Chapter 3

DYNAMIC ENVIRONMENT OPTIMISATION ALGORITHMS

This chapter considers approaches to adapting optimisation algorithms to solve dynamic optimisation problems. The fundamental problems involved with applying differential evolution to dynamic environments are explained. An outline of related research in the field is given to provide context on the current state of the field. Two algorithms for dynamic optimisation problems which are based on differential evolution are described in detail.

3.1 Introduction

Optimisation algorithms, developed for static environments, tend to be ineffective when applied to dynamic optimisation problems [Noroozi *et al.*, 2011][Li and Yang, 2009]. Section 3.2 describes how low diversity is the main reason for DE's lack of transferability from static to dynamic environments. A secondary problem of outdated information is also described.

A review of related work by other researchers on adapting population-based algorithms to dynamic optimisation problems (DOPs) is given in Section 3.3. Three main approaches for adapting static optimisation algorithms to dynamic environments are identified from existing literature in Section 3.3.1. These approaches are: increasing diversity, memory

and parallel search. The literature review revealed seven general strategies which are commonly used by optimisation algorithms to achieve the three general approaches. The seven strategies are discussed in Section 3.3.2. Specific algorithms based on GA, PSO, DE and other main algorithms are discussed in Section 3.3.3 and each is shown to employ one or more of the seven strategies.

This thesis investigates the use of DE in dynamic environments. Accordingly, two algorithms found in the literature, which are based on DE, are discussed in detail. DynDE, a multi-population algorithm, with measures to increase diversity, is described in Section 3.4.1. A more recent algorithm, *jDE*, is described in Section 3.4.2.

Dynamic optimisation algorithms, typically, have to react to changes in the environment, which makes it necessary for these algorithms to detect changes. Change detection is discussed in Section 3.5. Conclusions are drawn in Section 3.6.

3.2 Differential Evolution in Dynamic Environments

This section describes two properties of DE that make the algorithm ineffective when solving DOPs. The first, loss of diversity, is discussed in Section 3.2.1. Section 3.2.2 describes the problem of outdated information.

3.2.1 Loss of Diversity

The initial population of DE (and of most evolutionary algorithms) is formed from individuals that are randomly dispersed in the search space. This ensures that the algorithm explores a large area of the fitness landscape during the first generations. Eventually, however, DE converges to an optimum, with all the individuals clustered around a single point, resulting in a loss of diversity. However, the eventual loss of diversity in static environments is not a problem, because having all individuals clustered around the optimum assists with exploitation at the end of the search process.

The loss of diversity is, however, a major reason why evolutionary algorithms are ineffective in dynamic environments. The population lacks the diversity necessary to locate the position of new global optima when changes in the environment occur after the population starts to converge [Zaharie and Zamfirache, 2006]. This phenomenon can

be illustrated by means of an example using DE. Consider Figure 3.1 which depicts the offline error, current error and diversity of the populations (calculated using equation (2.7)) averaged over 30 independent runs on the MPB of the basic DE algorithm described in Section 2.4. Ten changes in the environment are illustrated. Changes occur once every 5 000 function evaluations.

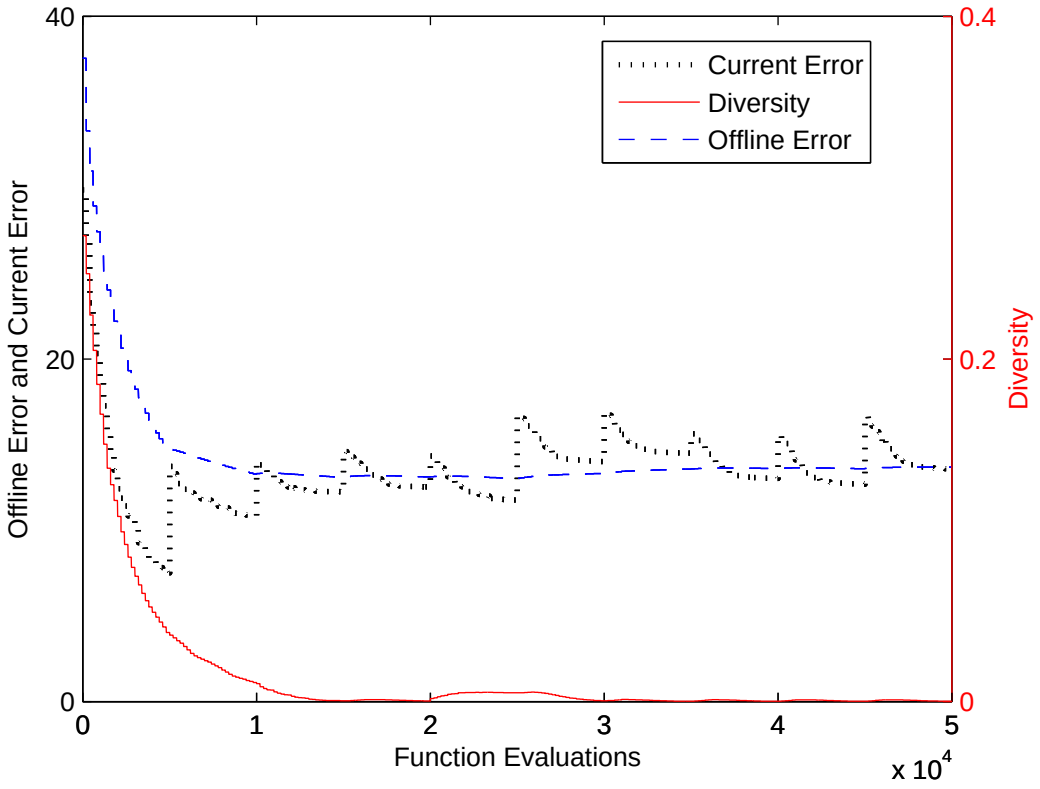

Figure 3.1: Diversity, current error and offline error of basic DE on the MPB

During the period between the commencement of the algorithm and the first change in the environment, the current error and the offline error drop sharply as the DE algorithm converges to one optimum in the environment. The fact that individuals are clustered increasingly closely together is illustrated by the sharp drop in diversity. Directly after the first change in the environment, the current error increases, since the optima have shifted. Although the current error does improve after the first and subsequent changes, it never reaches the low value that was found before the first change occurred. Diversity continues

to drop until it reaches a value close to zero shortly after the second change in the environment (at about 1000 function evaluations in Figure 3.1). This means that all individuals are clustered around a single optimum and that the rest of the fitness landscape is left completely unexplored. This clustering is especially detrimental in DE, since mutations are performed based on the spatial differences between individuals. Consequently, very small mutations are applied as a result of spatially close individuals which restricts further exploration of the fitness landscape. The average diversity per generation was 0.0017, averaged over 30 repeats of DE on the MPB. This value is relatively low and explains why normal DE is not effective in dynamic environments.

3.2.2 Outdated Information

DE performs selection as a competition between parents and offspring. Re-evaluation of parent individuals for each competition comparison in a static environment is unnecessary and ineffective, since the fitness values at particular points in the fitness landscape never change. Re-evaluation thus implies wasted function evaluations.

However, in dynamic environments, the function values of individuals become outdated when the environment changes. As a result, the algorithm can no longer rely on the fitness values of parent individuals to contribute meaningfully to the search process. The incorrect fitness values of the parent individuals may guide the search to regions of the fitness landscape with inferior fitness. A parent with an outdated, high fitness value, which in reality is less fit than its offspring, may be erroneously preserved by the parent versus offspring competition.

The outdated information problem can be solved by re-evaluating the function values of individuals when changes in the environment occur. The changed fitness landscape is reflected in the updated fitness values of parent individuals. Consequently, valid results can be found from the parent versus offspring competition.

3.3 Related Work

Several of the earlier investigations into optimisation in dynamic environments involved approaches based on genetic algorithms [Holland, 1975]. More recently, attention has been

given to other population-based optimisation algorithms like particle swarm optimisation (PSO) [Kennedy and Eberhart, 1995] and differential evolution [Price *et al.*, 2005][Storn, 1996].

This section provides a review of algorithms developed for solving DOPs, with the focus on those algorithms directly related to the objectives of this study. Figure 3.2 shows the layout of this section.

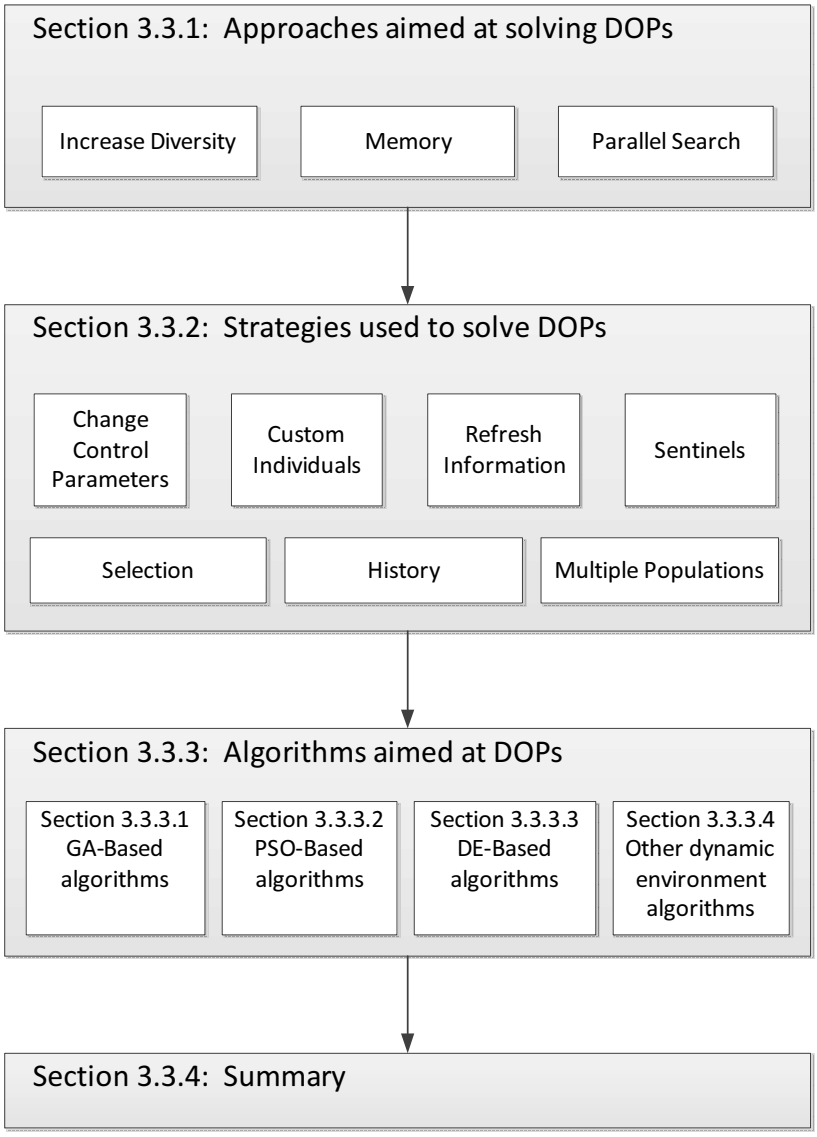

Figure 3.2: Layout of the related work section

The discussion is grouped into four parts. The first part, given in Section 3.3.1, de-

scribes the three main categories into which approaches, tailored to DOPs, can be categorised. The objective of the section is to explain *what* algorithms aimed at DOPs try to achieve. Three broad categories are identified and motivated. The second part, given in Section 3.3.2, describes specific algorithmic strategies aimed at DOPs. This section explains *how* the approaches described in Section 3.3.1 are achieved. Seven strategies are identified and explained. The strategies are discussed generically as far as possible, without details relating to specific implementations. The third part, given in Section 3.3.3, describes particular algorithms aimed at DOPs in terms of the underlying algorithms (GA, PSO, DE, and other). For each algorithm, the underlying strategies that are employed are highlighted. The categorisation of the algorithms of Section 3.3.3 into the strategies identified in Section 3.3.2 is summarised in Section 3.3.4.

3.3.1 Approaches aimed at solving Dynamic Optimisation Problems

A review of the literature identified three main categories of approaches used to adapt algorithms for dynamic environments. At least one of these is present in most algorithms aimed at DOPs. These categories can be seen as the goals which algorithms, aimed at DOPs, try to achieve. The three categories (adapted from [Jin and Branke, 2005]) are:

Increase Diversity: Section 3.2.1 argued that one of the main reasons why traditional population-based algorithms fail, when applied to dynamic environments, is that the algorithms converge to a solution and then lack the diversity required to locate new optima once the environment changes. High diversity results in a larger area of the search space being covered by the population, which increases the probability of relocating optima after a change in the environment occurs [Ursem, 2002]. Jin and Branke [2005] and Nguyen *et al.* [2012] pointed out that most algorithms try to remedy the low diversity problem by either explicitly increasing diversity after a change in the environment is detected, or by maintaining more diversity during the optimisation process.

Memory: Optima in dynamic environments shift, disappear, or new optima may appear when changes in the environment occur. However, unless the changes are large, information regarding where optima were located before a change can still be used

to locate new optima in their vicinity after the change. A change in the environment can also lead to the conversion of global optima to local optima and vice versa. Knowing the location of all (or at least several) of the local optima, can accelerate the location of a new global optimum after a change in the environment.

A cyclic dynamic environment (discussed in Section 2.5.2) periodically returns to the same state. A long-term memory of previously found, good solutions can be used to recognise locations of optima when a complete cycle has occurred. Memory has also been used to extrapolate future states of the environment in situations where a correlation between successive changes exists.

Preserving information, in the form of a memory of the search space characteristics from before a change in the environment, can thus be utilised to improve the exploration of the fitness landscape after a change in the environment [Nguyen *et al.*, 2012]. Note that the potential benefits of memory-approaches diminish as the severity of changes in the environment increases or becomes more chaotic.

Parallel Search: Evolutionary and swarm algorithms are inherently parallel search algorithms by virtue of multiple individuals in the population. Many algorithms further extend this parallelism by employing independent parallel searches to locate optima, mostly by using multiple independent sub-populations [Nguyen *et al.*, 2012]. Several promising optima can be investigated simultaneously by using multiple populations, which increases the likelihood of discovering a global optimum. It is common for algorithms to prevent sub-populations from converging to the same optimum and to control how individuals are distributed among the sub-populations.

The following section details the mechanisms employed by algorithms to achieve the above mentioned approaches.

3.3.2 Strategies used to solve Dynamic Optimisation Problems

Seven common strategies, used to create optimisation algorithms for dynamic environments, were identified in this study and are presented in this section. Strategies typically fall within one or more of the approach categories, which were identified in the previous

section. The strategies are described, as far as possible, independently of the base algorithms (specific GA, PSO and DE algorithms that employ these strategies are discussed in the following section). The seven strategies are:

1. **Change Control Parameters:** Several control parameters of evolutionary and swarm algorithms directly affect diversity, for example, the mutation rate used in a GA and the scale factor used in DE [Zaharie, 2002a]. Other parameters, for example, population size [Teo, 2006], DE scheme (refer to Section 2.4.2) and PSO neighbourhood structure, affect the convergence rate which indirectly influences diversity. Several researchers experimentally determined the most effective control parameters to use in conjunction with their algorithms on dynamic environments [Mendes and Mohais, 2005]. Other algorithms adapt control parameters during the course of the algorithm's execution [Brest *et al.*, 2009].

   Changed control parameters are generally used to increase the diversity of the population. This assists with locating global optima (since a larger area of the search space is explored) and helps prevent convergence of the entire population, which hampers the algorithm's ability to respond to changes in the environment.

2. **Custom Individuals:** Several approaches to increasing diversity have focused on changing the behaviour of all, or a portion of, individuals within the population [Cruz *et al.*, 2011]. The custom individuals are normally used to increase diversity by perturbating their locations using random noise [Blackwell and Branke, 2006]. By employing these custom individuals as a portion of the population, the normal individuals within the population can converge to an optimum while the population, as a whole, can still have a relatively high diversity due to the influence of the custom individuals.

3. **Refresh Information:** To prevent populations from converging to a single point (i.e. losing all diversity), steps can be taken to introduce fresh genetic material into the population [Grefenstette, 1999]. Common approaches include reinitialising the entire population, incorporating random individuals into the population, reinitialising the personal best value in PSO, and employing an aging metaphor to replace

individuals and populations that have survived for many generations with random individuals.

Refreshing information has the benefit of increasing diversity, but also of assisting the parallel search of the fitness landscape as new global optima may be discovered after a population is reinitialised. Depending on the algorithm, the potential exists of losing information when reinitialising individuals and populations. This approach is consequently used, mostly in conjunction with multiple populations, so that information regarding the locations of global optima would not be lost.

4. **Sentinels:** Individuals that are kept at constant positions within the search space are referred to as sentinels. Sentinels are commonly used to detect changes in the environments (see Section 3.5), but a uniformly distributed group of sentinels in a population can also increase diversity by providing genetic material from areas of the search space not represented by normal individuals [Morrison, 2004]. For example, a sentinel that acquired a relatively high fitness after a change in the environment can be used to guide the population towards an optimum.

5. **Selection:** Diversity can be explicitly controlled by employing a custom selection operator. For example, a selection operator that not only selects individuals based on fitness, but which is also more likely to select individuals that are located in underpopulated areas of the search space [Oppacher and Wineberg, 1999]. Individuals that have converged to a single point thus have a lower probability of surviving to the next generation. The resulting population is thus likely to be more widely dispersed within the search space.

6. **History:** Keeping track of good solutions within the fitness landscape can be beneficial in cyclic dynamic environments. An archive which acts as a memory of the search history is utilised by algorithms to store good solutions over several generations [Cruz *et al.*, 2011]. When optima reappear in the same position as previously, individuals from the archive can redirect the search to these positions by contributing genetic material to the main population [Ramsey and Grefenstette, 1993].

   A secondary benefit of an archive of previous solutions is that it can be used to

increase diversity by injecting genetic material from areas of the fitness landscape other than the area to which the main population has converged.

With reference to the three categories discussed in the previous section, the use of an archive is an example of an explicit memory of the fitness landscape.

7. **Multiple Populations:** The use of multiple populations falls into all three of the previously mentioned categories: diversity, memory and parallel search. By enforcing a spatial distance between sub-populations it can be ensured that sub-populations converge to different optima in the environment, which in turn lowers the risk of converging to a local optimum (especially in situations where the number of sub-populations is roughly equal to the number of optima). By virtue of more widely distributed individuals, diversity is increased [Cruz *et al.*, 2011]. Keeping track of several optima means that more information about the search space is available when changes in the environment occur; in effect a clearer memory of the previous shape of the fitness landscape is available [Ursem, 2000]. Multiple populations is a form of implicit memory of the locations of optima before a change in the environment. Knowing the locations of pre-change optima is useful when changes in the environment are relatively small, as these locations are a good starting point for searches for the new locations.

The majority of evolutionary algorithms make use of at least one of the seven strategies described in this section. The exact details of how these strategies are implemented in the various algorithms aimed at DOPs are given in the following section.

3.3.3 Algorithms aimed at Dynamic Optimisation Problems

An overview of specific algorithms for optimising dynamic environments is given in this section, with reference to the seven strategies identified in the previous section. These algorithms are discussed in the following sections in terms of their respective base algorithms. Section 3.3.3.1 describes algorithms based on genetic algorithms, Section 3.3.3.2 describes algorithms based on particle swarm optimisation, Section 3.3.3.3 describes algorithms based on differential evolution, and Section 3.3.3.4 describes algorithms with hybrids or other bases.

3.3.3.1 Genetic Algorithm-Based Algorithms

Early GA approaches to solving DOPs endeavoured to increase population diversity after changes occurred in the dynamic environment by increasing the mutation rate. Cobb [1990] suggested dramatically increasing the mutation rate of a GA after a change occurred, while Vavak *et al.* [1997] advocated a more gradual increase. Both these algorithms can be classified under the Change Control Parameters strategy as the mutation rate is one of a GA's control parameters. A high mutation rate encourages exploration by introducing genetic material from currently unexplored regions of the fitness landscape. This is useful after a change in the environment since optima would, potentially, have shifted to unpopulated areas of the search space.

Later algorithms sustain high diversity throughout the optimisation process as opposed to only increasing diversity directly after changes in the environment. Approaches aimed at maintaining a high amount of diversity during the entire execution of the GA algorithm include Grefenstette's random immigrants algorithm [Grefenstette, 1999], which introduces random individuals into a GA's population after each generation. The random individuals act as a steady source for new genetic material which assists in locating new optima. This approach can be classified as a Refresh Information strategy. Morrison [2004] proposed an algorithm which, in contrast to introducing random individuals, makes use of sentinels that are uniformly distributed throughout the search space to increase diversity. Sentinels are used in selection and crossover to create individuals for the next generation, but are not replaced or mutated. Sentinels placed in a grid, rather than random placement, guarantee a uniform sampling of the fitness landscape, making it more likely that the algorithms will be able to detect and respond to changes in the environment. The sentinels introduce diverse genetic material into the population that would otherwise have converged to a single point. This algorithm is classified under the Sentinels strategy. A disadvantage of Morrison's sentinel approach is that a large number of sentinels must be maintained to achieve a useful coverage of the fitness landscape. Function evaluations are effectively wasted on sentinels that are, for the most part, located in inferior regions of the fitness landscape.

The thermodynamic GA [Mori *et al.*, 1996], [Mori *et al.*, 1997], [Mori *et al.*, 1998]

explicitly controls the population's diversity throughout the optimisation process by selecting individuals for the next population based not only on their fitness, but also on the rarity of their genes. This Selection strategy helps to prevent the convergence of the population to a single point, thus improving the chance of locating new optima after a change in the environment.

Utilising the search history with a History strategy is useful in dynamic optimisation problems where the shape of the fitness landscape is oscillating or cyclic, i.e. changes in the dynamic environment will result in the environment's returning to the same configuration at a future stage. For these problems, maintaining an explicit memory of good solutions has been found to be especially useful. Ramsey and Grefenstette [1993] made use of a knowledge base of individuals that performed well in previous generations. These individuals are inserted into the population when a change occurs that results in a previously seen environment. Yang [2005] made use of a memory of individuals which are used as memory-based immigrants. During each generation, the individuals in the memory are re-evaluated and the best memory individual is repeatedly mutated to create a number of immigrant individuals. These immigrants then replace the worst-performing individuals in the normal population. This results in a constant influx of genetic material from previously found good solutions, which is useful when optima reappear at previous positions.

Algorithms typically use a memory archive of finite size. Simões and Costa [2009a] showed that the optimal memory size is algorithm and problem dependent. Once the memory reaches its maximum size, the algorithms must replace an element in the memory when a new element is added. In general, strategies for replacing memory individuals aim to keep the memory population as diverse as possible. Typical strategies include, replacing the memory individual that is spatially closest to the new individual, or removing the individual that will result in the highest variance in spatial positioning in the memory archive. Zhu *et al.* [2011] showed that the typical memory updating strategies may fail to accomplish the goal of high memory diversity since the same memory index can successively be replaced. Zhu *et al.* [2011] accordingly suggests a memory archive update strategy which replaces most similar individuals (if the new individual is more fit than the memory individual), but also keeps track of how often each memory archive index was updated in

the recent past. If an index was updated more than a specified number of times, the new individual rather replaces the oldest memory individual. This History strategy prevents the successive replacement of the same individual and was shown to result in a more diverse archive population.

Simões and Costa [2009b] used a memory of good individuals, each associated with a particular state of the environment (the location of a good optimum in a particular time in the dynamic environment). The memory individuals are used to predict when changes in the environment will occur and also what state the environment is likely to assume. Nonlinear regression is used to predict when changes are likely to occur. Markov chains are used to predict the next state of the environment. Individuals from the memory archive which represent the predicted state are introduced into the main population after a prediction. The memory individuals thus serve to guide the main population to solutions found earlier in the search process. The algorithm is consequently classified as using a History strategy. The techniques proposed by Simões and Costa [2009b] to predict the environment state will only work when a pattern between changes in the environment exists and if the environment can assume a relatively small number of states (roughly equal to the number of states stored in the Markov chain).

Memory is employed in the majority of algorithms to retain information regarding the location of optima in the environment before a change occurred, especially in non-cyclic dynamic environments. The goal is to track optima through the fitness landscape as changes occur. This is generally achieved by using multiple, independent populations to locate various optima. A key feature of these approaches is that independent populations are allowed to search for optima in parallel. Three of the seminal GA algorithms employing this strategy are self-organizing scouts (SOS) [Branke *et al.*, 2000][Branke, 2002][Branke and Schmeck, 2003], shifting balance GA (SBGA) [Oppacher and Wineberg, 1999] and the multinational GA (MGA) [Ursem, 2000]. All three of these approaches make use of a custom technique to intelligently distribute individuals among the populations.

SOS uses a large base population to search for optima in the fitness landscape. When an optimum is located, a small scout population is left to guard and further optimise the optimum while the base population continues to search for new optima. The area guarded by the scout population is then excluded from the fitness landscape of the base population

by reinitialising base population individuals that stray within a threshold distance from the best individual in the scout population. Diversity within scout populations is maintained by using a mutation step-size which is proportional to the area of the search space covered by the scout population (a Change Control Parameters strategy). This ensures that individuals from a scout population do not converge completely, but explore a relatively small area of the search space.

Individuals are distributed between the various scout populations and the base population based on a quality measure calculated for each sub-population. The quality measure depends on the proportional fitness and proportional increase in fitness in the last generation of each sub-population in comparison with the other sub-populations. Each population is allocated a subset of the total number of individuals equal to the proportion of the quality measure of the population over the sum of the quality measures of all populations. Populations with a high fitness and recent large improvements in fitness are thus allocated more individuals. Maximum and minimum population sizes of the base and scout populations are parameters to the algorithm. SOS can be classified as using a Multiple Population strategy by virtue of the base and scout populations.

SOS aims to keep the bulk of the individuals in a single base population searching for new optima, while SBGA groups a single core population around the best found optima and uses smaller populations, called colonies, to search for new optima. The purpose of a SOS scout population is to exploit an optimum that was found by the base population, while the goal of a SBGA colony population is to find new optima. Colonies are kept away from the core population by selecting parents for reproduction based on their distance from the core population. By selecting parents that are far from the core population, their offspring are generally also located far from the core population. SBGA thus uses a Selection and a Multiple Population strategy. The information contained in the colony populations is shared with the core population by means of migrant individuals that are periodically transferred from colonies to the core population. This influx of genetic material from the colonies can cause the core population to shift its location to promising regions of the fitness landscape discovered by colonies. Alternatively, if migrant individuals have poor fitness values, the selection operators will not consider these migrant individuals for parents, and these individuals will consequently die out. The SBGA process allows for

the broad exploration of the fitness landscape and the discovery of several optima by the colonies. Optima are consequently tracked by the colonies, which assists in the location of new global optima after changes in the environment occur.

The MGA algorithm [Ursem, 2000] allocates individuals to sub-populations based on the optimum on which each individual is located. MGA uses *hill-valley detection* to form sub-populations. Hill-valley detection randomly samples points between two individuals to determine whether the two individuals are located on the same optimum in order to group individuals into sub-populations. Multiple optima are detected if any of the intermediate points that are sampled yields a lower fitness value than the two individuals (assuming maximisation). Groupings ensure that each sub-population resides on a different optimum.

Parent individuals selected for the creation of offspring for a given sub-population are not only selected from that sub-population but also from other sub-populations (this is in contrast to SOS and SBGA). When offspring are created, hill-valley detection determines whether each offspring individual is to remain in the current sub-population, whether the offspring individual should join another sub-population, or whether the offspring individual should be placed in an entirely new sub-population. Sub-populations are prevented from losing all their individuals by a selection operator that divides the fitness of an individual by its sub-population size. Small sub-populations thus have an evolutionary advantage over larger sub-populations (since a smaller denominator is used) and are likely to continue to exist and contribute genetic material to successive generations. Independent sub-populations increase diversity, indirectly store information regarding the shape of the fitness landscape, and contribute to locating optima in parallel. MGA uses the Selection and Multiple Populations strategies.

3.3.3.2 Particle Swarm Optimisation-Based Algorithms

Considerable success has been achieved in applying modern optimisation algorithms, such as PSO, to dynamic optimisation problems. One of the first investigations into using PSO in dynamic environments is the algorithm suggested by Carlisle and Dozier [2000] wherein the personal best value of particles were periodically reset to their current position. This has the effect of refreshing information about the environment and encouraging particles to explore other parts of the search space (the algorithm thus uses the Refresh Information

strategy). This algorithm was subsequently improved by the authors [Carlisle and Dozier, 2002] by first determining whether the current position is in fact better than the existing personal best value before resetting the value.

The Refresh Information strategy was also used in another of the early PSO-based algorithms, created by Hu and Eberhart [2002]. The authors suggested that diversity should be increased by reinitialising the particles in the PSO algorithm when a change in the environment occurs.

Later diversity-increasing approaches include charged particles [Blackwell and Bentley, 2002], where each particle of a PSO is assigned a virtual charge and is then allowed to repel other particles based on the laws of electrostatics. The repulsive force between particles prevents the entire population from converging to a single point, which allows the discovery of multiple optima and supports exploration after a change in the environment. This is an example of an algorithm employing the Custom Individuals strategy. Another example of this strategy is the idea of increasing diversity by reinitialising a portion of the swarm of particles within a hyper-sphere, centred around the best particle within the swarm [Blackwell and Branke, 2004, 2006]. These particles are called *quantum particles* and were implemented in a PSO algorithm. The cloud radius determines how far from the best particle the quantum particles are dispersed.

Investigations into finding an appropriate neighbourhood structure for PSO [Li and Dam, 2003], [Janson and Middendorf, 2004] have been conducted, since these parameters greatly affect the diversity of the swarm. For example, using a star network topology in PSO results in faster convergence but greater loss of diversity than using a ring social structure where information regarding the global best position is spread through the swarm more gradually. In general, a neighbourhood structure that increases diversity is more effective in dynamic environments, since diverse particles are able to explore the fitness landscape more effectively after a change in the environment. These approaches are classified under the Change Control Parameters strategy.

The dynamic heterogeneous PSO (DHPSO) is an algorithm based on the idea that particles in the swarm do not necessarily need to follow the same behaviour, or even maintain the same behaviour during the entire optimisation process [Leonard *et al.*, 2011]. During a particular period in the optimisation process it may, for example, be useful for a

subset of the particles to focus on exploitation, while the other particles perform a more explorative role. DHPSO allows particles that have stagnated (i.e. particles that have not shown any improvement for ten iterations) to randomly select a new behaviour from one of the following five options: standard PSO, social PSO (the cognitive component of the standard PSO is discarded), cognitive PSO (the social component of the standard PSO is discarded), bare bones PSO [Kennedy, 2003], and modified bare bones PSO [Kennedy, 2003]. This process allows the particle behaviour to adapt to the most appropriate model during different stages of the optimisation process. DHPSO responds to changes in the environment by reinitialising half of all particles when change occurs and re-evaluating the other half to inhibit the problem of outdated information. The algorithm thus utilised the Change Control Parameters and Refresh Information strategies.

Parrott and Li [2004] suggested a multiple swarm PSO approach to solving DOPs, called speciation. Multiple swarms correspond to the idea of multiple populations in GAs, and have the same benefits: increased diversity and parallel discovery of optima. Speciation can thus be classified as a Multiple Populations strategy. The social component of PSO provides a simple method to divide the swarm into sub-swarms when using speciation. A particle is allocated to a sub-swarm if the Euclidean distance between the position of the particle and the best particle in the sub-swarm (referred to as the species seed) is below a certain threshold value. Species seeds are determined by processing a list of all particles sorted in descending order of fitness. The set of species seeds is constructed by adding each particle, encountered in the list, that is not within the threshold distance from any of the particles already in the set of species seeds [Parrott and Li, 2006]. The global best value of each particle within a sub-swarm is set to the personal best value of the species seed. The sizes of sub-swarms are dynamic. Particles can migrate to another sub-swarm by moving too far away from their current sub-swarm's best particle, or by moving closer to another sub-swarm's best particle. It is allowable for a sub-swarm to contain only one particle in which case the particle merely acts as scout guarding an optimum. As a mechanism to prevent sub-swarms from becoming too large, a maximum sub-swarm size is defined. When a sub-swarm's size exceeds this limit, the particles with the lowest fitness are randomly reinitialised and allocated to appropriate sub-swarms. The Refresh Information strategy is used by this algorithm when reinitialising particles, and also by re-evaluating the personal

best value of each particle after each generation. The speciation algorithm provides a natural mechanism for particles to form, join and split sub-swarms.

Blackwell and Branke [2006] introduced a multiple swarm PSO-based algorithm (MPSO) that is based on three components: exclusion, anti-convergence and quantum individuals. In contrast to earlier algorithms, all sub-swarms contain the same number of particles. The aim of having multiple sub-swarms is that each sub-swarm should be positioned on its own, promising optimum in the environment. Unfortunately, sub-swarms often converge to the same optimum, hence decreasing diversity. Exclusion [Blackwell and Branke, 2004] is a technique meant to prevent sub-swarms from clustering around the same optimum by means of reinitialising sub-swarms that stray within a threshold Euclidean distance from better performing sub-swarms. This threshold distance is called the exclusion radius which is calculated as $r_{excl} = (V_{max,F} - V_{min,F})/(2n_p^{\frac{1}{n_d}})$ where $V_{max,F}$ and $V_{min,F}$ are the upper and lower search range of function F in the n_d dimensions (assuming equal ranges for all dimensions), and n_p is the number of optima.

The anti-convergence component of the algorithm was incorporated to prevent stagnation of the particles in the search space. This is achieved by reinitialising the weakest sub-swarm if it is found that all sub-swarms have converged. Convergence of a sub-swarm is detected if all particles within the sub-swarm fall within a threshold Euclidean distance of each other. This threshold is called the convergence radius. MPSO thus uses the Multiple Populations, Refresh Information (through anti-convergence reinitialisation), and Custom Individuals (by merit of the quantum individuals) strategies.

MPSO has the disadvantage that the severity of changes in the environment is a parameter to the algorithm which is used to calculate the cloud radius. While this information is readily available when employing a benchmark function, it is unlikely that this information will be known to the algorithms in real-world dynamic optimisation problems.

The MPSO algorithm was adapted and improved by del Amo *et al.* [2010] to reinitialise or pause sub-swarms that perform badly. For each swarm a history of improvements (the difference in fitness from successive iterations) is stored. A swarm is flagged as performing badly if, for five iterations, its improvement falls below 15% of its best improvement since the last change in the environment and if the swarm is in the set containing the 20% of swarms that have the lowest fitness value. Once a swarm is thus flagged, the algorithm,

using a history of previous changes, then estimates the number of iterations before the next change in the environment. Should less than 20% of the estimated period between changes remain, the swarm is paused until the next change in the environment (hence preventing it from wasting function evaluations). Otherwise, the swarm is reinitialised to continue searching for new optima. This algorithm thus uses the Multiple Populations, Refresh Information, and Custom Individuals strategies present in MPSO and incorporates a History strategy. Disadvantages of this algorithm include the fact that equally sized periods between changes in the environment are assumed and that constants used by the algorithm are problem dependent. For example, using 20% of the period between changes to determine whether to reinitialise or to pause the swarm may not be an effective choice for all environments. An environment in which the change period is large may enable a swarm to locate a new optimum in less than 20% of the change period, in which case the swarm should be reinitialised rather than paused. Conversely, reinitialisation of the swarm would be less likely to lead to the discovery of an optimum in an environment with a low change period, and a value larger than 20% should be used as the cutoff for pausing unproductive sub-swarms.

Blackwell [2007] adapted MPSO of Blackwell and Branke [2006] by self-adapting the number of swarms in the search space. This algorithm (referred to as MPSO2) is aimed at situations where the number of optima in the dynamic environment is unknown. Swarms are generated when the number of free swarms that have not converged to an optimum (M_{free}) have dropped to zero. Conversely, swarms are removed if M_{free} is higher than n_{excess}, a parameter of the algorithm. A swarm is classified as converged if all particles are located within a diameter of $2r_{conv}$. The values of r_{conv} and r_{excl} are calculated using $r_{excl} = r_{conv} = (V_{max,F} - V_{min,F})/(2n_k{}^{\frac{1}{n_d}})$, where n_k is the number of sub-swarms, $V_{max,F}$ and $V_{min,F}$ is the upper and lower search range of function F in the n_d dimensions (assuming equal ranges for all dimensions). MPSO2 can find an effective value for the number of sub-swarms by introducing and removing a sub-swarm when necessary. This algorithm thus uses the Multiple Populations, Refresh Information, and Custom Individuals strategies present in MPSO and incorporates a Change Control Parameters strategy.

Blackwell *et al.* [2008] adapted MPSO2 by converting all particles to quantum particles for one iteration after a change in the environment occured. Experimental results showed

that this adaptation requires fewer quantum particles during the period between changes in the environment. By evaluating fewer particles during each iteration, the number of iterations that can be performed between changes in the environment is increased.

Li *et al.* [2006] improved the speciation algorithm of Parrott and Li [2004] by introducing ideas from [Blackwell and Branke, 2004], namely quantum individuals to increase diversity, and anti-convergence to detect stagnation and subsequently reinitialise the worst-performing populations. A Custom Individuals strategy was thus added to the algorithm of Parrott and Li [2004]. This algorithm is called Speciation-based PSO (SPSO). The same adaptation that was made to MPSO2, namely converting all particles to quantum particles for one iteration after a change in the environment and reducing the number of quantum particles for the other iterations, was suggested for SPSO [Blackwell *et al.*, 2008].

Cellular PSO [Hashemi and Meybodi, 2009a] incorporates ideas from cellular automata into a PSO algorithm aimed at DOPs. The search space is divided into equally sized cells. Particles are allocated to cells based on their spatial positions within the search space. The particles within each cell perform a local best search based on the best performing particle within the cell and the cell's neighbourhood. A particle migrates from one cell to another when position updates place it outside the bounds of its original cell. A threshold is used to prevent too many migrating to a specific cell. When the number of particles within a cell exceeds the threshold, randomly selected particles from the overcrowded cell are allocated to randomly selected cells within the search space. Parameters to the algorithm include the number of cells, the neighbourhood size, and the maximum number of particles per cell threshold. The motivation for partitioning particles into cells is similar to that of forming sub-swarms and also provides the benefits of parallel search and tracking multiple optima. Cellular PSO is thus classified as using a Multiple Populations strategy and, through reinitialising particles, a Refresh Information strategy.

A Custom Individuals strategy was incorporated into Cellular PSO by Hashemi and Meybodi [2009b]. This adaptation changes a portion of each cell's particles to quantum particles for a specified number of iterations after a change in the environment. The fraction of particles that are changed to quantum particles, and the number of iterations that quantum particles must be maintained, are parameters to the algorithm.

Kamosi *et al.* [2010b] proposed an algorithm that utilises a single parent swarm to

locate promising areas in the search space and a dynamic number of child swarms to exploit the promising areas. This multi-swarm optimisation algorithm is referred to here as MSO. MSO commences with zero child swarms. The local best particle of the parent swarm is monitored and each time that it improves, a child swarm is created around it. All the particles from the parent swarm, that are located within a certain radius, r_{cs}, of the local best particle, migrate to the newly created child swarm. If the migrating particles exceed the maximum child swarm size, the surplus particles are reinitialised. Conversely, if too few particles migrate from the parent population, particles are randomly created using a uniform distribution within a radius of $r_{cs}/3$ from the local best particle of the child swarm. Randomly initialised particles are added to the parent swarm to replace particles lost in the migration process. Child swarms optimise their respective areas of the fitness landscape. Exclusion is used to prevent child swarms from converging to the same optimum.

A particle from the parent swarm that strays within a threshold distance, r_{cs}, of the local best particle of a child swarm is compared to the local best particle in terms of fitness. If the parent swarm particle has a higher fitness, a copy of it replaces the local best particle within the child swarm before the parent swarm particle is reinitialised. The algorithm reacts to changes in the environment by re-evaluating all parent swarm particles. Particles in child swarms act as quantum particles for one iteration after a change in the environment. MSO thus uses the Multiple Populations, Custom Individuals and Refresh Information strategies. Besides the normal PSO parameters, MSO has the following parameters: the number of particles in the parent swarm, the number of particles in each child swarm, the child swarm radius (r_{cs}), the exclusion radius, and the cloud radius.

Kamosi *et al.* [2010a] improved MSO by introducing a hibernation metaphor to place child swarms into a "sleep" state when they become unproductive. This approach (referred to here as hibernating MSO (HMSO)) is motivated by the fact that function evaluations are wasted on a child swarm which have located the summit of the optimum that it tracks. Function evaluations are freed up for other swarms by stopping the optimisation process for child swarms that have converged. A child swarm hibernates if the following two conditions are both satisfied: Firstly, the difference in fitness between the local best solution in the child swarm and the global best solution found in the fitness landscape

exceeds a specified threshold (i.e. the swarm has a comparatively low fitness). Secondly, the swarm radius is smaller than a threshold value (the swarm has converged). Swarms are awakened from hibernation when a change in the environment occurs.

Kamosi *et al.* [2010a] showed that HMSO performs better than MSO on the majority of the benchmark instances that were investigated. The HMSO algorithm yields lower errors than MSO because, by hibernating weaker sub-swarms that have converged, more function evaluations are made available to other swarms to discover and exploit new optima. A disadvantage of HMSO is that it introduces two new parameters: the convergence radius and threshold difference between the local and global best solution fitness values that must be exceeded.

Novoa-Hernández *et al.* [2011] removed the quantum particle component of the MPSO algorithm of Blackwell and Branke [2006]. The authors introduced a diversity increasing scheme directly after a change in the environment by replacing the worst-performing particles within each sub-swarm (half of the sub-swarm was replaced). The replacement particles are randomly created in a hypersphere centred at the best particle in the sub-swarm formed using a uniform distribution with a specified radius. Novoa-Hernández *et al.* [2011] also introduced a control rule which allows bad sub-swarms to "sleep" and consequently save function evaluations. This algorithm is referred to as MPSOD.

Sub-swarms are placed in a sleep state (i.e. temporarily removed from the optimisation process) if the following three conditions hold. Firstly, no recent changes occurred in the environment. Secondly, the sub-swarm must have converged to within a specified radius. Thirdly, the fitness of the sub-swarm must be classified as "bad". Membership to a fuzzy set of bad sub-swarms is determined using a function which assigns a "badness" value to each sub-swarm based on the average fitness of all the sub-swarms (fit_a) and the most fit sub-swarm (fit_b). Sub-swarms with a fitness below average is assigned a badness value of one. Otherwise the badness value is equal to the ratio $(fit_b - fit_k)/(fit_b - fit_a)$ where fit_k is the fitness of the sub-swarm that is being classified. Sub-swarms for which the badness value is smaller than a specified threshold are classified as bad. The sleeping swarms of MPSOD is similar to the hibernating swarms of HMSO and only differs in when the swarms are removed from the optimisation process. Placing sub-swarms in a sleep state allows more function evaluations to be allocated to other swarms, which leads to

more exploitation of optima that have been found and the discovery of new optima. This algorithm uses the same strategies as MPSO: Multiple Populations, Refresh Information, and Custom Individuals.

The clustering PSO (CPSO) algorithm of Yang and Li [2010] clusters the main swarm into independent sub-swarms after each change in the environment. A Multiple Populations strategy is thus used. The particles are allocated to sub-swarms based on their spatial proximity. The total number of particles and the maximum sub-swarm size are parameters to the algorithm. CPSO transfers information from one environment to the next by means of a memory archive containing good solutions found before the change (a History strategy). The best particle within a sub-swarm is added to the memory archive when the convergence of a sub-swarm is detected (all particles fall within a threshold radius). Converged sub-swarms are subsequently removed. Sub-swarms which search overlapping areas of the search space are detected by calculating the percentage of particles from one sub-swarm that fall within the search area of another. Two sub-swarms are merged if the percentage of overlap exceeds a threshold value, whereupon the weakest excess particles are removed to prevent the size of the merged sub-swarm from exceeding the maximum sub-swarm size.

Yang and Li [2010] also use a custom replacement strategy for the best particle within each sub-swarm. When a new particle with a better fitness than the best particle within the sub-swarm is found, components from each dimension of the new particle are iteratively incorporated into the old best particle and evaluated. A dimensional component is only permanently incorporated into the best particle if it results in an improved fitness. The motivation for this strategy is that it prevents promising information from some of the dimensions from being lost due to bad information in other dimensions. The approach is classified as a Selection strategy as particles are not automatically flagged as the best particle within the sub-swarm based on fitness.

CPSO responds to changes in the environment by removing all sub-swarms, reinitialising the main swarm and replacing the weakest particles in the newly created main swarm with the particles in the memory archive. The memory archive is subsequently cleared and the clustering process is repeated to form sub-swarms.

CPSO differs from most other algorithms in that the total number of particles dimin-

ishes after each change in the environment due to the removal of sub-swarms that have converged and the removal of excess particles when sub-swarms are merged. The algorithm prevents a situation where zero particles remain by introducing random particles into the search space when all particles have been removed.

The main ideas from CPSO where extended by Li and Yang [2012] to form CPSOR, an algorithm aimed specifically at situations where changes in the environment are difficult or impossible to detect. CPSOR uses the same techniques for clustering and removing redundant particles as CPSO, but in contrast to CPSO, changes in the environment are not explicitly detected or responded to by CPSOR. A threshold value, which is a parameter to the algorithm, is used as a minimum ratio of current population size over the original population size. Random particles are introduced when the ratio falls below the the threshold value, whereupon the clustering and particle removal processes are allowed to proceed as in CPSO. A strength of CPSOR is that it does not rely on detecting changes. CPSOR was shown to be more effective than several other algorithms in environments where changes occurred only locally in the fitness landscape (as opposed to changes that affect the entire fitness landscape and are consequently easier to detect). A weakness of CPSOR is that it introduces a new parameter which may be problem dependent.

3.3.3.3 Differential Evolution-Based Algorithms

The first application of the DE algorithm to DOPs occurred relatively recently. Consequently, researchers could apply earlier successful DOP approaches from GA and PSO-based algorithms when adapting DE for dynamic environments.

Section 3.3.2 described how Custom Individuals can be used to increase diversity. An approach similar to quantum particles, called Brownian individuals, is utilised by DynDE, a DE-based algorithm for dynamic environments [Mendes and Mohais, 2005]. Use of Brownian individuals involves the creation of individuals close to the best individual by adding a small random value, sampled from a zero-mean normal distribution, to each component of the best individual. Mendes and Mohais [2005] adapted the ideas from [Blackwell and Branke, 2004] to create their multi-population algorithm, DynDE, which uses exclusion to prevent populations from converging to the same peak. Exclusion reinitialises a sub-population when it strays too closely to another and can thus be seen as a

Refresh Information strategy. Furthermore, it was determined that the DE/best/2/bin best scheme is the most effective for use by DynDE (a Change Control Parameters strategy). The DE/best/2/bin scheme actually reduces diversity, since DE/best/2/bin encourages fast convergence. However, since multiple, independent populations are used which increase diversity, it was found to be beneficial for sub-populations to converge quickly to their respective optima. A Multiple Populations strategy is thus also used in DynDE. Mendes and Mohais [2005] showed that DynDE was at least as effective as its PSO-based counterparts. DynDE is discussed in detail in Section 3.4.1.

The favoured populations DE (FPDE) [du Plessis and Engelbrecht, 2008] is an adaptation to DynDE created in a preliminary study to the algorithms presented in this thesis. The adaptation aims to prevent wasting function evaluations on sub-populations with inferior performance. Normal DynDE is executed for ϱ_1 generations after a change in the environment. Thereafter, only the ϱ_3 sub-populations with the lowest error are executed for ϱ_2 generations while all other sub-populations are paused. Normal DynDE resumes after the ϱ_2 generations. The parameters ϱ_1, ϱ_2 and ϱ_3 must be manually tuned and it was found that FPDE yields only minor improvements over DynDE.

Brest *et al.* [2009] proposed a self-adaptive multi-population DE algorithm, called *jDE*, for optimising dynamic environments. This work focused on adapting the DE scale factor and crossover probability and is based on a previous algorithm [Brest *et al.*, 2006] for optimising static environments. The motivation behind using self-adaptive control parameters is that the algorithm can change the scale factor and crossover probability during the optimisation process to appropriate values as the environment changes. A further benefit is that the self-adaptive appoach results in fewer parameters to tune manually. This approach is classified as a Change Control Parameters strategy. *jDE* also contains components that are similar to other dynamic optimisation algorithms. For example, exclusion is used to prevent sub-populations from converging to the same optimum (a Refresh Information strategy). An aging metaphor is used to reinitialise sub-populations that have stagnated on a local optimum. Each individual's age is incremented every generation. Offspring inherit the age of parents, but this age may be reduced if the offspring performs significantly better than the parent. Sub-populations of which the best individual is too old are reinitialised so that the sub-population can discover other optima. The aging mechanism

allows the algorithm to discover new optima continuously and is classified as a Refresh Information strategy.

A further mechanism is used to prevent convergence within sub-populations: An individual is reinitialised if the Euclidean distance between the individual and the best individual in the population is too small, thus increasing diversity. The algorithm also utilises a form of memory called an archive. The best individual is added to the archive every time a change in the environment occurs. One of the sub-populations is always created by randomly selecting an individual from the archive and by adding small random numbers to each of the individual's components. The archive is a History strategy and information from the archive is incorporated through a Custom Individuals strategy. *jDE* is discussed in more detail in Section 3.4.2.

Recently, Noroozi *et al.* [2011] proposed a DE-based algorithm aimed at dynamic environments, referred to as Cellular DE. Cellular DE is a DE implementation of Cellular PSO which was discussed in the previous section. This algorithm divides the search space into equally sized cells which are used to delineate sub-populations (a Multiple Populations strategy). A limit is placed on the maximum number of individuals that are allowed to be located in each cell. Once this limit is exceeded, the worst performing individual within the overpopulated cell is reassigned to a randomly selected cell. The random reassignment of individuals is classified as a Refresh Information strategy. The algorithm employs a variant of the DE/current-to-best/bin scheme to create mutant vectors (a Change Control Parameter strategy). The best individual within the target vector's cell and surrounding cells is used as $\vec{x}_{best}$ in the scheme. Cellular DE track multiple optima in the dynamic environment by allowing individuals to converge to cells representing the areas where optima occur.

3.3.3.4 Other Dynamic Environment Algorithms

This section describes algorithms for dynamic environments that are either hybrids of GA, PSO and DE algorithms, or those that are not based on these algorithms. Usage of the seven strategies of Section 3.3.2 is indicated where possible.

An algorithm that uses both a DE population and a PSO swarm was proposed by Lung and Dumitrescu [2007]. This algorithm is called collaborative evolutionary-swarm

optimisation (CESO) and uses a Multiple Populations strategy. A swarm optimised by using PSO refines the best found optimum, while a DE variant, called crowding differential evolution, is used to maintain diversity and locate local optima. Crowding DE [Thomsen, 2004] is an algorithm aimed at locating multiple optima in static environments. Crowding DE changes the offspring operator of DE so that the most similar individual in the parent population, rather than the target vector, is replaced. This idea is similar to an approach used in niching, where the fitness of an individual is divided by a sharing function which is proportional to the number of other individuals within a threshold distance of the individual [Goldberg and Richardson, 1987][Sareni and Krahenbuhl, 1998], and is classified as a Selection strategy. Crowding DE maintains a highly diverse population in which each individual is ideally located on an optimum. Multiple optima are thus tracked simultaneously. The particles in the PSO swarm are replaced by the individuals in the DE population when a change in the environment occurs or when the distance between the best particle in the swarm and the best individual in the population drops below a threshold value.

CESO was improved by incorporating a second DE population [Lung and Dumitrescu, 2010]. The new algorithm is referred to as the evolutionary swarm cooperative algorithm (ESCA). The second DE population is used to maintain a search history and acts as a memory of previously found optima for the first DE population. ESCA thus also uses a History strategy.

Several investigations have focused on the self-adaptation of control parameters (a Change Control Parameters strategy). Angeline *et al.* [1996] made use of a self-adaptive evolutionary programming (EP) algorithm to evolve finite machines to predict the next value of an input string. The string to be predicted was changed dynamically during the execution of the algorithm. Evolution strategies (ES) have been applied to DOPs by several researchers [Arnold and Beyer, 2002], [Weicker and Weicker, 1999], [Bäck, 1998].

Moser and Hendtlass [2007] proposed the multi-phase multi-individual extremal optimisation (MMEO) algorithm. Extremal optimisation (EO) [Boettcher and Percus, 1999] makes use of a single individual which is mutated. EO consequently finds the optimum of the fitness landscape through hill climbing. EO was adapted by Moser and Hendtlass [2007] to contain several individuals, each of which used five steps to find optima. Firstly,

a stepwise sampling of the fitness landscape is performed to locate areas that potentially contain optima. From these potential points, an individual performs a local search to find a local optimum. During the local search phase, the individuals are checked to ensure that no duplicates (individuals that are optimising the same peak) exist. If changes in the environment occur, individuals are optimised further using a local search. Finally, individuals are fine tuned using finer-grained hill climbing. Multiple individuals were introduced by Moser and Hendtlass [2007] to locate and track multiple optima in the dynamic environment. Accordingly MMEO can be classified as using the Multiple Populations strategy.

Moser [2009] refined the MMEO algorithm by making use of the Hooke-Jeeves search algorithm [Hooke and Jeeves, 1961] to perform the local search. Further improvements to the MMEO local search was reported in [Moser and Chiong, 2010] by tuning the step sizes to more problem appropriate values. Currently, Moser and Chiong [2010] report the lowest offline error on the moving peaks benchmark in the literature.

3.3.4 Summary

The previous sections described how sub-components of each algorithm can be categorised into the seven strategies identified in Section 3.3.2. This information is summarised in Table 3.1. The table shows that most of the algorithms reviewed can be classified as using at least one of the seven strategies.

The majority of the algorithms employ several of the strategies to realise the three main approaches to solving DOPs: increasing diversity, memory, and parallel search. As the field matured over time, strategies like Multiple Populations and Refresh Information have been used in an increasing number of algorithms, although the specific implementation details still vary considerably. Conversely, the Sentinel strategy has not been widely adopted.

Despite the large number of available algorithms the problem of optimisation in dynamic environments has not been completely solved. New research which reports on improvements over previous algorithms is continuously published and the field remains an active area of research internationally. DE, in comparison to GA and PSO, has only been applied to DOPs relatively recently and there is still a comparatively small number of algorithms that uses DE as a base. The potential of improving the current algorithms thus exists and is the focus of this research.

Table 3.1: General strategies followed per algorithm

Algorithm	Change Control Parameters	Custom Individuals	Refresh Information	Sentinels	Selection	History	Multiple Populations
Angeline *et al.* [1996]	✓						
Blackwell and Bentley [2002]		✓					
Blackwell and Branke [2006] (MPSO)		✓	✓				✓
Blackwell [2007] (MPSO2)	✓	✓	✓				✓
Branke [2002], Branke and Schmeck [2003] (SOS)	✓						✓
Brest *et al.* [2009] (*jDE*)	✓	✓	✓			✓	✓
Carlisle and Dozier [2000]			✓				
Cobb [1990]	✓						
del Amo *et al.* [2010]		✓	✓			✓	✓
Grefenstette [1999]			✓				
Hashemi and Meybodi [2009b] (Cellular PSO)		✓	✓				✓
Hu and Eberhart [2002]			✓				
Kamosi *et al.* [2010b] (MSO)		✓	✓				✓
Kamosi *et al.* [2010a] (HMSO)		✓	✓				✓
Leonard *et al.* [2011]	✓		✓				
Li and Dam [2003], Janson and Middendorf [2004]	✓						
Li *et al.* [2006] (SPSO)		✓	✓				✓
Lung and Dumitrescu [2007] (CESO)					✓		✓
Lung and Dumitrescu [2010] (ESCA)					✓	✓	✓
Mendes and Mohais [2005] (DynDE)	✓	✓	✓				✓
Mori *et al.* [1996], Mori *et al.* [1997], Mori *et al.* [1998]					✓		
Morrison [2004]				✓			
Moser and Hendtlass [2007], Moser and Chiong [2010]							✓
Noroozi *et al.* [2011] (Cellular DE)	✓	✓	✓				
Novoa-Hernández *et al.* [2011]		✓	✓				✓
Oppacher and Wineberg [1999] (SBGA)					✓		✓
Parrott and Li [2004]			✓				✓
Ramsey and Grefenstette [1993]						✓	
Simões and Costa [2009b]						✓	
Ursem [2000] (MGA)					✓		✓
Vavak *et al.* [1997]	✓						
Yang [2005]						✓	
Yang and Li [2010]					✓	✓	✓
Zhu *et al.* [2011]						✓	

3.4 Differential Evolution for Dynamic Optimisation Problems

The previous section gave an outline of common strategies used by algorithms aimed as DOPs and gave an overview of specific algorithms for dynamic environments. The focus of this study is on DE and this section consequently gives a more detailed description of two DE-based algorithms for solving DOPs. The first is DynDE which is described in Section 3.4.1. DynDE is used as the base algorithm for extensions and improvements presented in the following chapters. The second algorithm is *jDE* which is a recent state-of-the-art algorithm based on DE. *jDE* is discussed in Section 3.4.2.

3.4.1 DynDE

DynDE is a differential evolution algorithm developed by Mendes and Mohais [2005] to solve dynamic optimisation problems. DynDE makes use of approaches similar to those applied by Blackwell and Branke [2006] to PSO in dynamic environments. The most successful versions of DynDE make use of multiple populations, exclusion and Brownian individuals to adapt DE for dynamic environments. These three components are described in Sections 3.4.1.1 to 3.4.1.3. Section 3.4.1.4 gives a discussion on the DE scheme used by DynDE, followed by a general discussion on the DynDE algorithm in Section 3.4.1.5.

3.4.1.1 Multiple Populations

Typically, a static problem space may contain several local optima. These optima typically move around in a dynamic environment and also change in height and shape. This implies that a local optimum may become a global optimum when a change in the environment occurs. Consequently, not only the movement of global optima in the problem space must be tracked, but also the local optima. An effective method of tracking all optima is to maintain several independent sub-populations of DE individuals, one sub-population on each optimum. In their most successful experiments Mendes and Mohais [2005] used 10 sub-populations, each containing six individuals.

3.4.1.2 Exclusion

In order to track all optima, it is necessary to ensure that all sub-populations converge to different optima. If all populations converged to the global optimum it would defeat the purpose of having multiple populations. Mendes and Mohais [2005] used exclusion to prevent sub-populations from converging to the same optimum. Exclusion compares the locations of the best individuals from each sub-population. If the spatial difference between any two of these individuals becomes too small, the entire sub-population of the inferior individual is randomly reinitialised. A threshold is used to determine if two individuals are too close. This threshold, or exclusion radius, is calculated as

$$r_{excl} = \frac{V_{max,F} - V_{min,F}}{2n_p^{\frac{1}{n_d}}} \tag{3.1}$$

where $V_{max,F}$ and $V_{min,F}$ are the upper and lower search range of function F in the n_d dimensions (assuming equal ranges for all dimensions), and n_p is the number of peaks. Equation (3.1) shows that the exclusion threshold increases with an increase in number of dimensions and decreases if the number of peaks is increased.

Let $\vec{x}_{best,k}$ be the best individual in sub-population P_k, $k \in \{1, 2, \ldots, n_k\}$. Exclusion is thus performed as described in Algorithm 7 (assuming a minimisation function, F).

Algorithm 7: DynDE Exclusion

```
for k1 = 1, ..., nk do
    for k2 = 1, ..., nk do
        if ||x_best,k1 − x_best,k2||2 < r_excl and k1 ≠ k2 then
            if F(x_best,k1) < F(x_best,k2) then
                Reinitialise population P_k2
            else
                Reinitialise population P_k1
            end
        end
    end
end
```

3.4.1.3 Brownian Individuals

In cases where a change in the environment results in the positional movement of some of the optima, it is unlikely that all of the sub-populations will still be clustered around the optimal point of their respective optima, even if the change is small. In order for the individuals in the sub-populations to track the moving optima more effectively, the diversity of each population should be increased. Mendes and Mohais [2005] successfully used Brownian individuals for this purpose. In every generation, a predefined number of the weakest individuals is flagged as Brownian. These individuals are then replaced by new individuals created by adding a small random number, sampled from a zero centred Gaussian distribution, to each component of the best individual in the sub-population. A Brownian individual, $\vec{x}_{brown}$, is thus created from the best individual, $\vec{x}_{best}$, using

$$\vec{x}_{brown} = \vec{x}_{best} + \vec{r} \tag{3.2}$$

where $\vec{r}$ is a random vector with $r_j \sim N(0, r_{brown})$ and r_{brown}, the Brownian radius, is the standard deviation of the Gaussian distribution. Mendes and Mohais [2005] showed that a suitable value to use for r_{brown} is 0.2.

The addition of the small random values to the components of the Brownian individuals ensures that the population never completely converges. This is especially important in a DE-based algorithm, since mutation step-sizes depend on the vector differences between individuals. Without the Brownian individuals, it is conceivable that the entire population could converge to a single point in the fitness landscape which would result in all vector differences being zero. The evolution process would cease and the algorithm would be unable to respond to changes in the environment.

3.4.1.4 DE Scheme

Mendes and Mohais [2005] experimentally investigated several DE schemes to determine which one is the best to use in DynDE. The schemes investigated were DE/rand/1/bin, DE/rand/2/bin, DE/best/1/bin, DE/best/2/bin, DE/rand-to-best/1/bin, DE/current-to-best/1/bin and DE/current-to-rand/1/bin. It was shown that the most effective scheme to use in conjunction with DynDE is DE/best/2/bin, where each temporary individual is

created using

$$\vec{v} = \vec{x}_{best} + \mathcal{F} \cdot (\vec{x}_1 + \vec{x}_2 - \vec{x}_3 - \vec{x}_4) \tag{3.3}$$

with $\vec{x}_1 \neq \vec{x}_2 \neq \vec{x}_3 \neq \vec{x}_4$ and $\vec{x}_{best}$ being the best individual in the sub-population.

3.4.1.5 DynDE Discussion

Section 3.3 identified three categories of approaches used by dynamic optimisation algorithms (increasing diversity, memory and parallel search). All three of these categories are present in the subcomponents of DynDE, i.e. Brownian individuals increases diversity and multiple populations are used to maintain a memory of the fitness landscape and to encourage parallel searching.

The performance of DynDE is thoroughly investigated in Chapter 4. However, Figure 3.3 depicts the offline error, current error and diversity of the DynDE algorithm on the MPB with Scenario 2 settings for 10 changes in the environment to illustrate how effective DynDE is on dynamic environments compared to normal DE. Averages over 30 runs were used.

A comparison between Figures 3.1 and 3.3 shows the considerable improvement of DynDE over DE. Firstly, where the diversity of the DE algorithm sharply declines to a point close to zero, the diversity of DynDE remains high. The average diversity per generation over all repeats was found to be 0.2661 for DynDE, compared to the value of 0.0017 for DE. The frequent perturbations on the diversity curve in Figure 3.3 shows how diversity is perpetually increased by Brownian individuals and sub-populations reinitialised by exclusion.

Secondly, the graphs clearly show considerably lower offline and current errors for DynDE than for DE. The offline error illustrated in Figure 3.1 increases after the first few changes in the environment, while DynDE's offline error consistently decreases (see Figure 3.3). The current error of DynDE frequently approaches zero between changes in the environment, while the current error of DE remains high, especially after the first change in the environment.

Despite the relative effectiveness of DynDE, a disadvantage of this algorithm is that several new problem dependent parameters are introduced. A total of four parameters is

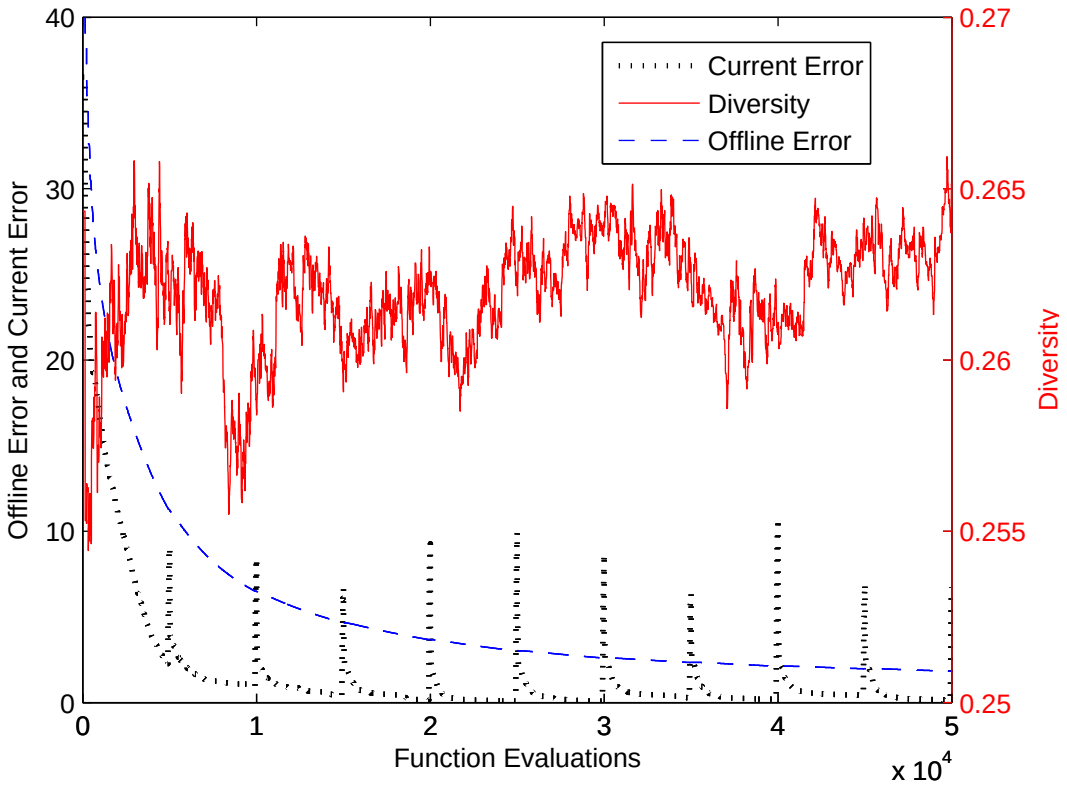

Figure 3.3: Diversity, Current error and Offline error of DynDE on the MPB

present in DynDE that are not present in DE:

1. the size of sub-populations,
2. the number of sub-populations,
3. the number of Brownian individuals per sub-population, and
4. the Brownian radius, r_{brown}, used to create Brownian individuals.

While Mendes and Mohais [2005] experimentally determined guidelines for these parameters, further fine tuning may be required for different problems. Furthermore, equation (3.1) requires knowledge about the number of optima in the environment. Even assuming that a constant number of optima is present, this information is generally not available in real-world problems.

3.4.2 jDE

The 2009 Congress on Evolutionary Computation (CEC2009) ran a dynamic optimisation competition. This competition used the generalised dynamic benchmark generator (GDBG) [Li *et al.*, 2008] to compare the results of competing algorithms. The winning algorithm was *jDE*, developed by Brest *et al.* [2009]. This section describes the five main algorithmic components that are utilised by *jDE* to solve dynamic optimisation problems. The components, self-adaptive control parameters, multiple populations, aging, localised exclusion, and custom reinitialisation are respectively discussed in Sections 3.4.2.1 to 3.4.2.5. A general discussion of the *jDE* algorithm is given in Section 3.4.2.6.

3.4.2.1 Self-Adaptive Control Parameters

The base algorithm of *jDE* is the self-adaptive DE algorithm of Brest *et al.* [2006] discussed in Section 2.4.4. Each individual, $\vec{x}_i$, stores its own value for the crossover factor, Cr_i, and scale factor, $\mathcal{F}_i$. Before the DE mutation and crossover steps, new scale and crossover factors ($\mathcal{F}_{new_i}$ and Cr_{new_i}) are calculated as follows:

$$\mathcal{F}_{new_i} = \begin{cases} \mathcal{F}_l + U(0,1) \cdot \mathcal{F}_u \text{ if } (U(0,1) < \tau_1) \\ \mathcal{F}_i \text{ otherwise} \end{cases} \tag{3.4}$$

$$Cr_{new_i} = \begin{cases} U(0,1) \text{ if } (U(0,1) < \tau_2) \\ Cr_i \text{ otherwise} \end{cases} \tag{3.5}$$

where τ_1 and τ_2 are the probabilities that the factors will be adjusted. Brest *et al.* [2006] used 0.1 for both τ_1 and τ_2. F_l and F_u are parameters to the algorithm. *jDE* uses $F_l = 0.36$ based on recommendations made by Zaharie [2002a], and $F_u = 0.9$. During initialisation, the initial values for the scale and crossover factors associated with each individual are $F_{initial} = 0.5$ and $Cr_{initial} = 0.9$.

The purpose of the self-adaptive component is to find values for the scale and crossover factors that are appropriate to the fitness landscape. Subsequently, the scale and crossover factors are constantly adapted to find values appropriate for each point in time as changes in the environment occur.

3.4.2.2 Multiple Populations

Multiple populations are used to increase diversity and facilitate independent parallel exploration of the fitness landscape. Each sub-population explores a different region of the fitness landscape and consequently serves as a memory of local and global optima. A local optimum may become a global optimum after a change in the environment, making it beneficial to track multiple optima. Brest *et al.* [2009] used 5 sub-populations containing 10 individuals each.

3.4.2.3 Aging

An aging metaphor is employed at the individual level to prevent stagnation of sub-populations. Individuals that are determined to be too old can be reinitialised in order to allow the influx of fresh genetic material. A sub-population tracking a local optimum is probabilistically reinitialised when the best individual in the sub-population becomes too old. This makes it possible for the sub-population to converge to another (potentially global) optimum. A particular sub-population may thus discover several optima between changes in the environment.

Let $\vec{x}_{i,k}$ be the i^{th} individual in sub-population P_k. Let $\vec{x}_{best,k}$ refer to the best individual in sub-population P_k, and $\vec{\hat{y}}$ be the global best individual over all populations. Each individual $\vec{x}_{i,k}$ stores its own age, denoted by $A_{i,k}$. Individuals or sub-populations are reinitialised based on their age as described in Algorithm 8.

Algorithm 8: Aging and Reinitialisation

if $\vec{x}_{i,k} \neq \vec{\hat{y}}$ **then**
  if $\vec{x}_{i,k} = \vec{x}_{best,k}$ **and** $A_{i,k} > \tau_3$ **and** $U(0,1) < \tau_4$ **then**
    Reinitialise sub-population P_k
  end
  else if $A_{i,k} > \tau_5$ **and** $U(0,1) < \tau_6$ **then**
    Reinitialise $\vec{x}_{i,k}$
  end
end

The parameter τ_3 is the age that the best individual in a sub-population must reach

before the sub-population is reinitialised with probability τ_4. τ_5 is the age a normal individual must reach before the individual is reinitialised with probability τ_6. Parameters used by Brest *et al.* [2009] are $\tau_3 = 30$, $\tau_4 = 0.1$, $\tau_5 = 25$ and $\tau_6 = 0.1$. There is thus a 10% chance that individuals that are too old, or populations of which the best individual is too old, are reinitialised. The age value of each individual is incremented after each generation.

Offspring partially inherit their age from parents. When a new offspring individual is created, it is assigned an age based on its fitness and Euclidean distance from the individual it is to replace. Offspring individuals that are close, either in fitness or in Euclidean distance, to the individual they are to replace, are likely to be assigned a high age. Diversity is increased in the population by this process, since individuals with a high age will likely survive for a shorter period (and contribute less genetic material) than individuals with a low age. The inheritance procedure is shown in Algorithm 9, where $\vec{u}_i$ is the offspring individual that is to replace $\vec{x}_i$.

Algorithm 9: Age Inheritance

if $\|\vec{x}_i - \vec{u}_i\|_2 < \tau_7$ **or** $F(\vec{x}_i) - F(\vec{u}_i) < \tau_8$ **then**
  $A_i = \min\{A_i, \tau_9\}$
else
  $A_i = \min\{A_i, \tau_{10}\}$
end

The thresholds, τ_7 and τ_8, in Algorithm 9 determine how spatially close or similar in fitness an offspring individual must be to receive a high age. τ_9 is the age that an individual is assigned when it is too close to its parent individual, and τ_{10} is the age it receives when it is not. Parameters used by Brest *et al.* [2009] are $\tau_7 = 0.01$, $\tau_8 = 0.1$, $\tau_9 = 20$ and $\tau_{10} = 5$.

3.4.2.4 Exclusion

Exclusion is used to prevent sub-populations from converging to the same optimum. This increases diversity and assists parallel optima discovery. Exclusion is performed in the same way as in DynDE (see Algorithm 7), with the one difference being that a constant

value $\tau_{11} = 0.05$ is used as the exclusion threshold.

A further exclusion measure is used within sub-populations to increase diversity, namely local exclusion. Local exclusion prevents the occurrence of spatial differences of zero between individuals within each sub-population. The DE mutation step-sizes depend on the distance between difference vectors. Convergence to a single point is thus undesirable as evolution would effectively end and the algorithm would not be able to respond to changes in the environment.

Individuals that are located below a small Euclidean threshold distance from the best individual in the sub-population are reinitialised. Local exclusion is summarised in Algorithm 10, where the number of individuals within sub-population k is $n_{I,k}$. τ_{12} is the local exclusion threshold and a value of $\tau_{12} = 0.0001$ was used by Brest *et al.* [2009].

Algorithm 10: Local Exclusion

for $k = 1, \ldots, n_k$ **do**
  for $i = 1, \ldots, n_{I,k}$ **do**
    if $\|\vec{x}_{best,k} - \vec{x}_{i,k}\|_2 < \tau_{12}$ **and** $\vec{x}_{best,k} \neq \vec{x}_{i,k}$ **then**
      Reinitialise $\vec{x}_{i,k}$
    end
  end
end

3.4.2.5 Custom Reinitialisation

jDE employs an archive population, P_{ar}, of good solutions as a form of memory (in addition to using multiple populations). This archive starts out empty and grows by one individual every time a change in the environment is detected, at which time the best individual found before the change is added to the archive. The archive is utilised when individuals are reinitialised as described in Algorithm 11.

The archive is analogous to the *hall of fame* used in co-evolution. Through the archive, the current population can effectively draw on the experience of previous generations by searching in areas of the fitness landscape where optima were previously found. This is particularly useful in situations where the change type of the environment is cyclic which

results in the periodic return of the environment to the same state. The archive also provides useful information when the changes in the environment are small, which makes the locations of previous optima relevant for several generations.

An approach similar to the creation of Brownian individuals is used to perturb individuals drawn from the archive. Algorithm 11 makes use of random vectors $\vec{r}_1$ (with $r_{1,j} \sim N(0,1)$) and $\vec{r}_2$ (with $r_{2,j} \sim U(-0.5, 0.5)$).

Algorithm 11: Custom Reinitialisation of $\vec{x}_{i,k}(t)$

if $U(0,1) < \tau_{13}$ **and** $k = 1$ **and** $|P_{ar}| > 0$ **then**
  $a = U(1, |P_{ar}|)$;
  $\vec{z} = P_{ar}[a]$;
  $\varsigma = \tau_{14}/i$;
  if $(i \bmod 2) = 1$ **then**
    $\vec{x}_{i,k}(t+1) = \vec{z} + \varsigma \cdot \vec{r}_1$
  else
    $\vec{x}_{i,k}(t+1) = \vec{z} + \varsigma \cdot \vec{r}_2$
  end
else
  Generate $\vec{x}_{i,k}(t+1)$ randomly from a uniform distribution
end

Note that only the first sub-population makes use of the archive, to avoid genetic material from the archive dominating the entire population. τ_{13} is the probability that an individual will be generated using information from the archive. τ_{14} controls the magnitude of random perturbations. Parameters used by Brest *et al.* [2009] are $\tau_{13} = 0.5$ and $\tau_{14} = 0.1$.

3.4.2.6 *jDE* Discussion

jDE contains several novel components, for example, aging and local exclusion, that have never been used in previous algorithms. A significant disadvantage of *jDE*, however, is the number of control parameters used in the algorithm. In addition to parameters such as the number of sub-populations and sub-population size which are found in other

algorithms, *jDE* introduces parameters F_l and F_u to control the adaptation of the scale factor. Fourteen other general parameters, labelled τ_1 through τ_{14} are also introduced. Although Brest *et al.* [2009] provide guideline values for these parameters, and although the algorithm may not be equally sensitive to all parameters, the likelihood that parameters would have to be fine-tuned for specific problems is high. Appendix A shows that *jDE* is very sensitive to at least two of the parameters, τ_3 and τ_5. Given the large number of parameters, a manual fine-tuning process could be extremely time consuming.

3.5 Detecting Changes in the Environment

Most evolutionary dynamic optimisation algorithms respond to changes in the environment. When the period between changes is unknown, it is necessary for the algorithm to detect changes. Change detection is not trivial, since changes could be localised in small areas of the fitness landscape, or could involve the introduction of an optimum into an area of the search space where no individuals are located. It is thus necessary to have an appropriate strategy to detect changes. Standard change detection approaches include the periodic re-evaluation of the best individual in the population [Hu and Eberhart, 2002], stationary sentinels [Morrison, 2004], or random points in the problem space [Carlisle and Dozier, 2002]. Discrepancies between current and previous fitness values indicate a change in the environment [Nguyen *et al.*, 2012]. Richter [2009] did a thorough study on change detection in environments where it is difficult to detect changes.

The effective detection of changes is a complex problem worthy of a study on its own. The change detection approach used by an algorithm aimed at DOPs impacts the results of the algorithm, because if changes are not correctly detected, the algorithm cannot respond appropriately to changes in the environment. This dependence on the detection strategy could bias results.

The problem of change detection was consequently delineated as outside the scope of the current study. The algorithms that are evaluated in the following chapter use an automatic detection strategy whereby the algorithm can test for changes in the environment without using any function evaluations. The change period is not given as a parameter to the algorithm, but the algorithms are allowed to detect changes flawlessly by interrogat-

ing the benchmark function once every generation. One of the normal change detection strategies can thus be integrated into the algorithms when needed. Throughout this study, where comparisons are drawn with the published results of another algorithm in the literature, the change detection approach used by the other algorithm is used to obtain results for the comparison.

3.6 Conclusions

The purpose of this chapter was to introduce problems experienced by static optimisation algorithms applied to dynamic environments and to discuss progress made in the EA and SI research community to address these problems.

Section 3.2 motivated why diversity is a major cause of poor performance of DE on DOPs. It was shown that DE suffers from a lack of diversity in dynamic environments. Outdated information was highlighted as a secondary problem of DE in dynamic environments.

Several algorithms that were developed to solve DOPs were discussed in Section 3.3. Three broad categories for dynamic optimisation approaches were identified, namely: increasing diversity, employing memory, and parallel searching. Seven general strategies commonly used to achieve the three approaches where highlighted.

Two state-of-the-art differential evolution algorithms, namely DynDE and *jDE*, were discussed in detail in Section 3.4. DynDE is used in the next chapter as base algorithm for the approaches proposed in this thesis. *jDE* is used for the purpose of comparing the new approaches to a state-of-the-art DE-based algorithm. Both algorithms contain appropriate components which make them more suited to dynamic environments than DE, but it was also found that several new parameters were introduced. These parameters may require fine-tuning when DynDE and *jDE* are applied to real-world problems. The performance of DynDE and *jDE* on benchmark functions is investigated in the next chapter.

Section 3.5 contained a discussion on the automatic detection dynamic environment changes. The detection of changes are outside the scope of this study and an automatic detection strategy is used.

The next chapter discusses new adaptations to DynDE which are proposed by the

author. These adaptations are aimed at improved performance in dynamic environments. A comparative evaluation of the new algorithm is performed to determine whether performance is improved.