Chapter 1
Self-adaptive Differential Evolution for dynamic environments with fluctuating numbers of optima

M.C. du Plessis and A.P. Engelbrecht

Department of Computer Science, University of Pretoria, Pretoria, South Africa

1.1 Introduction

Despite the fact that evolutionary algorithms often solve static problems successfully, dynamic optimization problems tend to pose a challenge to evolutionary algorithms [21]. Differential evolution (DE) is one of the evolutionary algorithms that does not scale well to dynamic environments due to lack of diversity [35]. A significant body of work exists on algorithms for optimizing dynamic problems (see Section 1.3). Recently, several algorithms based on DE have been proposed [19][26][28][27][10].

Benchmarks used to evaluate algorithms aimed at dynamic optimization (like the moving peaks benchmark [5] and the generalized benchmark generator [17] [16]), typically focus on problems where a constant number of optima moves around a multi-dimensional search space. While some of these optima may be obscured by others, these benchmarks do not simulate problems where new optima are introduced, or current optima are removed from the search space.

Dynamic Population DE (DynPopDE) [27] is a DE-based algorithm aimed at dynamic optimization problems where the number of optima fluctuates over time. This chapter describes the subcomponents of DynPopDE and then investigates the effect of hybridizing DynPopDE with the self-adaptive component of *jDE* [10] to form a new algorithm, Self-Adaptive DynPopDE (SADynPopDE).

The following sections describe dynamic environments and the benchmark function used in this study. Related work by other researchers is presented in Section 1.3. Differential evolution is described in Section 1.4. The components of DynPopDE, the base algorithm used in this study, are described and motivated in Section 1.5. The incorporation of self-adaptive control parameters into DynPopDE to form SA-

DynPopDE and the experimental comparison of these algorithms are described in Section 1.6. The main conclusions of this study are summarized in Section 1.7.

1.2 Dynamic Environments

It is not uncommon for the solution to real-world optimization problems to vary over time. In order to evaluate and compare algorithms aimed at dynamic optimization problems, researchers created benchmark functions. The benchmark used in this study is a variation of the moving peaks benchmark.

1.2.1 Moving Peaks Benchmark

Branke [5] created the moving peaks benchmark (MPB) to address the need for a single, adaptable benchmark that can be used to compare the performance of algorithms aimed at dynamic optimization problems. The multi-dimensional problem space of the moving peaks function contains several peaks, or optima, of variable height, width, and shape. These peaks move around with height and width changing periodically. The MPB allows the following parameters to be set:

- Number of peaks
- Number of dimensions
- Maximum and minimum peak width and height
- Change period (the number of function evaluations between successive changes in the environment)
- Change severity (how much the locations of peaks are moved within the search space)
- Height and width severity (standard deviations of changes made to the height and width of each peak)
- Peak function
- Correlation (between successive movements of a peak)

Three scenarios of settings of MPB parameters are suggested by Branke [5], however the majority of researchers using the MPB employ only variations of Scenario 2 settings. The Scenario 2 settings are listed in Table 1.1.

The performance measure suggested by Branke and Schmeck [7] is the *offline error*. The offline error is the running average of the lowest-so-far error found since the last change in the environment:

$$Offline\ Error = \frac{\sum_{t=1}^{T} E(\mathbf{x}_{best}(t), t)}{T} \tag{1.1}$$

Table 1.1 MPB Scenario 2 settings

Setting	Value
Number of Dimensions	5
Number of Peaks	10
Max and Min Peak height	[30,70]
Max and Min Peak width	[1.0,12.0]
Change period	5000
Change severity	1.0
Height severity	7.0
Width severity	1.0
Peak function	Cone
Correlation	[0.0,1.0]

where T is the total number of function evaluations and $E(\mathbf{x}_{best}(t),t)$ is the error of the best individual found since the last change in the environment (referred to as the *current error*).

1.2.2 Extensions to the Moving Peaks Benchmark

This study investigates dynamic environments in which the number of peaks fluctuates over time. The MPB was therefore adapted by the current authors to allow the number of peaks to change when a change occurs in the environment. For the adapted MPB, the number of peaks, $m(t)$, is calculated as:

$$m(t) = \begin{cases} \max\{1, m(t-1) - M \cdot U(0,1) \cdot \mathscr{C}\} \\ \qquad \text{if } U(0,1) < 0.5 \\ \min\{M, m(t-1) + M \cdot U(0,1) \cdot \mathscr{C}\} \\ \qquad \text{otherwise} \end{cases} \tag{1.2}$$

where M is the maximum number of peaks and $\mathscr{C}$ is the maximum fraction of M that can be added or removed from the population after a change in the environment. $\mathscr{C}$ thus controls the severity of the change in number of peaks. For example, $\mathscr{C} = 1$ will result in up to M peaks being added or removed, while $\mathscr{C} = 0.1$ will result in a change of up to 10% of M. M and $\mathscr{C}$ are included as parameters to the benchmark function.

1.3 Related Work

A considerable body of work exists on optimization for dynamic environments. Algorithms directly related to this study are briefly described in this section.

A change in the environment may cause a local optimum to become the global optimum. Information regarding the position of local optima can thus be useful to find the global optimum after a change in the environment. This information is generally gathered 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) [6], [4], [8], shifting balance GA (SBGA) [24] and multinational GA (MGA) [34]. All three of these approaches make use of different techniques to distribute individuals among the populations intelligently.

Parrott and Li [25] suggested a multiple swarm PSO approach to optimizing dynamic problems, 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. When using speciation, the social component of PSO provides an simple method to divide the swarm into sub-swarms. 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 is below a certain threshold value. The global best value of each particle within a sub-swarm is set to the personal best value of the best particle. The sizes of sub-swarms are dynamic. Particles can migrate to another population 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. As a mechanism to prevent sub-swarms 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 reinitialized and allocated to an appropriate sub-swarm.

Blackwell and Branke [3] introduced a multiple population PSO based algorithm that contains three components: exclusion, anti-convergence and quantum particles. In contrast to earlier algorithms, in their approach, 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 [2] is a technique meant to prevent sub-swarms from clustering around the same optimum by means of reinitializing sub-swarms that stray within a threshold Euclidean distance from better performing sub-swarms. This threshold distance is called the exclusion radius. Anti-convergence is meant to prevent stagnation of the particles in the search space. Consequently, if it is found that all sub-swarms have converged to their respective optima, the weakest sub-swarm is randomly reinitialized. Convergence of a sub-swarm is detected if all particles of the sub-swarm fall within a threshold Euclidean distance of each other. This threshold is called the convergence radius. *Quantum particles* is a means of increasing diversity by reinitializing a portion of the swarm of particles within a hyper-sphere, centered around the best particle in the swarm.

Blackwell [1] further adapted the PSO-based algorithm of Blackwell and Branke [3] by self-adapting the number of swarms in the search space. This algorithm 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 con-

verged to a optimum (M_{free}) have dropped to zero. Conversely, swarms are removed if M_{free} is higher than n_{excess}, a parameter of the algorithm. The algorithms of Blackwell and Branke [3] and Blackwell [1] have the disadvantage that the severity of changes in the environment is given as a parameter to the optimization algorithms. 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 optimization problems.

Li *et al.* [18] improved the speciation algorithm of Parrott and Li [25], by introducing ideas from Blackwell and Branke [2], namely quantum particles to increase diversity, and anti-convergence to detect stagnation and subsequently reinitialize the worst performing populations. This algorithm is called Speciation-based PSO (SPSO).

An approach similar to quantum individuals, called Brownian individuals, is utilized by DynDE, a DE based algorithm for dynamic environments [19]. Use of Brownian individuals involves the creation of individuals close to the best individual by adding a small random value, sampled from a normal distribution, to each component of the best individual. Mendes and Mohais [19] adapted the ideas from Blackwell and Branke [3] to create their multi-population algorithm, DynDE, which uses exclusion to prevent populations from converging to the same peak. Mendes and Mohais [19] showed that DynDE was at least as effective as its PSO based counterparts. DynDE will be discussed in detail in Section 1.5.

Recently, Brest *et al.* [10] proposed a self-adaptive multi-population DE algorithm for optimizing dynamic environments, called *jDE*. This work focused on adapting the DE scale factor and crossover probability and is based on a previous algorithm for static optimization [9]. *jDE* also contains several components that are similar to other dynamic optimization algorithms. A technique similar to exclusion is used to prevent sub-populations from converging to the same optimum. An aging metaphor is used to reinitialize 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 reinitialized. Within sub-populations a further mechanism is used to prevent convergence. An individual is reinitialized if the Euclidean distance between the individual and the best individual in the population is too small. The algorithm also utilizes 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 adding small random numbers to each of the individual's components.

For a survey of algorithms for dynamic optimization see [15].

1.4 Differential Evolution

The purpose of this section is to describe the Differential Evolution (DE) algorithm and to discuss variations of the basic algorithm. The original DE algorithm is discussed in Section 1.4.1, followed by a discussion on DE schemes in Section 1.4.2 and DE control parameters in Section 1.4.3. Section 1.4.4 describes the performance of DE in dynamic environments and Section 1.4.5 discusses methods of detecting changes in dynamic environments.

1.4.1 Basic Differential Evolution

Differential Evolution (DE) is an optimization algorithm based on Darwinian evolution [12], created by Storn and Price [30], [33]. Several variants to the DE algorithm have been suggested, but the original algorithm is given in Algorithm 1.

Algorithm 1: Basic Differential Evolution

Generate a population, P, of I individuals by creating vectors of random candidate solutions, $\mathbf{x}_i$, $i = 1, \cdots, I$ and $|\mathbf{x}_i| = J$;
Evaluate the fitness, $F(\mathbf{x}_i)$, of all individuals.;
while *termination criteria not met* **do**
  foreach $i \in \{1, \ldots, I\}$; `// Create` I `individuals for a trial population` **do**
    Select three individuals, $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$, at random from the current population such that $\mathbf{x}_1 \neq \mathbf{x}_2 \neq \mathbf{x}_3$;
    Create a new trial vector $\mathbf{v}_i$ using:

$$\mathbf{v}_i = \mathbf{x}_1 + \mathscr{F} \cdot (\mathbf{x}_2 - \mathbf{x}_3) \tag{1.3}$$

    where $\mathscr{F} \in (0, \infty)$ is known as the scale factor and $\mathbf{x}_1$ is referred to as the base vector;
    Add $\mathbf{v}_i$ to the trial population;
  end
  foreach $\mathbf{x}_i$ *in the current population (referred to as the target vector), select the corresponding trial vector* $\mathbf{v}_i$ *from the trial population;* `// Perform crossover` **do**
    Create offspring $\mathbf{u}_i$ as follows:

$$u_{i,j} = \begin{cases} v_{i,j} \text{ if } (U(0,1) \leq Cr \text{ or } j == j_{rand}) \\ x_{i,j} \text{ otherwise} \end{cases} \tag{1.4}$$

    where $Cr \in (0,1)$ is the crossover probability and j_{rand} is a randomly selected index, i.e. $j_{rand} \sim U(1, J+1)$;
    Evaluate the fitness of $\mathbf{u}_i$;
    If $\mathbf{u}_i$ has a better fitness value than $\mathbf{x}_i$ then replace $\mathbf{x}_i$ with $\mathbf{u}_i$;
  end
end

1.4.2 Differential Evolution Schemes

Most variations of DE (called schemes) are based on different approaches to creating each of the temporary individuals, $\mathbf{v}_i$ (see equation (1.3)), and different approaches to the method of creating offspring [31]. One of two crossover schemes are typically used to create offspring. The first, binary crossover, is used in equation (1.4). The second common approach is called exponential crossover.

By convention, schemes are labelled in the form DE/*a*/*b*/*c*, where *a* is the method used to select the base vector, *b* is the number of difference vectors and *c* is the method used to create offspring. The scheme used in Algorithm 1 is referred to as DE/rand/1/bin.

Several methods of selecting the base vector have been developed and can be used with either of the crossover methods. Popular base vector selection methods include [31][32][20] (in each case the selected vectors are assumed to be unique):

DE/rand/2: Two pairs of difference vectors are used:

$$\mathbf{v}_i = \mathbf{x}_1 + \mathscr{F} \cdot (\mathbf{x}_2 + \mathbf{x}_3 - \mathbf{x}_4 - \mathbf{x}_5) \tag{1.5}$$

DE/best/1: The best individual in the population is selected as the base vector:

$$\mathbf{v}_i = \mathbf{x}_{best} + \mathscr{F} \cdot (\mathbf{x}_1 - \mathbf{x}_2) \tag{1.6}$$

1.4.3 Differential Evolution Control Parameters

The Differential Evolution algorithm has several control parameters that can be set. Ignoring extra parameters introduced by some DE schemes, the main DE control parameters are population size, scale factor ($\mathscr{F}$) and crossover factor (Cr).

The scale factor ($\mathscr{F}$) controls the magnitude of the difference vector and consequently the amount by which the base vector is perturbed. Large values of $\mathscr{F}$ encourage large scale exploration of the search space but could lead to premature convergence, while small values result in a more detailed exploration of the local search space while increasing convergence time.

The crossover factor (Cr) controls the diversity of the population, since a large value of Cr will result in a higher probability that new genetic material will be incorporated into the population. Large values of Cr result in fast convergence while smaller values improve robustness [13].

General guidelines for the values of parameters that work reasonably well on a wide range of problems are known, however, best results in terms of accuracy, robustness and speed are found if the parameters are tuned for each problem individually [13].

1.4.4 Differential Evolution in Dynamic Environments

Like most evolutionary algorithms, the initial population of DE is formed from individuals that are randomly dispersed in the search space. This ensures that the algorithm explores a large area of the search space during the first generations. Eventually, however, the algorithm converges to an optimum, with all the individuals clustered around a single point.

The amount by which the locations of individuals in the population differ is referred to as diversity. If individuals are uniformly distributed in the search space then their diversity is high. Alternatively, when individuals are clustered around a single point, the diversity is low. The diversity measure used in this work calculates the diversity, D, of a population of size I in J dimensions as

$$D = \sum_{i=1}^{I} \|\mathbf{d} - \mathbf{x}_i\|_2 \tag{1.7}$$

where $\mathbf{d}$ is the average location of all individuals, calculated as

$$d_j = \frac{\sum_{i=1}^{I} x_{i,j}}{I}, \forall\ j \in J \tag{1.8}$$

In static environments this loss of diversity is not a bona fide problem and is in most cases desirable, since having all individuals clustered around the optimum assists with the fine-grained optimization at the end of the search process.

In dynamic environments, however, loss of diversity is a major cause of evolutionary algorithms being ineffective. When changes occur in the environment, the population lacks the diversity necessary to locate the position of the new global optimum [35]. Consider Figure 1.1 which depicts the offline error, current error and diversity averaged over 30 runs on the MPB of the basic DE algorithm described in Section 1.4. Ten changes in the environment are illustrated. Changes occur once every 5 000 function evaluations.

During the period between the commencement of the algorithm and the first change in the environment, the current error and the offline error drops sharply due to the DE algorithm converging to one or more optima in the environment. The fact that individuals are clustered increasingly closely together is proven 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. The diversity continues to drop until it reaches a value close to zero shortly after the second change in the environment. This means that all individuals are clustered around a single optimum and that the rest of the search space is left completely unexplored. In DE mutations are performed based on the spacial differences between individuals. If individuals are spatially close then very small mutations are applied which hampers further exploration of the search space. Over 30 runs the average diversity per generation was found to be 0.422. This

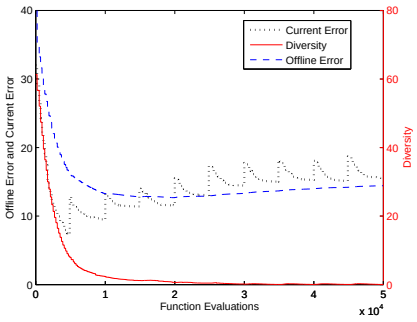

Fig. 1.1 Diversity, Current error and Offline error of DE with re-evaluation after changes

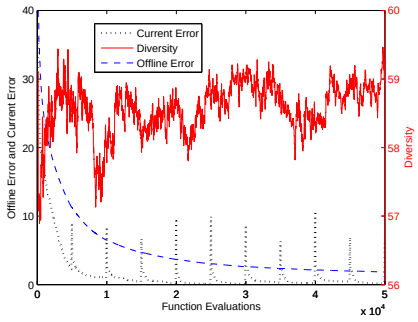

Fig. 1.2 Diversity, Current error and Offline error of DynDE on the MPB

value is extremely low and explains why normal DE is not effective in dynamic environments.

1.4.5 Detecting Changes in the Environment

Most evolutionary dynamic optimization algorithms respond to changes in the environment. When the period between changes is known beforehand, it is necessary for the algorithm to detect changes automatically. Detecting changes is not trivial, since changes could be localized in small areas of the search space, 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 approaches include periodically re-evaluating the best particle in the swarm [14], stationary sentinels [21], or random points in the problem space [11]. Discrepancies between current and previous fitness values indicate a change in the environment.

The MPB requires the optimization algorithm to automatically detect changes in the environment. It was experimentally determined by the authors that a simple change detection strategy, involving only the re-evaluation of a single sentinel individual once during every generation, is sufficient to detect all changes when using the MPB. This approach was followed throughout this study, but is not expected to be sufficient for all dynamic optimization problems. However, a more robust approach, for example re-evaluating the best individual in each sub-population, can be introduced relatively easily should it be required.

1.5 Dynamic Population Differential Evolution

Dynamic Population Differential Evolution (DynPopDE) [27], an extension of DynDE and CDE [28], is aimed at dynamic optimization problems where the number of optima in the search space is unknown or fluctuates over time.

DynDE is a differential evolution algorithm developed by Mendes and Mohais [19] to solve dynamic optimization problems. Sections 1.5.1 to 1.5.5 provides an overview of the components inherited by DynPopDE from DynDE. The two algorithmic components that make up CDE is described and motivated in Sections 1.5.6 and 1.5.7. DynPopDE is an extension of CDE consisting of three components, namely: spawning new populations (Section 1.5.8), removing populations (Section 1.5.9), and the introduction of a penalty factor on the performance value used to determine which sub-population is to evolve at a given time (Section 1.5.10).

1.5.1 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 an entirely different optimum may become the global optimum once a change in the environment occurs. Consequently, not only the movement of the global optimum 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 [19] used 10 sub-populations, each containing 6 individuals.

1.5.2 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 [19] 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 reinitialized. A threshold is used to determine if two individuals are too close. The threshold, or exclusion radius, is calculated as

$$r_{excl} = \frac{X}{2p^{\frac{1}{d}}} \tag{1.9}$$

where X is the range of the d dimensions (assuming equal ranges for all dimensions), and p is the number of peaks. Equation (1.9) shows that the exclusion threshold increases with an increase in number of dimensions and decreases if the number of peaks is increased. Because knowledge of the number of peaks is generally not available, it was proposed that the exclusion threshold be calculated using the number of sub-populations as follows [27]:

$$r_{excl} = \frac{X}{2K^{\frac{1}{d}}} \tag{1.10}$$

where K is the number of populations, thus making the threshold dependent on the number of available populations. The same equation was used by Blackwell [1] in the self-adapting multi-swarms algorithm.

1.5.3 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 [19] successfully used Brownian individuals for this purpose. In every generation a predefined number of the weakest individuals are flagged as Brownian. These individuals are then replaced by new individuals created by adding a small random number, sampled from a zero centered Gaussian distribution, to each component of the best individual in the sub-population. A Brownian individual, $\mathbf{x}_{brown}$, is thus created from the best individual $\mathbf{x}_{best}$ using

$$\mathbf{x}_{brown} = \mathbf{x}_{best} + \mathcal{N}(0, v) \tag{1.11}$$

where v is the standard deviation of the Gaussian distribution. Mendes and Mohais [19] showed that a suitable value of v to use is 0.2.

1.5.4 DE Scheme

Mendes and Mohais [19] 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

$$\mathbf{v} = \mathbf{x}_{best} + \mathscr{F} \cdot (\mathbf{x}_1 + \mathbf{x}_2 - \mathbf{x}_3 - \mathbf{x}_4) \tag{1.12}$$

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

1.5.5 DynDE Discussion

The performance of DynDE was thoroughly investigated by Du Plessis and Engelbrecht [28]. However, to illustrate how effective DynDE is on dynamic environments compared to normal DE, Figure 1.2 depicts the offline error, current error and diversity of DynDE algorithm on the MPB for 10 changes in the environment. Averages over 30 runs are used.

A comparison between Figures 1.1 and 1.2 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 59.496 for DynDE, compared to the value of 0.422 for DE. The frequent perturbations on the diversity curve in Figure 1.2 shows how diversity is perpetually increased by Brownian individuals and sub-populations reinitialized by exclusion.

Secondly, the graphs clearly show considerably lower offline and current errors for DynDE than for DE. In Figure 1.1, the offline error increases after the first few changes in the environment, while DynDE's offline error consistently decreases (see Figure 1.2). 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.

1.5.6 Competitive Population Evaluation

For most static optimization problems, the effectiveness of an algorithm is measured by the error of the best solution found at the end of the optimization process. In contrast, optimization in dynamic environments implies that a solution is likely to be required at all times (or at least just before changes in the environment occur), not just at the termination of the algorithm. In these situations, it is imperative to find the lowest error value as soon as possible after changes in the environment have occurred.

A dynamic optimization algorithm can thus be improved, not only by reducing the error, but also by making the algorithm reach its lowest error value (before a change occurs in the environment) in fewer function evaluations.

The above argument is the motivation for the component of DynPopDE named Competitive Population Evaluation (CPE). The primary goal of CPE is not to decrease the error value found by DynDE, but rather to make the algorithm reach the lowest error value in fewer function evaluations. It is proposed that this can be

achieved by initially allocating all function evaluations after a change in the environment to the sub-population that is optimizing the global optimum; thereafter function evaluations are allocated to other sub-populations to locate local optima.

The mechanism used by CPE to allocate function evaluations is based on the performance of sub-populations. The best-performing sub-population is evolved on its own until its performance drops below that of another sub-population. The new best performing sub-population is then evolved on its own until its performance drops below that of another sub-population. This process is repeated until a change occurs in the environment. CPE allows the location of the global optimum to be discovered early, while the sub-optimum peaks are located later. CPE thus differs from DynDE in that peaks are not located in parallel, but sequentially. The CPE process is detailed in Algorithm 2.

Algorithm 2: Competitive Population Evaluation

while *termination criteria not met* **do**
  Allow the standard DynDE algorithm to run for two generations;
  repeat
    for $k = 1,\ldots,K$; `// all K sub-populations`
    do
      Calculate the performance value, $\mathscr{P}(k,t)$
    end
    Select a such that $\mathscr{P}(a,t) = \min_{k=1,\ldots,K}\{\mathscr{P}(k,t)\}$;
    Evolve only sub-population a using DE;
    $t = t+1$;
    Calculate $\mathscr{P}(a,t)$;
    Perform Exclusion;
  until *Change occurs in the environment*;
end

The performance value, $\mathscr{P}$, of a sub-population depends on two factors: The current fitness of the best individual in the sub-population and the error reduction of the best individual during the last evaluation of the sub-population. Let K be the number of sub-populations, $\mathbf{x}_{best,k}$ the best individual in sub-population k, and $F(\mathbf{x}_{best,k},t)$ the fitness of the best individual in sub-population k during iteration t. The performance $\mathscr{P}(k,t)$ of population k after iteration t is given by:

$$\mathscr{P}(k,t) = (\Delta F(\mathbf{x}_{best,k},t)+1)(R_k(t)+1) \tag{1.13}$$

where

$$\Delta F(\mathbf{x}_{best,k},t) = |F(\mathbf{x}_{best,k},t) - F(\mathbf{x}_{best,k},t-1)| \tag{1.14}$$

For function maximization problems, $R_k(t)$ is calculated as:

$$R_k(t) = |F(\mathbf{x}_{best,k},t) - \min_{q=1,\ldots,K}\{F(\mathbf{x}_{best,q},t)\}| \tag{1.15}$$

and for function minimization problems,

$$R_k(t) = |F(\mathbf{x}_{best,k},t) - \max_{q=1,\ldots,K}\{F(\mathbf{x}_{best,q},t)\}| \tag{1.16}$$

Therefore, the best performing sub-population is the sub-population with the highest product of fitness and improvement. The motivation for the addition of 1 to the first and second terms in equation (1.13) is to prevent a population being assigned a performance value of zero. Without the addition in the second term, the least fit population will always be assigned a performance value of zero (since the product of the first and second term will be zero) and will never be considered for searching for an optimum. Similarly, without the addition in the first term, a good performing population that does not show any improvement during a specific iteration, is assigned a performance value of zero and will never be considered for evaluation again. The addition of 1 to the first and second terms is thus included to ensure that every sub-population could potentially have the highest performance value and subsequently be given function evaluations.

The absolute values of $\Delta F(\mathbf{x}_{best,k},t)$ and $R_k(t)$ are taken to ensure that the performance values are always positive. When a population is reinitialized due to exclusion (see Section 1.5.2), the fitness of the best individual is likely to be lower than before reinitialization (since the sub-population now consists of randomly generated individuals), resulting in a large $\Delta F(\mathbf{x}_{best,k},t)$ value. The population is consequently assigned a relatively large performance value, making it likely that it would be allocated fitness evaluations in the near future.

The CPE approach can be clarified by an example. Figure 1.3 plots the error per function evaluation of the best individual in each of three sub-populations (labelled Population 1, Population 2 and Population 3) found during an actual run of the DynDE algorithm on the MPB. During the period that is depicted, no changes occurred in the environment. Note that one of the sub-populations (Population 3) converged to the global optimum, as is evidenced by its error value approaching zero, while the others likely converged to local optima.

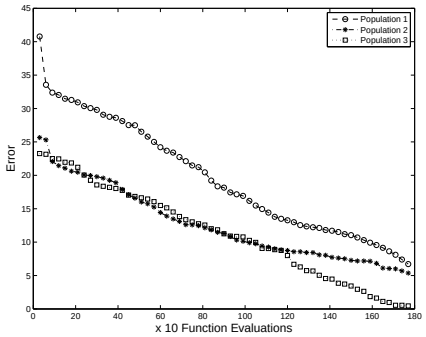

Fig. 1.3 Error profile of three DynDE populations in a static environment

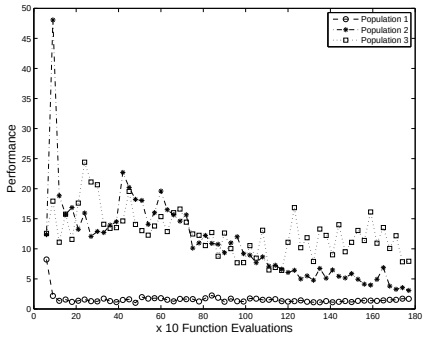

Fig. 1.4 Performance, $\mathscr{P}$, of each population

The performance of each of the populations was calculated using equation (1.13). A plot of the performance of each of the populations is given in Figure 1.4. For roughly the first half of the period depicted, Population 2 and Population 3 alternated between having the highest performance value. After the first half of the depicted period, Population 3 consistently received the highest performance value, as it converged to the global optimum. Population 3 received, on average, a higher performance value than the other two populations. The population that had the highest overall error received a relatively low average performance value. Note that it was only after about 1 000 function evaluations that Population 2 and Population 3 received a lower performance value than the initial performance value of Population 1.

It is now possible, by using the calculated performance values, to observe the behaviour of each of the populations when the CPE algorithm (refer to Algorithm 2) is used to selectively evolve sub-populations on their own based on their respective performance values. The plots of the error of each of the populations per function evaluation when using CPE are given in Figure 1.5. For the first few iterations, Population 2 and Population 3 were alternately evolved, followed by a period where only Population 2 was evolved. After about 250 function evaluations, Population 3 was evolved (except for a few intermittent iterations around 700 function evaluations) until it converged to the global optimum at about 1 000 function evaluations. Population 2 was evolved during the next period which lasted until about 1 200 function evaluations, while Population 1 was evolved during the last period.

Note that the lowest error was reached after 1 000 function evaluations in Figure 1.5, while this point was only reached after 1 800 function evaluations in Figure 1.3. The performance values of the three populations when using CPE are given in Figure 1.6. Observe that for considerable periods the performance values of some of the sub-populations remain constant. This occurs because only one sub-population is evolved at a time, during which the performance value of the other populations remains unchanged.

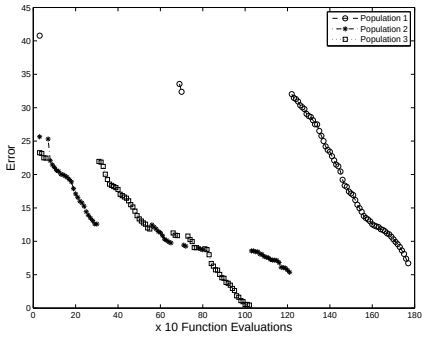

Fig. 1.5 Error profile of three populations when using competitive evaluation

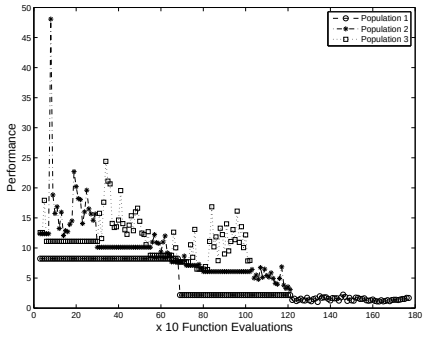

Fig. 1.6 Performance, $\mathscr{P}$, of each population when using competitive evaluation

It is clear how the more successful sub-populations are allocated more function evaluations earlier in the evolution process. Optima are located sequentially by CPE and in parallel by DynDE. Figure 1.7 depicts the effect that using CPE had on the offline error for the three sub-population example. The curve of the offline error when competing sub-populations are used, exhibits a steeper downward gradient than the curve of normal DynDE, due to earlier discovery of the global optimum. Overall, the offline error was reduced by more than 30% after 1 800 function evaluations.

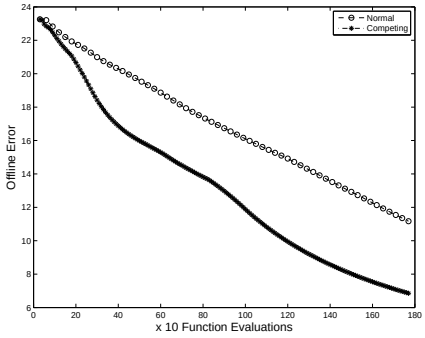

Fig. 1.7 Comparison of offline error for normal and competitive evaluation

By competitively choosing the better performing populations to evolve before other populations, the lowest error value could be reached sooner, thus reducing the average error. This technique has the added advantage that better performing populations will receive more function evaluations that would otherwise have been wasted on finding the maximum of the sub-optimal peaks. The overall error value should consequently also be reduced.

An advantage of CPE is that it only utilizes information that is available in normal DynDE, so that no extra function evaluations are required.

1.5.7 Reinitialization Midpoint Check

Section 1.5.2 explained how DynDE determines when two sub-populations are located on the same optimum, which results in the weaker sub-population being reinitialized. This approach does not take into account the case when two optima are located extremely close to each other, i.e. within the exclusion threshold from one another. In these situations, one of the sub-populations will be reinitialized, leaving one of the optima unpopulated. It was shown [28] that this problem can be partially remedied by determining whether the midpoint between the best individuals in each sub-population constitutes a higher error value than the best individuals of both sub-populations. If this is the case, it implies that a trough exists between the

two sub-populations and that neither should be reinitialized (see Figure 1.8, scenario A). This approach is referred to as the Reinitialization Midpoint Check (RMC) approach. It is apparent that RMC does not work in all cases. Scenarios B and C of Figure 1.8 depict situations where multiple optima within the exclusion threshold are not detected by a midpoint check. Scenario C further constitutes an example where no point between the two optima will give a higher error, thus making it impossible to detect two optima by using any number of intermediate point checks.

This approach is similar to, but simpler, than *hill-valley detection* suggested by [34], since only one point is checked between sub-populations.

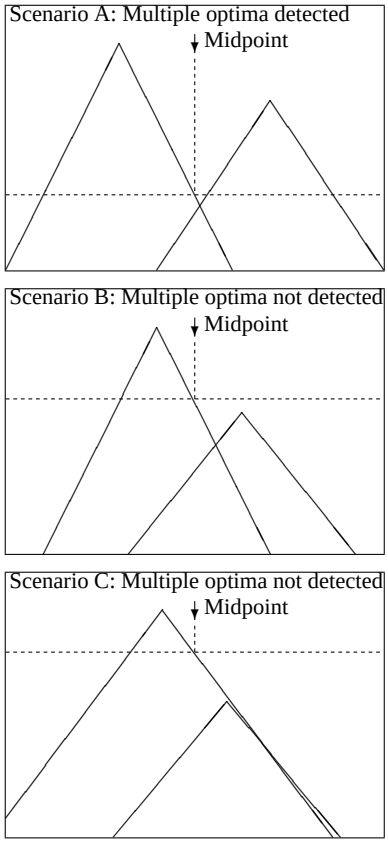

Fig. 1.8 Midpoint checking scenarios

The midpoint check approach provides a method of detecting multiple optima within the exclusion threshold without being computationally expensive or using too many function evaluations, since only one point is evaluated.

1.5.8 Spawning Populations

It was shown [27] that even if the number of peaks is known, creating an equal number of sub-populations as peaks is not always an effective strategy. When the number of peaks is unknown, choosing the number of sub-populations to use would be, at best, an educated guess. It was therefore suggested [27] that the number of sub-populations should not be a parameter of the algorithm, but that sub-populations should be spawned as needed. The question that must be answered is: When should new populations be spawned?

CPE is based on allocating processing time and function evaluations to populations based on a performance value, $\mathscr{P}(k,t)$ (refer to equation (1.13) in Section 1.5.6). Sub-populations are evolved in sequence until all sub-populations converge to their respective optima in the search space. It was proposed that an appropriate time to introduce an additional sub-population is when little further improvement in fitness is found for all the current sub-populations. Introducing new sub-populations earlier would be contrary to the competing population approach of CPE, where inferior sub-populations are deliberately excluded from the evolution process so that optima can be discovered earlier.

A detection scheme is suggested to indicate when evolution has reached a point of little or no improvement in fitness of current sub-populations. This point will be referred to as stagnation. When stagnation is detected, DynPopDE introduces a new population of randomly created individuals so that previously undiscovered optima can be located. CPE calculates the performance value of a sub-population, k, as the product of its current relative fitness, $R_k(t)$, and the improvement that was made in fitness during the previous generation, $\Delta F(\mathbf{x}_{best,k},t)$ (see equation (1.13)). It is suggested that a meaningful indicator of stagnation is when all of the current sub-populations receive a zero improvement of fitness after their last respective function evaluations. Let $\mathscr{K}$ be the set of current populations. Define a function, $\Upsilon(t)$, that is true if stagnation has occurred, as follows:

$$\Upsilon(t) = \begin{cases} true \text{ if} (\Delta F(\mathbf{x}_{best,k},t) = 0) \; \forall \; k \in \mathscr{K} \\ false \text{ otherwise} \end{cases} \tag{1.17}$$

$\Delta F(\mathbf{x}_{best,k},t)$ is defined in equation (1.14). Note that equation (1.17) does not guarantee that stagnation of all populations has permanently occurred, since the fitness of some of the sub-populations may improve in subsequent generations. However, when $\Upsilon(t)$ is true, it does mean that none of the sub-populations has improved its fitness in the previous generation. Since function evaluations are not effectively used by the current sub-populations, a new sub-population should be created which may lead to locating more optima in the problem space.

After each generation, DynPopDE determines the value of $\Upsilon(t)$. If $\Upsilon(t) =$ *true* then a new randomly generated sub-population is added to the set of sub-populations. The sub-population spawning approach allows DynPopDE to commence with only a single population and adapt to an appropriate number of pop-

ulations. The number of sub-populations is thus removed as a parameter from the algorithm.

1.5.9 Removing Populations

The previous section explained how new populations are spawned when necessary. However, it is possible that equation (1.17) detects stagnation incorrectly since it can not guarantee that stagnation for all sub-populations has occurred indefinitely. Consequently, more sub-populations than necessary may be created. Furthermore, in problems where the number of optima fluctuate, it would be desirable to remove superfluous sub-populations when the number of optima decreases. It thus becomes necessary to detect and remove redundant sub-populations.

DynDE reinitializes sub-populations through exclusion when the spatial difference between the population and a more fit population drops below the exclusion threshold. It is reasonable to assume that when redundant populations are present (i.e. more sub-populations exist than optima), these sub-populations will perpetually be reinitialized and will not converge to specific optima since exclusion prevents more than one sub-population converging to the same optima. Consequently, redundant sub-populations can be detected by finding sub-populations that are successively reinitialized through exclusion without reaching a point of apparent stagnation (i.e. there are no more optima available for the sub-population to occupy).

A sub-population, k, will be discarded when it is flagged for reinitialization due to exclusion and if

$$\Delta F(\mathbf{x}_{best,k}, t) \neq 0 \tag{1.18}$$

The RMC exclusion process is thus further adapted to also remove sub-populations (refer to Algorithm 3 where $\mathbf{x}_{best,k}$ is the best individual in sub-population $k \in \{1,2,\ldots,K\}$).

This approach may at times incorrectly classify populations as redundant in that it does not guarantee the removal of populations only when the number of populations outnumber the number of optima. In some cases a sub-population will be discarded for converging to an optimum, occupied by another sub-population, even when undiscovered optima still exist in the the problem space. However, no optimum is ever left unguarded through the discarding process since a sub-population is discarded only when converging to an optimum already occupied by another sub-population. No information about the search space is thus lost through discarding a sub-population. If all sub-populations have stagnated, a new sub-population will be created through the spawning process. The spawning and the discarding components of DynPopDE thus reach a point of equilibrium where sub-populations are created when function evaluations are not being used effectively by the current sub-populations, and sub-populations are removed when converging to optima that are already guarded by other sub-populations.

Algorithm 3: DynPopDE Exclusion

```
for k = 1,...,K do
    for q = 1,...,K do
        if ||x_best,k − x_best,q||_2 < r_excl and k ≠ q then
            Let x_mid = (x_best,k + x_best,q)/2;
            if F(x_mid) < F(x_best,k) and F(x_mid) < F(x_best,q) then
                if F(x_best,k) < F(x_best,q) then
                    if ΔF(x_best,q, t) = 0 then
                        Reinitialize population q
                    else
                        Discard population q
                    end
                else
                    if ΔF(x_best,k, t) = 0 then
                        Reinitialize population k
                    else
                        Discard population k
                    end
                end
            end
        end
    end
end
```

1.5.10 Penalty Factor

The detection of stagnation is essential to DynPopDE's population spawning process. The detection strategy described in Section 1.5.8 is based on all sub-populations not improving their respective fitness values in the previous generation. However, in CPE, sub-populations are evolved alone based on performance value. It is possible that a situation could occur where some of the weaker sub-populations are not allocated enough fitness evaluation to reach the stagnation point as detected by equation (1.17). It was found experimentally that in order to detect stagnation effectively, it is necessary to distribute function evaluations more uniformly among all populations [27]. The final component of DynPopDE is the introduction of a penalty factor into the performance value, $\mathscr{P}(k,t)$ given in equation (1.13), to penalize populations for successively receiving the highest performance value without showing any improvement in fitness.

Each population, k, maintains a penalty factor $pen_k(t)$ which is calculated as follows:

$$pen_k(t) = \begin{cases} pen_k(t-1)+1 \text{ if}(\Delta F(\mathbf{x}_{best,k},t)=0) \\ 0 \text{ otherwise} \end{cases} \tag{1.19}$$

The penalty factor is thus reset to zero as soon as an improvement of fitness is found. The performance with penalty $\mathscr{P}_{pen}(k,t)$ is calculated as:

$$\mathscr{P}_{pen}(k,t) = \begin{cases} \frac{\mathscr{P}(k,t)}{pen_k(t)} \text{ if}(pen_k(t) > 0) \\ \mathscr{P}(k,t) \text{ otherwise} \end{cases} \tag{1.20}$$

Populations that stagnate but still receive a large performance value, $\mathscr{P}(k,t)$, will eventually receive a low value for performance with penalty, $\mathscr{P}_{pen}(k,t)$, and will consequently not dominate the evolution process.

The necessity of using a penalty implies that there is an intrinsic cost to detecting stagnation effectively, since it is necessary to waste more function evaluations on weaker sub-populations to ensure a reasonable chance to stagnate.

1.6 Self-Adaptive DynPopDE

The main goal of this study is an investigation into the effect of incorporating self-adaptive control parameters into DynPopDE. This section describes the self-adaptive approach that is incorporated into DynPopDE to form the Self-Adaptive Dynamic Population Differential Evolution (SADynPopDE) algorithm.

1.6.1 The Self-Adaptive DynPopDE Algorithm

In a previous study, it has been shown that CDE can be improved by self-adapting the DE scale and crossover factors [29]. Three different DE-based approaches to self-adaptation were investigated [23][9][22], and it was found that the the approach of Brest *et al.* [9] is the most effective of the three when used in conjunction with CDE.

The approach of Brest *et al.* [9] self-adapts the values of both the crossover factor and the scale factor. Each individual, $\mathbf{x}_i$, stores its own value for the crossover factor, Cr_i, and scale factor, $\mathscr{F}_i$. Before equation (1.3) is used to create a new trial individual, the scale factor and crossover factor of the target individual ($\mathbf{x}_i$ in equation (1.4)) are used to create new values for the scale factor and crossover factor to be used in equations (1.3) and (1.4). The new scale and crossover factors ($\mathscr{F}_{new}$ and Cr_{new}) are calculated as follows:

$$\mathscr{F}_{new} = \begin{cases} \mathscr{F}_l + U(0,1) \cdot \mathscr{F}_u \text{ if } (U(0,1) < \tau_1) \\ \mathscr{F}_i \text{ otherwise} \end{cases} \tag{1.21}$$

$$Cr_{new} = \begin{cases} U(0,1) \text{ if } (U(0,1) < \tau_2) \\ Cr_i \text{ otherwise} \end{cases} \tag{1.22}$$

where τ_1 and τ_2 are the probabilities that the factors will be adjusted. Brest *et al.* [9] used 0.1 for both τ_1 and τ_2. Other values used were $\mathscr{F}_l = 0.1$ and $\mathscr{F}_u = 0.9$. $\mathscr{F}_l$ is a constant introduced to avoid premature convergence by ensuring that the scale

factor is never too small (see Section 1.4.3), while $\mathscr{F}_u$ determines the range of scale factors that can be explored by the algorithm.

This approach is the basis for the successful *jDE* algorithm [10] which is aimed at dynamic environments. Brest *et al.* [10] used $\mathscr{F}_l = 0.36$ rather than $\mathscr{F}_l = 0.1$ for the dynamic algorithm. Initial values of 0.9 for the crossover factor and 0.5 for the scale factor were used [10]. *jDE* makes use of DE/rand/1/bin, but it was found that DE/best/2/bin yielded the best results in the CDE-based version of this algorithm [29].

The self-adaptive approach of Brest *et al.* [9][10] was incorporated into DynPopDE to form SADynPopDE. The following section presents experimental evidence that shows that SADynPopDE is an improvement over DynDE and DynPopDE.

1.6.2 Results and Discussion

This section experimentally compares the performance of SADynPopDE to that of DynDE and DynPopDE, in order to determine whether self-adaptive control parameters yields a more effective algorithm.

The modified MPB described in Section 1.2.2 was used to model problems with a fluctuating number of peaks. For these experiments, DynDE was given 10 sub-populations, since this number was shown yield reasonable performance on this type of problem [27]. The number of individuals in the sub-populations of all algorithms were set to six. For all algorithms, DynDE and DynPopDE were compared to SADynPopDE for different values of maximum number of peaks and percentage change in number of peaks. Additionally, the effect of varying dimension and change period was investigated. All combinations of settings listed in Table 1.2 were investigated.

Table 1.2 MPB settings for fluctuating number of peaks experiments.

Setting	**Values**
Number of Dimensions	5, 10, 15
Change period	1000, 2000, 3000, 4000, 5000
Maximum Number of Peaks	20, 40, 60, 80, 100, ..., 180, 200
Percentage Change in Number Peaks	10, 20, 30, 40, 50

Each experiment was repeated 50 times. Average offline error and 95% confidence interval for DynDE, DynPopDE and SADynPopDE along with the results of Mann-Whitney U tests are summarized in Tables 1.3 and 1.4. For the sake of brevity, some of the results were omitted, but are available from the authors. The tables list the maximum number of peaks (**Max Peaks**), percentage change in number of peaks (**% Change**), change period (**CP**), offline error of DynDE, offline error of

DynPopDE, offline error of SADynPopDE, the p-values of Mann-Whitney U tests comparing the differences in offline error between DynDE and DynPopDE (**p-A**), and DynPopDE and SADynPopDE (**p-B**). Values for which the differences were not statistically significant at a 95% confidence level are given in boldface. Cases where the offline error of DynPopDE is lower than that of SADynPopDE are given in italics.

Table 1.3 Results in 5 dimensions

Max Peaks	% Change	CP	DynDE	DynPopDE	p-A	SADynPopDE	p-B
20	10	1000	15.73 ± 1.22	11.39 ± 0.69	0.00	6.78 ± 0.32	0.00
20	10	3000	3.85 ± 0.33	3.21 ± 0.33	0.00	2.14 ± 0.13	0.00
20	10	5000	2.42 ± 0.19	1.96 ± 0.19	0.00	1.38 ± 0.09	0.00
20	30	1000	28.92 ± 1.46	19.88 ± 0.88	0.00	13.53 ± 0.56	0.00
20	30	3000	7.84 ± 0.64	5.61 ± 0.53	0.00	3.56 ± 0.26	0.00
20	30	5000	3.97 ± 0.45	3.24 ± 0.34	0.01	2.22 ± 0.21	0.00
20	50	1000	37.73 ± 1.15	26.15 ± 0.88	0.00	17.73 ± 0.57	0.00
20	50	3000	11.35 ± 0.86	9.10 ± 0.60	0.00	5.61 ± 0.38	0.00
20	50	5000	6.30 ± 0.59	5.32 ± 0.46	0.04	2.89 ± 0.26	0.00
40	10	1000	16.70 ± 1.44	10.68 ± 0.99	0.00	8.62 ± 0.58	0.01
40	10	3000	5.09 ± 0.48	3.94 ± 0.47	0.00	2.60 ± 0.15	0.00
40	10	5000	3.07 ± 0.20	2.38 ± 0.20	0.00	1.64 ± 0.09	0.00
40	30	1000	28.97 ± 0.99	21.94 ± 1.24	0.00	15.11 ± 0.72	0.00
40	30	3000	10.14 ± 0.92	6.54 ± 0.46	0.00	5.00 ± 0.41	0.00
40	30	5000	6.37 ± 0.71	4.11 ± 0.44	0.00	2.70 ± 0.27	0.00
40	50	1000	38.19 ± 1.22	28.00 ± 0.88	0.00	20.74 ± 0.70	0.00
40	50	3000	15.00 ± 1.18	10.53 ± 0.69	0.00	6.96 ± 0.58	0.00
40	50	5000	8.52 ± 0.89	5.41 ± 0.58	0.00	4.20 ± 0.39	0.00
80	10	1000	16.30 ± 1.70	11.82 ± 1.19	0.00	9.19 ± 0.75	0.00
80	10	3000	6.64 ± 0.80	3.95 ± 0.34	0.00	3.16 ± 0.27	0.00
80	10	5000	3.62 ± 0.22	2.76 ± 0.29	0.00	2.18 ± 0.14	0.00
80	30	1000	29.26 ± 1.35	20.44 ± 0.94	0.00	16.75 ± 0.85	0.00
80	30	3000	13.29 ± 1.13	7.73 ± 0.67	0.00	5.78 ± 0.38	0.00
80	30	5000	6.42 ± 0.58	4.95 ± 0.57	0.00	3.44 ± 0.33	0.00
80	50	1000	37.94 ± 1.17	28.09 ± 0.78	0.00	21.46 ± 0.57	0.00
80	50	3000	17.12 ± 1.12	11.75 ± 0.93	0.00	8.33 ± 0.56	0.00
80	50	5000	9.83 ± 0.87	7.60 ± 0.72	0.00	5.58 ± 0.40	0.00
160	10	1000	16.60 ± 1.55	12.03 ± 1.22	0.00	9.55 ± 0.81	0.00
160	10	3000	7.94 ± 1.29	5.03 ± 0.57	0.00	3.54 ± 0.29	0.00
160	10	5000	4.53 ± 0.46	3.37 ± 0.61	0.00	2.66 ± 0.30	0.00
160	30	1000	28.60 ± 1.31	22.61 ± 1.05	0.00	16.80 ± 0.76	0.00
160	30	3000	14.38 ± 1.26	9.43 ± 0.94	0.00	6.48 ± 0.51	0.00
160	30	5000	8.95 ± 1.04	5.67 ± 0.78	0.00	4.22 ± 0.42	0.01
160	50	1000	36.32 ± 1.18	29.31 ± 0.93	0.00	22.10 ± 0.70	0.00
160	50	3000	18.71 ± 1.20	12.14 ± 0.84	0.00	9.32 ± 0.50	0.00
160	50	5000	12.58 ± 1.10	8.15 ± 0.72	0.00	5.72 ± 0.43	0.00

The results show that DynDE generally yields large offline errors when the number of optima fluctuates. Increasing the number of dimensions greatly increases the

Table 1.4 Results in 15 dimensions

Max Peaks	% Change	CP	DynDE	DynPopDE	p-A	SADynPopDE	p-B
20	10	1000	118.51 ± 5.70	63.59 ± 5.33	0.00	68.97 ± 4.89	***0.14***
20	10	3000	47.52 ± 8.12	19.39 ± 2.74	0.00	16.95 ± 2.36	**0.23**
20	10	5000	25.82 ± 5.15	11.66 ± 2.10	0.00	8.43 ± 1.18	0.02
20	30	1000	146.05 ± 2.30	98.36 ± 4.83	0.00	101.37 ± 3.30	***0.11***
20	30	3000	80.93 ± 6.07	42.26 ± 4.71	0.00	32.34 ± 2.08	0.00
20	30	5000	47.70 ± 5.81	24.73 ± 3.93	0.00	17.64 ± 2.18	0.00
20	50	1000	157.85 ± 2.53	120.04 ± 4.06	0.00	117.58 ± 2.69	**0.72**
20	50	3000	107.02 ± 5.33	51.94 ± 4.49	0.00	41.19 ± 3.12	0.00
20	50	5000	67.92 ± 5.41	33.87 ± 3.07	0.00	26.50 ± 2.26	0.00
40	10	1000	103.93 ± 5.47	64.02 ± 6.34	0.00	58.14 ± 4.89	**0.18**
40	10	3000	37.39 ± 7.52	21.58 ± 3.94	0.00	15.64 ± 2.32	**0.08**
40	10	5000	21.77 ± 4.48	13.66 ± 2.39	0.00	9.62 ± 1.75	0.00
40	30	1000	137.11 ± 3.09	91.94 ± 4.25	0.00	92.80 ± 3.65	***0.60***
40	30	3000	72.50 ± 5.20	40.85 ± 3.86	0.00	32.65 ± 2.94	0.00
40	30	5000	52.98 ± 4.95	24.85 ± 3.93	0.00	20.24 ± 2.45	**0.13**
40	50	1000	146.45 ± 2.19	112.54 ± 3.89	0.00	110.23 ± 3.29	**0.48**
40	50	3000	98.18 ± 4.94	52.51 ± 3.15	0.00	43.75 ± 2.97	0.00
40	50	5000	65.42 ± 5.49	34.49 ± 2.69	0.00	27.34 ± 2.43	0.00
80	10	1000	95.19 ± 6.04	55.47 ± 5.40	0.00	56.74 ± 4.94	***0.49***
80	10	3000	41.76 ± 7.72	27.93 ± 5.55	0.00	16.13 ± 2.43	0.00
80	10	5000	24.61 ± 6.25	12.44 ± 3.29	0.00	7.90 ± 1.06	0.01
80	30	1000	128.25 ± 2.83	89.70 ± 4.70	0.00	84.20 ± 3.49	**0.16**
80	30	3000	78.10 ± 5.63	41.28 ± 4.51	0.00	32.97 ± 3.11	0.01
80	30	5000	43.62 ± 5.92	25.94 ± 3.28	0.00	20.64 ± 2.08	0.03
80	50	1000	138.18 ± 2.17	107.12 ± 3.54	0.00	108.03 ± 2.56	***0.32***
80	50	3000	92.44 ± 4.16	59.88 ± 5.61	0.00	43.56 ± 2.66	0.00
80	50	5000	62.89 ± 4.50	34.59 ± 4.25	0.00	30.44 ± 2.30	**0.29**
160	10	1000	87.46 ± 5.89	56.10 ± 6.78	0.00	48.07 ± 5.54	**0.11**
160	10	3000	33.20 ± 5.60	22.77 ± 3.74	0.00	14.77 ± 2.51	0.00
160	10	5000	25.33 ± 6.24	12.41 ± 3.10	0.00	12.84 ± 2.62	***0.72***
160	30	1000	120.91 ± 2.52	85.79 ± 4.21	0.00	81.53 ± 3.57	**0.22**
160	30	3000	69.23 ± 5.86	38.10 ± 3.93	0.00	34.71 ± 3.42	**0.23**
160	30	5000	47.35 ± 6.68	27.42 ± 3.56	0.00	21.79 ± 3.08	0.02
160	50	1000	132.74 ± 2.26	104.89 ± 3.03	0.00	102.15 ± 2.75	**0.27**
160	50	3000	88.49 ± 4.18	54.33 ± 2.69	0.00	45.54 ± 2.76	0.00
160	50	5000	66.22 ± 6.10	37.94 ± 3.25	0.00	31.54 ± 2.61	0.00

offline error, because the size of the search space is effectively increased which makes it harder to locate optima. In addition, lowering the number of function evaluations between changes in the environment has a negative effect on the offline error, since fewer generations can be performed between changes in the environment.

Fluctuating the number of optima in the solution space results in severe changes in the environment. When optima vanish from the solution space, entire subpopulations are left with a low fitness value. Similarly, optima are introduced in areas of the search space which is not guarded by individuals. As the maximum number of optima in the environment increases (and subsequently the number of

optima that are introduced or removed when a change in the environment occurs), the offline error of DynDE increases.

Increasing the percentage change in number of peaks negatively affected the performance of DynDE. This trend is expected, since, when peaks are removed, the sub-populations guarding those peaks are reduced to relatively low fitness values. Similarly, new peaks that are introduced will initially not be located by sub-populations. When the percentage of peaks that are removed or introduced is increased, either, more sub-populations are left not guarding a peak, or more peaks are introduced that are initially unguarded. The detrimental effect of percentage change in number of peaks was more pronounced for larger values of maximum number of peaks, since a greater number of peaks are introduced or removed.

The same general trends that are visible in the DynDE results were also found for both DynPopDE and SADynPopDE, i.e. the offline error deteriorates when the change period is reduced, the percentage change in number of peaks is increased, or the number of dimensions is increased.

A comparison of the results of DynDE and DynPopDE (refer to Tables 1.3 and 1.4) shows DynPopDE yielded a clear improvement over DynDE. Over all experiments, it was found that 746 of the 750 experiments resulted in a statistically significantly better result for DynPopDE. On average, DynPopDE yielded an improvement over DynDE of 29.22% in 5 dimensions, 37.02% in 10 dimensions and 40.55% in 15 dimensions. The magnitudes of the improvements are greater when larger values for maximum number of peaks were used, as population spawning ability of DynPopDE is more beneficial when in these regions.

Figure 1.9 graphically depicts the surface found when subtracting the average offline error of SADynPopDE from the average offline error of DynPopDE for various settings of the MPB when a 50% maximum change in the number of peaks was used. For 5 dimensions almost all the SADynPopDE results were statistically significantly better than those of DynPopDE. For the 10 dimensional experiments the majority of the SADynPopDE results were statistically significantly better than DynPopDE while in 15 dimensions the majority of the differences were not statistically significant. SADynPopDE did not perform statistically significantly worse than DynPopDE in any of the 5 and 10 dimensional experiments. In 15 dimensions, SADynPopDE was only statistically significantly worse then DynPopDE in 1 of the 250 experiments.

Figure 1.9 shows how the improvement of SADynPopDE over DynPopDE in 5 dimensions becomes more pronounced as the change period becomes smaller. This trend is reversed in 10 and 15 dimensions where the improvement of SADynPopDE over DynPopDE was relatively small for change periods of 1000. The improvements of SADynPopDE over DynPopDE were less frequently significant in the higher dimensions, but differences in offline error are nonetheless often larger than 10. It was found that SADynPopDE was more often better than DynPopDE when the percentage change in the number of peaks is larger then 30%.

From the results it can be concluded that SADynPopDE is an improvement over DynPopDE, especially in low dimensional problems.

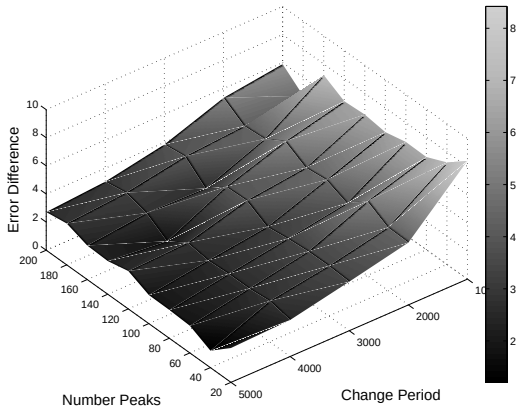

Fig. 1.9 Difference in offline error between DynPopDE and SADynPopDE for various settings of maximum number of peaks and change period in 5 dimensions.

1.7 Summary

This chapter introduced SADynPopDE, a self-adaptive multi-population DE-based optimization algorithm, aimed at dynamic optimization problems in which the number of optima in the environment fluctuates over time. SADynPopDE was experimentally compared to two algorithms upon which it is based: DynDE and DynPopDE. DynDE extends DE for dynamic environments by utilizing multiple sub-populations which are encouraged to converge to distinct optima by means of exclusion. DynPopDE extends DynDE by: using competitive population evaluation to selectively evolve sub-populations, using a midpoint check during exclusion to determine whether sup-populations are indeed converging to the same optimum, dynamically spawning and removing sub-populations, and using a penalty factor to aid the stagnation detection process.

Experimental evidence suggests that DynPopDE is more effective on problems in which the number of optima fluctuate than DynDE. In turn, it was shown that SADynPopDE is more effective than DynPopDE on these types of problems, especially on low dimensional cases.

In conclusion, the incorporation of self-adaptive control parameters into DynPopDE, not only yielded a more effective algorithm, but also removes the need to fine-tune the DE crossover and scale factors.

References

1. Blackwell, T.: Particle swarm optimization in dynamic environments. In: Evolutionary Computation in Dynamic and Uncertain Environments, pp. 29–49. Springer (2007)
2. Blackwell, T., Branke, J.: Multiswarm optimization in dynamic environments. Applications of Evolutionary Computing **3005**, 489–500 (2004)
3. Blackwell, T., Branke, J.: Multiswarms, exclusion, and anti-convergence in dynamic environments. IEEE Transactions on Evolutionary Computation **10**(4), 459–472 (2006)
4. Branke, J.: Evolutionary Optimization in Dynamic Environments. Kluwer Academic Publishers, Norwell, MA, USA (2002)
5. Branke, J.: The moving peaks benchmark. http://www.aifb.uni-karlsruhe.de/ ˜jbr/MovPeaks/ (2007)
6. Branke, J., Kauler, T., Schmidt, C., Schmeck, H.: A multi-population approach to dynamic optimization problems. In: In Adaptive Computing in Design and Manufacturing, pp. 299–308. Springer (2000)
7. Branke, J., Schmeck, H.: Designing evolutionary algorithms for dynamic optimization problems. In: S. Tsutsui, A. Ghosh (eds.) Theory and Application of Evolutionary Computation: Recent Trend s, pp. 239–262. Springer (2002)
8. Branke, J., Schmeck, H.: Designing evolutionary algorithms for dynamic optimization problems pp. 239–262 (2003)
9. Brest, J., Greiner, S., Boskovic, B., Mernik, M., Zumer, V.: Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. Evolutionary Computation, IEEE Transactions on **10**(6), 646–657 (2006)
10. Brest, J., Zamuda, A., Boškovic, B., Maučec, M.S., Žumer, V.: Dynamic optimization using self-adaptive differential evolution. In: CEC'09: Proceedings of the Eleventh conference on Congress on Evolutionary Computation, pp. 415–422. IEEE Press, Piscataway, NJ, USA (2009)
11. Carlisle, A., Dozier, G.: Tracking changing extrema with adaptive particle swarm optimizer. In: Proc. World Automation Congress, pp. 265–270 (2002)
12. Darwin, C.: The origin of species, (1859)
13. Engelbrecht, A.P.: Computational Intelligence An Introduction, 2nd edn. John Wiley and Sons (2007)
14. Hu, X., Eberhart, R.: Adaptive particle swarm optimisation: detection and response to dynamic systems. In: Proceedings Congress on Evolutionary Computation, pp. 1666–1670 (2002)
15. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments - a survey. IEEE Transactions on Evolutionary Computation **9**(3), 303–317 (2005)
16. Li, C., Yang, S.: A generalized approach to construct benchmark problems for dynamic optimization. In: SEAL '08: Proceedings of the 7th International Conference on Simulated Evolution and Learning, pp. 391–400. Springer-Verlag, Berlin, Heidelberg (2008)

17. Li, C., Yang, S., Nguyen, T.T., Yu, E.L., Yao, X., Jin, Y., g. Beyer, H., Suganthan, P.N.: Benchmark Generator for CEC2009 Competition on Dynamic Optimization. University of Leicester, University of Birmingham, Nanyang Technological University, Technical Report (2008)
18. Li, X., Branke, J., Blackwell, T.: Particle swarm with speciation and adaptation in a dynamic environment. In: GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pp. 51–58. ACM, New York, NY, USA (2006)
19. Mendes, R., Mohais, A.: Dynde: a differential evolution for dynamic optimization problems. In: Congress on Evolutionary Computation, pp. 2808–2815. IEEE (2005)
20. Mezura-Montes, E., Velázquez-Reyes, J., Coello, C.A.: A comparative study of differential evolution variants for global optimization. In: GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pp. 485–492. ACM, New York, NY, USA (2006)
21. Morrison, R.: Designing Evolutionary Algorithms for Dynamic Environments. Springer (2004)
22. Omran, M., Engelbrecht, A., Salman, A.: Bare bones differential evolution. European Journal of Operational Research **196**(1), 128 – 139 (2009)
23. Omran, M., Salman, A., Engelbrecht, A.: Self-adaptive differential evolution. In: Proceedings of the International Conference on Computational Intelligence and Security, p. 192199. Xian, China (2005)
24. Oppacher, F., Wineberg, M.: The shifting balance genetic algorithm: Improving the ga in a dynamic environment. In: W.B. et al. (ed.) Genetic and Evolutionary Computation Conference (GECCO), vol. 1, pp. 504 – 510. Morgan Kaufmann, San Francisco (1999)
25. Parrott, D., Li, X.: A particle swarm model for tracking multiple peaks in a dynamic environment using speciation. In: Congress on Evolutionary Computation, pp. 98–103. IEEE (2004)
26. du Plessis, M., Engelbrecht, A.: Improved differential evolution for dynamic optimization problems. IEEE Congress on Evolutionary Computation. CEC 2008. pp. 229–234 (2008)
27. du Plessis, M., Engelbrecht, A.: Differential evolution for dynamic environments with unknown numbers of optima. Submitted to Journal of Global Optimization (2010)
28. du Plessis, M., Engelbrecht, A.: Using competitive population evaluation in a differential evolution algorithm for dynamic environments. Submitted to European Journal of Operational Research (2010)
29. du Plessis, M., Engelbrecht, A.: Self-adaptive competitive differential evolution for dynamic environments. IEEE Symposium Series on Computational Intelligence. SSCI 2011. (2011)
30. Price, K., Storn, R., Lampinen, J.: Differential evolution - A practical approach to global optimization. Springer (2005)
31. Storn, R.: On the usage of differential evolution for function optimization. In: Biennial Conference of the North American Fuzzy Information Processing Society, pp. 519–523. IEEE (1996)
32. Storn, R., Price, K.: Minimizing the real functions of the icec96 contest by differential evolution. In: In: IEEE Conference on Evolutionary Computation, pp. 842–844. IEEE (1996)
33. Storn, R., Price, K.: Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization **11**, 341–359 (1997)
34. Ursem, R.K.: Multinational GA optimization techniques in dynamic environments. In: D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, H.G. Beyer (eds.) Genetic and Evolutionary Computation Conference, pp. 19–26. Morgan Kaufmann (2000)
35. Zaharie, D., Zamfirache, F.: Diversity enhancing mechanisms for evolutionary optimization in static and dynamic environments. In: 3rd Romanian-Hungarian Joint Symposium on Applied Computational Intelligence, pp. 460–471 (2006)