} \leftarrow$ *maximum mutation*

Step 1 : *Initialize probability vector*(P)
for $i \leftarrow 1$ **to** l
  do $P[i] \leftarrow 0.5$
while t<*max_generation*
  Step 2 : *Sample solutions from* P
    if $t \leftarrow 1$
      then {*elite solution* $\leftarrow$ *generate*(P)
      else if $t > 1$ **and** *envChanged* $\leftarrow$ *true*
        then {*mutated elite solution* $\leftarrow$ *generate*(mP)
                 random solution $\leftarrow$ *mutated elite solution*
      else {*random solution* $\leftarrow$ *generate*(P)
  Step 3 : *Let solutions compete*
    winner, loser $\leftarrow$ *compete*(*elite solution, random solution*)
    elite solution $\leftarrow$ *winner*
    if *new elite found* $\leftarrow$ *true* & $t > 1$
      then $mP \leftarrow P$
               compute t_{chg}(bt_{chg} *or* rt_{chg})
               new elite found $\leftarrow$ *false*
    Step 4 : *Detect change*
      compute c_d
      if $t>1$ **and** $c_d >$ *cThres*
        then {*envChanged* $\leftarrow$ *true*
  then **Step 5** : *Apply mutation scheme*
    if *envChanged* $\leftarrow$ *true*
      then *amcGA4 or amcGA5*
               (*please see eqns* 3.10 & 3.11 *respectively*)
               $t_m \leftarrow t_m + 1$
               if $t_m \geq M_{max}$
                 then $t_m \leftarrow 0$
                          envChanged $\leftarrow$ *false*
  Step 6 : *Update working* P
    if *envChanged* $\leftarrow$ *false*
      then **for** $i \leftarrow 1$ **to** l
                 do **if** *winner*$[i] \neq$ *loser*$[i]$
                        then **if** *winner*$[i] \leftarrow 1$
                                 then $\{P[i] \leftarrow P[i] + \frac{1}{Np}$
                                 else $\{P[i] \leftarrow P[i] - \frac{1}{Np}$
  $t \leftarrow t + 1$

TABLE 3.4: Pseudo-code of amcGA4 and amcGA5

current working $\overrightarrow{P}$. The mutation scheme is repeated for a defined number of generations similar to the hypermutation scheme. After the mutation operation, the algorithm continues as a conventional cGA unless another environmental change occur. Table 3.5 shows a summary of the dynamic schemes embedded within each variant of the amcGA.

TABLE 3.5: Summary of the change detection, adaptive mutation and change trend schemes.

	amcGA1	**amcGA2**	**amcGA3**	**amcGA4**	**amcGA5**
Change detection	✓	✓	✓	✓	✓
Adaptive mutation 1	✓				
Adaptive mutation 2		✓			✓
Adaptive mutation 3			✓	✓	
Trend scheme 1				✓	
Trend scheme 2					✓

The adaptive mutation scheme presented in this section is somewhat similar to the well established Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen and Ostermeier, 1996, 2001; Hansen and Kern, 2004), which is a standard in continuous domain evolutionary optimisation (in static environments). Both algorithms share the same idea of storing information about a candidate solution which is used to update or bias the solution sampling process. However, in the CMA-ES evolution cycle, after the evaluation and ranking of the solutions, their relative steps are used to update the parameters of the sampling distribution. This process is repeated throughout the evolution cycle until a termination criteria is met. On the other hand, the amcGA only updates $\overrightarrow{mP}$ when an elite is discovered and samples a mutated elite from it when a dynamic change occurs. This sampling and update process is only repeated for a defined number of iterations. In addition, the amcGA is developed for combinatorial optimisation problems and the embedded adaptive mutation scheme can be considered as an improved form of the popular hyper-mutation scheme.

3.1.5 Discussion

Mutation promotes diversity and can be used to improve the ability of an optimisation algorithm to react to dynamic changes. However, mutation on its own can

reduce the performance of an algorithm if the rate of mutation is not controlled appropriately.

Comparing the adaptive mutation scheme presented here with other mutation schemes applied to the cGA, it is clear the adaptive-mutation scheme operates on population-level unlike that of Silva et al. (2007, 2008), where mutation is applied to potential solutions. When addressing DOPs using EAs, diversity in the population is useful for adapting in a changing environment, since members of the population represent potential solutions that can be applied to different environmental circumstances. The adaptive mutation scheme is embedded within all amcGA variants to promote diversity in dynamic environments, to ensure that the population (the probability vector) maintains diversity while solving a DOP and slowly move towards the optimal solution.

The change detection scheme presented in this section can be viewed as a fusion between the change detection using a dedicated detector and change detection based on an algorithm's behaviour (i.e. using a change trend scheme).

3.2 Summary

In this chapter, a novel variant of the cGA denoted as amcGA, suitable for solving dynamic optimisation problems was introduced. The amcGA is made up of 3 core components: a novel change detection scheme, novel adaptive mutation schemes and a novel change trend scheme. These components guide the amcGA to exploit local optima, explore new promising regions in a search space, converge to best global optimum and enable the algorithm escape local optima.

In order to regulate the amount a mutation operation alters the probability vector, novel mutation strategies are implemented by means of an adaptive mutation scheme at a population level. The adaptive mutation scheme is based on the degree of change in a dynamic environment such that a high degree of change results to a high mutation rate.

Change patterns or trends in a dynamic environment are also captured by the algorithm so as to improve overall performance. The novel change trend scheme enables the amcGA to automatically monitor direction and rate of change exhibited by the current working probability vector.

Next chapter will present and discuss the set-up for all experiments including benchmark problems and control optimisation problems in a dynamic environment using a physical system.

Chapter 4

Set-up Scene for Experiments

The experimental study of the amcGA algorithm is performed based on two dynamic benchmark problems: 1.) a DOP using a synthetic dynamic benchmark generator (i.e. XOR DOP generator and 2.) a DOP using a physical system, more specifically a torsional mass-spring damper system in a dynamic environment. These dynamic test environments are described in this chapter as well as associated parameter settings for all experiments. In this thesis, the optimisation problems considered are problems of single objective and without constraints in the decision space, except the constraint of the search domain using the physical system. It is important to state that all variants of the amcGA described in Chapter 3 were used in all experiments described in this Chapter.

4.1 Artificial benchmark generators

Researchers have developed several dynamic problem generators (dynamic synthetic problems) for the purpose of comparing the performance of EAs developed for dynamic environments. As stated in Chapter 2, DOPs are constructed by changing parameters and properties of static optimisation problems. Through proper control and modification of certain parameters, different dynamic environments (and levels of dynamism) can be simulated for static problems such as change dynamism (severity, frequency, cyclicity and predictability of change) and problem dimension.

Dynamic benchmark problems are essential in the process of developing, evaluating and comparing the performance of dynamic optimisation algorithms. Nguyen

(2011); Morrison (2004) and Yang (2004) highlighted the characteristics of a good benchmark generator:

- ***Applicability***: It should closely (to an extent) resemble or model real-world problems
- ***Flexibility***: It should be easy to configure by a user for different dynamic settings (such as change severity, frequency and periodicity) and different scales (such as the number of optima, dimensions and domain ranges).
- ***Simplicity***: It should be efficient and easy to adapt to user specifications.
- ***Generalization***: It should allow the representation of different dynamic scenarios. it should not be problem/algorithm specific.

Apart from the above characteristics, there are other properties associated with a DOP generator which can be classified based on the following criteria (Nguyen, 2011):

- ***Time linkage***: If a current solution found by a dynamic optimisation algorithm influences future dynamic changes.
- ***Detectability***: Whether dynamic changes are made visible to the optimisation algorithm (using dedicated detectors or based on the behaviour of the algorithm).
- ***Dimensionality***: Whether dynamic changes affect the problem dimension.
- ***Predictability***: If changes in a dynamic environment are predictable.
- ***Randomized/Cyclic***: Whether dynamic changes are random or cyclic, where same environment re-appear.
- ***Properties that change***: Whether dynamic changes affect parameters of the objective function and constraints.

Different dynamic benchmark generators have been developed over the years and used in various literature to construct a dynamic test environment to evaluate performance of EAs. In general, existing DOP generators can be classified into two categories: 1.) The first category of dynamic benchmark generators, constructs dynamic environments by modifying a predefined fitness landscape. An example

is the moving peaks benchmark problem which consists of a multi-dimensional landscape with several peaks, where the position, width and height of each peak is altered every time a dynamic change occurs (Branke, 1999; Li et al., 2008; Li and Yang, 2008; Li et al., 2011). 2.) The second category of dynamic benchmark generators, switches between several states of a static optimisation problem. A good example of the later is the travelling salesman problem where the number of cities, locations and distance changes over time i.e. switches between static instances (Cobb and Grefenstette, 1993; Mavrovouniotis et al., 2012). In addition, DOP generators in this category are characterised by how fast the environment changes.

In this section, the XOR DOP generator and two popular dynamic benchmark problems are described, and used to evaluate the performance of the amcGA.

4.1.1 XOR DOP generator

This dynamic generator has been used by several researchers to evaluate the performance of dynamic optimisation algorithms with modifications (Yang and Yao, 2008; Yang, 2008). The XOR DOP proposed by Yang (2003) is the only known benchmark generator for combinatorial space that constructs a dynamic environment from any binary-encoded static optimisation problem. The XOR DOP simply shifts the population of solutions into a different location in a fitness landscape. Given a static optimisation problem $f(x)$ $(x \in \{0,1\}^l)$ where l is the length of a binary string, a dynamic environment is generated by applying a binary XOR mask $\overrightarrow{M}$ to each candidate solution before evaluating at every τ generations.

$$f(x,t) = f(x \oplus \overrightarrow{M}(k)) \tag{4.1}$$

where $f(x,t)$ represents the fitness value of a candidate solution x, $k = t/\tau$ is the index of the time of change, t is the current count of generations and $\overrightarrow{M}(k)$ is a binary mask of environment period k. The bitwise exclusive-or operator $\oplus$ is applied to the x and $M(k)$ according to the following principle:

$$x_i \oplus x_j = \begin{cases} 0 & \text{if } x_i = x_j, \quad (4.2a) \\ 1 & \text{otherwise.} \quad (4.2b) \end{cases}$$

$\overrightarrow{M}(k)$ is incrementally generated as follows:

$$\overrightarrow{M}(k) = \overrightarrow{M}(k-1) \oplus \overrightarrow{T}(k) \tag{4.3}$$

where $\overrightarrow{T}(k) \in \{0, 1.0\}^l$ is a binary template randomly generated for environment period b. $\overrightarrow{T}(k)$ is made up of $\rho \times l$ 1s, where ρ controls the intensity or magnitude of change. $\overrightarrow{M}(k)$ is initially set to all zeros to indicate no dynamic change. If $\rho = 0$, the environment is considered stationary since $\overrightarrow{T}$ will contain only $0s$ and no change will occur. On the other hand, when $p = 1$ results in a high degree of change (or high change severity). Also a small τ means faster environment change while a large τ means slow environment change.

It is important to note that by default environmental changes in the XOR DOP generator occur in a random pattern. Also in this thesis experiments were carried out in 2 additional dynamic environments; cyclic and cyclic with noise dynamic environments. Using the XOP DOP generator, a cyclic dynamic environment is generated as follows: a 2K XORing mask is randomly generated as the base states in the search space. This way the environment can cycle among the base states in a fixed logical cyclic order. Suppose an environment change occurs periodically every τ generations, then individuals at generation t are evaluated as follows:

$$f(\overrightarrow{x}, t) = f(\overrightarrow{x} \oplus \overrightarrow{M}(I_t)) = f(\overrightarrow{x} \oplus \overrightarrow{M}(k\%(2K))) \tag{4.4}$$

where $k = \lfloor \frac{t}{\tau} \rfloor$ is the index of the current environmental period and $I_t = k\%(2K)$ is the index of the base state the environment is in at generation t (please refer to Yang (2005b) for a detailed explanation).

The cyclic with noise environment used follows a deterministic approach as described in Yang and Yao (2008). Each time the environment is about to move from one base state to another (i.e $\overrightarrow{M}(i)$), a noise template $\overrightarrow{T}(n)$ with a small portion of 1s is randomly created and integrated into $\overrightarrow{M}'(i)$ i.e:

$$\overrightarrow{M}'(i) = \overrightarrow{M}(i) \oplus \overrightarrow{T}(n) \tag{4.5}$$

where $\overrightarrow{M}'(i)$ is the next new base state. The number of 1s in $\overrightarrow{T}(n)$ is set to be linear with the Hamming distance between base states i.e $\gamma \times \rho \times l$, $\gamma \in (0.0, 1.0)$.

In the next section, two popular benchmark problems are introduced to test the amcGA performance in dynamic environments. The benchmark problems are: 1.) Dynamic decomposable unitation-based functions and 2.) Dynamic knapsack problems.

4.1.1.1 Decomposable unitation-based functions (DUFs)

The decomposable unitation-based functions have been used as benchmark problems by the EA community in an attempt to study what creates difficult optimisation problems for EAs (Kaedi et al., 2013; Peng et al., 2011; Tinós and Yang, 2007a). This type of function return the total number of 1s in a binary string (i.e. unitation function of binary string). Three DUFs, denoted as DUF1, DUF2 and DUF3, are used as static problems to construct dynamic test environments, to evaluate the performance of algorithms discussed in chapter 3 and are described below:

DUF1:

DUF1 is a OneMax problem that simply aims to maximise the number of 1s in a binary string. The fitness of a binary string of length l is the total number of 1s contained in the string.

$$f_{DUF1} = u(x) \tag{4.6}$$

where $u(x)$ is the unitation function.

DUF2:

The DUF2 evaluates the performance of an EA on the level of building blocks (BB) interaction. The DUF2 consists of 4 bit BB, where an optimal solution consists of four sub-optimal solutions and the remaining solutions form a wide plateau with fitness value of zero. This property inherent in the DUF2 makes it a difficult problems for EAs (Fernandes et al., 2009; Kaedi et al., 2013).

$$f_{DUF2} = \begin{cases} 4 & \text{if } u(x) = 4, \quad (4.7a) \\ 2 & \text{if } u(x) = 3, \quad (4.7b) \\ 0 & \textit{otherwise} \quad (4.7c) \end{cases}$$

DUF3:

The DUF3 is a fully deceptive function, which is considered a hard optimisation problem for EAs. This is because the low-order BBs within the function do not combine to form a high-order BB, rather the low-order BBs combine to form a deceptive sub-optimal BB (Fernandes et al., 2009; Peng et al., 2011).

$$f_{DUF3} = \begin{cases} 3 - u(x) & \text{if } u(x) < 4 \quad (4.8a) \\ 4 & \textit{otherwise} \quad (4.8b) \end{cases}$$

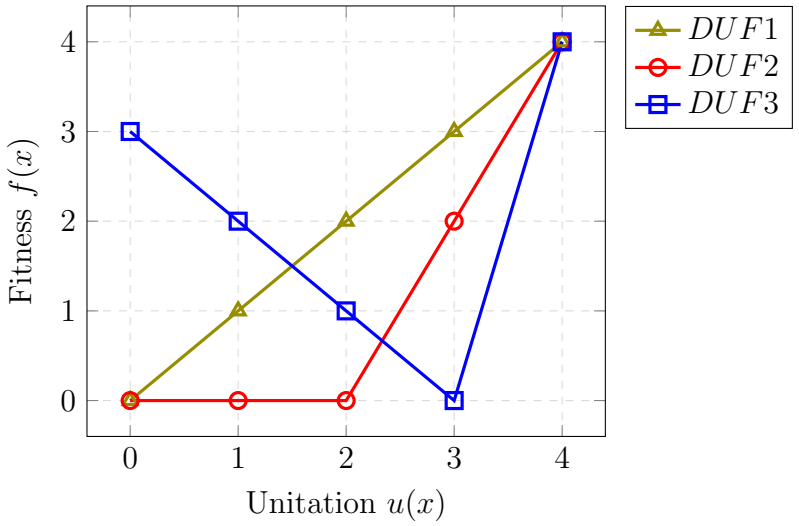


FIGURE 4.1: The building blocks used for the three DUFs

It is important to note that each DUF made up of 25 copies of 4-bit BB. Each BB contributes a maximum value of 4 to the total fitness value. Fig. 4.1 shows the BB of the selected DUFs based on Eq.(4.6),(4.7) and(4.8). Using the XOR DOP generator discussed, dynamic test environments are constructed based on the DUFs and are denoted as DDUF1, DDUF2 and DDUF3.

4.1.1.2 Dynamic knapsack problem (DKP)

The knapsack problem is a classic NP-complete optimisation problem that has been rigorously studied by the EA community in the last few decades (Puchinger et al., 2010; Li and Yang, 2008). The main aim of this problem is to fill a knapsack with the best subset of items among a larger set so as to maximize the value of contents in the knapsack without exceeding the knapsack capacity. This benchmark problem has been studied in both static (e.g. Puchinger et al. (2010); Zou et al. (2011)) and dynamic environments (e.g. Ben-Romdhane et al. (2013); Li and Yang (2008)) with different modifications. The dynamic property of the knapsack problem is achieved when the problem parameters (such as item weight, value and knapsack capacity) are time dependent and subject to variation.

Given n items, each of which has a weight $w_i(t)$ and a value $v_i(t)$ and a knapsack of capacity C. The main goal of the knapsack problem is to load the items that guarantees maximum value without exceeding the knapsack capacity C. A dynamic test environment is constructed for the knapsack problem and is denoted

as DKP. Mathematically DKP can be defined as follows:

$$Maximize\ f(x,t) = \sum_{i=1}^{n} p_i(t)x_i(t) \tag{4.9}$$

$$s.t. \begin{cases} \sum_{i=1}^{n} w_i(t)x_i(t) & \leq C \quad (4.10a) \\ x_i(t) \in \{0,1\}, & i = 1,...,n \quad (4.10b) \end{cases}$$

where x_i is the binary decision variable used to indicate if item i is included or discarded. In this thesis, all values and weights are positive, and all weights are less than the knapsack capacity C.

$$C = 0.45 \times \sum_{i=1}^{n} w_i \tag{4.11}$$

From Eq.(4.11), the knapsack capacity C of n items was set to 45% of the sum of all item weight. The test instance generator by Pisinger (1999) was used to generate the parameters of static K. Table 4.1 shows the global optima for the static phase of the K instances. In this thesis, optimality of the K is defined as the max fitness value obtained by an objective function without exceeding the constraint capacity.

TABLE 4.1: Knapsack problem instance and optimal fitness value

n-items	Total weight	Total profit
100	1646	1946

A knapsack problem with n items using randomly generated data was constructed as follows:

$$w_i = random\ integer[2,20] \tag{4.12}$$

$$p_i = random\ integer[1,30] \tag{4.13}$$

The total profit of items selected is the fitness of a candidate solution if the total item weight is within the knapsack capacity. However, if a candidate solution selects many items such that the total weight is more than the knapsack capacity, then a penalty function is used to judge how much the candidate solution exceeds

the knapsack capacity. The penalty is calculated as:

$$f(x,t) = \begin{cases} \sum_{i=1}^{n} p_i x_i & \sum_{i=1}^{n} w_i x_i \leq C \quad (4.14a) \\ f_p & \text{else} \quad (4.14b) \end{cases}$$

where $f_p = 10^{-7} \times \left(\sum_{i=1}^{n} w_i - \sum_{i=1}^{n} w_i x_i \right)$, similar to the penalty function used in Yang and Yao (2005). This penalty function ensures that any solution that exceeds C will not be competitive with solutions that do not.

4.1.1.3 Comparison of dynamic benchmark problems

These dynamic test problems have been chosen to evaluate the performance of the amcGA (and competing algorithms) under different dynamic scenarios. Specifically the DDUFs and DKP are academic combinatorial problems suitable for evaluating dynamic (binary-encoded) optimisation algorithms. The DDUFs has an increasing difficulty for the amcGA in the order from DDUF1 to DDUF2 to DDUF3. This is due to the deceptive property inherent in DDUF2 and DDUF3 which makes it a difficult problem for dynamic EAs. The DKP is another difficult problem, because the profit and weight of each item selected changes over time based on the XOR mask (i.e $\overrightarrow{M}$ in XOR DOP generator).

4.1.2 Parameter settings and performance measures

The XOR DOP generator is governed by different dynamic components, which may interact in a very complex way. The characteristics of the XOR DOP generator, the DDUFs & DKP, the change ratio ρ, the rate of change τ as well as the total number of runs, are all components which affect the behaviour of a dynamic optimisation algorithm. In order to cover a wide range of possible environment dynamics, different settings for each of these components have been considered so as to assess how well the amcGA performs.

Experiments were carried out on the selected DOPs in order to evaluate the effect of the change detection, change trend and mutation schemes on the performance of the amcGA. An additional experiment was carried out to compare the amcGA with a cGA with hypermutation denoted as mcGA, a GA with hypermutation denoted as GAm and a probability based incremental learning algorithm with hypermutation denoted as PBILm (Yang and Richter, 2009).

The GAm (Cobb (1990); Cobb and Grefenstette (1993)) makes use of an alternative mutation strategy (i.e. hypermutation) as a means of increasing diversity in order to track the optimum of a changing environment. The hypermutation in GAm, increases the probability of mutation whenever there is a degradation in the performance of elite solutions or timed-average best performance.

The PBILm is an estimation of distribution algorithm (EDA) that combines the mechanisms of the generational genetic algorithm with competitive learning. The hypermutation scheme within PBILm aims to maintain diversity. The mutation operation always alters and shifts the probability vector towards the central point in the search space (Yang and Richter, 2009). After a mutation process, solutions are sampled using the mutated probability vector and this mutation operation is repeated for a defined number of generations.

For all algorithms some common parameters have been obtained from Yang and Richter (2009); Tinós and Yang (2010) and some from preliminary experiments. The population size $n = 100$, speed of change $\tau = 20, 60$ and 100, and change ratio $\rho = 0.1, 0.2, 0.5$ and 1.0. The sensitivity level for detecting change is $\Delta f > 0$ for the amcGA described in Chapter 3, so as to detect any change since most of the DOPs (DDUF1-3) involves the maximization of 1s in a binary string (candidate solution).The probability of mutation for PBILm $p_m = 0.05$ with mutation shift $\delta = 0.05$, same as Yang and Richter (2009).

All algorithms use the elitism approach (in the case of PBILm an elite of size 1 was used). For the mcGA, GAm and PBILm the probability of mutation was set to a base level $p_m^l = 0.05$ and a high value $p_m^h = 0.3$ when the hypermutation scheme is triggered due to change in environment (i.e. drop in fitness of an elite solution) and this lasts for 5 generations ($g_{hm} = 5$). However, due to the DOP properties, these parameter settings may result in better or less performance. This is because the DDUFs and DKP have different optimal values and there is no rule that can be applied to find out the optimal parameter settings for a general problem. Best-of-generation fitness (see Eq.(2.2)) and modified offline performance were used as performance measures for all algorithms.

4.2 Torsional mass spring damper system (TMSDS)

A torsional mass-spring damper system (TMSDS) platform in a dynamic environment as shown in Fig. 4.2 was used to evaluate the amcGA described in chapter 3. The TMSDS is described in detail in this section to ensure understanding and reproducibility of the experiments and results.

The control of the mass-spring system is a classic control problem which is widely used as a testbench for testing different control and optimisation algorithms. There are many examples of the mass-spring damper system in practical real-world applications such as, tuned mass dampers in tall buildings, bridges and car suspension system which absorbs and prevents vibration from being transmitted to the car.

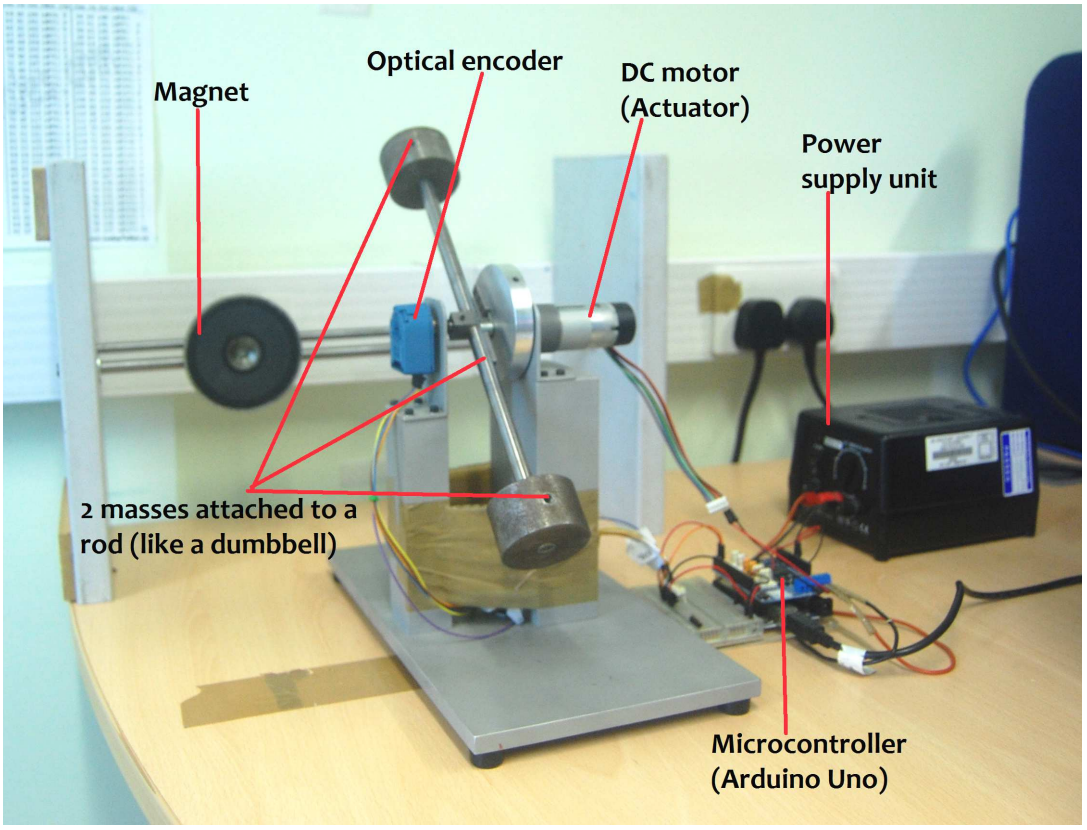


FIGURE 4.2: Experimental set-up

From Fig. 4.2 and 4.3, the TMSDS variables are: T = torque applied to the system, θ = angular position of the mass, k = spring constant, c = damping due to moving parts, I = moment of inertia of the mass, $\dot{\theta}$ = angular speed and $\ddot{\theta}$ = angular acceleration. Applying Newton's second law of rotation and Laplace

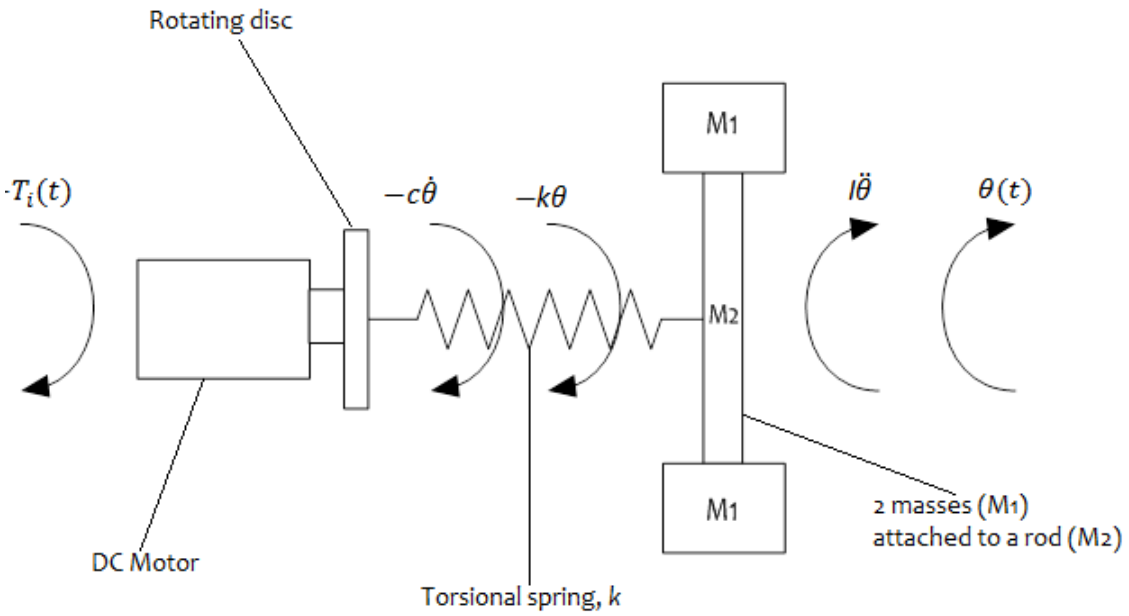


FIGURE 4.3: Diagram of TMSDS.
The spring is regarded linear although it has little non-linear properties

TABLE 4.2: Actual values of the TMSDS properties

Component	**Value**
Mass M1	0.15 kg
Mass M2	0.2062 kg
Rod length	0.3 m
Spring torque	310.805 Nmm
Spring constant, k	0.0989 $Nm/rads$
Spring max deflection	180°

transform assuming zero initial conditions. The transfer function of the TMSDS can be deduced as:

$$\frac{\theta(s)}{T_i(s)} = \frac{1}{Is^2 + cs + k} \tag{4.15}$$

where $T_i(s)$ is the system input and $\theta(s)$ is the system output.

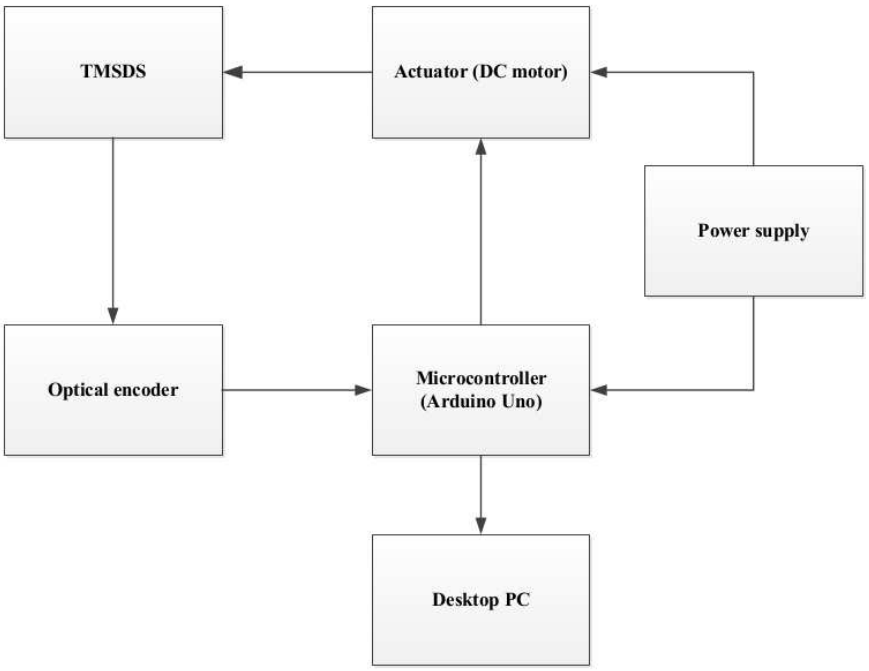


FIGURE 4.4: Overall structure of the TMSDS

4.2.1 TMSDS structure

The TMSDS provides a rotational oscillation. One end of a torsion spring is attached to a mass and the other end to a DC motor which acts as an actuator. An oscillatory behaviour is given through a mass that consists of two cylindrical blocks attached at opposite ends of a rod. Whenever the mass is displaced a certain angle, an optical encoder records the position and direction of rotation. Information from the optical encoder is sent to a micro-controller which processes the information and provides a calculated output to the actuator. The active damping required to dampen the oscillations of the TMSDS is provided by the actuator (i.e. DC motor). The micro-controller sends the telemetry to a host computer for further performance analysis (see Table 4.4). It is important to state that in this thesis all control actions and optimisation process are carried out using the micro-controller only so as to demonstrate the efficiency of the amcGA.

A strong magnet is used to cause interference which disrupts the mass-spring system from reaching a desired setpoint. This addition to the TMSDS, creates a dynamic problem (scenario). The magnet is placed exactly 1.5cm away from the mass to avoid sticking to the mass (see Fig. 4.2). This way the respective dynamic algorithm is expected to detect any change (the presence of the magnet)

and respond accordingly. The actual values of the properties of the TMSDS are shown in Table 4.2

4.2.2 TMSDS components

The TMSDS embedded system consists of individual parts which are: an optical encoder (sensor) and a micro-controller with a software implemented feedback controller. This section describes these components:

4.2.2.1 Optical encoder (sensor input)

The TMSDS makes use of an optical encoder as an input source to measure the position of the mass while the system is active. The encoder is an incremental optical encoder which belongs to the HEDS-5700 series, specifically the HEDS-5700 C10 (see Fig. 4.5). The HEDS-5700 C10 has two 90°phase shifted channels to provide direction and position information at 100 counts per revolution. Also, the sensor is a high performance, low cost optical encoder with mounted bushings and shafts. The HEDS-5700 C10 contains a special detection circuit that permits high resolution, encoding performance, reliability and long rotational life.

FIGURE 4.5: HEDS-5700 C10 optical encoder

4.2.2.2 Micro-controller

The micro-controller is an integral part of the TMSDS that processes information received from sensor and provides a calculated output to the actuator. All control

actions and optimisation process run only on the micro-controller. The micro-controller used for all experiments is the Arduino Uno (see Fig. 4.6) which is an open-source, low cost electronics prototyping platform based on an ATmega328, clocked at 16MHz, powered at 5V and contains 32KB of flash memory of which 0.5KB is used by the boot-loader. This micro-controller was chosen to identify possible limitations of the amcGA and to serve as a proof that the amcGA is capable of solving real-time, real-world DOPs.

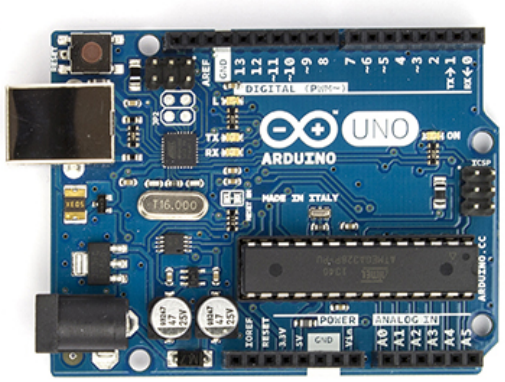


FIGURE 4.6: Arduino Uno micro-controller

4.2.2.3 Actuator (DC motor)

An EGM30 12V DC motor was used as an actuator to dampen the oscillation of the TMSDS (see Fig. 4.7). This DC motor is made up of a 30:1 reduction gearbox and an encoder. The actuator has standard noise suppression capacitor across the motor windings (see Gonçalves et al. (2013)). The encoder within the actuator was not used in this application although it can be used to determine the position of the DC motor.

4.2.2.4 Controller

There exists different feedback control techniques but the most common among all is the Proportional Integral Derivative (PID) controller. Despite having been around for a long time, majority of the industrial applications still use PID controllers because of its simplicity and ease of use.

PID controllers continue to survive the evolution of technology, starting from pneumatics and mechanic to microprocessors. Over the past decades, implementations

FIGURE 4.7: EMG30 12V DC motor

on microprocessors continues to increase and this has given opportunities to provide additional features like continuous adaptation, auto-tuning and gain scheduling. The PID control is an active feedback method that is implemented using a combination of Proportional, Integral and Derivative controllers (see Fig.4.8 and Table 4.2.1).

Combinations of these controllers are frequently used as proportional derivative (PD) controller e.g. Gao et al. (2014) and proportional integral (PI) controller e.g. Roy et al. (2012). Below are equations describing the PID control system:

$$Proportional = k_p e(t) \tag{4.16}$$

$$Integral = k_i \int_0^t e(t)dt \tag{4.17}$$

$$Derivative = k_d \frac{de(t)}{dt} \tag{4.18}$$

$$u(t) = k_p e(t) + k_i \int_0^t e(t)dt + k_d \frac{de(t)}{dt} \tag{4.19}$$

where k_p, k_i and k_d are the proportional, integral and derivative gain respectively, and

$$e(t) = pv_t - sp \tag{4.20}$$

where $e(t)$ is the measure of error at time t, pv_t is the processes measurement at time t and sp is a desired setpoint. PID functionalities can be summarized as follows:

- The proportional controller (P) responds to an error by adjusting the overall control action proportional to the error.
- The integral controller (I) reduces steady-state error taking into account the duration of the error.
- The derivative controller (D) generates a response using the change in error from a previous error value.

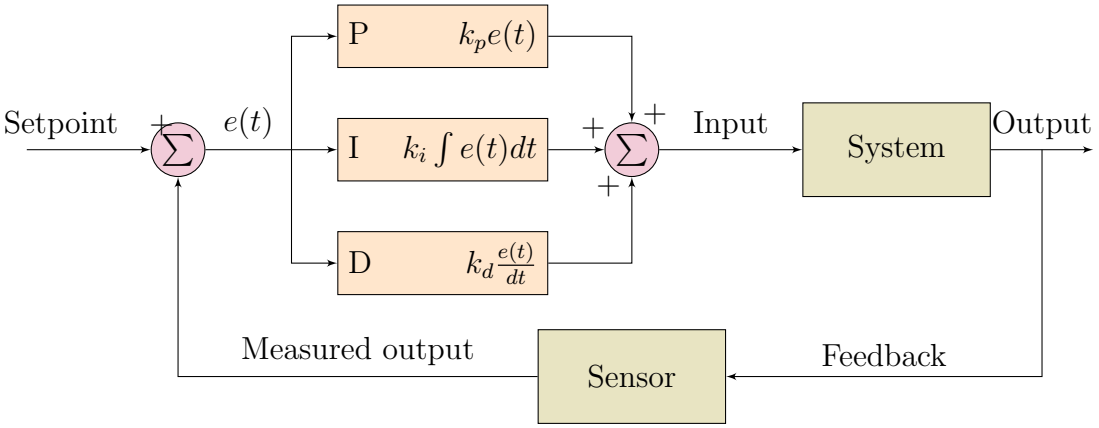


FIGURE 4.8: Overview of the Proportional-Integral-Derivative (PID) Controller

For a PID controller to work in the right way, it is important that its gains are tuned properly. Tuning PID controllers has been an active area in the research community (e.g. Caraffini et al. (2013); Passow and Gongora (2008)). Great effort have been made to develop methods to reduce the time spent on optimising the choice of PID parameters. There are various types of techniques applied for PID tuning which are broadly classified as classical (such as Ziegler-Nichols) and computational techniques such as EAs (Iruthayarajan and Baskar, 2009; Passow et al., 2008). However, one disadvantage of the classical method is the necessary prior knowledge of the process and once tuned the controller performs good but optimum system response is not always achieved.

Experiments were carried out using the TMSDS to evaluate the performance of the amcGA on the optimisation of an integral-state PID controller. The integral-state PID controller is a variant of the PID controller that limits the integral state variable to a lower (I_l) and upper (I_u) bound values (Wescott, 2000). The bound values permits the mitigation of undesired overshoots from integral windup (if the bound values are properly selected).

Algorithm 4.2.1: *Intergral-state PID controller* (*setpoint*, *process*)

```
integral error ← 0
previous error ← 0
while process active
      do { error ← setpoint − process
           integral error ← integral error + error
           if integral error > integral max
             then {integral error ← integral max
           if integral error < integral min
             then {integral error ← integral min
           derivative error ← error − previous error
           P out ← error × proportional gain
           I out ← integral error × integral gain
           D out ← derivative error × derivative gain
           output ← P out + I out + D out
           previous error ← error
```

TABLE 4.3: Pseudo-code of integral-state PID controller

4.2.3 Parameter settings and performance measures

Based on preliminary experiments, parameters of the amcGA (for all experiments) were set as follows: the population size $s = 100$, speed of change $\tau = 20$, binary string length l of candidate solution $= 24bits$ and the sensitivity level (or threshold) for detecting change was $\Delta f > 1.0$ for the change detection scheme described in Chapter 3. All candidate solutions are created with chromosomes within the range specified in Table 4.4 and the summary of parameter configuration are shown in Table 4.5. Also a block illustrating the dynamic evolutionary controller is shown in Fig. 4.9.

4.2.3.1 Solution encoding and decoding

The amcGA candidate solutions represents the control parameters. The control parameter consists of the P,I and D gain parameters. Each candidate solution is

encoded as a binary string of length $l = n \times m$. Where $n = 3$ is the total number of variables, $m = 8bits$ is the binary string length of each variable.

Mapping of the binary string to a real value x is realised in two steps:

- Convert the binary representation b of each parameter from base 2 to base 10:
$$x^{'} = \left(\sum_{i=0}^{m} b_i \times 2^i \right)_{10} \tag{4.21}$$
- Solve for the corresponding real value based on domain by using the equation below:
$$x = x_l + x^{'} \times \left(\frac{x_h}{2^m - 1} \right) \tag{4.22}$$
where x_l and x_h represent domain boundaries of each gain parameter (see Table 4.4).

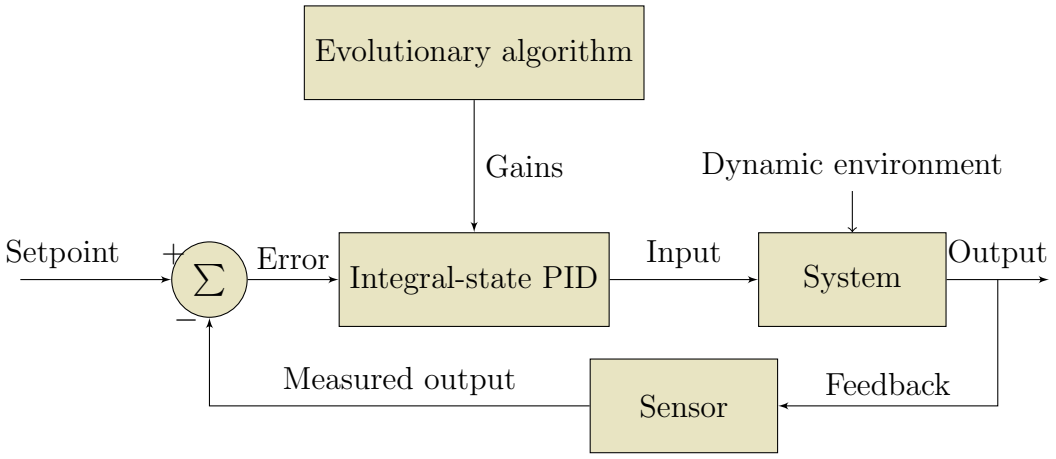


FIGURE 4.9: Overview of the dynamic evolutionary feedback controller loop

TABLE 4.4: PID parameter value range (in continuous domain)

Parameter	**Domain**
Proportional gain	0 - 10.00
Integral gain	0 - 10.00
Integral state max	100
Integral state min	-100
Derivative gain	0 - 4.00

The rotation of the torsional mass-spring system was restricted to turn between -40 and 40 degrees from its middle position 0. The evaluation of each candidate solution took approximately 20 seconds. Each solution was tested by perturbing

TABLE 4.5: Summary of configuration of parameters of the competing algorithms

	amcGA1-5	**mcGA**	**r-rcGA**	**r-cDE**
Encoding scheme	Binary	Binary	Real-valued	Real-valued
Population size	100	100	100	100
Dynamic strategy	Adaptive mutation	Hyper-mutation	Restart	Restart

the mass to each side and analysing the controller's reaction to reach a setpoint (-20 and 20). The setpoint of the torsional mass-spring system is initially set to -20 degrees, at this point the controller takes over and tries to maintain the setpoint. After 500 control cycles, the setpoint of the system is set to 20 degrees for another 500 control cycles while the controller continues to maintain the new setpoint. This set-up enables the automatic implementation and evaluation of candidate solutions on the testbench while the amcGA monitors the environment for any changes.

Each candidate solution generated by the amcGA is evaluated using the dynamic testbench described rather than in a simulated environment. An additional experiment was carried out to compare the performance of the amcGA schemes with a real-valued cGA with a restart scheme (and the elitism method) denoted as r-rcGA where the probability vector is re-initialized whenever the environment changes, a compact differential evolution with a restart scheme and elitism method denoted as r-cDE and a cGA with a hypermutation scheme denoted as mcGA. For the mcGA , the probability of mutation was set to a base level $p_m^l = 0.01$ for normal generations and a high $p_m^h = 0.3$ for interim generations when the hypermutation scheme is triggered due to change in environment (which lasts for 5 generations).

It is important to state that in this thesis, the amcGA is compared to the above mentioned compact optimisation algorithm in order to evaluate memory consumption and performance in a dynamic environment. Others have compared the cGA with the GA and proved that the cGA is computationally more efficient (in terms of memory requirements) than the GA and suitable for memory constrained applications (please refer to Ahn and Ramakrishna (2003); Mininno et al. (2008)).

For each experimental run, the root mean square error (RMSE) associated with the respective algorithm is calculated. This gives a measure of the dynamic tuning of the PID controller for that particular run. The variation in the RMSE gives a measure of the consistency of the evolved controller performance. Therefore, the following two performance measures were used to assess the properties of each dynamically tuned controller performance:

- RMSE. The ability to maintain the setpoint which is defined as:

$$RMSE = \sqrt{\frac{\sum_{i=0}^{t} e(t)^2}{t}} \tag{4.23}$$

$$e(t) = pv_t - sp \tag{4.24}$$

where t is the control cycle, $e(t)$ is the measure of error at control cycle t, pv_t is the angle of the mass at control cycle t and sp is the setpoint. From the above equation, it can be observed that this is a minimisation problem.

- Best-of-generation performance measure (Eq. 2.2) was evaluated every generation as well as the overall performance of all algorithms. For each experiment using all algorithms on the testbench, 30 independent runs were executed and for each run 10 dynamic changes were permitted which is equivalent to 200 generations. Parameters of Eq. 2.2 were set as follows: FBOG_{ij} expresses the fitness value of the best solution at generation i of run j and $G = 20$ x τ (i.e. $\tau = 10$) is the total number of generation for each run.

4.3 Summary

This chapter presented and discussed the set-up scene for all experiments including a control optimisation problem in a dynamic environment using a physical system i.e. the TMSDS. These test problems were chosen to verify the impact of the schemes described in chapter 3 on the performance of the amcGA i.e. the efficiency of the change detection scheme in regulating the mutation rate of the amcGA. More specifically, these tests were chosen to verify if a dynamic optimisation algorithm would take advantage of simulation discrepancies to achieve unrealistic performance and if the same algorithm would show similar performance when evaluated using a physical system (such as TMSDS).

It is important to state that experiments using the TMSDS were carried out using only the Arduino micro-controller. All software implementation of the PID controller, competing dynamic optimisation algorithms and control of TMSDS were written in C programming language using the Arduino IDE and uploaded onto the Arduino micro-controller through a serial port. Experiments using the XOR DOP generator, DOPs and competing dynamic optimisation algorithms were written in C language and simulation carried out using a desktop PC.

The results obtained from all experiments are presented, analysed and discussed in the next chapter.

Chapter 5

Analysis and Discussion

5.1 Analysis of XOR DOP experiments

The experimental and statistical test results on the DOPs described in Section 4.1.1 are shown in Figures 5.1- 5.6 and Tables 5.1- 5.4 respectively. In general, evolutionary algorithms act as a stochastic process which yields a solution expected to be as close as possible to an optimal solution. Therefore it is important to provide a thorough analysis of the algorithms performance statistically. The Kruskal-Wallis tests followed by pairwise comparisons using Wilcoxon rank-sum tests with the Bonferroni correction was used to test for the null hypothesis whether or not there are significant differences in the ability of the dynamic algorithms to tackle the XOR DOPs. The statistical test results at 0.05 level of significance are shown in Tables 5.1 - 5.4. Performance of all algorithms are analysed based on overall performance and behaviour on the selected XOR DOPs.

5.1.1 Experimental study regarding overall performance

From all figures and tables in this section, several behaviours can be observed: First, GAm and PBILm show a constant performance across all DOPs regardless of the dynamics of the environment. This is because PBILm and GAm evaluate 100 candidate solutions every generation and have a greater chance of finding better solutions than the mcGA and amcGAs, which only evaluate 2 candidate solutions every generation. With the increasing of τ, GAm and PBILm has more time to search for solutions with higher fitness values before the next dynamic change. However, in environment with a high change ratio ρ, PBILm was outperformed by

the amcGA (variants) as can be observed from Fig. 5.1-5.3. This is as a result of the lack of information transfer from the last environment of the last dynamic change. Also, PBILm applies the mutation scheme to the current working $\overrightarrow{P}$ which has no information of the previous environment, this means PBILm is focused more on preventing premature convergence of $\overrightarrow{P}$. Therefore, it is important to state that this comparison was carried out in order to see which of the competing compact dynamic algorithm was less behind the PBILm and GAm in terms of the overall performance.

Second, mcGA outperforms some of the amcGA variants in some of the DOPs. This is due to the fact that whenever a change occurs, mcGA tries to find a better solution for the current environment (which is the effect of rapid increase in the probability of mutation p_m) but does not ensure diversity as seen in Fig. 5.1(a). Also for some dynamic settings, mcGA shows similar performance to some of the amcGA variants. Given a value of τ when the environment changes with respect to the change ratio, the performance of the algorithms can be considered similar (see Figs. 5.1(b)- 5.2(c)).

Third, among the five variants of the amcGA, amcGA1 and amcGA3 exhibit high performance. From Fig. 5.1(a)- 5.6, amcGA3 shows stable performance across different environment dynamics. This is as a result of how the mutation scheme within both algorithms alter $\overrightarrow{mP}$. The amcGA3 mutation scheme either increase or decrease $\overrightarrow{mP}$ at a reduced scale (see Eq. 3.7), while amcGA1 is a randomised mutation based on the current values of elements of $\overrightarrow{mP}$ and the degree of change c_d (see Eq. 3.5). Performance of all amcGA variants based on $\overline{F}_{BOG}$ is shown on Figs. 5.1(a)-5.3(c) for different environment dynamics. Although Figs. 5.1(a)-5.3(c) shows general performance of all algorithms, it is difficult to draw out conclusions about the final result of the compared algorithms by just visual inspection of the performance curves. Using the Wilcoxon rank-sum test, it can be observed that the performance of the amcGA1 and amcGA3 achieved higher fitness than that of the mcGA. In addition to this when the change ratio ρ [1] is set to high i.e 1.0 (see Fig. 5.5) and speed of change τ is set to low and medium in some environments, most of the amcGA variant outperformed both mcGA and PBILm. This behaviour is a result of how the amcGA handles $\overrightarrow{P}$.

[1]From the description of the XOR DOP in chapter 4, it can be observed that XOR DOP generator creates stationary environments when $\rho = 0.0$. However, when $\rho = 1.0$, respective DOPs shifts between two environments. This implies that the intermediate binary template $\overrightarrow{T}$ (Eq. 4.3) is randomly generated and for each $\overrightarrow{T}$ generated the environment is different (i.e. for $0.0<\rho<1.0$). This property is common amongst all DOPs generated using the XOR DOP generator.

The amcGA maintains a moderate convergence rate as it explores the search space. This can be considered as an advantage over the hypermutation scheme since the amcGA not only carries information from one stage of the problem to the next stage but also retains this information in the form of $\overrightarrow{mP}$, which represents properties and dynamics of a particular environment. Since the mutation scheme is only applied to $\overrightarrow{mP}$, it ensures that the current working $\overrightarrow{P}$ maintains its diversity unless a solution generated by the mutated $\overrightarrow{mP}$ (whenever a change is detected) outperforms the current best solution generated by $\overrightarrow{P}$, therefore replacing $\overrightarrow{P}$ with $\overrightarrow{mP}$. This can be considered as an advantage over the hypermutation scheme. The $\overrightarrow{P}$ in mcGA is mainly updated based on the solutions sampled from it. This implies that genetic diversity is encouraged at an individual level since the mutation scheme is applied directly to a candidate solution to create another for selection.

Finally, the performance of the amcGA4 and amcGA5 in the cyclic environment is similar to that of the mcGA on the DDUF1, DDUF2 and DKP (see Figs. 5.1(c), 5.2(b) and 5.4(a)). This behaviour is the result of the change trend scheme within the algorithm. Although this scheme does not make use of any external training data, it has a positive effect on the performance of the amcGA4 and amcGA5. The change trend scheme ensures that the amcGA retains information about past environments (i.e. $\overrightarrow{mP}$) while searching for promising region (using $\overrightarrow{P}$) in the search space of a new environment. This can be observed when $\tau = 100$ and ρ is between 0.1 and 0.5 (see Table 5.1- 5.4), the algorithms are given more time to explore the search space before the next dynamic change, but experience slow convergence rate.

On the other hand, convergence deprives mcGA of the adaptability to changing environments because the $\overrightarrow{P}$ within mcGA learns from the best hyper-mutated solution whenever a change occurs. However, the mutation mechanism and change trend scheme embedded in amcGA4 and amcGA5 grants more diversity than mcGA (and PBILm), thus better adaptability to dynamic changes changes (see Fig. 5.4(a)-(c)). The change trend scheme within amcGA4 and amcGA5 makes use of change patterns exhibited by the current working probability vector (i.e. $\overrightarrow{P}$) to alter $\overrightarrow{mP}$. This way, both amcGA4 and amcGA5 responds to dynamic changes based on how often new elites are obtained.

5.1.2 Analysis of algorithms' behaviour on selected DOPs

In order to have a clear understanding of the experimental results, we need to look further into the dynamic behaviour of all algorithms. The dynamic behaviour of all algorithms on the selected DOPs are shown in Figs. 5.5- 5.6, where the data were averaged over 30 runs, τ is set to 60 and $\rho = 0.1$, 0.2, 0.5 and 1.0. Several behaviours can be observed when analysing the effect of the dynamic environments on the performance of the competing algorithms.

From Figs. 5.5- 5.6, it can be observed that for a fixed τ with increasing value of ρ, PBILm and GAm outperform other algorithms on several cases and maintains almost the same performance across the four DOPs. The behaviour is a result of the high adaptability brought in by the hypermutation scheme (and population-based structure) within PBILm. However, the performance of GAm and PBILm decreases on the cyclic DDUF2, cyclic DDUF3, random DDUF2 and random DDUF3. This is due to the fact that when a dynamic change occurs, the deceptive BBs inside DDUF2 and DDUF3 draws the population into the new environment slowly. This is because the deceptive attractors are suboptimal with relatively high fitness (Fernandes et al., 2009).

An interesting behaviour is that on DDUF1, the performance of the amcGA variants drops when ρ is between 0.1 and 0.5 but soon stabilizes. This is because when $\rho = 1.0$, the environment changes between two landscapes. This makes it difficult for the algorithm to converge well since the algorithm may wait during one environment state. Also, among the amcGA variants, the amcGA3 shows high performance in DDUF1. The reason for this lies in the way the mutation scheme operates within amcGA, it ensures that the amcGA3 adapts to the changing environment regardless of the change severity. The mutation scheme only increases (or decreases) $\overrightarrow{mP}$ by $\left(r - \frac{p_m}{2}\right)$ which is determined by a random number r unlike the mutation scheme in PBILm which is determined by the probabilities in $\overrightarrow{P}$.

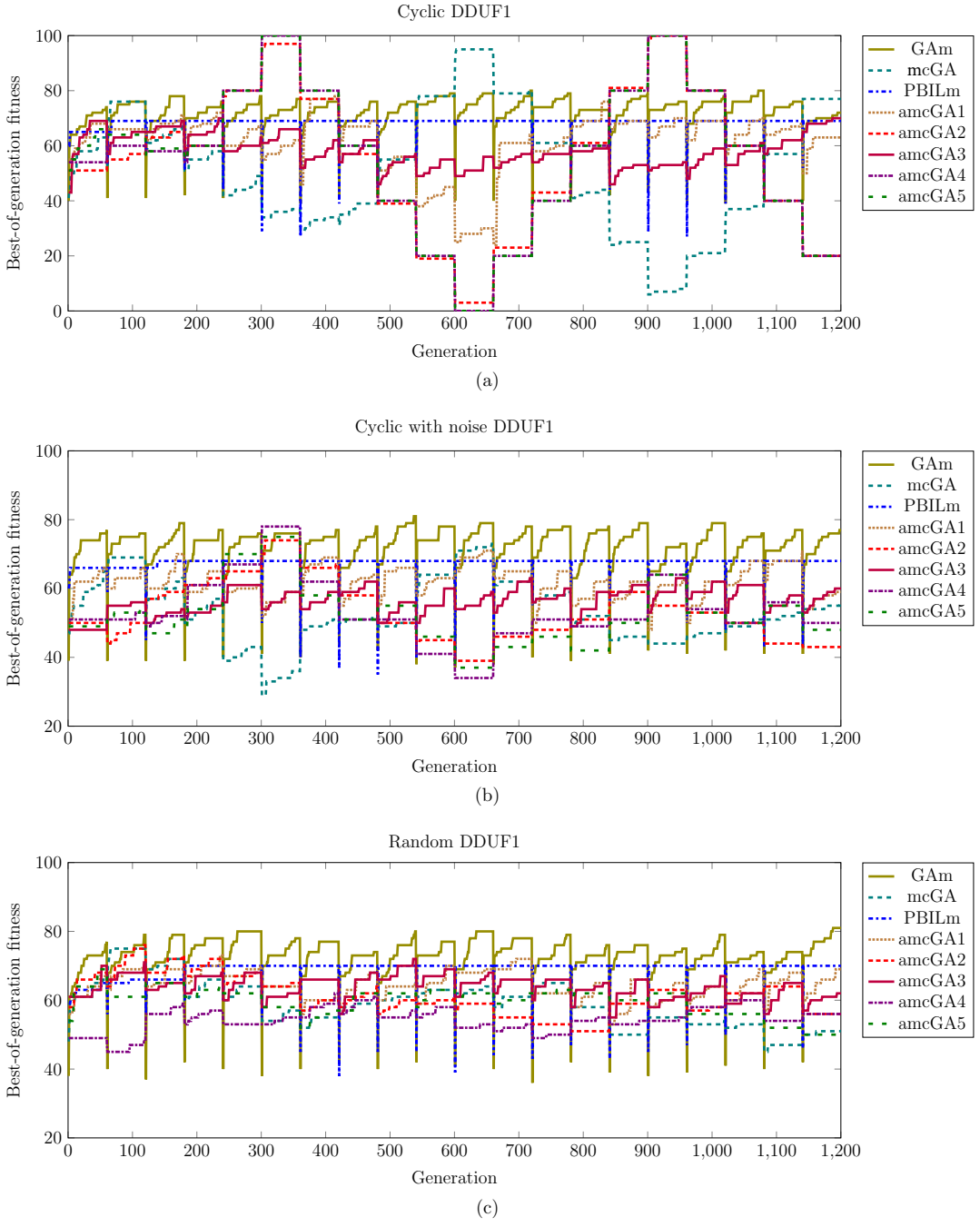


FIGURE 5.1: Dynamic performance of all algorithms on DDUF1 when $\tau = 60$ and $\rho = 0.2$ in three different dynamic environments. Best fitness values achieved each generation are shown in the figure, where the optimum is 100. Please refer to Appendix C for the individual dynamic behaviour of competing algorithms.

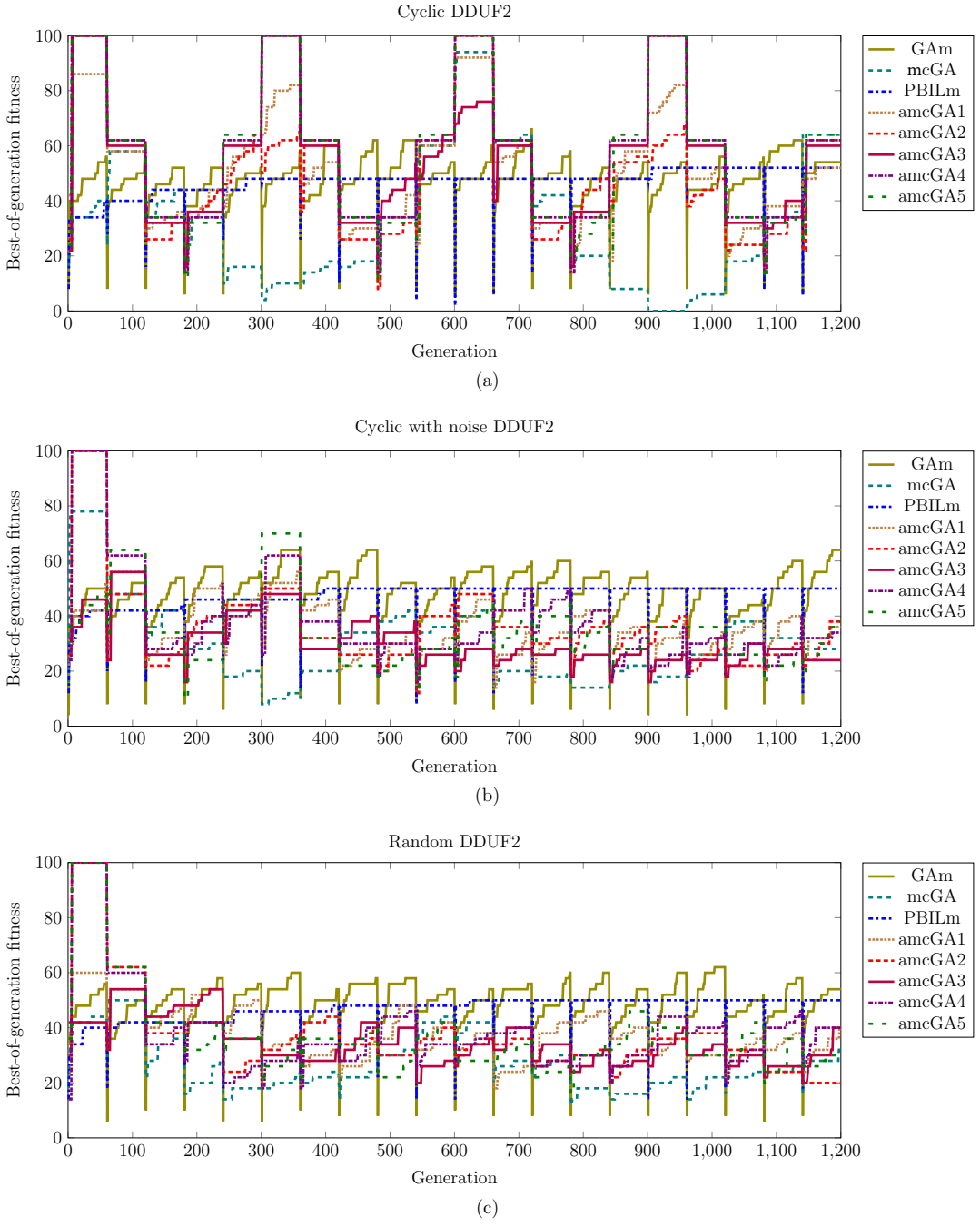


FIGURE 5.2: Dynamic performance of all algorithms on DDUF2 when $\tau = 60$ and $\rho = 0.2$ in a three different dynamic environments. Best fitness values achieved each generation are shown in the figure, where the optimum is 100. Please refer to Appendix C for the individual dynamic behaviour of competing algorithms.

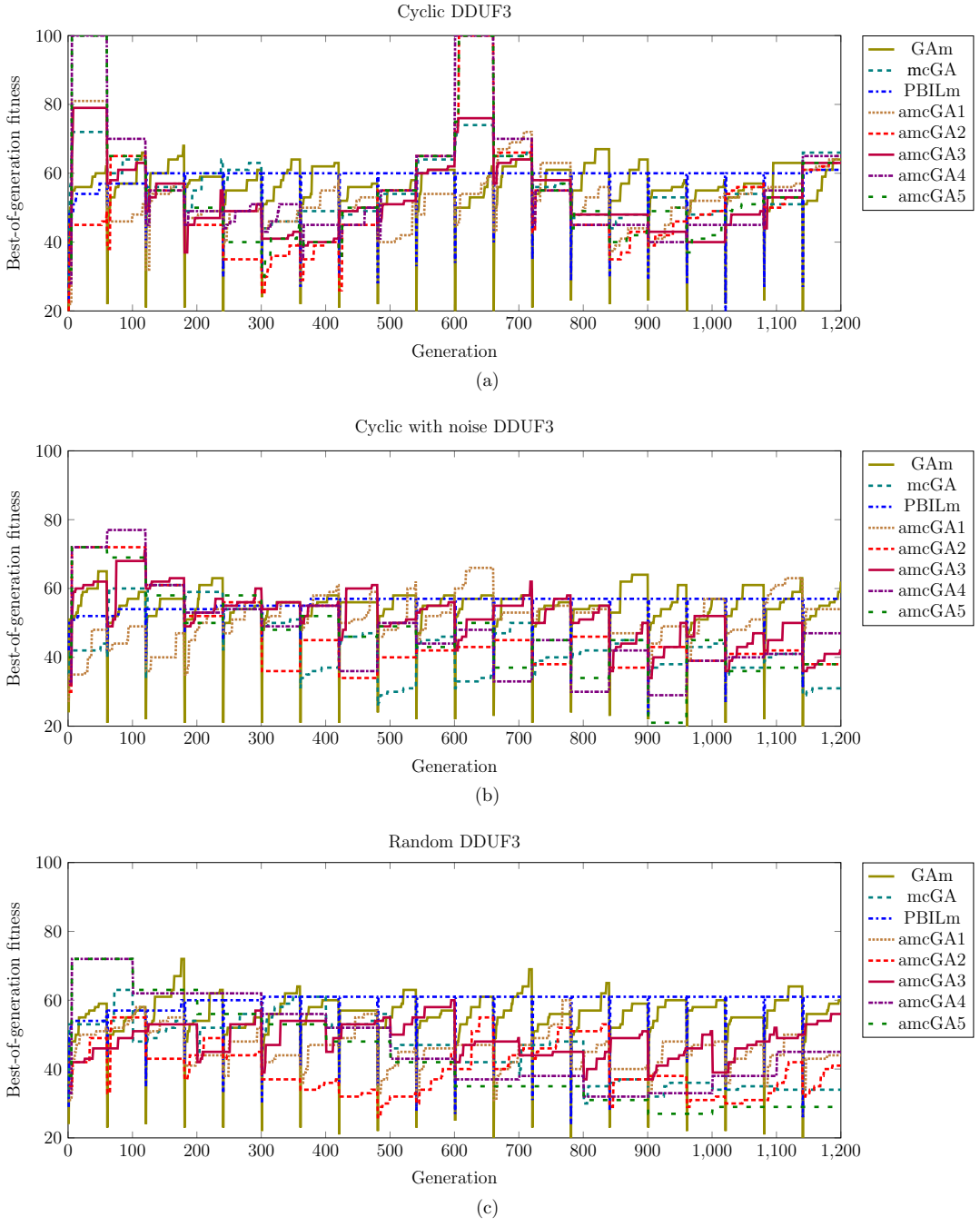


FIGURE 5.3: Dynamic performance of all algorithms on DDUF3 when $\tau = 60$ and $\rho = 0.2$ in a three different dynamic environments. Best fitness values achieved each generation are shown in the figure, where the optimum is 100. Please refer to Appendix C for the individual dynamic behaviour of competing algorithms.

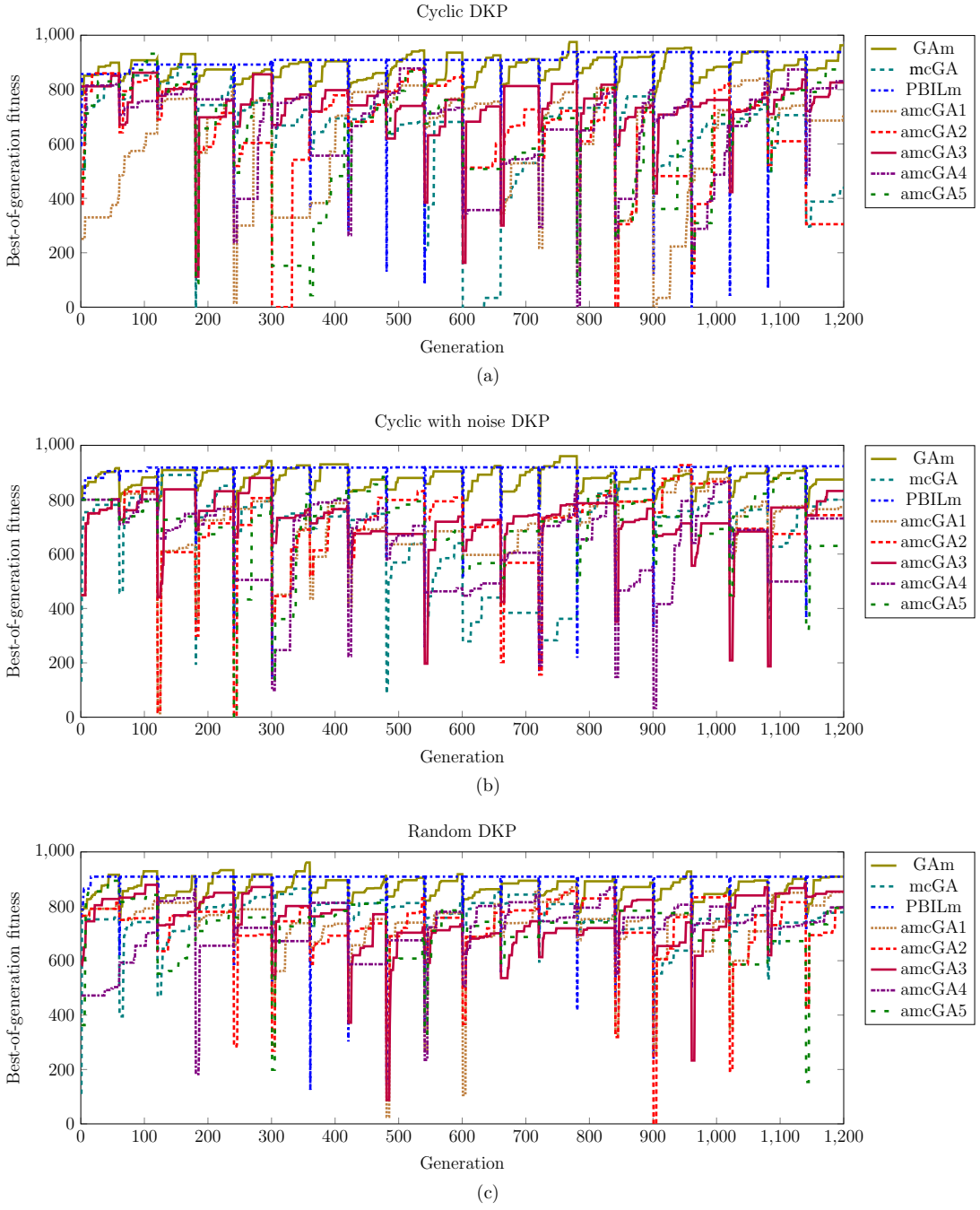


FIGURE 5.4: Dynamic performance of all algorithms on DKP when $\tau = 60$ and $\rho = 0.2$ in a three different dynamic environments. Best fitness values achieved each generation are shown in the figure, where the optimum is 1000. Please refer to Appendix C for the individual dynamic behaviour of competing algorithms.

TABLE 5.1: Statistical results regarding the offline performance of the amcGA variants against other algorithms on the DDUFs in a cyclic environment. "+", "−" and "∼" indicates that Algorithm 1 is better than, worse than or statistically equivalent to Algorithm 2 (Algorithm 1 − Algorithm 2).

Algorithms and DOPs	**DDUF1**																**DDUF2**												**DDUF3**							
Environment dynamics	$\tau = 20$				$\tau = 60$				$\tau = 100$				$\tau = 20$				$\tau = 60$				$\tau = 100$				$\tau = 20$				$\tau = 60$				$\tau = 100$			
Cyclic, $\rho \Rightarrow$	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**
amcGA1 - GAm	−	−	−	−	−	∼	−	−	−	−	−	∼	−	−	−	−	−	−	−	∼	−	−	−	∼	∼	−	−	−	−	∼	−	−	−	∼	−	−
amcGA1 - mcGA	+	+	∼	∼	+	+	∼	∼	+	+	+	∼	+	+	+	∼	+	+	+	+	∼	∼	+	+	∼	∼	∼	+	∼	∼	−	+	+	∼	∼	∼
amcGA1 - PBILm	−	−	−	−	∼	−	−	−	∼	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	+	∼	−	∼	+	∼	−	∼	+
amcGA2 - GAm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	−	−	−	−	−	−	∼	−
amcGA2 - mcGA	+	∼	∼	∼	+	+	∼	∼	+	+	∼	∼	+	+	+	∼	∼	∼	∼	+	∼	∼	∼	∼	∼	∼	∼	+	−	∼	∼	+	∼	∼	∼	∼
amcGA2 - PBILm	−	−	−	−	∼	−	−	−	−	−	−	−	−	−	−	∼	−	−	−	−	−	∼	−	−	−	−	∼	+	−	−	∼	+	−	−	+	+
amcGA3 - GAm	−	−	∼	−	−	−	∼	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA3 - mcGA	∼	∼	∼	∼	+	+	∼	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	−	−	−	∼	∼	∼	−	−	∼	∼	−	−
amcGA3 - PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	−	−	∼	−	−	−	∼	−	−	−	−	−	−	+	∼	−	−	+	−	−	−	+
amcGA4 - GAm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	−	−	−	−	−	−	−	−	−	−	∼	−	−	−	−	−	−	−	−
amcGA4 - mcGA	+	∼	∼	−	+	+	∼	∼	+	+	−	−	+	∼	∼	+	∼	+	∼	∼	+	+	∼	+	∼	∼	∼	+	∼	∼	∼	+	∼	∼	∼	∼
amcGA4 - PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	+	−	−	∼	+	−	−	∼	+
amcGA5 - GAm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	∼	−	−	−	−	∼
amcGA5 - mcGA	+	∼	∼	−	+	+	∼	∼	+	+	−	−	+	+	∼	∼	∼	+	∼	∼	+	−	∼	+	∼	∼	∼	+	∼	∼	∼	+	∼	∼	∼	∼
amcGA5 - PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	−	−	−	−	−	−	−	−	−	−	−	∼	−	+	−	−	∼	+	−	−	∼	+

TABLE 5.2: Statistical results regarding the offline performance of the amcGA variants against other algorithms on the DDUFs in a cyclic with noise environment. "+", "−" and "∼" indicates that Algorithm 1 is better than, worse than or statistically equivalent to Algorithm 2 (Algorithm 1 − Algorithm 2).

Algorithms and DOPs	$DDUF1$																																			
Environment dynamics	$\tau = 20$				$\tau = 60$				$\tau = 100$				$\tau = 20$				$\tau = 60$				$\tau = 100$				$\tau = 20$				$\tau = 60$				$\tau = 100$			
Cyclic with noise, $\rho \Rightarrow$	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**
amcGA1 - GAm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA1 - mcGA	+	+	∼	−	+	+	+	∼	+	+	+	+	+	∼	∼	∼	∼	∼	−	+	∼	∼	∼	∼	∼	∼	∼	−	+	∼	∼	+	+	+	+	+
amcGA1 - PBILm	−	−	−	−	−	−	−	−	∼	∼	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA2 - GAm	−	−	−	−	−	∼	−	−	−	−	∼	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA2 - mcGA	+	+	∼	−	+	∼	∼	∼	+	∼	∼	∼	∼	+	∼	+	−	−	∼	+	−	∼	∼	−	−	∼	∼	−	∼	∼	∼	+	+	∼	∼	∼
amcGA2 - PBILm	−	−	−	−	−	−	−	−	∼	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA3 - GAm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	−	−	−	−	−	−	∼	−
amcGA3 - mcGA	∼	∼	∼	−	+	+	+	+	+	+	+	+	∼	+	+	∼	∼	∼	+	+	∼	∼	+	+	∼	−	∼	−	−	−	−	−	−	+	+	+
amcGA3 - PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−
amcGA4 - GAm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	−	−	−	−	−	−	−	−
amcGA4 - mcGA	∼	∼	−	−	∼	∼	∼	∼	+	∼	∼	∼	∼	∼	∼	−	−	−	∼	+	∼	∼	∼	+	∼	∼	∼	+	∼	∼	∼	+	∼	∼	∼	∼
amcGA4 - PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	+	−	−	∼	+	−	−	∼	+
amcGA5 - GAm	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	−	−	∼	−	−	−	−	−	−	∼	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA5 - mcGA	∼	∼	−	−	∼	∼	∼	∼	+	∼	∼	∼	+	+	∼	∼	∼	+	∼	∼	∼	+	+	+	−	−	−	−	−	−	−	−	−	−	−	−	∼
amcGA5 - PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−

TABLE 5.3: Statistical results regarding the offline performance of the amcGA variants against other algorithms on the DDUFs in a random environment. "+", "−" and "∼" indicates that Algorithm 1 is better than, worse than or statistically equivalent to Algorithm 2 (Algorithm 1 − Algorithm 2).

Algorithms and DOPs	$DDUF1$																																	
Environment dynamics	$\tau = 20$				$\tau = 60$				$\tau = 100$				$\tau = 20$				$\tau = 60$				$\tau = 100$				$\tau = 20$				$\tau = 60$				$\tau = 100$			
Random, $\rho \Rightarrow$	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**
amcGA1 - GAm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA1 - mcGA	−	∼	∼	∼	+	+	+	−	+	+	+	+	+	∼	∼	∼	∼	∼	∼	∼	∼	+	−	∼	∼	∼	∼	+	+	+	∼	+	+	+	+	∼
amcGA1 - PBILm	−	−	−	−	∼	−	−	−	+	∼	∼	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	+	−	−	−	+
amcGA2 - GAm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA2 - mcGA	−	−	∼	∼	∼	∼	−	−	∼	+	+	+	∼	∼	∼	∼	∼	∼	+	+	+	+	∼	∼	∼	−	∼	+	∼	−	−	+	∼	∼	−	∼
amcGA2 - PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	+	−	−	−	+
amcGA3 - GAm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA3 - mcGA	−	∼	∼	∼	+	+	+	∼	+	+	+	+	+	+	+	+	∼	+	∼	+	+	+	+	+	+	∼	−	∼	+	∼	∼	∼	+	+	+	−
amcGA3 - PBILm	−	−	−	−	∼	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	+	−	−	−	+
amcGA4 - GAm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	∼	−	−
amcGA4 - mcGA	+	∼	∼	−	+	+	∼	∼	+	+	−	−	+	∼	∼	∼	∼	+	−	∼	−	+	+	∼	∼	∼	∼	+	∼	∼	∼	+	∼	∼	∼	∼
amcGA4 - PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	+	−	−	∼	+	−	−	∼	+
amcGA5 - GAm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	−	−	−	−	−	−	+
amcGA5 - mcGA	−	−	−	−	∼	−	−	−	∼	∼	∼	+	∼	∼	∼	∼	∼	+	−	∼	∼	∼	+	∼	∼	−	−	+	∼	−	−	+	∼	−	−	−
amcGA5 - PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	+	−	−	−	−

TABLE 5.4: Statistical results regarding the offline performance of the amcGA variants against other algorithms on the DKP in different environments. "+", "−" and "∼" indicates that Algorithm 1 is better than, worse than or statistically equivalent to Algorithm 2 (Algorithm 1 − Algorithm 2).

Algorithms and DOPs	**Cyclic DKP**																																			
Environment dynamics	$\tau = 20$				$\tau = 60$				$\tau = 100$				$\tau = 20$				$\tau = 60$				$\tau = 100$				$\tau = 20$				$\tau = 60$				$\tau = 100$			
$\rho \Rightarrow$	0.1	**0.2**	**0.5**	**1.0**	0.1	**0.2**	**0.5**	**1.0**	0.1	**0.2**	**0.5**	**1.0**	0.1	**0.2**	**0.5**	**1.0**	0.1	**0.2**	**0.5**	**1.0**	0.1	**0.2**	**0.5**	**1.0**	0.1	**0.2**	**0.5**	**1.0**	0.1	**0.2**	**0.5**	**1.0**	0.1	**0.2**	**0.5**	**1.0**
amcGA1 - GAm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA1 - mcGA	−	−	−	+	+	∼	−	−	+	+	+	−	−	∼	−	−	+	+	+	−	+	+	+	−	−	−	−	−	∼	∼	+	−	+	∼	+	−
amcGA1 - PBILm	−	−	−	−	∼	∼	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA2 - GAm	−	−	−	−	−	∼	−	−	−	−	∼	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA2 - mcGA	−	−	−	−	∼	∼	−	−	+	+	+	−	−	−	−	−	+	+	∼	−	+	+	+	−	−	−	−	−	∼	∼	+	−	+	−	+	−
amcGA2 - PBILm	−	−	−	−	−	−	−	−	∼	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA3 - GAm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	−	−	−	−	−	−	∼	−
amcGA3 - mcGA	+	−	+	∼	+	+	+	+	+	+	+	+	∼	+	+	∼	+	+	+	+	+	+	+	+	∼	∼	∼	∼	−	∼	+	+	+	∼	+	+
amcGA3 - PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	−	−	−	−	−
amcGA4 - GAm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	−	−	−	−	−	−	−	−
amcGA4 - mcGA	−	−	−	−	∼	∼	−	−	+	+	+	−	−	∼	∼	−	+	+	∼	−	∼	+	+	−	−	−	−	−	−	−	+	−	+	−	+	−
amcGA4 - PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	+	−	−	∼	+	−	−	∼	+
amcGA5 - GAm	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	−	−	∼	−	−	−	−	−	∼	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA5 - mcGA	−	−	−	−	∼	∼	−	−	+	+	+	∼	−	∼	−	−	+	+	∼	−	∼	+	+	−	−	−	−	−	−	−	+	−	+	−	+	∼
amcGA5 - PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−

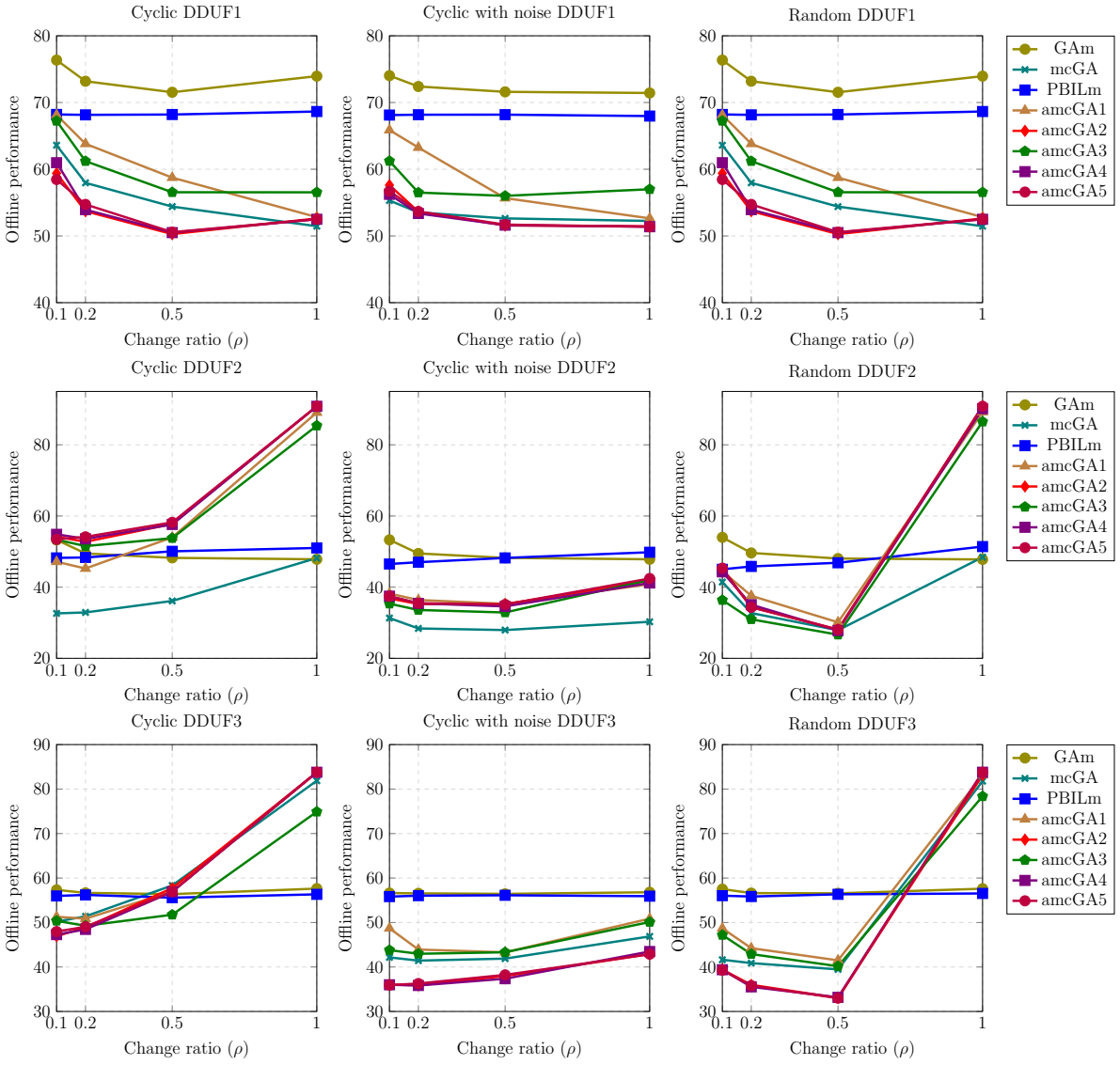


FIGURE 5.5: This figure shows the best offline performance of each algorithm under different change ratios and environments (i.e. Cyclic, Cyclic with noise and Random) with $\tau = 60$. Since the DDUFs are maximisation problems, it can be observed that for all DDUFs the optimum value is 100.

In Figs. 5.2 and 5.3, it can be observed that on the cyclic DDUF2 and cyclic DDUF3 (with and without noise), all amcGA variants show low performance when ρ= 0.1 to 0.5 but exhibit rapid increase in performance when ρ=1.0. This is due to the deceptive nature of DDUF2 and DDUF3, since a low-order BBs within the function do not clearly lead to a high-order BBs and the amcGAs seems to be sensitive to low ρ. However, all amcGA variants cope well with high ρ (1.0) because the environment shifts between two states which in turn gives more time

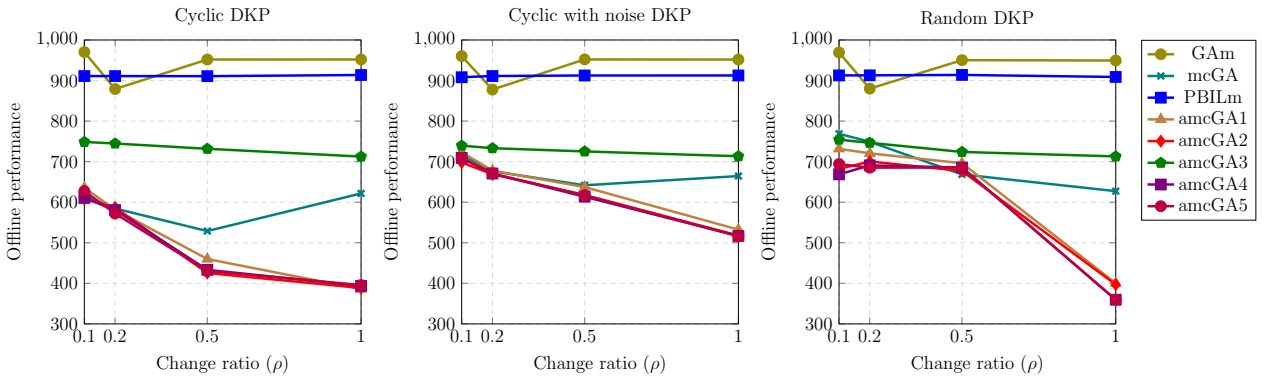


FIGURE 5.6: This figure shows the best offline performance of each algorithm under different change ratios and environments (Cyclic, Cyclic with noise and Random) with $\tau = 60$. Since the DKP is a maximization problem, it can be observed that for all environments the optimum value is 1000.

for the algorithm to obtain a better solution suitable for the environment before the next change occurs.

Looking at DKP in Figs. 5.4, it can be observed that for all dynamic environments, the performance of all variants of the amcGA reduces as ρ increases. This can be considered normal, since an increase in ρ implies more severe environment changes. When the nature of the dynamic environment increases from cyclic to cyclic with noise, the performance of all amcGA variant and mcGA increases slightly. Despite the fact that a cyclic environment with noise is generally more difficult to deal with than a cyclic environment, the amcGA variants showed better performance. But in the random environment, the performance of some of the amcGA variants dropped (when $\rho = 0.5$ and 1.0). This implies that even though the existence of noise in a cyclic environment may over weigh randomness (in terms of difficulty), it favours the performance of all amcGA variants.

Finally, from Fig. 5.1(a) - 5.6, it can be observed that the amcGA variants (i.e. amcGA1 to amcGA5) performed better on the DDUF2 and DDUF3 problem, especially when τ is large (see Table 5.1 - 5.4). This implies that the performance of the amcGA not only depends on the dynamics of the environment, but also on the DOP being considered. Overall, the experimental results indicate the amcGA variants can be considered as the best compact approach (among the tested) when solving DOPs with deceptive properties[2].

[2]These are functions where there exists low-order building blocks that do not combine to form the higher-order building blocks. Instead, low-order building blocks may lead astray an optimisation algorithm towards local optima (Fernandes et al., 2009)

5.1.3 Discussion

The effect of change trend and different mutation schemes on the performance of the amcGA in dynamic environments was studied in this section. From experimental results, several conclusions can be drawn on the overall performance of the algorithms:

- First, the mutation scheme has a positive effect on the performance of the amcGA. Information about an environment (before or after a dynamic change) is retained and reprocessed whenever a dynamic change occurs (instead of storing individual solutions in a dedicated memory or using training data).
- Second, statistical results highlight that variants of the amcGA display best performance on a number of DOPs (with respect to environment dynamics) when compared with GAm, mcGA and PBILm. In several cases, the change in environment had minimal effect on the performance of the amcGA variants while the algorithms tries to find a suitable solution. The interaction between the change trend and mutation depends on the DOP (see Figs. 5.1- 5.6 and Table 5.1- 5.4).
- Third, the addition of a change trend scheme to the amcGA improves the algorithms performance in dynamic environments. The change trend scheme ensures that the amcGA responds to dynamic changes based on the change pattern exhibited by the current working probability. This allows the algorithm to update its mutation strategy using the change pattern. However, the observed effect of these in the experiments, although not conclusive, seems to have less of an effect than the hypermutation on the performance of the GAm and PBILm.
- Finally, the mutation scheme embedded within all amcGA variants encourages diversity in dynamic environments. It ensures that the algorithm maintains its population diversity while tackling a DOP and gradually moves towards the optimal solution (see Appendix C for individual performance plots).

5.2 Analysis of TMSDS experiments

The algorithms presented in Chapter 3 were used for tuning an integral-state PID controller in order to evaluate the performance of each algorithm in the dynamic environment described in Chapter 4. All algorithms were evaluated using the TMSDS platform without the need for any manual (or human) intervention. Each algorithm starts with a randomly generated initial solution (chromosome) and default position (zero initial position of the TMSDS). The respective results are presented and discussed below.

As mentioned earlier, the amcGAs and competing algorithms makes use of the elitism approach to copy the best individual of every generation into the next generation. Each algorithm evaluates 2 candidate solutions every generation and runs for 200 generations (i.e. 10 dynamic changes $\times$ 20 generation). Each solution requires about 20 seconds for evaluation plus an additional 5 seconds for the TMSDS to reset to the initial position before the next evaluation. The overall time for each algorithm to complete a run is about 10000 seconds. The experimental results of all algorithms based on RMSE (i.e. Eq. 4.23), mean, standard deviation and Best-of-generation performance using Eq. 2.2 are shown in Table 5.6 and 5.7. A summary of the memory requirements of each algorithm on the TMSDS are shown in Table 5.5.

5.2.1 Analysis regarding overall performance of the amcGA

From Table 5.6 and 5.7, it can be observed that all amcGA variants outperforms the mcGA, r-rcGA and r-cDE. The reason for this behaviour is because the restart scheme mainly resets the algorithm (i.e. re-initializing $\overrightarrow{P}$) whenever a change is detected so as to distribute the search force. This method only increases the diversity in the population at the beginning of each new environment. But restarting an algorithm whenever a change occurs does not assure finding the optimum solution in a reasonable time frame. Also, r-rcGA and r-cDE do not exploit any useful information from old environments and the frequent restart sacrifices its evolving capability.

The effect of the hypermutation in mcGA is only noticed at the start of a new environment. While tuning the controller using mcGA, it was observed that the

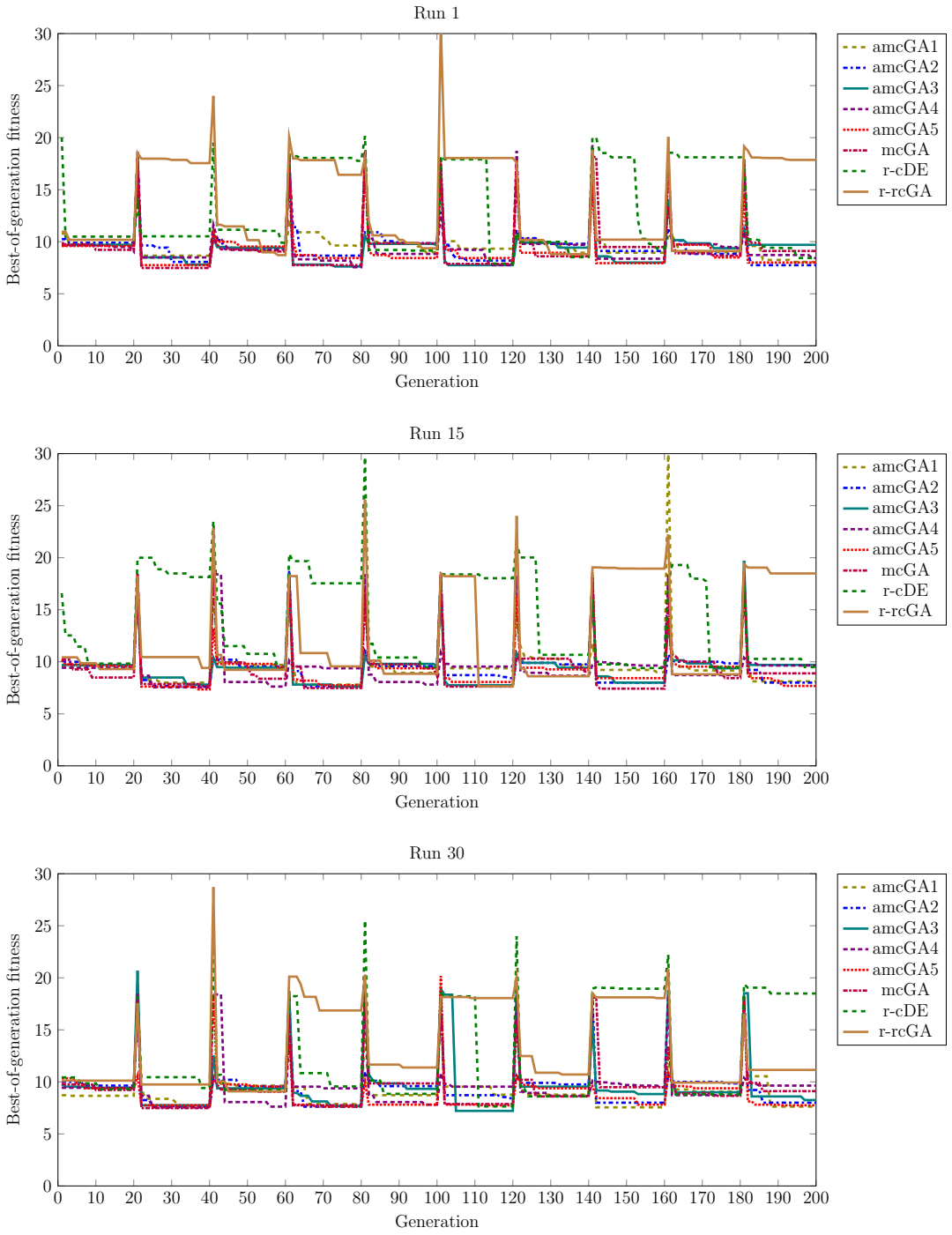


FIGURE 5.7: The figure shows the performance of all competing algorithms using the TMSDS in 3 different runs. Best-or-generation fitness (based on RMSE) achieved each generation is shown. The main objective is to minimize error (i.e the TMSDS is a minimization problem). Please refer to Appendix C for the individual step response of competing algorithms.

probability vector converged faster than the other algorithms. This is because, the $\overrightarrow{P}$ updates based on the best hyper-mutated solution which makes $\overrightarrow{P}$ loose its diversity (the hypermutation scheme on its own does not encourage genetic diversity and can lead to early convergence). Since the mcGA requires that a base and high level mutation rate be picked a priori, introducing a proper level of diversity becomes difficult to achieve. This can be considered as one of the disadvantages of the hypermutation scheme.

The dynamic behaviour of all algorithms (based on Eq. 2.2) are shown in Fig. 5.7 for the number of environmental changes with respect to best-of-generation fitness (i.e. elite fitness RMSE) against the number of generations. From Figs. 5.7, several behaviours can be observed. The spikes in the graph signifies a change in environment which is when the magnet is placed in front of the mass. The amcGA tries to find a solution suitable for both environments (i.e in the presence of a magnet and when away from the magnet) unlike the r-rcGA, r-cDE and mcGA which exhibits unstable performance (see Fig. 5.7).

5.2.1.1 Memory consumption

In order to have a clear indication of the memory requirements in the TMSDS problem, the memory footprint of the competing algorithms were evaluated. The embedded software excluding the compact dynamic optimisation algorithms consumes 6386 byte, which is 19% of the total flash memory. The Arduino Uno contains 32KB of flash memory of which 0.5KB is used by the boot-loader. The addition of each algorithm results to an increase in memory. For the mcGA and r-rcGA, the additional increase in memory is 16% and 20% respectively while the rcDE requires an additional 21% of total flash memory. The r-cDE requires more memory due to the solution sampling mechanism within the algorithm (please refer to Mininno et al. (2011)).

From Table 5.5, it is clear that the amcGA is not computationally expensive, in terms of memory. The amcGA1-3 takes 19% of the flash memory, while amcGA4 and amcGA5 takes 21%. In the amcGA4 and amcGA5, the 2% additional increase in memory is as result of the addition of the change trend scheme to the adaptive mutation scheme. From Table 5.5, it can be observed that the amcGAs are more memory efficient than the competing algorithms. The addition of the adaptive mutation, change detection and trend scheme do not result to increased memory requirement. This implies that the amcGA is significantly better than

the competing compact algorithms because integrating similar dynamic techniques into the competing algorithms would make them computationally expensive and unsuitable for applications using memory constrained devices.

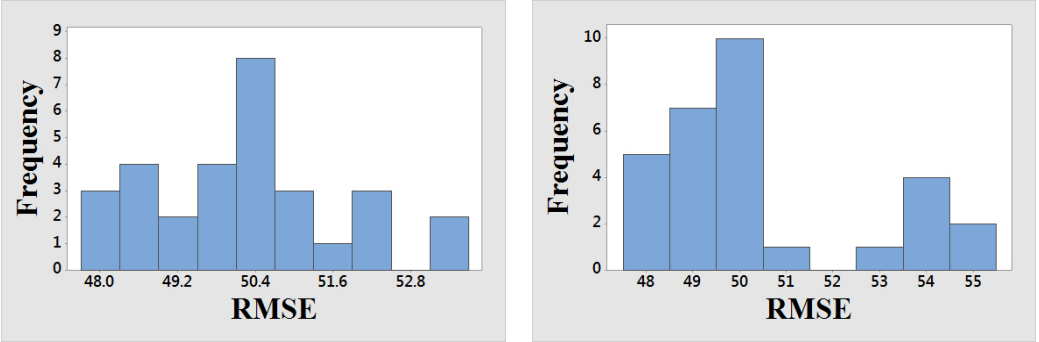
TABLE 5.5: Summary of memory requirements on the TMSDS. This table shows the size of each algorithm (without the integral-state PID controller) in the Arduino memory. This is expressed as size (byte) in Flash memory, the SRAM usage which is a combination of initialized data (data) and non-initialized data (bss) SRAM = data + bss.

Algorithms	***Flash (byte)***	***data (byte)***	***bss (byte)***	***SRAM (byte)***	***Total memory (%)***
mcGA	4916	28	509	537	16
r-cDE	6706	22	347	369	21
r-rcGA	6402	22	275	297	20
amcGA1	5868	28	658	686	19
amcGA2	5952	28	658	686	19
amcGA3	6096	28	658	686	19
amcGA4	6304	29	754	783	21
amcGA5	6278	29	754	783	21

5.2.2 Statistical Analysis of TMSDS Experiment

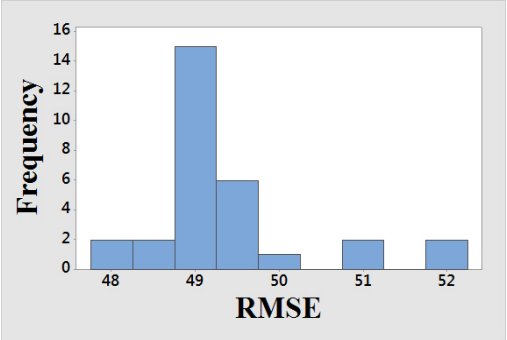
The first step in the analysis of the data is to test for the null hypothesis whether or not there are significant differences in the ability of the evolutionary PID controller to maintain the setpoint in the dynamic environment. To achieve this, a parametric statistic test such as the t-test can be used to accept or reject the null hypothesis. Parametric statistics assumes that all populations have equal standard deviations and normal distribution of values. However, it is important to test normality so as to confirm these assumptions. Therefore a histogram of each results from the evolved PID controller (using the respective compact algorithms) is depicted in Fig. 5.8. It can be observed that results obtained do not show a normal distribution. Also, from Table 5.6, it is clear that the standard deviations are not equal. In addition, the Shapiro-Wilk normality test was used to further analyse the results for normality. This test confirmed that all results were from a non-normal distribution.

Since the data does not show a normal distribution and equal standard deviations, a non-parametric statistic, the Wilcoxon Rank-Sum statistic test with 0.05 level of significance was used to verify the null hypothesis h_0 ($h_0 : A = B$) against the alternative hypothesis h_a ($h_a : A \neq B$). More precisely, the null hypothesis is rejected if the p-value is smaller than 0.05.

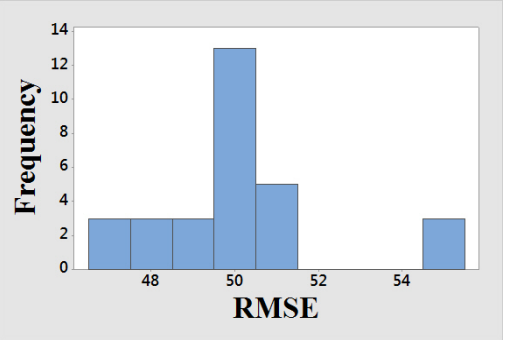


(a) *amcGA1*.

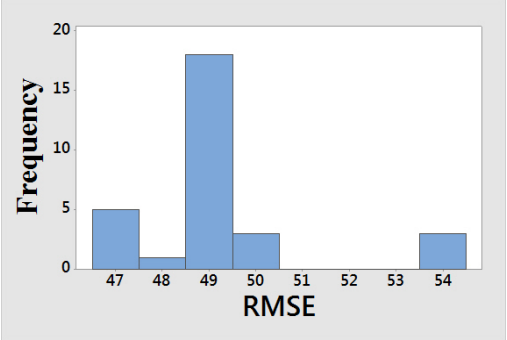
(b) *amcGA2*.



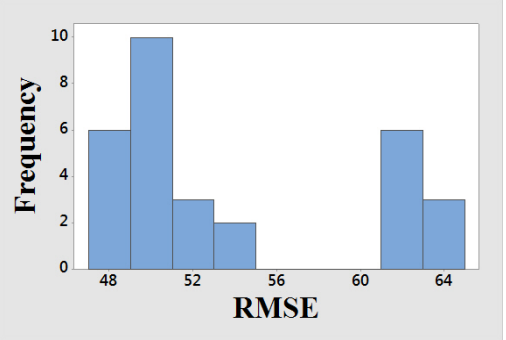
(c) *amcGA3*.



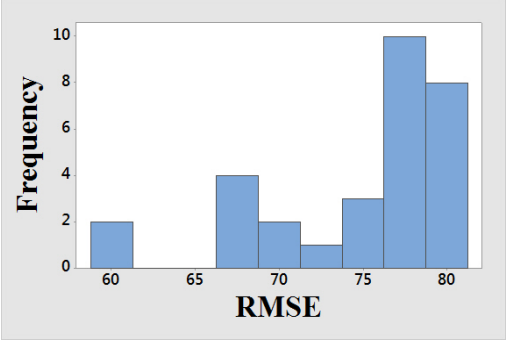
(d) *amcGA4*.



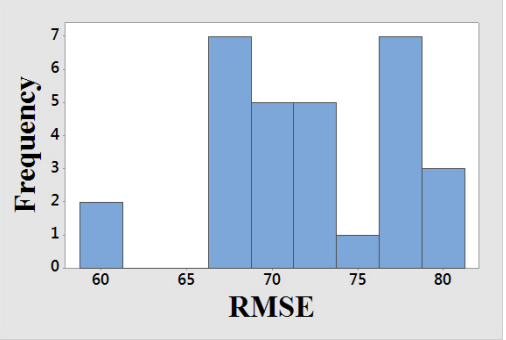
(e) *amcGA5*.



(f) *mcGA*.



(g) *r-cDE*.



(h) *r-rcGA*.

FIGURE 5.8: Histogram of the best PID performance observed over all 30 test runs using the respective compact dynamic algorithm

With the amcGA variants set as reference, the statistic test led to the null hypothesis being rejected except when amcGA1 was compared with amcGA5. The test indicated that there are significant differences between the compact dynamic optimisation tuning techniques. Results of these comparisons are shown in Table 5.7. From Table 5.7, the Wilcoxon Rank-Sum test do give information as to where the difference are amongst the algorithms:

- amcGA1 outperformed r-cGA, r-cDE, mcGA and amcGA2
- amcGA2 outperformed r-cGA, r-cDE and mcGA
- amcGA3 outperformed all competing algorithms
- amcGA4 outperformed r-cGA, r-cDE, mcGA and amcGA5
- amcGA5 outperformed r-cGA, r-cDE and mcGA. Also amcGA5 is statistically equivalent to amcGA1.

From all results shown in this section, the amcGA3 exhibited the best performance amongst the amcGA variants, showing low RMSE, Mean, *Stdev*, $\overline{F}_{BOG}$ and closely tracked the changing environment (please refer to Figs C.13- C.20 in Appendix C). This can be considered a significant finding since it demonstrates that the amcGA variants outperform the competing compact dynamic algorithms.

TABLE 5.6: Experimental results of all algorithms with respect to total RMSE, overall mean, standard deviation and performance based on $\overline{F}_{BOG}$

Algorithms	***RMSE***	***Mean***	***Stdev***	$\overline{F}_{BOG}$
amcGA1	54236.26	1807.87	65.19	50.24
amcGA2	55107.04	1836.90	67.14	50.49
amcGA3	**52238.19**	**1807.94**	**29.74**	**45.58**
amcGA4	54543.86	1818.12	61.25	50.14
amcGA5	54492.58	1816.08	65.85	50.15
mcGA	58103.11	1936.77	120.27	53.62
r-cDE	82589.42	2105.98	150.58	74.76
r-rcGA	79880.14	2662.67	168.67	72.30

TABLE 5.7: Statistical results regarding the overall performance of the amcGA variants on the TMSDS. The table shows the statistical results of the Wilcoxon rank-sum test at 0.05 level of significance. The results of Algorithm 1 (in rows) and Algorithm 2 (in columns) are shown as "+", "−" and "∼" when Algorithm 1 is better than, worse than or statistically equivalent to Algorithm 2.

Algorithms	***r-rcGA***	***r-cDE***	***mcGA***	***amcGA5***	***amcGA4***	***amcGA3***	***amcGA2***
amcGA1	+	+	+	∼	−	−	+
amcGA2	+	+	+	−	−	−	
amcGA3	+	+	+	+	+		
amcGA4	+	+	+	+			
amcGA5	+	+	+				
mcGA	+	+					
r-cDE	∼						

5.2.3 Discussion

In this section the amcGA and competing algorithms were used for tuning the gains of an integral-state PID controller for the TMSDS in a dynamic environment. Performance of the algorithms were evaluated using an actual system instead of an artificial benchmark problem. This made any system identification that would create an approximate but imperfect model of the actual real-world system implicit.

From experimental results shown, several conclusions can be drawn on the overall performance of the amcGA:

- The amcGA can be an effective means to perform self-tuning function for control algorithms in dynamic environments.
- The amcGA maintains the small footprint of the conventional cGA which allows direct implementation on the same processor running the control algorithm, thus overcoming the limitations related to typical population-based dynamic optimisation algorithms (see Table 5.5).
- The schemes within the amcGAs encourages diversity in a dynamic environment. It ensures that the algorithm maintains population diversity when tackling the DOP instead of a random-walk like the r-rcGA and r-cDE.
- Instead of searching for an optimal solution for a particular environment, the amcGA tries to find a solution suitable for both environments i.e. with respect to the position of the magnet (please refer to Fig. C.13 - C.20 in Appendix C).

5.3 Summary

This chapter evaluated the performance of the amcGA using an artificial dynamic benchmark problem and a physical system in a dynamic environment. Overall, from all experiments presented, as stated earlier, it can be observed that some of the competing dynamic algorithms exploit artificial simulation discrepancies in an opportunistic manner to achieve high fitness values with deceitful performance (please see mcGA performance in both benchmark problems in Section 5.1 and 5.2).

In general, when designing a dynamic optimisation algorithm it is important to achieve a proper balance between exploration and exploitation. Controlling exploration and exploitation of an optimisation algorithm is difficult since these abilities are implicit. Also, there is a trade off between exploration and exploitation of a search space. Higher selection pressure means more exploitation while lower selection pressure means more exploration. This implies that maintaining a good balance between exploration and exploitation is highly significant because inadequate choice of parameter values can limit the ability of a dynamic algorithm to locate the optimum. This is evident in the performance of mcGA, when the mutation rate increases due to a dynamic change, much of the search space is explored. However, the mcGA loses promising solution and in some cases experiences difficulty converging to an optimum due to insufficient exploitation. It was noticed that this behaviour in some dynamic environment settings led to early convergence of $\overrightarrow{P}$.

On the other hand, all amcGA variants explore the defined search space before a dynamic change occurred. After a dynamic change, the amcGA variants exploit the search based on information from previous environments (for a defined number of generations) so as to adapt quickly in the new environment before further exploration. This behaviour comes from the schemes embedded with the amcGA. The level of diversity applied to the algorithm after a dynamic change depends on the degree of change. This means that the exploration ability of the amcGA (to an extent) is directly proportional to degree of change, whilst the exploitation ability of the algorithm to an extent, depends on information from previous environments (based on $\overrightarrow{P}$ and $\overrightarrow{mP}$).

From the experiments carried out, it can be affirmed that the amcGA3 performed best in both benchmark problems. This is because the mutation scheme embedded in the amcGA3 ensures that $\overrightarrow{mP}$ is altered at a reduced rate in order to reflect

the degree of dynamic change. This implies that the amcGA3 has better evolving capability and fast response to dynamic changes (see Appendix C for individual performance plots).

However, it was observed that the schemes presented in Chapter 3 are problem dependent. For example, the amcGA4 and amcGA5 are suitable for reoccurring (cyclic and cyclic with noise) dynamic changes. This is because the change trend scheme embedded in amcGA4 and amcGA5 keeps track of the change pattern in a dynamic environment so as to enable the algorithm to learn and adapt to similar subsequent dynamic changes. On the other hand, amcGA1 and amcGA2 are suitable for applications that have no time constraints as they require more time to adapt properly. Also, the amcGA1 can be used to access/determine which of the other variants are suitable for a particular DOP.

Chapter 6

Conclusions and Future Work

Despite the steady increase in high performance computational devices, there are applications where it is required to solve optimisation problems in memory constrained embedded systems. For example, the optimization of parameters for unmanned aerial vehicles and robotic systems, where cost and size of the embedded system being used is very important. In such situations, the amcGA presented in this thesis is considered suitable. This is because the algorithmic structure of the amcGA permits direct implementation on a memory constrained embedded hardware system without a significant increase in memory. Also the amcGA can be seen as an advancement in developing a compact dynamic optimization algorithm suitable for memory constrained devices and applications. This way the existing gap between academic research and real-world dynamic application reduces.

Therefore, this chapter concludes this thesis by summarizing the results, re-evaluates their utility, and discussing key points of the research. Contribution to knowledge are presented and an outlook at future improvements and topics that will be the focus of future research in the field will be given.

6.1 Concluding remarks

The main outcome of this thesis is the novel variant of the compact genetic algorithm suitable for solving DOPs. More specifically, an adaptive-mutation compact genetic algorithm (amcGA) was created and presented. This is a significant step for tackling DOPs using a compact dynamic optimisation algorithm. Furthermore the amcGA has been compared to other compact and population-based dynamic

optimisation algorithms. This research was assessed against the research hypothesis stated in sections 1.2 and 3.1.2.

Initially, a literature survey in chapter 2 identified limitations of existing dynamic optimisation algorithms and control optimisation techniques. Most dynamic optimisation algorithms are evaluated using artificial benchmark problems which in some cases do not relate to a real world problem sufficiently. There have been studies on the use of data originating from real world scenarios. Although, the algorithms used are capable of solving DOPs, the complex structure of the algorithms limit their application in small-scale embedded hardware systems with limited memory. These predecessors are the basis for the development of the amcGA.

This thesis developed and presented a novel compact dynamic optimisation algorithm suitable for solving artificial DOPs and direct implementation on embedded hardware with limited memory (see chapter 3). In order to evaluate the performance of the algorithm, dynamic test problems were introduced and used (see chapter 4). Two different types of DOPs, an artificial DOP and a DOP based on a physical system were used, primarily because of their relevance in verifying the research hypothesis. The XOR-DBG is a synthetic dynamic benchmark generator that constructs dynamic environments for any binary encoded static optimisation problem. The TMSDS is a physical system operating in a dynamic environment.

Experimental analysis was used to support the verification of the research hypothesis. The limitation of existing dynamic optimisation algorithm and techniques i.e. hypermutation and restart schemes discussed in this thesis instigated a novel compact approach. The resulting compact dynamic optimisation algorithm is suitable for direct implementation on small scale embedded system with limited memory (see table 5.5) as well as simulations using a desktop PC.

The novel algorithm developed and presented in this thesis is based on the idea of maintaining population diversity at a population-level by applying an intelligent adaptive mutation and effective change detection schemes. It was demonstrated that these techniques support extracting and transferring of key information from one environment to another and that this can effectively minimize the exploration exploitation dilemma commonly faced by most dynamic optimisation algorithms.

6.2 Contributions

Contributions of this thesis in the field of cGA and dynamic optimisation covers the development of techniques that: enhance the adaptation capabilities of the cGA in dynamic environments, encourage diversity within the cGA i.e preventing premature convergence of the probability vector $\overrightarrow{P}$ and allow knowledge transfer without using a dedicated memory storage.

Specifically, the main technical contributions of this research can be summarized as follows:

- The development of a novel change detection scheme that measures the degree of change in a dynamic environments. The change detection scheme also tracks the performance of an elite solution in a dynamic environment. This was confirmed through experimentation and analysis carried out in chapter 5.

- The development of a novel adaptive mutation scheme that operates on a population level (based on the degree of change), such that a high degree of change results to a high mutation rate so as to diversify solutions sampled from the probability vector. This way, exploration and exploitation properties are implicitly regulated by the algorithm. This can be observed in chapter 5.

- The development of a novel change trend scheme that tracks and monitors change pattern exhibited by the algorithm. This way the algorithm adapts quickly after a dynamic change. The change trend scheme can be applied to combinatorial DOPs and real-valued DOPs with some increase in memory requirements. In chapter 3, two change trend schemes were presented; bt_{chg} and rt_{chg}. The bt_{chg} was applied to the XOR-DOPs since this is a combinatorial problem while the TMSDS DOP made use of rt_{chg}. Candidate solutions were generated as binary strings for easy manipulation of individual bits. This is a substantial contribution since results of experiments presented in chapter 5 confirms that these schemes are computationally efficient (see table 5.5).

- The novel algorithm developed in this thesis has been evaluated using a synthetic dynamic benchmark generator and a physical system. These tests were used to verify the research hypothesis. Others have confirmed that most

dynamic optimisation algorithms exploit simulation discrepancies to achieve performance that are unrealistic. Competing algorithms showed stable performance when evaluated using the synthetic dynamic benchmark problem. However, this is not the case when evaluated using the TMSDS (please refer to the performance of mcGA on both test problems in chapter 5 and Appendix C).

- The schemes developed in this thesis have been studied in different dynamic test scenes. The amcGA shows overall better performance when compared to the hypermutation scheme applied to a cGA (mcGA).

6.3 Future research directions

For DOPs, locating and tracking an optimum in a defined search space is an effective method for dynamic evolutionary algorithms. Experimental results confirm that the main objective of optimisation algorithms in dynamic environments should be to find an optimum as quickly as possible before further exploration. The work presented here provides an effective platform for pursuing future avenues of research. The key areas of further research are listed and discussed below:

Scheme extension:

It would be interesting to integrate the schemes developed in this thesis into other compact and non-compact optimisation algorithms to further improve performance when solving dynamic optimisation problems. For example, the mutation factor in the cDE has an impact on the overall performance of the algorithm. Further research will be carried out to look at self-adaptation or adaptation of the mutation factor and diversity of the cDE to adapt them based on the dynamics of a DOP. Extending the schemes used in amcGA to the rcGA may also be valuable to improve performance when solving DOPs. This would encourage the use of memory efficient compact optimisation algorithms for DOPs. Also, studying the efficiency and suitability of the amcGAs in different types of real-world applications will be a pursuit in further research.

Compact intelligence:

At this point one can imagine that a true intelligent dynamic optimisation algorithm can self-evolve according to the dynamic nature of a given problem. This implies that the algorithm should be able to learn, store and re-use any relevant information about a dynamic environment in order to adapt quickly to subsequent dynamic changes. Although there will be challenges on the way to realise such algorithms, further research will look at what constitutes a true intelligent self-evolving dynamic optimisation algorithm based on a compact structure.

Dynamic parameter updates:

It is notable that the change threshold *cThres* and standard deviation σ (Eq 3.2) of the amcGA are fixed (see chapter 3) during an optimisation process. Since these parameters determine how sensitive the amcGA is to dynamic changes, further research will look at the dynamic control of these parameters during an optimisation process. This means updating these parameters in real-time according to the level of dynamism of physical test-bench or artificial benchmark problems.

Bibliography

Abramowitz, M., Stegun, I. A., 1972. Handbook of mathematical functions: with formulas, graphs, and mathematical tables. No. 55. Courier Corporation.

Ahn, C. W., Ramakrishna, R., Aug 2003. Elitism-based compact genetic algorithms. Evolutionary Computation, IEEE Transactions on 7 (4), 367–385.

Alba, E., Sarasola, B., July 2010. Abc, a new performance tool for algorithms solving dynamic optimization problems. In: Evolutionary Computation (CEC), 2010 IEEE Congress on. pp. 1–7.

Altin, L., Topcuoglu, H., Dec 2014. Performance evaluation of sensor-based detection schemes on dynamic optimization problems. In: Computational Intelligence in Dynamic and Uncertain Environments (CIDUE), 2014 IEEE Symposium on. pp. 24–31.

Alvarez-Gallegos, J., Villar, C. A. C., Flores, E. A. P., 2005. Evolutionary dynamic optimization of a continuously variable transmission for mechanical efficiency maximization. In: MICAI 2005: Advances in Artificial Intelligence. Springer, pp. 1093–1102.

Angeline, P. J., 1995. Adaptive and self-adaptive evolutionary computations. In: Computational intelligence: a dynamic systems perspective. Citeseer.

Aporntewan, C., Chongstitvatana, P., 2001. A hardware implementation of the compact genetic algorithm. In: Evolutionary Computation, 2001. Proceedings of the 2001 Congress on. Vol. 1. pp. 624–629 vol. 1.

Atkin, J. A., Burke, E. K., Greenwood, J. S., Reeson, D., 2008. On-line decision support for take-off runway scheduling with uncertain taxi times at london heathrow airport. Journal of Scheduling 11 (5), 323–346.

Baluja, S., 1994. Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning. Tech. rep., DTIC Document.

Bao, Y., Zhao, E., Gan, X., Luo, D., Han, Z., Aug 2009. A review on cutting-edge techniques in evolutionary algorithms. In: Natural Computation, 2009. ICNC '09. Fifth International Conference on. Vol. 5. pp. 347–351.

Baraglia, R., Hidalgo, J. I., Perego, R., 2001. A hybrid heuristic for the traveling salesman problem. Evolutionary Computation, IEEE Transactions on 5 (6), 613–622.

Barlow, G. J., Smith, S. F., 2008. A memory enhanced evolutionary algorithm for dynamic scheduling problems. In: Applications of Evolutionary Computing. Springer, pp. 606–615.

Battiti, R., Passerini, A., Oct 2010. Brain-computer evolutionary multiobjective optimization: A genetic algorithm adapting to the decision maker. Evolutionary Computation, IEEE Transactions on 14 (5), 671–687.

Ben-Romdhane, H., Alba, E., Krichen, S., 2013. Best practices in measuring algorithm performance for dynamic optimization problems. Soft Computing 17 (6), 1005–1017.

Branke, J., 1999. Memory enhanced evolutionary algorithms for changing optimization problems. In: Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on. Vol. 3. pp. –1882 Vol. 3.

Branke, J., 2001. Evolutionary Optimization in Dynamic Environments. Kluwer Academic Publishers, Norwell, MA, USA.

Branke, J., Kaußler, T., Smidt, C., Schmeck, H., 2000. A multi-population approach to dynamic optimization problems. In: Evolutionary Design and Manufacture. Springer, pp. 299–307.

Bui, L. T., Michalewicz, Z., Parkinson, E., Abello, M., April 2012. Adaptation in dynamic environments: A case study in mission planning. Evolutionary Computation, IEEE Transactions on 16 (2), 190–209.

Caraffini, F., Neri, F., Passow, B. N., Iacca, G., 2013. Re-sampled inheritance search: high performance despite the simplicity. Soft Computing 17 (12), 2235–2256.

Cartis, C., Gould, N. I., Toint, P. L., 2010. On the complexity of steepest descent, newton's and regularized newton's methods for nonconvex unconstrained optimization problems. SIAM Journal on Optimization 20 (6), 2833–2852.

Cedeno, W., Vemuri, V., Apr 1997. On the use of niching for dynamic landscapes. In: Evolutionary Computation, 1997., IEEE International Conference on. pp. 361–366.

Chaparro, B., Thuillier, S., Menezes, L., Manach, P., Fernandes, J., 2008. Material parameters identification: Gradient-based, genetic and hybrid optimization algorithms. Computational Materials Science 44 (2), 339 – 346.

Chen, L., Ding, L., Du, X., March 2011. Genetic algorithm with particle filter for dynamic optimization problems. In: Computer Research and Development (ICCRD), 2011 3rd International Conference on. Vol. 1. pp. 452–457.

Cheng, H., May 2012. Genetic algorithms with hyper-mutation for dynamic load balanced clustering problem in mobile ad hoc networks. In: Natural Computation (ICNC), 2012 Eighth International Conference on. pp. 1171–1176.

Cheng, H., Yang, S., June 2012. Hyper-mutation based genetic algorithms for dynamic multicast routing problem in mobile ad hoc networks. In: Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on. pp. 1586–1592.

Chiang, C.-H., May 2010. A genetic programming based rule generation approach for intelligent control systems. In: Computer Communication Control and Automation (3CA), 2010 International Symposium on. Vol. 1. pp. 104–107.

Chiang, M., 2008. Nonconvex optimization of communication systems. Advances in Mechanics and Mathematics 3, 137–196.

Chuang, C.-Y., Smith, S., June 2013. Diversity allocation for dynamic optimization using the extended compact genetic algorithm. In: Evolutionary Computation (CEC), 2013 IEEE Congress on. pp. 1540–1547.

Cobb, H. G., 1990. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Tech. rep., DTIC Document.

Cobb, H. G., Grefenstette, J. J., 1993. Genetic algorithms for tracking changing environments. Tech. rep., DTIC Document.

Colaço, M. J., Dulikravich, G. S., 2009. A survey of basic deterministic, heuristic and hybrid methods for single objective optimization and response surface generation. Thermal Measurements and Inverse Techniques 1.

Collingwood, E., Corne, D., Ross, P., 1996. Useful diversity via multiploidy. In: Evolutionary Computation, 1996., Proceedings of IEEE International Conference on. IEEE, pp. 810–813.

Dai, Y.-H., 2011. Nonlinear conjugate gradient methods. Wiley Encyclopedia of Operations Research and Management Science.

Dam, H. H., Lokan, C., Abbass, H. A., 2007. Evolutionary online data mining: An investigation in a dynamic environment. In: Evolutionary computation in dynamic and uncertain environments. Springer, pp. 153–178.

Dantzig, G. B., Ramser, J. H., 1959. The truck dispatching problem. Management science 6 (1), 80–91.

Das, S., Suganthan, P., Feb 2011. Differential evolution: A survey of the state-of-the-art. Evolutionary Computation, IEEE Transactions on 15 (1), 4–31.

Dorigo, M., Birattari, M., Stutzle, T., Nov 2006. Ant colony optimization. Computational Intelligence Magazine, IEEE 1 (4), 28–39.

Eiben, A., Schut, M. C., De Wilde, A., 2006. Boosting genetic algorithms with self-adaptive selection. In: Evolutionary Computation, 2006. CEC 2006. IEEE Congress on. IEEE, pp. 477–482.

Eiben, A., Smit, S., 2012. Evolutionary algorithm parameters and methods to tune them. In: Autonomous Search. Springer, pp. 15–36.

Eiben, A. E., Hinterding, R., Michalewicz, Z., 1999. Parameter control in evolutionary algorithms. Evolutionary Computation, IEEE Transactions on 3 (2), 124–141.

Fernandes, C. M., Guervós, J. J. M., Rosa, A. C., 2009. Using dissortative mating genetic algorithms to track the extrema of dynamic deceptive functions. CoRR abs/0904.3063.

Fletcher, R., 1987. Practical Methods of Optimization; (2Nd Ed.). Wiley-Interscience, New York, NY, USA.

Fliess, M., Join, C., 2009. Model-free control and intelligent pid controllers: towards a possible trivialization of nonlinear control? arXiv preprint arXiv:0904.0322.

Fogel, L. J., Owens, A. J., Walsh, M. J., 1966. Artificial Intelligence through Simulated Evolution. John Wiley & Sons, New York.

Fossati, L., Lanzi, P. L., Sastry, K., Goldberg, D. E., Gomez, O., 2007. A simple real-coded extended compact genetic algorithm. In: Evolutionary Computation, 2007. CEC 2007. IEEE Congress on. IEEE, pp. 342–348.

Franceschini, G., Macchietto, S., 2008. Model-based design of experiments for parameter precision: State of the art. Chemical Engineering Science 63 (19), 4846–4872.

Fu, H., Sendhoff, B., Tang, K., Yao, X., June 2012. Characterizing environmental changes in robust optimization over time. In: Evolutionary Computation (CEC), 2012 IEEE Congress on. pp. 1–8.

Gallagher, J. C., Vigraham, S., Kramer, G., 2004. A family of compact genetic algorithms for intrinsic evolvable hardware. Evolutionary Computation, IEEE Transactions on 8 (2), 111–126.

Gao, Q., Zheng, L., Chen, J., Wang, L., Hou, Y., 2014. Development of a novel disturbance observer based fractional order pd controller for a gun control system. The Scientific World Journal 2014.

Gédouin, P.-A., Delaleau, E., Bourgeot, J.-M., Join, C., Chirani, S. A., Calloch, S., 2011. Experimental comparison of classical pid and model-free control: position control of a shape memory alloy active spring. Control Engineering Practice 19 (5), 433–441.

Goldberg, D. E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning, 1st Edition. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Goldberg, D. E., Smith, R. E., 1987. Nonstationary function optimization using genetic algorithm with dominance and diploidy. In: Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, pp. 59–68.

Gonçalves, J., Lima, J., Costa, P. J., Moreira, A. P., 2013. Modeling and simulation of the emg30 geared motor with encoder resorting to simtwo: the official robot@ factory simulator. In: Advances in Sustainable and Competitive Manufacturing Systems. Springer, pp. 307–314.

Gongora, M., Passow, B., Hopgood, A., May 2009. Robustness analysis of evolutionary controller tuning using real systems. In: Evolutionary Computation, 2009. CEC '09. IEEE Congress on. pp. 606–613.

Gonzalez-Sieira, A., Bugarin, A., Mucientes, M., Moran, J., June 2013. A tabu search optimization module for scheduling: Design and integration in the open source tool libreplan for project management. In: Information Systems and Technologies (CISTI), 2013 8th Iberian Conference on. pp. 1–6.

Halder, U., Das, S., Maity, D., June 2013. A cluster-based differential evolution algorithm with external archive for optimization in dynamic environments. Cybernetics, IEEE Transactions on 43 (3), 881–897.

Hameed, I. A., 2011. Using gaussian membership functions for improving the reliability and robustness of students evaluation systems. Expert Systems with Applications 38 (6), 7135 – 7142.

Hansen, N., Kern, S., 2004. Evaluating the cma evolution strategy on multimodal test functions. In: Parallel problem solving from nature-PPSN VIII. Springer, pp. 282–291.

Hansen, N., Ostermeier, A., May 1996. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: Evolutionary Computation, 1996., Proceedings of IEEE International Conference on. pp. 312–317.

Hansen, N., Ostermeier, A., 2001. Completely derandomized self-adaptation in evolution strategies. Evolutionary computation 9 (2), 159–195.

Harik, G., 1999. Linkage learning via probabilistic modeling in the ecga. Urbana 51 (61), 801.

Harik, G. R., Lobo, F. G., Goldberg, D. E., 1999. The compact genetic algorithm. Evolutionary Computation, IEEE Transactions on 3 (4), 287–297.

Hatzakis, I., Wallace, D., 2006. Dynamic multi-objective optimization with evolutionary algorithms: A forward-looking approach. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation. GECCO '06. ACM, New York, NY, USA, pp. 1201–1208.

Haugwitz, S., Åkesson, J., Hagander, P., 2009. Dynamic start-up optimization of a plate reactor with uncertainties. Journal of Process Control 19 (4), 686–700.

Haupt, R. L., Haupt, S. E., 2004. Practical genetic algorithms. J. Wiley, Hoboken, N.J.

Heath, M., 1998. Computing: An introductory survey.

Hu, X., Eberhart, R., 2002. Adaptive particle swarm optimization: detection and response to dynamic systems. In: Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on. Vol. 2. pp. 1666–1670.

hui Chang, G., fei Li, Y., Gang, Q., Aug 2011. Intelligent controller design for pm dc motor position control using evolutionary programming. In: Computing, Control and Industrial Engineering (CCIE), 2011 IEEE 2nd International Conference on. Vol. 1. pp. 37–40.

Iacca, G., Caraffini, F., Neri, F., 2012. Compact differential evolution light: high performance despite limited memory requirement and modest computational overhead. Journal of Computer Science and technology 27 (5), 1056–1076.

Iruthayarajan, M. W., Baskar, S., 2009. Evolutionary algorithms based design of multivariable pid controller. Expert systems with Applications 36 (5), 9159–9167.

Isaacs, A., Puttige, V., Ray, T., Smith, W., Anavatti, S., June 2008. Development of a memetic algorithm for dynamic multi-objective optimization and its applications for online neural network modeling of uavs. In: Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on. pp. 548–554.

Jatmiko, W., Mursanto, P., Kusumoputro, B., Sekiyama, K., Fukuda, T., 2008. Modified pso algorithm based on flow of wind for odor source localization problems in dynamic environments. Wseas transactions on Systems 7 (2), 106–113.

Jensen, J. C., Chang, D. H., Lee, E. A., 2011. A model-based design methodology for cyber-physical systems. In: Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International. IEEE, pp. 1666–1671.

Jin, Y., Branke, J., 2005. Evolutionary optimization in uncertain environments-a survey. Evolutionary Computation, IEEE Transactions on 9 (3), 303–317.

Kaedi, M., Aghaee, N., Ahn, C. W., 2013. Evolutionary optimization in dynamic environments: Bringing the strengths of dynamic bayesian networks into bayesian optimization algorithm. Int J Innov Comput Inf Control 9, 2485–2503.

Kanoh, H., 2007. Dynamic route planning for car navigation systems using virus genetic algorithms. International Journal of Knowledge-based and Intelligent Engineering Systems 11 (1), 65–78.

Karsai, G., Sztipanovits, J., Ledeczi, A., Bapty, T., 2003. Model-integrated development of embedded software. Proceedings of the IEEE 91 (1), 145–164.

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., 1983. Optimization by simulated annealing. SCIENCE 220 (4598), 671–680.

Korejo, I., Yang, S., Li, C., 2009. A comparative study of adaptive mutation operators for metaheuristics.

Koussa, B., Bachir, S., Perrine, C., Duvanaud, C., Vauzelle, R., Dec 2012. A comparison of several gradient based optimization algorithms for papr reduction in ofdm systems. In: Communications, Computing and Control Applications (CCCA), 2012 2nd International Conference on. pp. 1–6.

Kramer, G., Gallagher, J., July 2003. Improvements to the *cga enabling online intrinsic evolution in compact eh devices. In: Evolvable Hardware, 2003. Proceedings. NASA/DoD Conference on. pp. 225–231.

Lanzi, P. L., Nichetti, L., Sastry, K., Voltini, D., Goldberg, D. E., 2008. Real-coded extended compact genetic algorithm based on mixtures of models. In: Linkage in evolutionary computation. Springer, pp. 335–358.

Larraanaga, P., Lozano, J. A., 2001. Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Kluwer Academic Publishers, Norwell, MA, USA.

Lauwerys, C., Swevers, J., Sas, P., June 2005. A model free control design approach for a semi-active suspension of a passenger car. In: American Control Conference, 2005. Proceedings of the 2005. pp. 2206–2211 vol. 3.

Lewis, A. S., Overton, M. L., 2013. Nonsmooth optimization via quasi-newton methods. Mathematical Programming 141 (1-2), 135–163.

Lewis, J., Hart, E., Ritchie, G., 1998. A comparison of dominance mechanisms and simple mutation on non-stationary problems. In: Parallel Problem Solving from Nature PPSN V. Springer, pp. 139–148.

Li, C., Nguyen, T. T., Yang, M., Yang, S., Zeng, S., 2015. Multi-population methods in unconstrained continuous dynamic environments: The challenges. Information Sciences 296, 95–118.

Li, C., Yang, S., 2008. A generalized approach to construct benchmark problems for dynamic optimization. In: Proceedings of the 7th International Conference on Simulated Evolution and Learning. SEAL '08. Springer-Verlag, Berlin, Heidelberg, pp. 391–400.

Li, C., Yang, S., Nguyen, T., Yu, E., Yao, X., Jin, Y., Beyer, H., Suganthan, P., 2008. Benchmark generator for cec 2009 competition on dynamic optimization. University of Leicester, University of Birmingham, Nanyang Technological University, Tech. Rep.

Li, C., Yang, S., Nguyen, T. T., June 2012. A self-learning particle swarm optimizer for global optimization problems. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 42 (3), 627–646.

Li, C., Yang, S., Pelta, D. A., 2011. Benchmark generator for the ieee wcci-2012 competition on evolutionary computation for dynamic optimization problems. China University of Geosciences, Brunel University, University of Granada, Tech. Rep.

Li, X., Branke, J., Blackwell, T., 2006. Particle swarm with speciation and adaptation in a dynamic environment. In: Proceedings of the 8th annual conference on Genetic and evolutionary computation. ACM, pp. 51–58.

Lin, M.-H., Tsai, J.-F., Yu, C.-S., 2012. A review of deterministic optimization methods in engineering and management. Mathematical Problems in Engineering 2012.

Makaitis, D., July 2003. Evolving fuzzy controllers through evolutionary programming. In: Fuzzy Information Processing Society, 2003. NAFIPS 2003. 22nd International Conference of the North American. pp. 50–54.

Manner, R., Mahfoud, S., Mahfoud, S. W., 1992. Crowding and preselection revisited. In: Parallel Problem Solving From Nature. North-Holland, pp. 27–36.

Mavrovouniotis, M., Li, C., Yang, S., Yao, X., 2013. Benchmark generator for the ieee wcci-2014 competition on evolutionary computation for dynamic optimization problems.

Mavrovouniotis, M., Yang, S., 2013a. Ant colony optimization with immigrants schemes for the dynamic travelling salesman problem with traffic factors. Applied Soft Computing 13 (10), 4023–4037.

Mavrovouniotis, M., Yang, S., June 2013b. Genetic algorithms with adaptive immigrants for dynamic environments. In: Evolutionary Computation (CEC), 2013 IEEE Congress on. pp. 2130–2137.

Mavrovouniotis, M., Yang, S., Dec 2014a. Ant colony optimization with self-adaptive evaporation rate in dynamic environments. In: Computational Intelligence in Dynamic and Uncertain Environments (CIDUE), 2014 IEEE Symposium on. pp. 47–54.

Mavrovouniotis, M., Yang, S., July 2014b. Elitism-based immigrants for ant colony optimization in dynamic environments: Adapting the replacement rate. In: Evolutionary Computation (CEC), 2014 IEEE Congress on. pp. 1752–1759.

Mavrovouniotis, M., Yang, S., 2015. Ant algorithms with immigrants schemes for the dynamic vehicle routing problem. Information Sciences 294, 456–477.

Mavrovouniotis, M., Yang, S., Yao, X., 2012. A benchmark generator for dynamic permutation-encoded problems. In: Parallel Problem Solving from Nature-PPSN XII. Springer, pp. 508–517.

Meza, J. C., 2010. Steepest descent. Wiley Interdisciplinary Reviews: Computational Statistics 2 (6), 719–722.

Michalewicz, Z., Schmidt, M., Michalewicz, M., Chiriac, C., 2007. Adaptive business intelligence: three case studies. In: Evolutionary Computation in Dynamic and Uncertain Environments. Springer, pp. 179–196.

Mills-Tettey, G. A., Stentz, A., Dias, M. B., 2008. Continuous-field path planning with constrained path-dependent state variables.

Mininno, E., Cupertino, F., Naso, D., Apr. 2008. Real-valued compact genetic algorithms for embedded microcontroller optimization. Trans. Evol. Comp 12 (2), 203–219.

Mininno, E., Neri, F., Cupertino, F., Naso, D., Feb 2011. Compact differential evolution. Evolutionary Computation, IEEE Transactions on 15 (1), 32–54.

Mitra, P., Venayagamoorthy, G. K., 2008. Real time implementation of an artificial immune system based controller for a dstatcom in an electric ship power system. In: Industry Applications Society Annual Meeting, 2008. IAS'08. IEEE. IEEE, pp. 1–8.

Morrison, R., De Jong, K., 2000. Triggered hypermutation revisited. In: Evolutionary Computation, 2000. Proceedings of the 2000 Congress on. Vol. 2. pp. 1025–1032 vol.2.

Morrison, R. W., 2004. Designing Evolutionary Algorithms for Dynamic Environments. SpringerVerlag.

Mustonen, V., Lässig, M., 2009. From fitness landscapes to seascapes: non-equilibrium dynamics of selection and adaptation. Trends in Genetics 25 (3), 111–119.

Mustonen, V., Lässig, M., 2010. Fitness flux and ubiquity of adaptive evolution. Proceedings of the National Academy of Sciences 107 (9), 4248–4253.

Nakata, M., Lanzi, P., Takadama, K., June 2013. Simple compact genetic algorithm for xcs. In: Evolutionary Computation (CEC), 2013 IEEE Congress on. pp. 1718–1723.

Narushima, Y., Yabe, H., Ford, J. A., 2011. A three-term conjugate gradient method with sufficient descent property for unconstrained optimization. SIAM Journal on Optimization 21 (1), 212–230.

Nelson, A. L., Barlow, G. J., Doitsidis, L., 2009. Fitness functions in evolutionary robotics: A survey and analysis. Robotics and Autonomous Systems 57 (4), 345–370.

Neri, F., Mininno, E., Kärkkäinen, T., 2010. Noise analysis compact genetic algorithm. In: Applications of Evolutionary Computation. Springer, pp. 602–611.

Neri, F., Tirronen, V., Feb. 2010. Recent advances in differential evolution: A survey and experimental analysis. Artif. Intell. Rev. 33 (1-2), 61–106.

Nguyen, T. T., 2011. Continuous dynamic optimisation using evolutionary algorithms. Ph.D. thesis, University of Birmingham.

Nguyen, T. T., Yang, S., Branke, J., 2012. Evolutionary dynamic optimization: A survey of the state of the art. Swarm and Evolutionary Computation 6, 1–24.

Nguyen, T. T., Yang, S., Branke, J., Yao, X., 2013. Evolutionary dynamic optimization: Methodologies. In: Evolutionary Computation for Dynamic Optimization Problems. Springer, pp. 39–64.

Nguyen, T. T., Yao, X., May 2009a. Benchmarking and solving dynamic constrained problems. In: Evolutionary Computation, 2009. CEC '09. IEEE Congress on. pp. 690–697.

Nguyen, T. T., Yao, X., 2009b. Dynamic time-linkage problems revisited. In: Applications of Evolutionary Computing. Springer, pp. 735–744.

Nguyen, T. T., Yao, X., 2010. Solving dynamic constrained optimisation problems using repair methods. IEEE Transactions on Evolutionary Computation (submitted).

Panda, S., 2011. Multi-objective pid controller tuning for a facts-based damping stabilizer using non-dominated sorting genetic algorithm-ii. International Journal of Electrical Power & Energy Systems 33 (7), 1296–1308.

Passow, B., Gongora, M., Coupland, S., Hopgood, A., June 2008. Real-time evolution of an embedded controller for an autonomous helicopter. In: Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on. pp. 2538–2545.

Passow, B. N., Gongora, M. A., 2008. Optimising a flying robot: controller optimisation using a genetic algorithm on a real-world robot.

Patel, R., Raghuwanshi, M., Nov 2010. Review on real coded genetic algorithms used in multiobjective optimization. In: Emerging Trends in Engineering and Technology (ICETET), 2010 3rd International Conference on. pp. 610–613.

Pazderin, A., Yuferev, S., Nov 2009. Power flow calculation by combination of newton-raphson method and newton's method in optimization. In: Industrial Electronics, 2009. IECON '09. 35th Annual Conference of IEEE. pp. 1693–1696.

Pelikan, M., Goldberg, D., Lobo, F., 2000. A survey of optimization by building and using probabilistic models. In: American Control Conference, 2000. Proceedings of the 2000. Vol. 5. pp. 3289–3293 vol.5.

Peng, X., Gao, X., Yang, S., 2011. Environment identification-based memory scheme for estimation of distribution algorithms in dynamic environments. Soft Computing 15 (2), 311–326.

Pham, N., Malinowski, A., Bartczak, T., Nov 2011. Comparative study of derivative free optimization algorithms. Industrial Informatics, IEEE Transactions on 7 (4), 592–600.

Phiromlap, S., Rimcharoen, S., 2013a. A frequency-based updating strategy in compact genetic algorithm. In: Computer Science and Engineering Conference (ICSEC), 2013 International. IEEE, pp. 207–211.

Phiromlap, S., Rimcharoen, S., Sept 2013b. A frequency-based updating strategy in compact genetic algorithm. In: Computer Science and Engineering Conference (ICSEC), 2013 International. pp. 207–211.

Pisinger, D., 1999. Core problems in knapsack algorithms. Operations Research 47 (4), 570–575.

Polyak, B. T., 2007. Newtons method and its use in optimization. European Journal of Operational Research 181 (3), 1086–1096.

Puchinger, J., Raidl, G. R., Pferschy, U., 2010. The multidimensional knapsack problem: Structure and algorithms. INFORMS Journal on Computing 22 (2), 250–265.

Rahmati, S., Mallakzadeh, M., Dec 2011. Determination of the optimum objective function for evaluating optimal body and barbell trajectories of snatch weightlifting via genetic algorithm optimization. In: Biomedical Engineering (ICBME), 2011 18th Iranian Conference of. pp. 21–26.

Ramsey, C. L., Grefenstette, J. J., 1993. Case-based initialization of genetic algorithms. In: Proceedings of the 5th International Conference on Genetic Algorithms. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 84–91.

Rand, W., Riolo, R., 2005. Measurements for understanding the behavior of the genetic algorithm in dynamic environments: A case study using the shaky ladder hyperplane-defined functions. In: Proceedings of the 7th Annual Workshop on Genetic and Evolutionary Computation. GECCO '05. ACM, New York, NY, USA, pp. 32–38.

Richter, H., May 2009. Detecting change in dynamic fitness landscapes. In: Evolutionary Computation, 2009. CEC '09. IEEE Congress on. pp. 1613–1620.

Richter, H., 2010. Memory design for constrained dynamic optimization problems. In: Applications of Evolutionary Computation. Springer, pp. 552–561.

Rohlfshagen, P., Yao, X., 2009. The dynamic knapsack problem revisited: A new benchmark problem for dynamic combinatorial optimisation. In: Applications of evolutionary computing. Springer, pp. 745–754.

Rossi, C., Abderrahim, M., Díaz, J. C., 2008. Tracking moving optima using kalman-based predictions. Evolutionary Computation 16 (1), 1–30.

Roy, R., Konar, A., Tibarewala, D., March 2012. Eeg driven artificial limb control using state feedback pi controller. In: Electrical, Electronics and Computer Science (SCEECS), 2012 IEEE Students' Conference on. pp. 1–5.

Sastry, K., Abbass, H. A., Goldberg, D., 2005. Sub-structural niching in non-stationary environments. In: AI 2004: Advances in Artificial Intelligence. Springer, pp. 873–885.

Silva, R. R., Lopes, H. S., Lima, C. R. E., 2007. A new mutation operator for the elitism-based compact genetic algorithm. In: Adaptive and Natural Computing Algorithms. Springer, pp. 159–166.

Silva, R. R., Lopes, H. S., Lima, C. R. E., 2008. A compact genetic algorithm with elitism and mutation applied to image recognition. In: Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence. Springer, pp. 1109–1116.

Simoes, A., Costa, E., Sept 2007. Improving memory usage in evolutionary algorithms for changing environments. In: Evolutionary Computation, 2007. CEC 2007. IEEE Congress on. pp. 276–283.

Simões, A., Costa, E., 2008. Evolutionary algorithms for dynamic environments: Prediction using linear regression and markov chains. In: Parallel Problem Solving from Nature–PPSN X. Springer, pp. 306–315.

Simões, A., Costa, E., 2009a. Improving prediction in evolutionary algorithms for dynamic environments. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation. ACM, pp. 875–882.

Simões, A., Costa, E., 2009b. Prediction in evolutionary algorithms for dynamic environments using markov chains and nonlinear regression. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation. ACM, pp. 883–890.

Smit, S., Eiben, A., May 2009. Comparing parameter tuning methods for evolutionary algorithms. In: Evolutionary Computation, 2009. CEC '09. IEEE Congress on. pp. 399–406.

Tawdross, P., Lakshmanan, S., Konig, A., Dec 2006. Intrinsic evolution of predictable behavior evolvable hardware in dynamic environment. In: Hybrid Intelligent Systems, 2006. HIS '06. Sixth International Conference on. pp. 60–60.

Tinós, R., Yang, S., 2007a. Genetic algorithms with self-organizing behaviour in dynamic environments. In: Evolutionary Computation in Dynamic and Uncertain Environments. Springer, pp. 105–127.

Tinós, R., Yang, S., 2007b. A self-organizing random immigrants genetic algorithm for dynamic optimization problems. Genetic Programming and Evolvable Machines 8 (3), 255–286.

Tinós, R., Yang, S., 2010. An analysis of the xor dynamic problem generator based on the dynamical system. In: Parallel Problem Solving from Nature, PPSN XI. Springer, pp. 274–283.

Tominaga, Y., Okamoto, Y., Wakao, S., Sato, S., May 2013. Binary-based topology optimization of magnetostatic shielding by a hybrid evolutionary algorithm combining genetic algorithm and extended compact genetic algorithm. Magnetics, IEEE Transactions on 49 (5), 2093–2096.

Turky, A. M., Abdullah, S., 2014. A multi-population harmony search algorithm with external archive for dynamic optimization problems. Information Sciences 272, 84–95.

Ursem, R. K., 2000. Multinational gas: Multimodal optimization techniques in dynamic environments. In: GECCO. pp. 19–26.

Uyar, A. Ş., Harmanci, A. E., 2005. A new population based adaptive domination change mechanism for diploid genetic algorithms in dynamic environments. Soft Computing 9 (11), 803–814.

Vavak, F., Fogarty, T. C., Jukes, K., 1996. A genetic algorithm with variable range of local search for tracking changing environments. In: Parallel problem solving from nature PPSN IV. Springer, pp. 376–385.

Vavak, F., Jukes, K., Fogarty, T., Apr 1997. Learning the local search range for genetic optimisation in nonstationary environments. In: Evolutionary Computation, 1997., IEEE International Conference on. pp. 355–360.

Črepinšek, M., Liu, S.-H., Mernik, M., Jul. 2013. Exploration and exploitation in evolutionary algorithms: A survey. ACM Comput. Surv. 45 (3), 35:1–35:33.

Wang, J., Tao, X., Cho, H., 2008. Microassembly of micro peg and hole using an optimal visual proportional differential controller. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture 222 (9), 1171–1180.

Wang, P., Zhao, Q., Yang, R., Oct 2006. Using accelerated evolutionary programming in self-turning control for uncertainty systems. In: Intelligent Systems Design and Applications, 2006. ISDA '06. Sixth International Conference on. Vol. 1. pp. 456–460.

Weicker, K., 2002. Performance measures for dynamic environments. In: Parallel Problem Solving from Nature PPSN VII. Springer Verlag, pp. 64–73.

Wescott, T., 2000. Pid without a phd. Embedded Systems Programming 13 (11), 1–7.

Woldesenbet, Y. G., Yen, G. G., 2009. Dynamic evolutionary algorithm with variable relocation. Evolutionary Computation, IEEE Transactions on 13 (3), 500–513.

Woodley, B. R., How, J. P., Kosut, R. L., 2001. Subspace based direct adaptive h inf control. International Journal of Adaptive Control and Signal Processing 15 (5), 535–561.

Xiao, L., Han, C., Xu, X., Huang, W., May 2010. Hybrid genetic algorithm and application to pid controllers. In: Control and Decision Conference (CCDC), 2010 Chinese. pp. 586–590.

Yang, S., Dec 2003. Non-stationary problem optimization using the primal-dual genetic algorithm. In: Evolutionary Computation, 2003. CEC '03. The 2003 Congress on. Vol. 3. pp. 2246–2253 Vol.3.

Yang, S., 2004. Constructing dynamic test environments for genetic algorithms based on problem difficulty. In: Evolutionary Computation, 2004. CEC2004. Congress on. Vol. 2. IEEE, pp. 1262–1269.

Yang, S., 2005a. Memory-based immigrants for genetic algorithms in dynamic environments. In: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation. GECCO '05. ACM, New York, NY, USA, pp. 1115–1122.

Yang, S., 2005b. Memory-enhanced univariate marginal distribution algorithms for dynamic optimization problems. In: Evolutionary Computation, 2005. The 2005 IEEE Congress on. Vol. 3. IEEE, pp. 2560–2567.

Yang, S., 2006. On the design of diploid genetic algorithms for problem optimization in dynamic environments. In: Evolutionary Computation, 2006. CEC 2006. IEEE Congress on. pp. 1362–1369.

Yang, S., 2007a. Genetic algorithms with elitism-based immigrants for changing optimization problems. In: Applications of Evolutionary Computing. Springer, pp. 627–636.

Yang, S., 2007b. Learning the dominance in diploid genetic algorithms for changing optimization problems.

Yang, S., Sep. 2008. Genetic algorithms with memory-and elitism-based immigrants in dynamic environments. Evol. Comput. 16 (3), 385–416.

Yang, S., Jiang, Y., Nguyen, T. T., 2013. Metaheuristics for dynamic combinatorial optimization problems. IMA Journal of Management Mathematics 24 (4), 451–480.

Yang, S., Richter, H., 2009. Hyper-learning for population-based incremental learning in dynamic environments. In: Evolutionary Computation, 2009. CEC'09. IEEE Congress on. IEEE, pp. 682–689.

Yang, S., Tinos, R., June 2008. Hyper-selection in dynamic environments. In: Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on. pp. 3185–3192.

Yang, S., Yao, X., 2003. Dual population-based incremental learning for problem optimization in dynamic environments. Asia Pacific Symposium on Intelligent and Evolutionary Systems.

Yang, S., Yao, X., 2005. Experimental study on population-based incremental learning algorithms for dynamic optimization problems. Soft Computing 9 (11), 815–834.

Yang, S., Yao, X., 2008. Population-based incremental learning with associative memory for dynamic environments. Evolutionary Computation, IEEE Transactions on 12 (5), 542–561.

Ying, Z., Lei, Z., Gui-li, W., Qian, Y., Nov 2009. A nonlinear pid controller based on an adaptive genetic algorithm. In: Intelligent Networks and Intelligent Systems, 2009. ICINIS '09. Second International Conference on. pp. 98–101.

Yoshida, Y., Adachi, N., 1994. A diploid genetic algorithm for preserving population diversity - pseudo-meiosis ga. In: Davidor, Y., Schwefel, H.-P., Manner, R. (Eds.), PPSN. Vol. 866 of Lecture Notes in Computer Science. Springer, pp. 36–45.

Yu, E. L., Suganthan, P. N., 2009. Evolutionary programming with ensemble of explicit memories for dynamic optimization. In: Proceedings of the Eleventh Conference on Congress on Evolutionary Computation. CEC'09. IEEE Press, Piscataway, NJ, USA, pp. 431–438.

Yu, X., Jin, Y., Tang, K., Yao, X., July 2010. Robust optimization over time - a new perspective on dynamic optimization problems. In: Evolutionary Computation (CEC), 2010 IEEE Congress on. pp. 1–6.

Yu, X., Tang, K., Chen, T., Yao, X., 2009. Empirical analysis of evolutionary algorithms with immigrants schemes for dynamic optimization. Memetic Computing 1 (1), 3–24.

Zanakis, S. H., Evans, J. R., 1981. Heuristic optimization: Why, when, and how to use it. Interfaces 11 (5), 84–91.

Zanella, F., Varagnolo, D., Cenedese, A., Pillonetto, G., Schenato, L., June 2012. Multidimensional newton-raphson consensus for distributed convex optimization. In: American Control Conference (ACC), 2012. pp. 1079–1084.

Zhang, J., Zhuang, J., Du, H., et al., 2009. Self-organizing genetic algorithm based tuning of pid controllers. Information Sciences 179 (7), 1007–1018.

Zhou, C., Meng, K., Qiu, Z., 2002. Compact genetic algorithm mutated by bit. In: Intelligent Control and Automation, 2002. Proceedings of the 4th World Congress on. Vol. 3. IEEE, pp. 1836–1839.

Zhu, H., Jiao, L., Pan, J., 2006. Multi-population genetic algorithm for feature selection. In: Advances in Natural Computation. Springer, pp. 480–487.

Zhu, J., 2014. Optimization of power system operation. John Wiley & Sons.

Zou, D., Gao, L., Li, S., Wu, J., 2011. Solving 0–1 knapsack problem by a novel global harmony search algorithm. Applied Soft Computing 11 (2), 1556–1564.

Zou, X., Wang, M., Zhou, A., McKay, B., 2004. Evolutionary optimization based on chaotic sequence in dynamic environments. In: Networking, Sensing and Control, 2004 IEEE International Conference on. Vol. 2. pp. 1364–1369 Vol.2.

Appendices

Appendix A

List of publications

List of publications by Chigozirim J. Uzor directly related to this PhD

Published:

- C.J. Uzor, M. Gongora, S. Coupland, and B.N. Passow. Adaptive mutation in dynamic environments. In Computational Intelligence (UKCI), 2014 14th UK Workshop on, pages 17, Sept 2014.
- C.J. Uzor, M. Gongora, S. Coupland, and B.N. Passow. Real-world dynamic optimization using an adaptive-mutation compact genetic algorithm. In Computational Intelligence in Dynamic and Uncertain Environments (CIDUE), 2014 IEEE Symposium on, pages 1723, Dec 2014.

Currently under review/revision:

- C.J. Uzor, M. Gongora, S. Coupland, and B.N. Passow. Adaptive-mutation compact genetic algorithm for dynamic environments. Soft Computing, 2015.

Appendix B

Additional tables

Statistical results on the DDUF1

TABLE B.1: Statistical results regarding the offline performance of the amcGA variants on the DDUF1. "+", "−" and "∼" indicates that *Algorithm 1* is better than, worse than or statistically equivalent to *Algorithm 2* (i.e. *Algorithm 1* − *Algorithm 2*).

Algorithms and DOPs	**DDUF1**											
Environment dynamics	τ **= 20**				τ **= 60**				τ **= 100**			
Cyclic ρ	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**
amcGA1 - amcGA2	+	+	+	+	+	∼	∼	+	+	+	+	+
amcGA1 - amcGA3	-	-	-	-	-	-	-	-	-	∼	-	-
amcGA1 - amcGA4	+	-	-	+	+	-	-	-	-	+	-	∼
amcGA1 - amcGA5	-	+	∼	-	+	-	+	-	+	∼	-	+
amcGA2 - amcGA3	-	-	-	-	-	-	∼	-	-	-	-	-
amcGA2 - amcGA4	-	∼	-	+	∼	-	∼	∼	-	∼	∼	+
amcGA2 - amcGA5	+	-	-	-	-	-	∼	-	-	+	-	-
amcGA3 - amcGA4	+	+	+	+	+	+	+	+	+	+	+	∼
amcGA3 - amcGA5	+	+	∼	+	+	+	∼	+	+	+	+	+
amcGA4 - amcGA5	∼	∼	∼	+	∼	+	∼	+	∼	∼	∼	-
Cyclic with noise, ρ	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**
amcGA1 - amcGA2	-	+	+	-	+	-	+	-	+	+	∼	+
amcGA1 - amcGA3	-	-	-	-	-	-	-	-	-	-	-	-
amcGA1 - amcGA4	-	+	-	∼	+	-	+	-	+	-	∼	-
amcGA1 - amcGA5	+	+	∼	-	∼	-	∼	-	+	-	-	-
amcGA2 - amcGA3	-	-	-	-	-	-	-	-	-	-	∼	-
amcGA2 - amcGA4	-	∼	∼	-	-	-	-	-	-	-	-	-
amcGA2 - amcGA5	∼	-	-	-	-	-	-	∼	+	-	∼	-
amcGA3 - amcGA4	+	+	+	+	+	+	+	+	+	+	+	+
amcGA3 - amcGA5	+	+	-	+	+	+	-	∼	+	+	∼	+
amcGA4 - amcGA5	-	∼	∼	-	-	∼	-	-	-	∼	-	-
Random, ρ	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**
amcGA1 - amcGA2	+	+	+	+	+	-	+	∼	+	-	+	+
amcGA1 - amcGA3	-	-	-	-	-	-	-	∼	-	-	∼	-
amcGA1 - amcGA4	+	-	-	-	-	∼	-	-	∼	+	+	+
amcGA1 - amcGA5	∼	-	-	∼	-	-	-	∼	-	∼	-	-
amcGA2 - amcGA3	-	-	-	∼	-	-	-	-	-	-	-	-
amcGA2 - amcGA4	-	∼	-	-	-	-	-	∼	-	-	-	∼
amcGA2 - amcGA5	-	-	-	-	∼	-	-	-	∼	-	∼	-
amcGA3 - amcGA4	+	-	∼	+	+	-	+	+	+	∼	+	+
amcGA3 - amcGA5	+	+	+	+	+	+	∼	∼	+	+	∼	+
amcGA4 - amcGA5	+	-	+	+	-	-	-	+	-	-	-	+

Statistical results on the DDUF2

TABLE B.2: Statistical results regarding the offline performance of the amcGA variants on the DDUF2. "+", "−" and "∼" indicates that *Algorithm 1* is better than, worse than or statistically equivalent to *Algorithm 2* (i.e. *Algorithm 1* − *Algorithm 2*).

Algorithms and DOPs	**DDUF2**											
Environment dynamics	τ **= 20**				τ **= 60**				τ **= 100**			
Cyclic ρ	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**
amcGA1 - amcGA2	∼	+	+	+	+	∼	∼	+	∼	∼	+	+
amcGA1 - amcGA3	-	-	-	-	-	-	-	-	-	-	-	-
amcGA1 - amcGA4	-	-	-	+	+	-	-	∼	-	+	∼	∼
amcGA1 - amcGA5	-	-	∼	-	+	-	∼	-	+	∼	-	∼
amcGA2 - amcGA3	-	-	-	-	-	-	-	-	-	-	-	-
amcGA2 - amcGA4	-	-	-	+	∼	-	+	-	-	-	∼	+
amcGA2 - amcGA5	-	-	-	∼	-	-	∼	-	-	+	-	-
amcGA3 - amcGA4	+	+	+	∼	+	+	∼	+	∼	+	+	∼
amcGA3 - amcGA5	∼	+	+	+	+	∼	+	+	+	+	+	+
amcGA4 - amcGA5	+	∼	-	+	∼	+	+	+	∼	-	∼	-
Cyclic with noise, ρ	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**
amcGA1 - amcGA2	∼	+	∼	-	+	-	+	∼	+	+	-	+
amcGA1 - amcGA3	-	-	-	-	-	-	-	-	-	-	-	-
amcGA1 - amcGA4	∼	+	∼	∼	+	+	-	-	+	-	∼	∼
amcGA1 - amcGA5	+	+	∼	-	-	-	∼	-	+	+	-	-
amcGA2 - amcGA3	-	-	-	-	-	-	-	-	-	-	-	-
amcGA2 - amcGA4	-	-	-	-	-	-	∼	-	∼	-	-	∼
amcGA2 - amcGA5	-	-	-	∼	-	-	∼	-	+	-	∼	-
amcGA3 - amcGA4	+	+	+	+	+	+	+	+	+	+	+	+
amcGA3 - amcGA5	+	+	-	+	+	+	+	∼	∼	+	+	+
amcGA4 - amcGA5	∼	+	-	-	-	∼	-	+	+	∼	-	-
Random, ρ	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**
amcGA1 - amcGA2	+	+	+	+	+	-	+	∼	+	-	+	+
amcGA1 - amcGA3	-	-	-	-	-	-	-	-	-	-	-	-
amcGA1 - amcGA4	-	-	-	-	-	∼	∼	-	-	-	-	-
amcGA1 - amcGA5	-	-	-	-	-	-	-	-	-	-	∼	∼
amcGA2 - amcGA3	-	-	-	-	-	-	-	-	-	-	-	-
amcGA2 - amcGA4	-	-	-	-	-	-	∼	-	∼	-	-	-
amcGA2 - amcGA5	∼	-	-	-	-	-	∼	-	-	-	∼	-
amcGA3 - amcGA4	+	+	+	+	+	-	+	∼	+	+	∼	∼
amcGA3 - amcGA5	+	∼	+	+	+	+	∼	+	+	+	+	+
amcGA4 - amcGA5	∼	-	+	+	∼	-	-	∼	-	∼	∼	+

Statistical results on the DDUF3

TABLE B.3: Statistical results regarding the offline performance of the amcGA variants on the DDUF3. "+", "−" and "∼" indicates that *Algorithm 1* is better than, worse than or statistically equivalent to *Algorithm 2* (i.e. *Algorithm 1* − *Algorithm 2*).

Algorithms and DOPs	**DDUF3**											
Environment dynamics	τ **= 20**				τ **= 60**				τ **= 100**			
Cyclic ρ	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**
amcGA1 - amcGA2	∼	+	+	-	-	∼	+	+	+	-	+	+
amcGA1 - amcGA3	-	-	-	∼	-	-	-	∼	-	-	-	-
amcGA1 - amcGA4	+	∼	-	+	-	+	-	+	-	∼	∼	+
amcGA1 - amcGA5	+	-	-	+	+	-	+	+	+	-	+	-
amcGA2 - amcGA3	-	-	-	-	-	∼	-	-	-	-	-	-
amcGA2 - amcGA4	∼	∼	+	-	-	-	-	+	-	+	-	-
amcGA2 - amcGA5	∼	-	-	-	∼	-	-	+	-	∼	-	+
amcGA3 - amcGA4	+	+	+	∼	∼	+	+	∼	-	∼	+	+
amcGA3 - amcGA5	+	+	∼	+	+	+	∼	+	+	+	+	∼
amcGA4 - amcGA5	∼	+	∼	+	+	-	+	+	∼	+	∼	+
Cyclic with noise, ρ	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**
amcGA1 - amcGA2	+	∼	∼	+	+	+	∼	-	+	+	+	+
amcGA1 - amcGA3	-	-	-	-	-	-	-	-	-	-	-	-
amcGA1 - amcGA4	∼	-	+	-	∼	-	-	+	+	∼	∼	∼
amcGA1 - amcGA5	+	-	-	-	+	-	-	+	-	∼	∼	-
amcGA2 - amcGA3	-	-	-	-	-	-	-	-	-	-	-	-
amcGA2 - amcGA4	-	∼	∼	-	-	-	-	-	-	-	-	-
amcGA2 - amcGA5	-	∼	∼	-	-	∼	-	-	-	-	∼	-
amcGA3 - amcGA4	+	-	∼	-	+	-	∼	-	-	∼	-	+
amcGA3 - amcGA5	∼	+	+	-	+	∼	+	∼	+	∼	∼	-
amcGA4 - amcGA5	-	-	-	-	-	∼	-	-	-	-	∼	-
Random, ρ	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**
amcGA1 - amcGA2	+	+	∼	+	+	-	-	+	+	-	+	+
amcGA1 - amcGA3	-	-	-	-	-	∼	-	-	-	∼	-	-
amcGA1 - amcGA4	+	+	∼	-	+	-	-	+	+	∼	+	∼
amcGA1 - amcGA5	∼	-	-	+	-	-	-	∼	+	-	∼	-
amcGA2 - amcGA3	-	-	-	-	-	-	-	-	-	-	-	-
amcGA2 - amcGA4	-	-	-	-	-	-	-	-	-	-	-	-
amcGA2 - amcGA5	-	-	-	∼	-	-	-	∼	-	-	-	∼
amcGA3 - amcGA4	+	+	+	∼	+	+	+	∼	+	+	+	-
amcGA3 - amcGA5	∼	+	+	+	∼	+	+	+	∼	+	∼	+
amcGA4 - amcGA5	-	+	+	+	-	-	+	+	+	+	-	∼

Statistical results on the DKP

TABLE B.4: Statistical results regarding the offline performance of the amcGA variants on the DKP. "+", "−" and "∼" indicates that *Algorithm 1* is better than, worse than or statistically equivalent to *Algorithm 2* (i.e. *Algorithm 1* − *Algorithm 2*).

Algorithms and DOPs	**DKP**											
Environment dynamics	τ **= 20**				τ **= 60**				τ **= 100**			
Cyclic ρ	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**
amcGA1 - amcGA2	+	+	+	+	+	+	-	+	+	+	+	+
amcGA1 - amcGA3	-	-	-	-	-	∼	-	-	-	-	∼	-
amcGA1 - amcGA4	+	∼	∼	-	∼	∼	-	-	+	-	-	+
amcGA1 - amcGA5	-	∼	∼	-	+	-	∼	-	+	+	∼	∼
amcGA2 - amcGA3	-	-	-	-	-	-	-	-	-	-	-	-
amcGA2 - amcGA4	-	+	-	-	-	∼	-	-	∼	∼	-	-
amcGA2 - amcGA5	-	∼	∼	-	∼	+	-	-	+	-	∼	-
amcGA3 - amcGA4	+	+	+	∼	+	+	+	+	∼	+	+	+
amcGA3 - amcGA5	+	+	+	∼	∼	+	+	+	+	+	+	∼
amcGA4 - amcGA5	+	+	+	∼	∼	∼	∼	+	+	∼	∼	+
Cyclic with noise, ρ	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**
amcGA1 - amcGA2	+	∼	+	+	+	+	+	∼	-	+	+	-
amcGA1 - amcGA3	-	-	-	-	∼	-	-	-	-	∼	-	-
amcGA1 - amcGA4	+	-	∼	+	+	-	∼	∼	+	∼	+	∼
amcGA1 - amcGA5	∼	-	-	∼	+	-	∼	∼	+	-	+	-
amcGA2 - amcGA3	-	-	-	-	-	-	-	-	-	-	-	-
amcGA2 - amcGA4	-	-	-	-	-	-	-	-	-	-	-	-
amcGA2 - amcGA5	-	-	-	-	-	-	∼	-	∼	-	-	-
amcGA3 - amcGA4	-	∼	+	+	-	∼	+	+	-	+	∼	∼
amcGA3 - amcGA5	+	+	∼	-	∼	+	∼	+	∼	-	+	∼
amcGA4 - amcGA5	-	-	-	-	-	∼	-	-	-	∼	-	-
Random, ρ	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**	**0.1**	**0.2**	**0.5**	**1.0**
amcGA1 - amcGA2	+	+	∼	∼	+	+	-	∼	+	+	+	-
amcGA1 - amcGA3	-	-	-	-	-	-	-	-	-	-	-	-
amcGA1 - amcGA4	-	∼	+	-	-	∼	+	-	+	-	+	-
amcGA1 - amcGA5	-	∼	-	-	∼	∼	-	-	-	-	∼	∼
amcGA2 - amcGA3	-	-	-	-	∼	-	-	-	-	∼	-	-
amcGA2 - amcGA4	-	-	∼	-	-	-	∼	-	-	∼	-	-
amcGA2 - amcGA5	-	-	-	-	-	-	∼	-	∼	-	∼	-
amcGA3 - amcGA4	+	∼	+	+	+	+	∼	+	+	+	+	+
amcGA3 - amcGA5	+	+	+	+	+	+	+	∼	+	∼	+	+
amcGA4 - amcGA5	+	-	-	∼	-	∼	+	∼	+	-	∼	∼

Appendix C

Additional performance plots

Individual plots of competing algorithm on the XOR DOP experiments

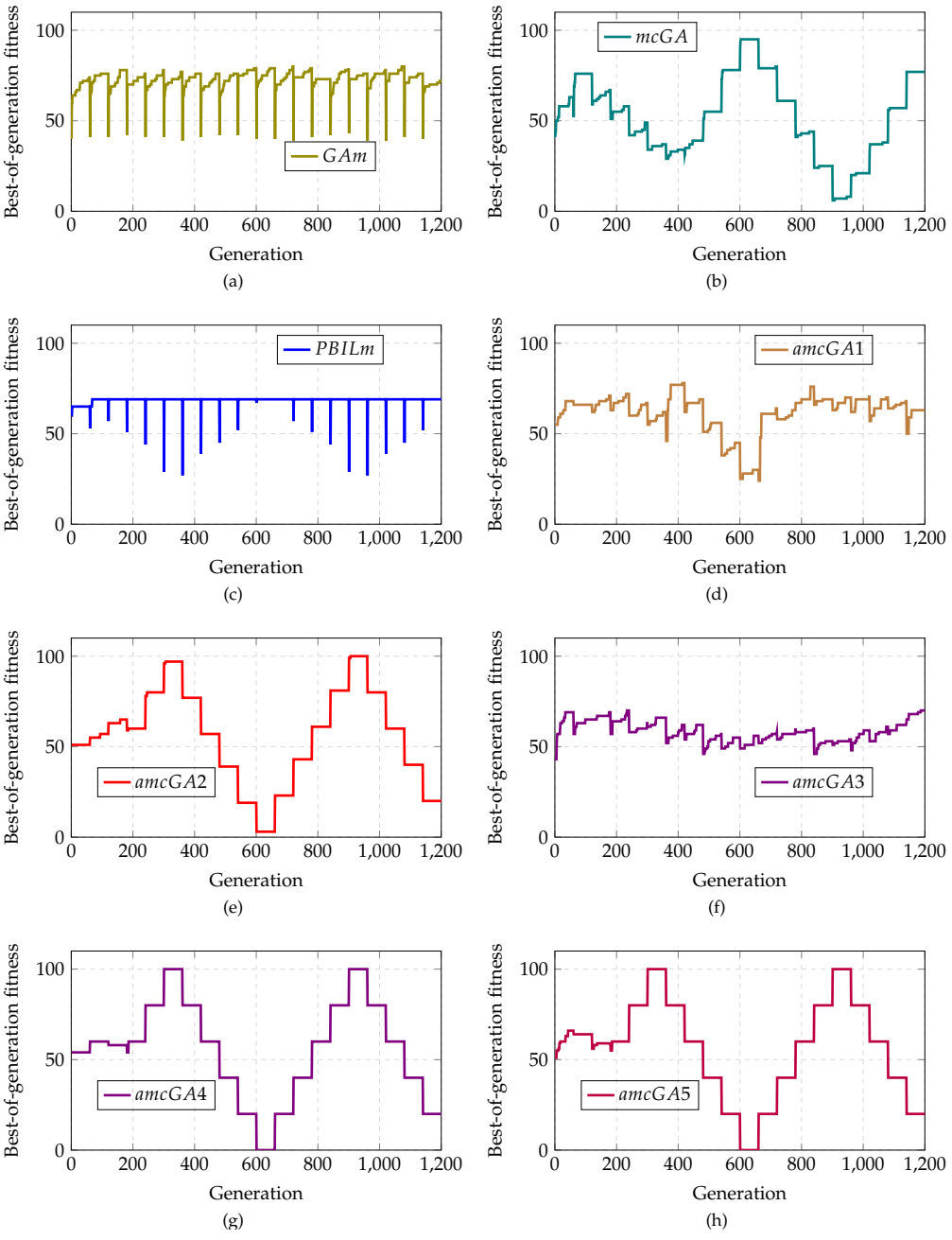


FIGURE C.1: Individual dynamic behaviour of competing algorithms on DDUF1 with $\tau = 60$ and $\rho = 0.2$ in a cyclic environment. Best fitness values achieved each generation are shown in the figure, where the optimum is 100.

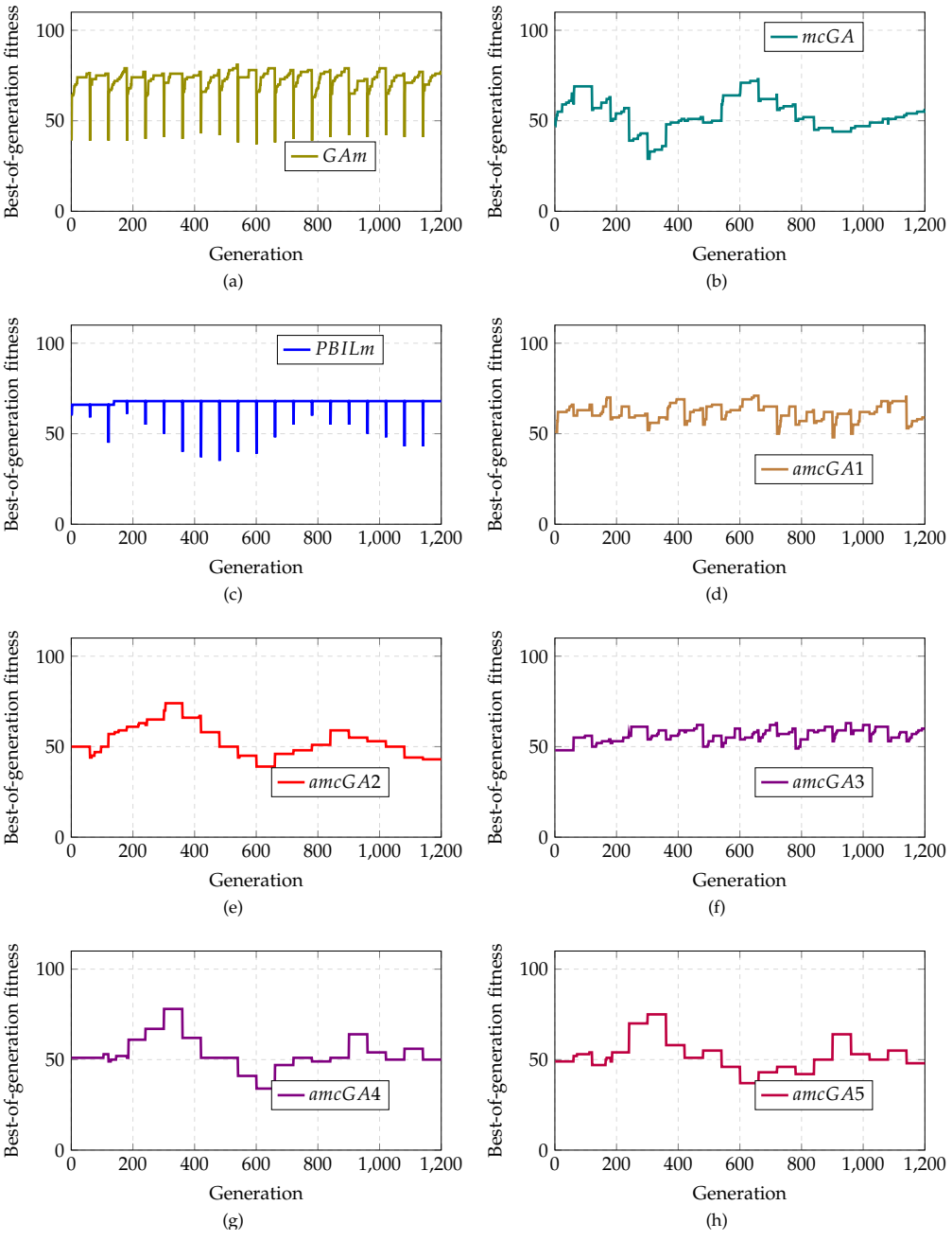


FIGURE C.2: Individual dynamic behaviour of competing algorithms on DDUF1 with $\tau = 60$ and $\rho = 0.2$ in a noisy cyclic environment. Best fitness values achieved each generation are shown in the figure, where the optimum is 100.

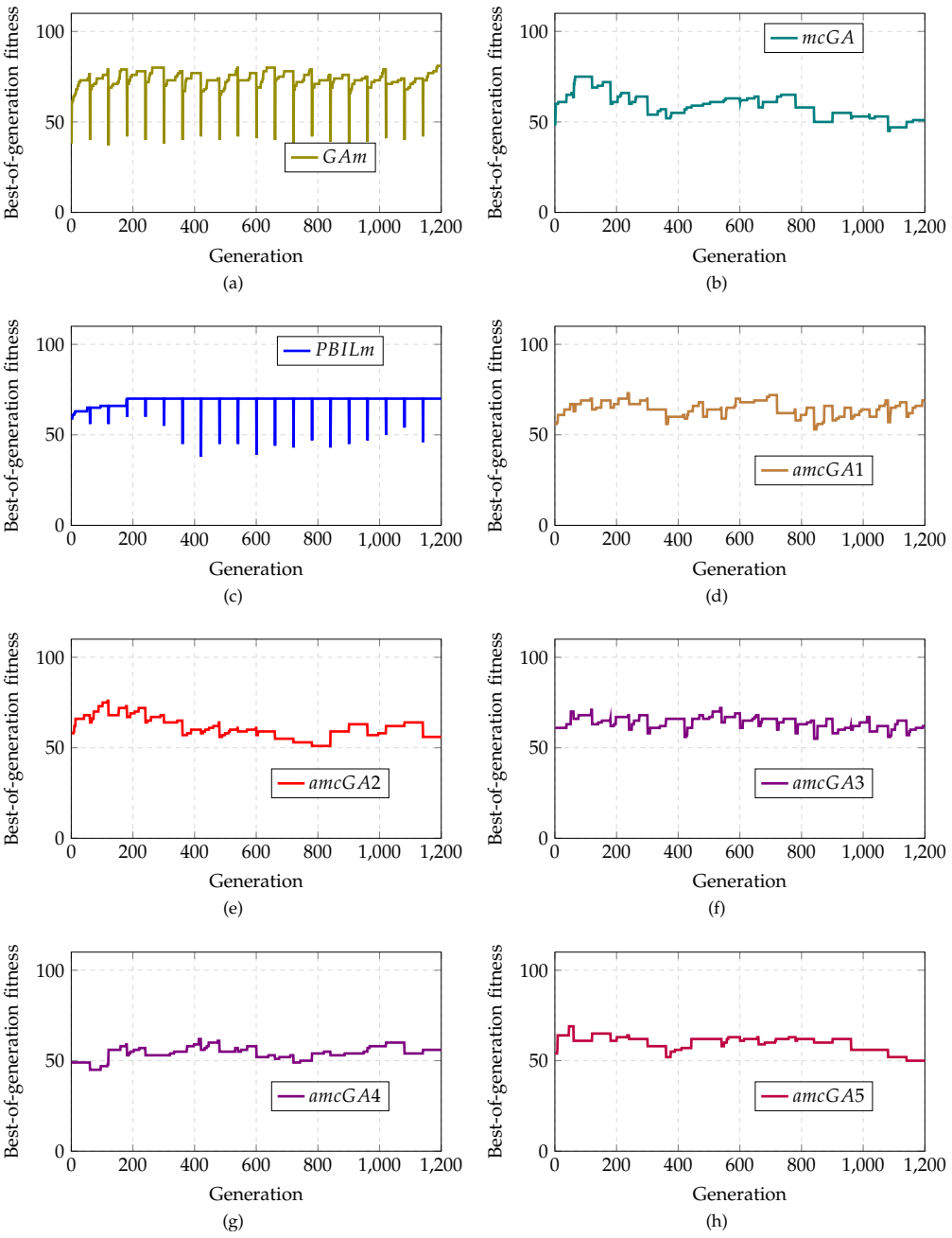


FIGURE C.3: Individual dynamic behaviour of competing algorithms on DDUF1 with $\tau = 60$ and $\rho = 0.2$ in a random environment. Best fitness values achieved each generation are shown in the figure, where the optimum is 100.

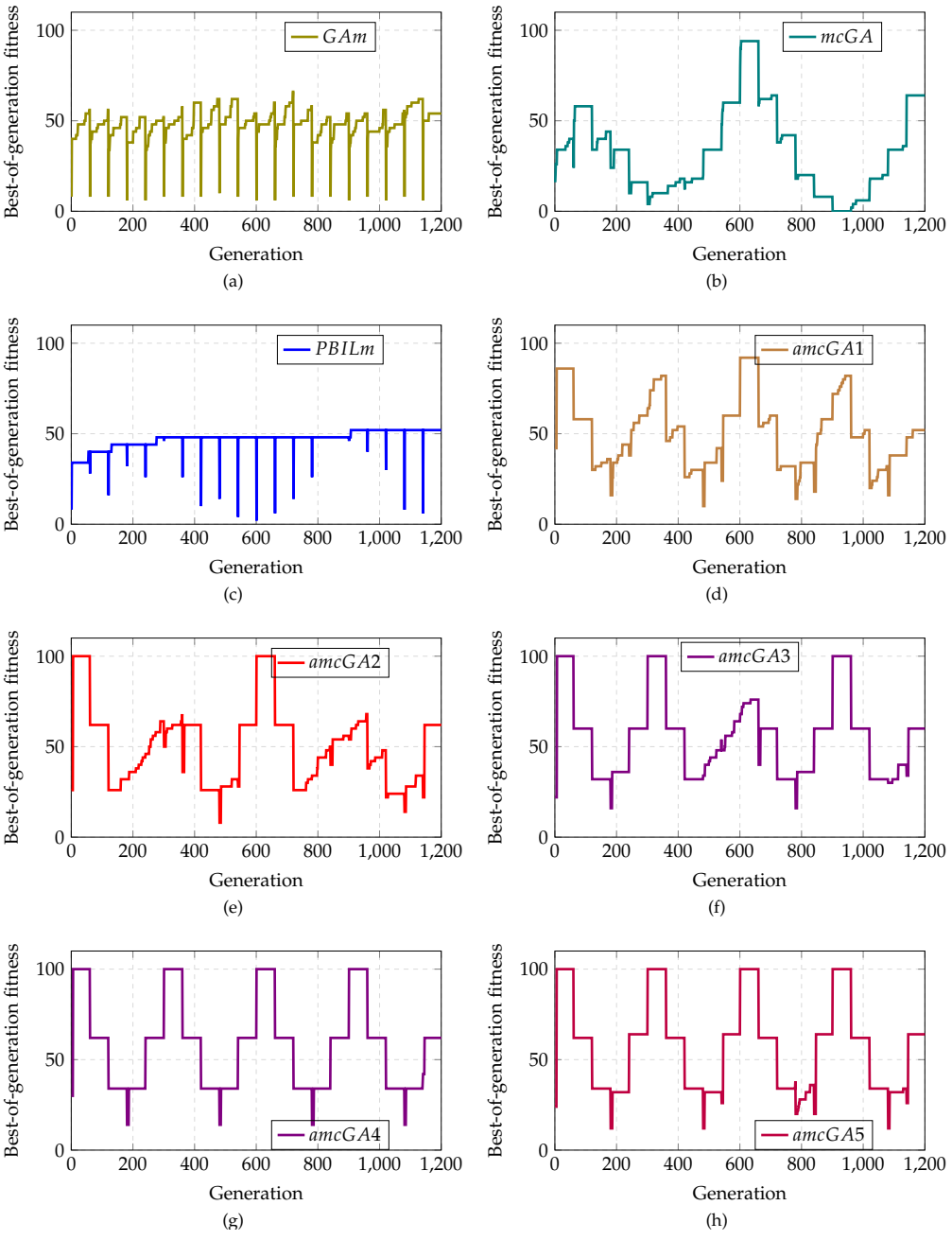


FIGURE C.4: Individual dynamic behaviour of competing algorithms on DDUF2 with $\tau = 60$ and $\rho = 0.2$ in a cyclic environment. Best fitness values achieved each generation are shown in the figure, where the optimum is 100.

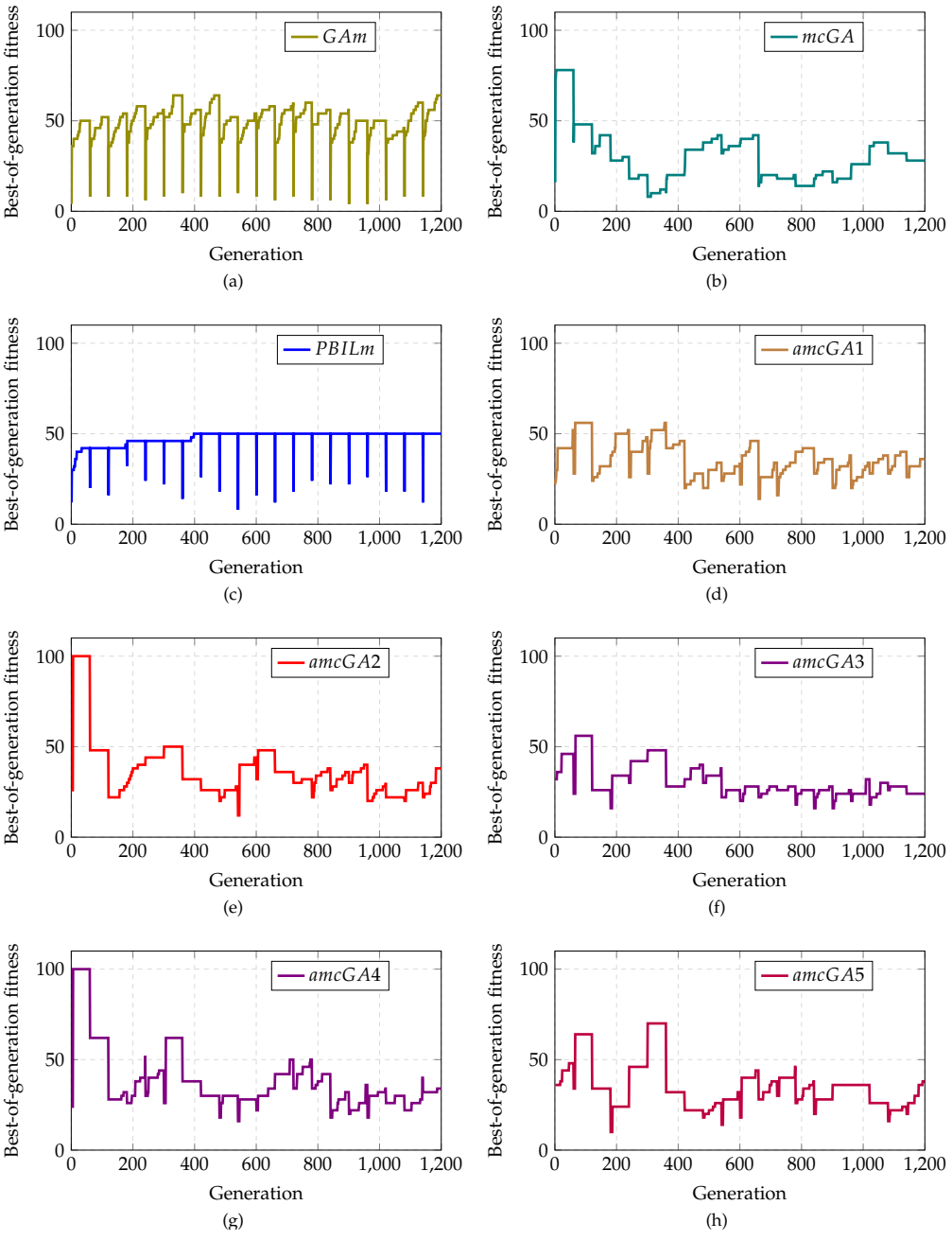


FIGURE C.5: Individual dynamic behaviour of competing algorithms on DDUF2 with $\tau = 60$ and $\rho = 0.2$ in a noisy cyclic environment. Best fitness values achieved each generation are shown in the figure, where the optimum is 100.

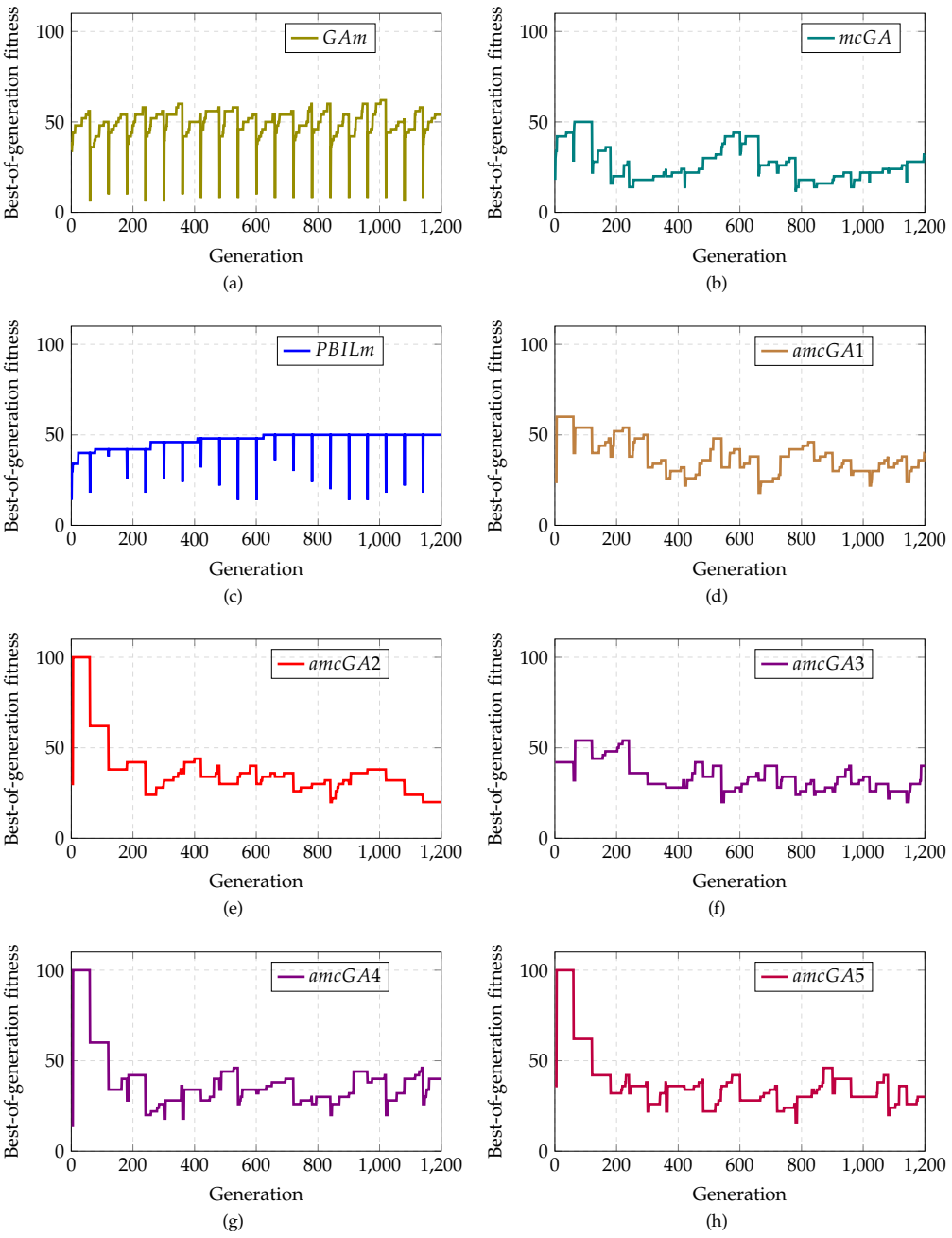


FIGURE C.6: Individual dynamic behaviour of competing algorithms on DDUF2 with $\tau = 60$ and $\rho = 0.2$ in a random environment. Best fitness values achieved each generation are shown in the figure, where the optimum is 100.

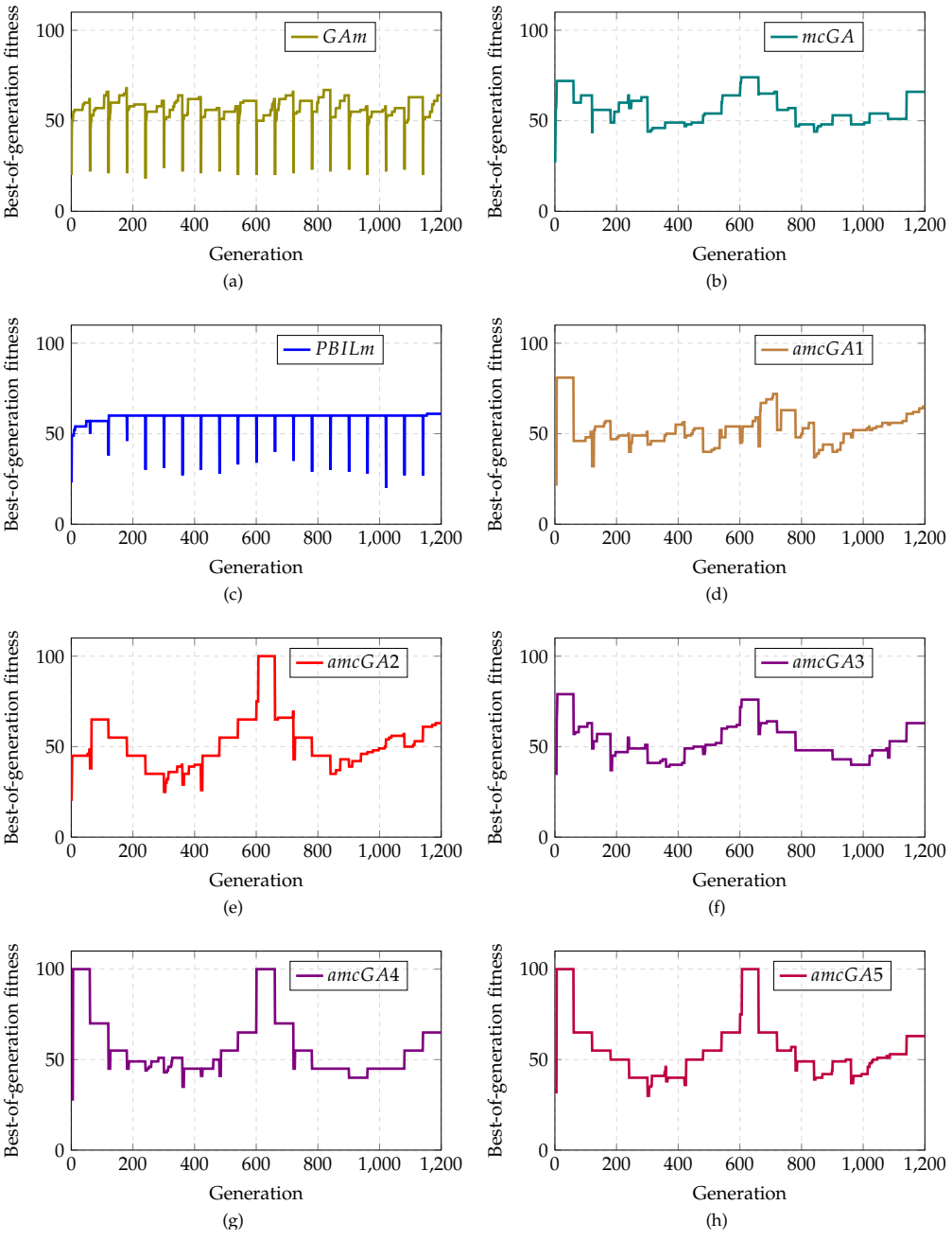


FIGURE C.7: Individual dynamic behaviour of competing algorithms on DDUF3 with $\tau = 60$ and $\rho = 0.2$ in a cyclic environment. Best fitness values achieved each generation are shown in the figure, where the optimum is 100.

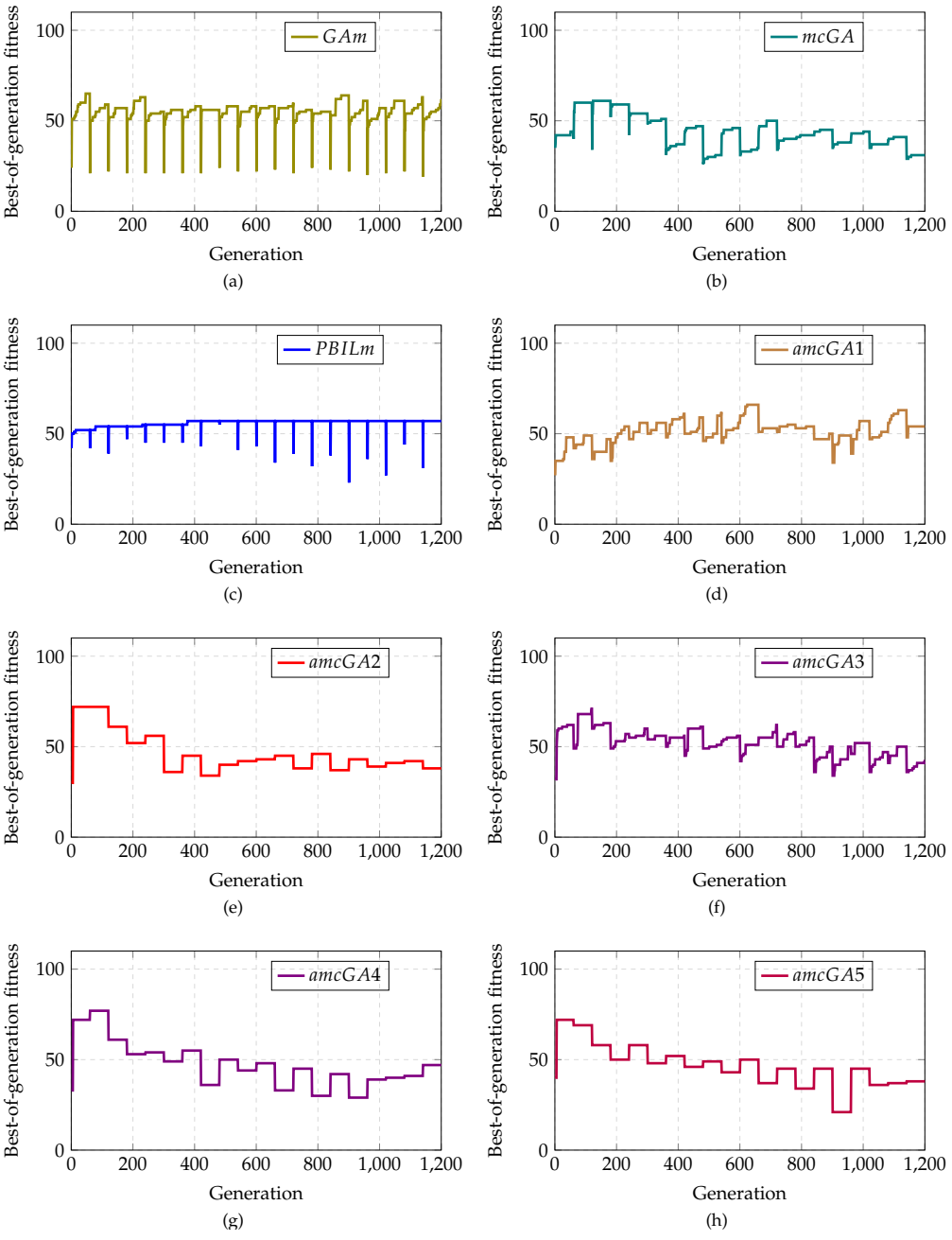


FIGURE C.8: Individual dynamic behaviour of competing algorithms on DDUF3 with $\tau = 60$ and $\rho = 0.2$ in a noisy cyclic environment. Best fitness values achieved each generation are shown in the figure, where the optimum is 100.

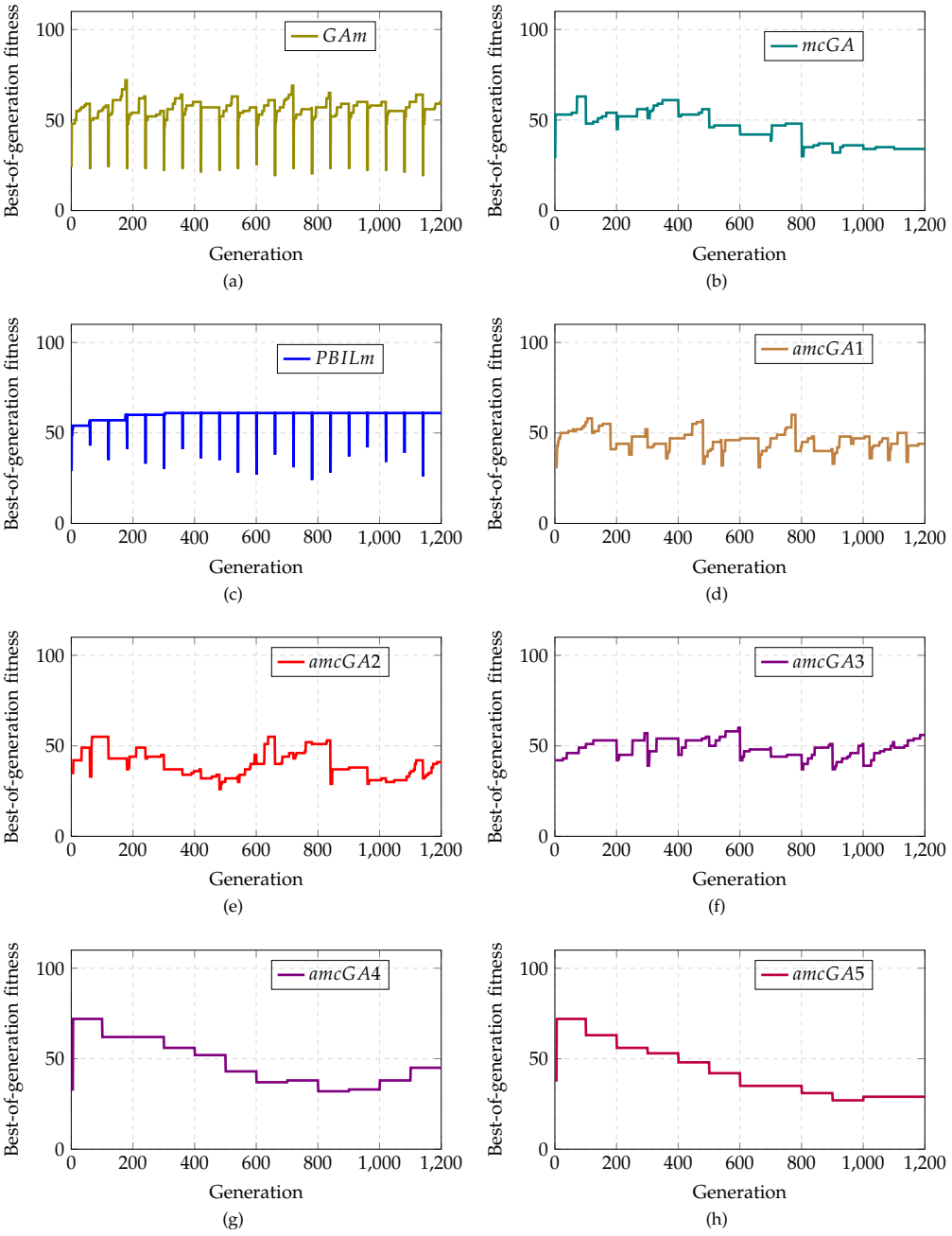


FIGURE C.9: Individual dynamic behaviour of competing algorithms on DDUF3 with $\tau = 60$ and $\rho = 0.2$ in a random environment. Best fitness values achieved each generation are shown in the figure, where the optimum is 100.

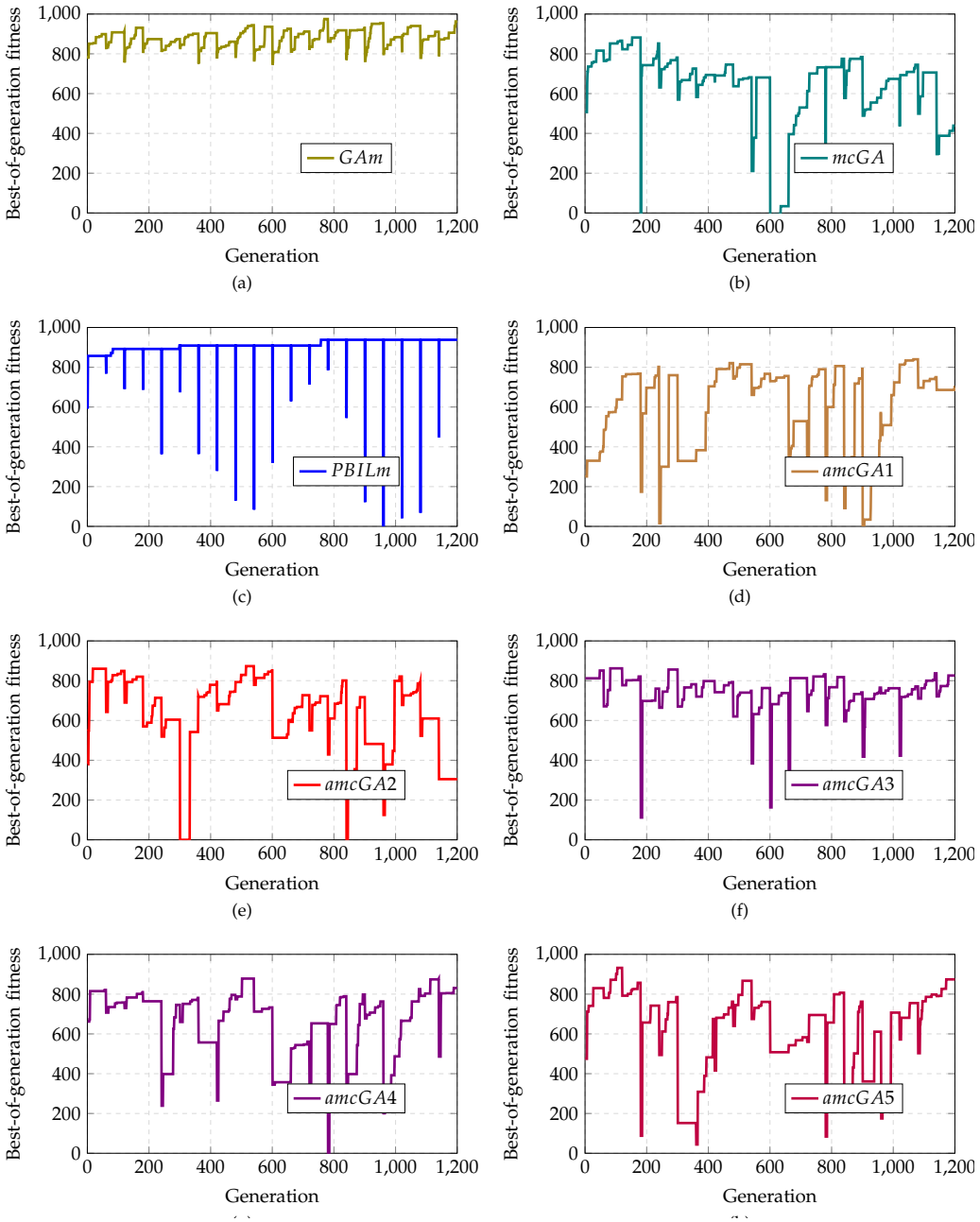


FIGURE C.10: Individual dynamic behaviour of competing algorithms on DKP with $\tau = 60$ and $\rho = 0.2$ in a cyclic environment. Best fitness values achieved each generation are shown in the figure, where the optimum is 1000.

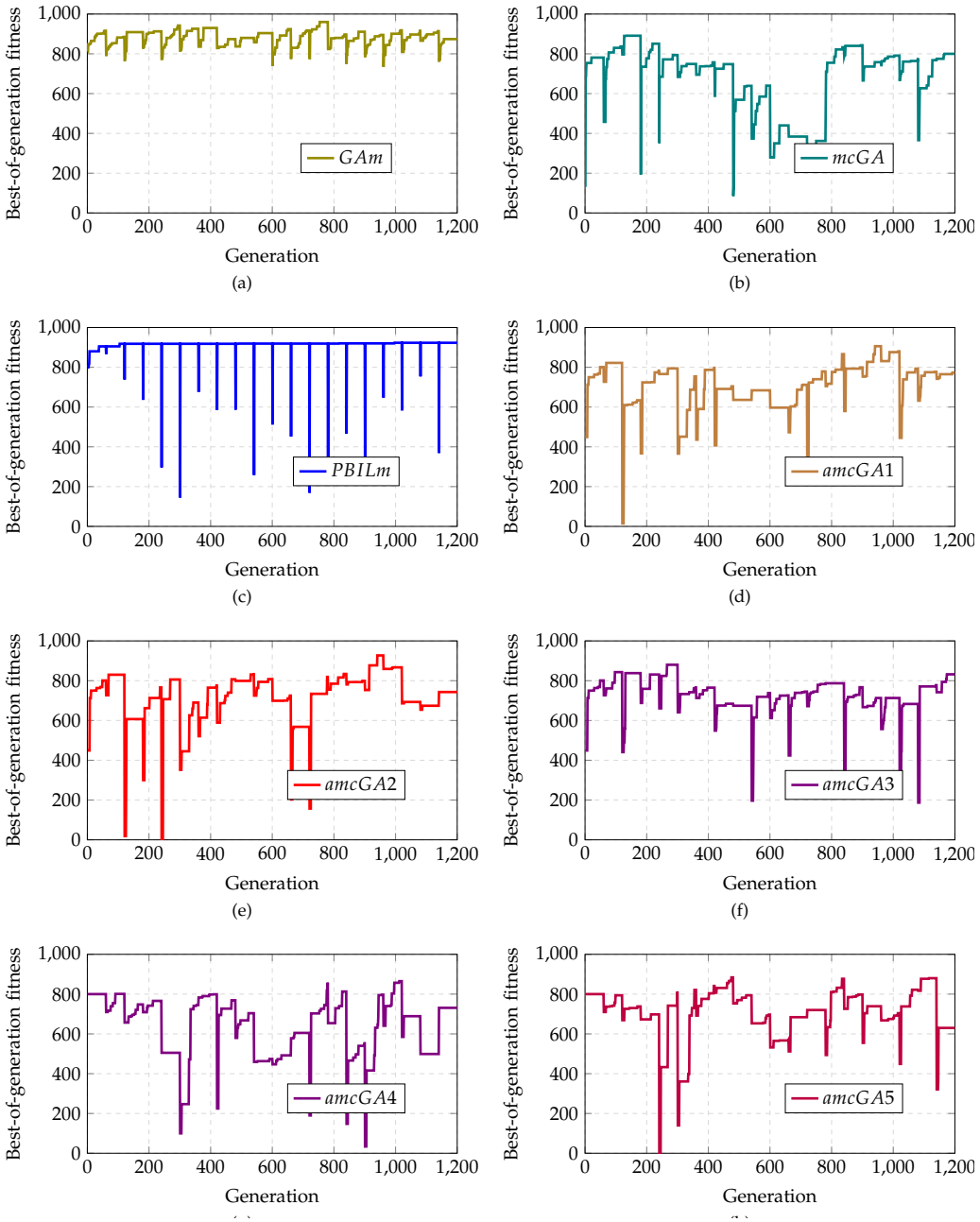


FIGURE C.11: Individual dynamic behaviour of competing algorithms on DKP with $\tau = 60$ and $\rho = 0.2$ in a noisy cyclic environment. Best fitness values achieved each generation are shown in the figure, where the optimum is 1000.

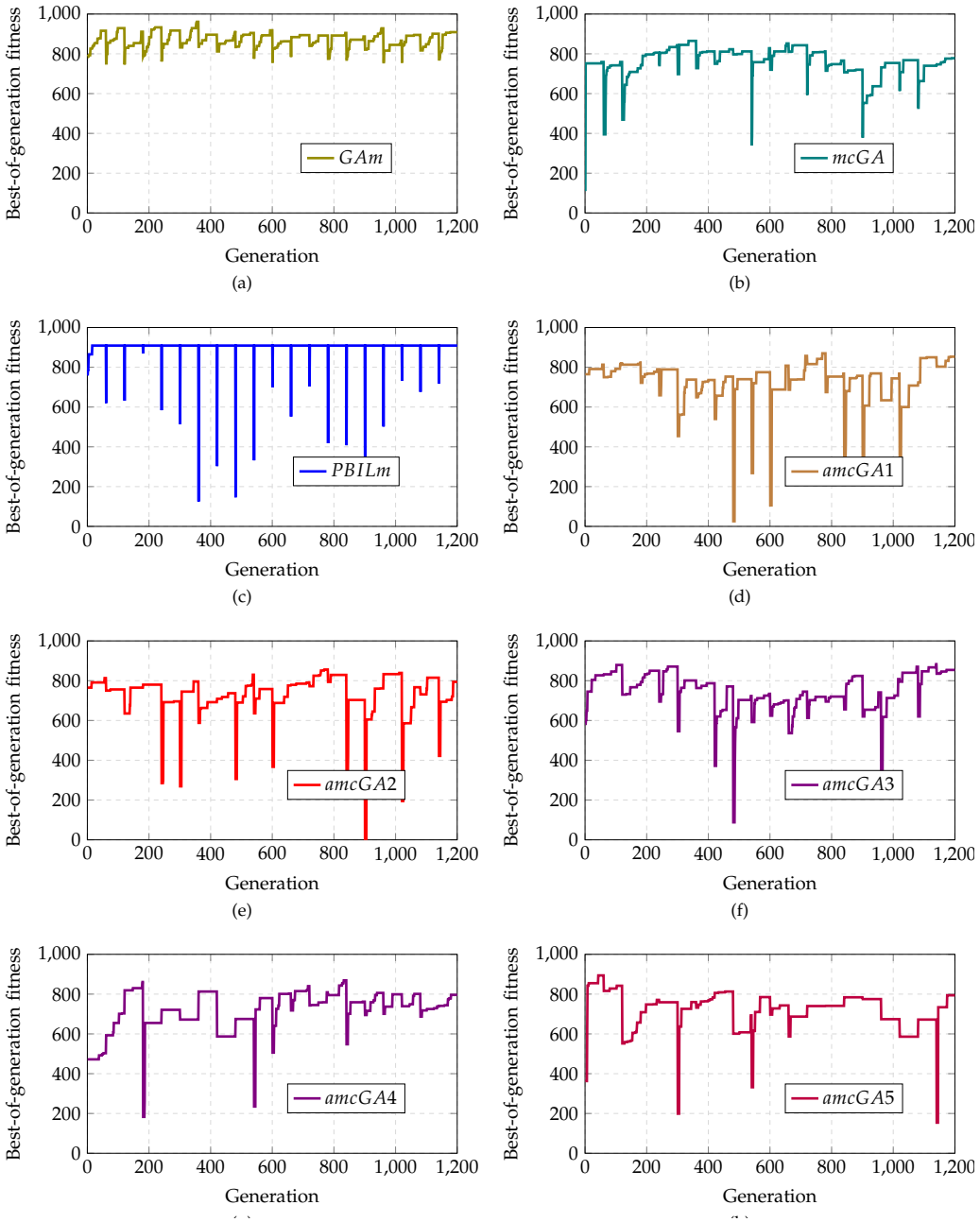


FIGURE C.12: Individual dynamic behaviour of competing algorithms on DKP with $\tau = 60$ and $\rho = 0.2$ in a random environment. Best fitness values achieved each generation are shown in the figure, where the optimum is 1000.

Best of all step-response of all compact dynamic algorithms on TMSDS experiment

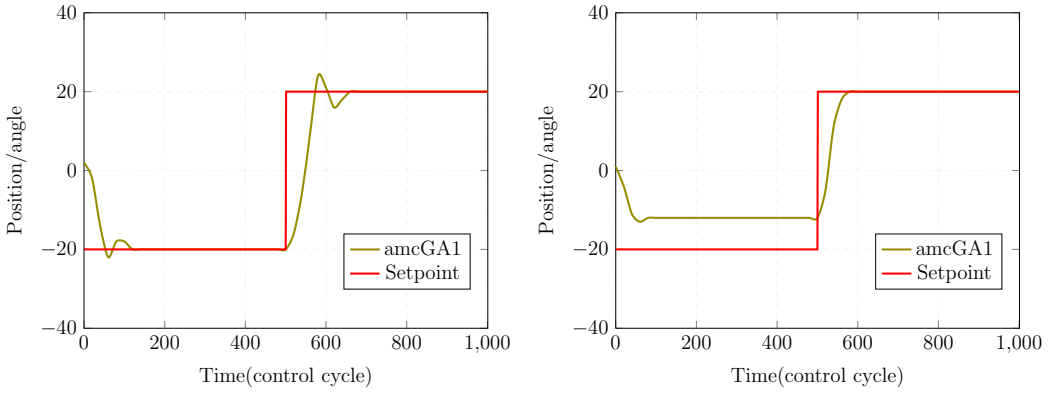


FIGURE C.13: Best control response of the amcGA1: away from magnet (*left*), close to magnet (*right*)

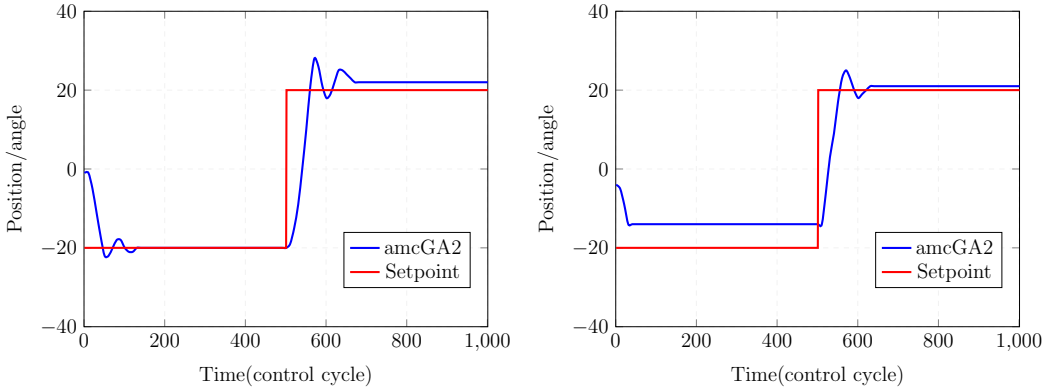


FIGURE C.14: Best control response of the amcGA2: away from magnet (*left*), close to magnet (*right*)

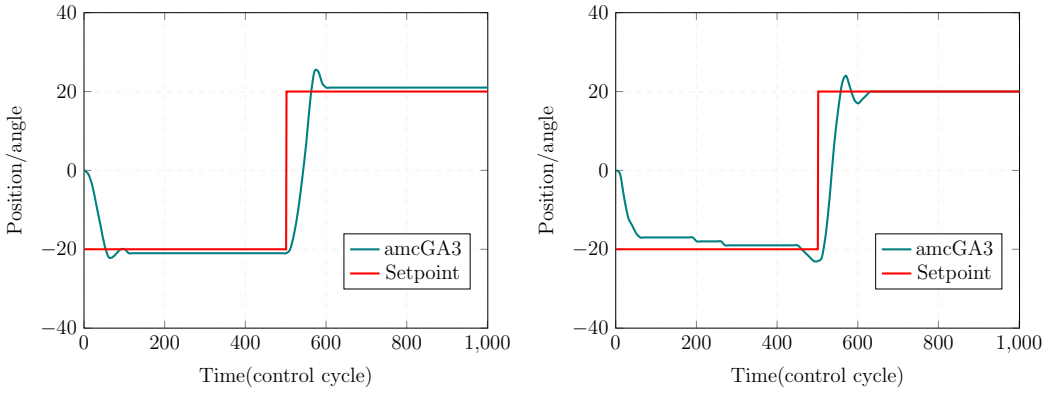


FIGURE C.15: Best control response of the amcGA3: away from magnet (*left*), close to magnet (*right*)

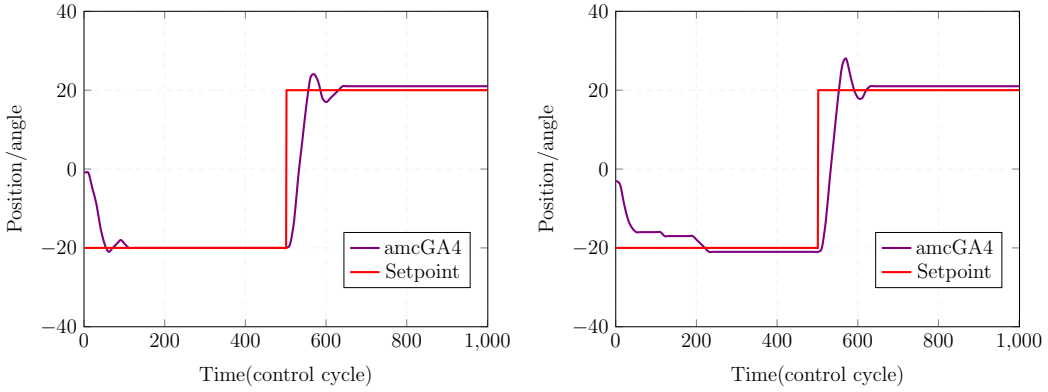


FIGURE C.16: Best control response of the amcGA4: away from magnet (*left*), close to magnet (*right*)

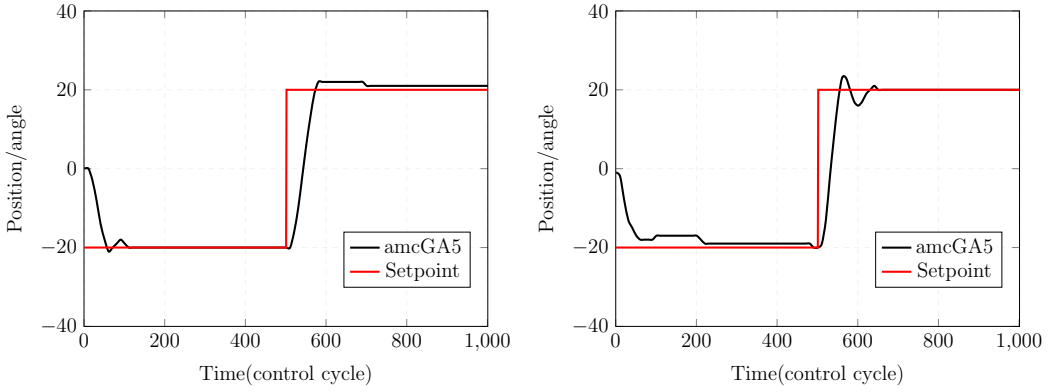


FIGURE C.17: Best control response of the amcGA5: away from magnet (*left*), close to magnet (*right*)

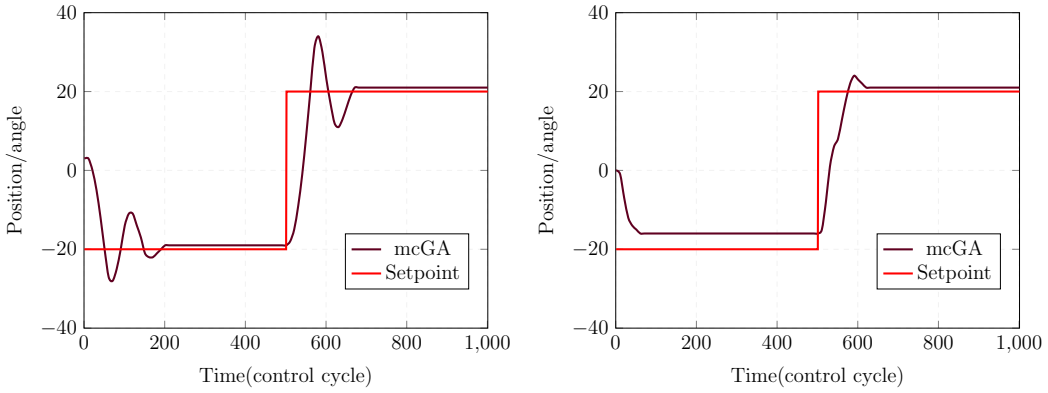


FIGURE C.18: Best control response of the mcGA: away from magnet (*left*), close to magnet (*right*)

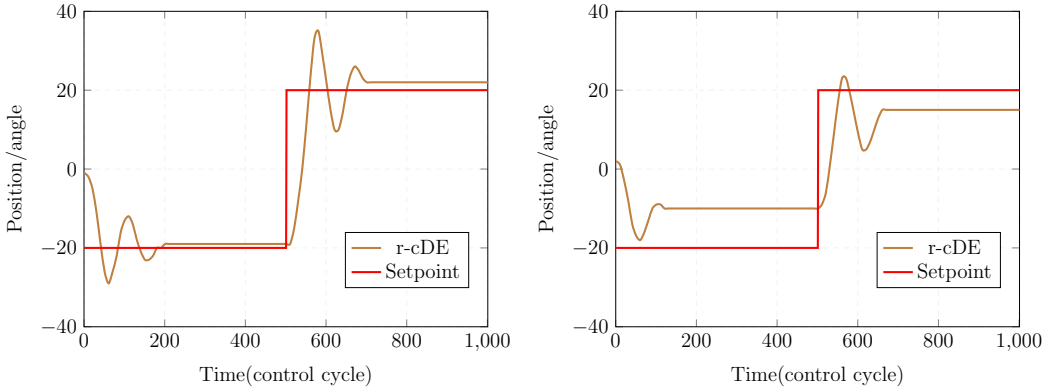


FIGURE C.19: Best control response of the r-cDE: away from magnet (*left*), close to magnet (*right*)

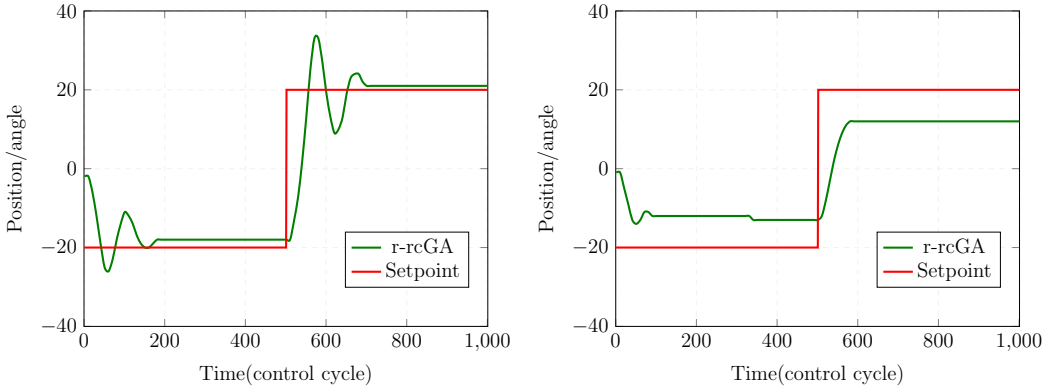


FIGURE C.20: Best control response of the r-rcGA: away from magnet (*left*), close to magnet (*right*)