COOPERATIVE CO-EVOLUTINARY ALGORITHMS FOR LARGE SCALE OPTIMISATION

MOHAMMAD NABI OMIDVAR

Supervisor: DR XIAODONG LI

Honours Thesis

School of Computer Science and Information Technology
RMIT University
Melbourne, AUSTRALIA

June, 2010

Abstract

The need for large scale global optimisation is growing with an astonishing pace. However, the exponential growth of the search space makes large scale optimisation a very challenging task. Interdependent variables in large scale problems makes them even harder to optimise. A popular divide-and-conquer approach for tackling large scale optimisation problem is to use a cooperative co-evolutionary model, where two techniques, *random grouping* and *adaptive weighting*, are adopted. In particular, the random grouping technique is shown to be effective in handling problems with variable interactions. In this thesis we show that adaptive weighting is ineffective and wastes considerable amount of computational resources. Furthermore, by employing more frequent random grouping and a new scheme for dynamically determining the subcomponent group sizes, we demonstrate that a substantial amount of computation can be saved, without scarifying the performance accuracy. In fact, in many instances, a better performance accuracy can be achieved. Our experimental results further vindicated these observations. This thesis also contributes to cooperative co-evolutionary modeling by generalising an existing mathematical study on random grouping. This mathematical study further demonstrates the importance of more frequent random grouping, and the need to develop a more intelligent variable grouping technique, than random grouping. However the mathematical study revealed that even the increased frequency of random grouping does not help when the number of interacting variables increases. Consequently, we propose a new grouping method called *delta grouping* that deals with the interacting variables in a more systematic and intelligent manner. The empirical study of *delta grouping* confirmed its efficiency in capturing interacting variables with a high success rate on large scale non-separable problems with up to 1000 real-valued variables.

Acknowledgements

I would like to take this opportunity to convey my special thanks to my supervisor Dr Xiaodong Li for his incredible support during the past year. His support is beyond an ordinary supervision. Without his help publishing two papers in a high profile conference within a period of six month was absolutely impossible.

Thanks to Professor Xin Yao, and Mr Zhenyu Yang for their useful comments on my work, to all the ECML group members, in particular, Assoc. Professor Vic Ciesielski and Dr Andy Song for their great discussions and feedbacks, to my great friends Mr Amir H. Fassihi and Alireza Vahdati for commenting on my work and their great suggestions, to my parents for their continual support throughout my life, and last but not least, to $\Phi\alpha\rho\iota\beta\alpha$ for giving me endless support, motivation and courage to overcome the hardships.

Contents

List of Figures

List of Tables

List of Publications

1. Omidvar M. N. and Li X. and Yao X. (2010). *Cooperative Co-evolution with Delta Grouping for Large Scale Non-separable Function Optimisation*. Proc. IEEE Congress on Evolutionary Computing (CEC).

2. Omidvar M. N. and Li X. and Yang X. and Yao X. (2010). *Cooperative Co-evolution for Large Scale Optimisation Through More Frequent Random Grouping*. Proc. IEEE Congress on Evolutionary Computing (CEC).

1 Introduction

Optimisation problems are ubiquitous, from science to engineering, from manufacturing to economics, or even our daily life. In its simplest form, optimisation can be defined as the process of finding the best solution to a problem in a set of available alternatives. Finding the best shape for a turbine blade, the optimum shape of an aeroplane wing to maximise the lift power, the best way of running water or gas pipes in a city with lowest budget, or even planning our journey to work in order to minimise the travelling time are just a few examples. Many optimisation techniques have been developed in the past. However, many factors exist that can make optimisation a very challenging task. Non-linear relation between the decision variables, noise, non-convexity of search landscape, and high number of decision variables are sources of complication. Various mathematical techniques have been developed for optimisation. However, these traditional optimisation techniques usually fail when the problem is non-linear, non-convex, high dimensional or when there is no mathematical formula to represent the problem (Price et al., 2005). Evolutionary Algorithms (EAs) (Holland, 1975; Goldberg, 1989; Bäck, 1996; Bäck et al., 1997) have been developed to combat these problems and can be used as an alternative to traditional optimisation techniques.

When applying EAs to difficult optimisation problems, their performance tend to deteriorate as the dimensionality of the problem increases (Liu et al., 2001), which is commonly referred to as the *curse of dimensionality* (Bellman, 1957). There are two major reasons for difficulty of large scale optimisation problems. First, it may is due to possible changes in the properties of the search space. For example, a unimodal function may turn into a multimodal function in higher dimensions (Rosenbrock, 1960). Second, the size of the search space grows exponentially as the dimensionality increases. However, not all large scale problems are difficult to optimise, for example when the decision variables are independent, a large scale problem could be decomposed into a set of one dimensional problems, but this is not necessarily true for many real-world problems. In many instances the problem of finding the global optimum becomes even more challenging when some or all of the decision variables have interaction amongst themselves. This class of problems are called non-separable problems and their indecomposable nature makes them extremely hard to optimise (Salomon, 1995). Variable interaction in large scale problems drastically increases the total number of function evaluations in order to find a reasonable solution.This makes large scale non-separable function optimisation a very challenging task.

The difficult nature of large scale non-separable problems gets magnified when dealing with real-world problems mainly because evaluating the candidate solutions for such problems is often very costly. For example, in the optimisation of gas turbine stator blades, the fitness evaluation has to be performed on a simulation software that may take from several minutes to several days, depending on the dimensionality of the problem (Hasenjäger et al., 2005). Evolutionary Robotics (Nolfi and Floreano, 2000), Multidisciplinary Design Optimisation (MDO) (Sobieszczanski-Sobieski and Haftka, 1997), Target Shape Design Optimisation Problems (TSDOPs) (Olhofer et al., 2001) are other areas with potential expensive evaluation. This demands new techniques to be developed in order to cut down the computational cost.

1.1 Research Questions

Many techniques have been proposed for solving large scale problems. Cooperative Co-evolution (CC) proposed by Potter and De Jong (1994) uses a divide-and-conquer approach to divide the decision variables into subpopulations of smaller sizes and each of these subpopulations is optimised with a separate EA in a round robin fashion. CC seems to be a promising framework for large scale

problems, but its performance degrades when applied to non-separable problems. The reason for this performance loss is highly related to the decomposition strategy. In the original CC implementation, each variable is placed in a separate subcomponent (i.e. one variable per subcomponent). This reduces the probability of optimising two interacting variables at the same time. van den Bergh and Engelbrecht (2004) further extended the Potter and De Jong (1994)'s decomposition method by allowing a n-dimensional problem to be divided into m static s-dimensional problems. This static decomposition of decision variables allows several (i.e., more than one) possibly interacting variables to be placed into a common subcomponent, therefore it has a better performance compared to the original CC decomposition strategy (van den Bergh and Engelbrecht, 2004). In an ideal setting, all the interacting variables should be grouped in one subcomponent in order to further enhance the performance of optimisation, but in real-world problems there is almost no prior information about how the variables are interacting. This demands for better techniques capable of capturing the interacting variables through the course of evolution and grouping them together in a common subcomponent.

Recently a new technique called random grouping has been proposed by Yang et al. (2008a) which shows significant improvement over previous CC techniques for large scale non-separable optimisation problems. In random grouping, every decision variable is randomly assigned to any of the subcomponents with equal probability and the whole process is repeated at the beginning of every cycle. It has been shown that by using random grouping the probability of grouping *two* interacting variables in the same subcomponent will significantly increase (Yang et al., 2008a). Random grouping is incorporated into an algorithm called DECC-G. This algorithm also evolves a weight vector for co-adaptation of subcomponents for further improving the solutions. Adaptive weighting and random grouping are described in more details in Chapter 2.

MLCC (Yang et al., 2008b) is another new technique for large scale non-separable function optimisation which extends DECC-G (Yang et al., 2008a) by self-adapting the subcomponent sizes. MLCC has shown superior performance over DECC-G (Yang et al., 2008b).

Although considerable amount of research has been conducted by researchers in the field of large scale global optimisation, it still remains as a very challenging problem and there is ample room for further improvements. *Random grouping* is a naive technique in dealing with variable interaction, and we speculate that increasing the number of interacting variables has a significant detrimental effect on its performance, especially in higher dimensions. In general, we aim to answer the following questions in this thesis:

- Does the *random grouping* scheme scales properly when the number of interacting variables increases?
- How can we design a simpler and more effective alternative to MLCC for self-adaptation of the subcomponents sizes in Cooperative Co-evolution?
- How can we develop a new technique, capable of capturing the interacting variables in a more systematic way?
- How effective does the new technique scales up when the number of interacting variables increases?

In order to answer the above questions, in this thesis, we first investigate the internal mechanisms of DECC-G and MLCC algorithms and demonstrate two major bottlenecks that degrades the performance of these algorithms. We also propose two techniques for further improving the performance of MLCC. It is also shown – by use of mathematical proof – that even the improved version of MLCC does not scale properly when the number of interacting variables increases. Ultimately we propose

a new technique for more systematic identification of the interacting variables and grouping them in one subcomponent. We call this new technique *delta grouping* which is described in further details in Chapter 4. Next section contains an elaborated list of research contributions.

1.2 Research Contributions

This research has made the following contributions to the field of large scale continuous global optimisation using a cooperative co-evolutionary framework:

- Extending the theoretical background of random grouping to include more than two interacting variables.
- Showing that more frequent random grouping is beneficial, especially when there are more than two interacting variables.
- Identifying two major bottlenecks in MLCC and showing how to increase the frequency of random grouping to substantially improve its performance without additional computational cost.
- Demonstrating that *adaptive weighting* is ineffective, wasting valuable fitness evaluations, and show that how the overall performance will be significantly improved by simply disabling this feature.
- Developing a simpler, more intuitive and yet more efficient alternative for self-adaptation of subcomponent sizes. This new algorithm is called DECC-ML
- Showing that DECC-ML provides faster convergence compared to previous techniques. This faster convergence results in saving up to $\frac{2}{3}$ of fitness evaluations which is a significant improvement especially in expensive optimisation problems.
- Proposing a more intelligent and systematic method than *random grouping* in order to better capture and group the interacting variables.

Next we will provide a survey of the related works in the literature. The main focus would be on the shortcomings of the current techniques and how they are related to the current study.

2 Related Works

In this chapter we describe the basic techniques and algorithms which are central to the current research. We present a survey of related works, and explain how they are related to our study. We also touch upon the shortcomings of previous techniques to further motivate the reason for undertaking this research. The structure of the rest of this chapter is as follows. Section 2.1 briefly describes the field of function optimisation and evolutionary approaches to function optimisation. Section 2.2 describes the Cooperative Co-evolutionary EAs. Section 2.3 gives a detailed description of Differential Evolution which is the evolutionary technique used in our experiments. Section 2.4 describes the *random grouping* approach to large scale global optimisation and emphasises its shortcomings. Section 2.5 describes Multilevel Cooperative Co-evolution (MLCC) which is an implementation of *random grouping* and *adaptive weighting* that self-adapts the subcomponent sizes through the course of evolution. Finally, Section 2.6 concludes this chapter.

2.1 Evolutionary Continuous Global Optimisation and Non-separability

Optimisation is an important area in applied mathematics, the aim of which is to find the best solution to a problem. Optimisation is usually in the form of minimisation or maximisation of a mathematical function. This mathematical function can represent a problem in engineering, manufacturing, economy, or other areas. These functions are usually called the *objective (or cost) function*, and are expressed as $f(\vec{x})$, where $\vec{x} =< x_1, x_2, x_3, ..., x_n >$. The elements of the vector $\vec{x}$ are called *parameters* or *variables*, and n is the dimension of the function. More formally, global optimisation is defined as the process of finding $\vec{x}_{min}$ such that $\forall \vec{x} \in \mathbb{D}, f(\vec{x}_{min}) < f(\vec{x})$, where $\mathbb{D}$ is the domain of the function $f : \mathbb{D} \rightarrow \mathbb{R}$ which is bounded by $\mathbb{R}^n$. In this context there is no distinction between minimisation and maximisation. That is because any maximisation problem could be changed to a minimisation problem by optimising the $-f$ instead of function f.

As it was mentioned earlier, non-separable (or indecomposable) functions are those functions where there are correlations between a subset of the variables. According to Salomon (1995), a function is separable if:

$$\forall i, j \in \{1, 2, ..., n\} | f(..., x_i^o, ...) = O_i \wedge f(..., x_j^o, ...) = O_j \Rightarrow f(..., x_i^o, ..., x_j^o, ...) = O_{i,j} \quad (1)$$

and non-separable otherwise. In Equation (1), x_i^o is the optimal value of the i^{th} dimension and O_i is the corresponding function value when the value of all the other variables are kept constant. This means that the optimal value of a dimension is totally independent of the values of the variables in other dimensions. As an example, all the functions with the following general form are decomposable:

$$f(\vec{x}) = \sum_{i=1}^{n} f_i(x_i) \quad (2)$$

Accordingly, the function $f(\vec{x}) = x_1^2 + \lambda_1 x_2^2$ is an example of a decomposable function, and $f(\vec{x}) = x_1^2 + \lambda_0 x_1 x_2 + \lambda_1 x_2^2$ is an example of an indecomposable (non-separable) functions (Salomon, 1995), where λ_0 and λ_1 are arbitrary non-zero constants.

Evolutionary Algorithms (EAs) (Bäck, 1996; Bäck et al., 1997) could be used for function optimisation. EAs are based on the Darwin's theory of evolution which is centred around the idea of survival of the fittest. Most of EAs maintain a population of individuals, and each individual is a potential solution to the problem. In mathematical terms, an individual can be denoted as the vector $\vec{x}$. The main idea of EA is to evolve the population in a systematic way in order to create new individuals

which are fitter than the previous individuals. The fitness of every individual is calculated by the objective function f. The main motivation for using EAs for function optimisation is because of their ability to escape from the local minima and converge to the global solution, which is often hard to achieve using traditional mathematical optimisation techniques (Goldberg, 1989). Many of the traditional mathematical optimisation techniques such as Conjugate Gradient (Hestenes and Stiefel, 1952), and Quasi-Newton (Nocedal and Wright, 2006) rely on calculation of the derivative of the objective function for the optimisation process. In contrast, EAs do not rely on the derivative and this property makes them suitable for both differentiable and non-differentiable objective functions.

2.2 Cooperative Co-evolution

Divide-and-conquer is an effective technique in solving complex problems. Cooperative Co-evolution (Potter and De Jong, 1994) is such a technique that can be used to decompose a complex problem into several simpler sub-problems.

Potter and De Jong (1994) made the first attempt to incorporate CC into Genetic Algorithm (Goldberg, 1989) for function optimisation. The success of CC attracted many researchers to incorporate CC into other evolutionary techniques such as Evolutionary Programming (Liu et al., 2001), Evolutionary Strategies (Sofge et al., 2002), Particle Swarm Optimisation (van den Bergh and Engelbrecht, 2004; Li and Yao, 2009), and Differential Evolution (Shi et al., 2005; Yang et al., 2008a).

The original CC decomposes a n-dimensional decision vector into n subcomponents and optimises each of the subcomponents using GA in a round robin fashion. This algorithm was called CCGA. However, CCGA was only evaluated on functions with up to 30 dimensions.

The first attempt for applying CC to large scale optimisation was made by Liu et al. (2001) using Fast Evolutionary Programming with Cooperative Co-evolution (FEPCC) where they tackled problems with up to 1000 dimensions, but it converged prematurely for one of the non-separable functions, confirming that Potter and De Jong (1994) decomposition strategy is ineffective in dealing with variable interaction for high dimensional problems.

van den Bergh and Engelbrecht (2004) were first to apply CC to PSO. They developed two dialects of Cooperative PSO (CPSO) based on the original Potter's framework, but unlike the original CCGA (Potter and De Jong, 1994) they divided a n-dimensional problem into m s-dimensional subcomponents as depicted in Figure 1.

CPSO is based on the following CC decomposition strategy (van den Bergh and Engelbrecht, 2004):

1. Divide the variables of the objective function into m s-dimensional subcomponents.

2. Optimise each of the *m* subcomponents in a round-robin fashion with a certain EA. Each individual is combined with the global best individuals from other subcomponents to form a n-dimensional vector for evaluation by the objective function. This is where the cooperation happens.

3. Stop the evolutionary process once the halting criteria is satisfied or the maximum number of evaluations is exceeded.

Since an individual in a subcomponent only represents a partial solution, it is not possible to evaluate it directly using the objective function. An individual must be evaluated in conjunction with individuals in other subcomponents in order to form a complete solution. It is only within this context that the 'cooperation' happens. A suitable choice of a *context vector* is the vector of all the global best

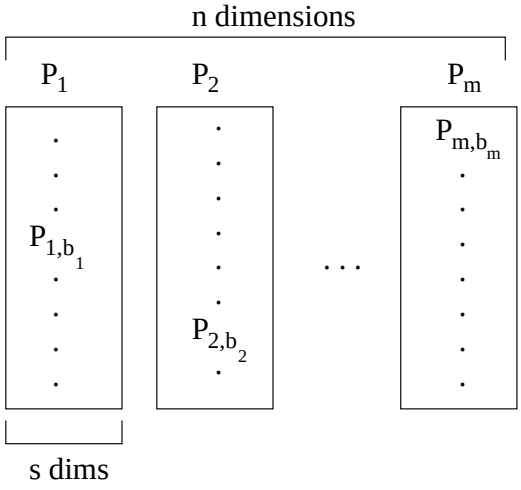


Figure 1: The n-dimensional objective vector is divided into m s-dimensional subcomponents. $P_{m,b}$ denotes the best individual in m^{th} subcomponent.

individuals in every subcomponent within which an individual is evaluated (van den Bergh and Engelbrecht, 2004). In mathematical terms a context vector could be defined as, $< P_{1,b_1}, P_{2,b_2}, ..., P_{m,b_m} >$, where b_i is the index of the best individual in the i^{th} subcomponent, and P_{i,b_i} refers to the best individual in the i^{th} subcomponent. Now in order to evaluate $P_{j,i}$ the i^{th} individual in the j^{th} subcomponent, P_{j,b_j} in the context vector should be replaced by $P_{j,i}$. So the trial vector will be $< P_{1,b_1}, P_{2,b_2}, ..., P_{j,i}, ..., P_{m,b_m} >$. This is also depicted in Figure 1.

One major drawback of CPSO is that its performance degrades rapidly as the dimensionality of the problem increases, mainly due to exponential growth in the size of the search space. This problem is magnified when applied to non-separable problems due to parameter interactions.

CC has also been applied to Differential Evolution in Yang et al. (2008a) and Shi et al. (2005). Shi et al. (2005), used splitting-in-half strategy where they divide the variables into two equally sized subcomponents each of which is optimised using a separate DE. However, their decomposition strategy does not scale up efficiently as the number of dimensions increases. Yang et al. (2008a) made the first attempt to develop a more effective way of dealing with variable interactions by random grouping of decision variables into different subcomponents. They have shown that random grouping of variables will increase the probability of grouping interacting variables in one subcomponent. *Random grouping* is further explained in Section 2.4.

Ray and Yao (2009) proposed a Cooperative Co-evolutionary Algorithm using Correlation based Adaptive Variable Partitioning (CCEA-AVP) technique in which they optimised all of the decision variables in one subcomponent for 5 generations before calculating the correlation coefficients of the top 50% of the individuals in the population. The variables with correlation coefficient value greater than a predetermined threshold are grouped in one subcomponent and the rest in another subcomponent. It has been shown that this technique performs better than traditional CCEAs on many non-separable benchmark functions. CCEA-AVP relies on calculation of correlation matrix for grouping interacting variables. A major disadvantage of CCEA-AVP is that it relies on a splitting-in-half strategy for decomposition of decision variables which has limited scalability to higher dimensions. This is similar to the scalability problem of the algorithm proposed by Shi et al. (2005). The other issue of CCEA-AVP is that it remains unclear as to what type of correlation coefficient has been used

in CCEA-AVP. To the best of our knowledge most statistical correlation coefficients such as Pearson's product-moment coefficient are used to measure the linear dependence between any two variables and it is independent of the slope of the line that they form. In other words two variables might be highly linearly correlated to each other yet they might be completely separable, i.e., correlation coefficients is not a proper measure for separability of variables in the context of large scale non-separable optimisation.

2.3 Differential Evolution

Differential Evolution (DE) (Storn and Price, 1995) is a relatively new EA for global optimisation. Despite its simplicity it has shown to be very effective on a set of benchmark functions (Vesterstrom and Thomsen, 2004). DE is a population based EA that works on a population of size $N_{popsize}$ as shown in Equation (3).

$$\vec{x}_{i,G}, i = \{1, 2, ..., N_{popsize}\}, \tag{3}$$

where i is the individual's index, and G is the current generation.

The main operation in DE is mutation, which is performed by creating a vector using the Equation (4) which is called the donor vector.

$$\vec{v}_{i,G} = \vec{x}_{r1,G} + F(\vec{x}_{r2,G} - \vec{x}_{r3,G}), \tag{4}$$

where $r1, r2, r3$ are mutually different integers that are randomly chosen from range $[1, 2, ..., N_{popsize}]$, and F is real valued constant factor that controls the amplification of the differential variation (Storn and Price, 1995). This mutation strategy is called DE/rand/1/bin (Storn, 1996) which is the classic mutation strategy proposed by Storn and Price (1995).

To get the target vector $\vec{x}'_{i,G}$, the crossover is performed by combining the donor vector (4) and the vector $\vec{x}_{i,G} = [x_{1,i,G}, x_{2,i,G}, ..., x_{D,i,G}]$ using Equation (5).

$$x'_{j,i,G} = \begin{cases} v_{j,i,G}, \text{ if } rand_j(0,1) \leq CR \text{ or } j = j_{rand} \\ \\ x_{j,i,G}, \text{otherwise} \end{cases} \tag{5}$$

where $j \in [1, D]$, and $rand_j(0, 1)$ is a uniform random number generator with values between zero and one. CR is the crossover rate which is in the range $[0, 1]$, and j_{rand} is also a uniform random number generator with values in range $[1, D]$.

Finally, the selection is done by Equation (6).

$$\vec{x}_{i,G+1} = \begin{cases} \vec{x}'_{i,G}, \text{if } \Phi(\vec{x}'_{i,G}) < \Phi(\vec{x}_{i,G}) \\ \\ \vec{x}_{i,G}, \text{otherwise}, \end{cases} \tag{6}$$

where Φ is the fitness function.

DE (Storn and Price, 1995) is designed to be less greedy compared to other EA techniques such as PSO (Kennedy and Eberhart, 1995) which makes it a good choice for large scale optimisation. This is because of the fact that in large scale optimisation, and due to the vastness of the search space more exploration is needed in order to find the global optimum. More greedy techniques run the risk of finding a suboptimal solution.

DE's control parameters are problem dependent and hard to determine (Gamperle et al., 2002; Qin and Suganthan, 2005). Self-Adaptive Differential Evolution with Neighbourhood Search (SaNSDE)

(Yang et al., 2008c) self-adapts crossover rate CR, scaling factor F, and mutation strategy. It has been shown that SaNSDE performs significantly better than other similar algorithms.

2.4 Random Grouping and Adaptive Weighting

In the framework proposed by Yang et al. (2008a), following a CC approach, the problem is decomposed into m s-dimensional subcomponents, and each subcomponent is optimised using a certain EA. Their method differs from ordinary CC approaches in two major ways. First they use random grouping of decision variables in order to increase the probability of grouping two interacting variables in one subcomponent, and secondly, the co-adaptation of subcomponents which is done using a technique known as *adaptive weighting*.

The motivation for random grouping is that, in most of the real-world problems, only a proportion of variables are interdependent.If an optimisation algorithm manages to group highly dependent variables in one subcomponent there will be a better chance to further improve the performance of the algorithm. Since there is no prior information on interacting variables, Yang et al. (2008a) showed that by random grouping of the decision variables, one can increase the probability of grouping two interacting variables in one subcomponent for at least some predetermined number of cycles. They demonstrated that random grouping results in a probability of 96.62% in grouping two interacting variables together for at least two cycles in a CC setting with 10 subcomponents and a total of 1000 decision variables over 50 iterations. However, it remains unclear what the probability will be if more than 2 variables interacting with each other, which is more likely the case in most optimisation problems. In Chapter 3 we show how the probability of grouping interacting variables drops significantly when there are more than two interacting variables. We also show – given the same number of evaluations – how to increase the probability of grouping more than two variables together using a modified version of DECC-G.

In adaptive weighting, a numeric weight value is applied to each subcomponent. All of these weight values form a vector called *weight vector*. This weight vector is optimised using a separate optimiser for some predetermined number of FEs. The motivation behind adaptive weighting is to co-adapt interdependent subcomponents. It is obvious that optimising the weight vector is far simpler than the original problem because the dimensionality of the weight vector with only m variables is smaller than the original problem with $m \times s$ variables.

Yang et al. (2008a) outline the steps of weight vector co-adaptation as follows.

1. set $i = 1$ to start a new *cycle*.
2. Randomly split a n-dimensional objective vector into m s-dimensional vector. This essentially means that any variable has equal chance of being assigned to any of the subcomponents.
3. Optimise the i^{th} subcomponent with a certain EA for a predefined number of FEs.
4. If $i < m$ then $i++$, and go to Step 3.
5. Construct a weight vector and evolve it using a separate EA for the best, worst, and a random member of the current population.
6. Stop if the maximum number of FEs is reached or go to Step 2 for the next *cycle*. Note that a *cycle* in a co-evolutionary context is equivalent to an iteration in classic EAs.

This scheme that uses a cooperative co-evolutionary EA with weight co-adaptation was named EACC-G in (Yang et al., 2008a). Since SaNSDE (Yang et al., 2008c) is used as the subcomponent optimiser in Step 3 the algorithm is called DECC-G.

2.5 Multilevel Cooperative Co-evolution

One major problem with DECC-G is determining the size of subcomponents. This parameter is problem dependent and is hard to determine. Multilevel Cooperative Co-evolution (MLCC) (Yang et al., 2008b) is an extension of DECC-G that dynamically self-adapts the subcomponent sizes.

MLCC uses a more flexible way of choosing the subcomponent sizes by choosing a group size from a set of predefined group sizes, $\mathbb{S} = \{s_1, ..., s_t\}$. The selection probability is calculated based on the performance history of each of decomposers through the course of evolution which is calculated by Equation (7).

$$r_k = \frac{(v - v')}{|v|}, k = \{1, 2, ..., t\} \tag{7}$$

where v is the best fitness of the last cycle and v' is the best fitness of the current cycle.

MLCC uses Equation (8) for calculating the selection probability of decomposers at the beginning of each cycle.

$$p_i = \frac{e^{7r_i}}{\sum_{j=1}^{t} e^{7r_j}}, i = \{1, 2, ..., t\} \tag{8}$$

where e and 7 are empirical values, and r_i, r_j are the performance records of the corresponding group sizes from the set $\mathbb{S}$ which are calculated by Equation (7). These values are updated at the beginning of each cycle and a new group size is chosen from the set accordingly. As it can be seen from Equation (8), MLCC uses a sophisticated formula for calculation of probabilities to select a decomposer from the set $\mathbb{S}$. There is no strong theoretical justification about the use of this formula and also the choice of the two arbitrary constants makes it less generalisable to various types of problems.

In this thesis we propose a new technique for self-adaptation of the subcomponent sizes which uniformly chooses a random group size from the set $\mathbb{S}$ when there is no fitness improvement. A typical choice of $\mathbb{S}$ is $\{5, 10, 25, 50, 100\}$ which is also used by Yang et al. (2008b). This new techniques has shown substantial improvement over MLCC (Section 5.1.2).

2.6 Chapter Summary

In this chapter we reviewed the previous works on large scale evolutionary optimisation. We mentioned that the performance of these algorithms deteriorate as the dimensionality of the problem increases. This is magnified when a subset of the decision variables are interdependent. *Random grouping* has been proposed by Yang et al. (2008a) in order to capture variable interactions. Some implementations of random grouping such as MLCC (Yang et al., 2008b) self-adapts the subcomponent sizes through the course of evolution. In the next chapter we show that the performance of *random grouping* will also deteriorate as the number of interacting variables increases. Yang et al. (2008a) assumed that there are only two interacting variables in the decision vector and centred their analysis around this assumption. In next chapter we identify a major bottleneck in DECC-G and MLCC that significantly reduces the probability of grouping interacting variables. Finally in Section 3.3 we propose a simpler and more efficient alternative to the self-adaptation technique used in MLCC.

3 More Frequent Random Grouping

The focus of this chapter is on proposing two techniques for optimising the performance of DECC-G. We also propose an alternative to MLCC for self-adapting subcomponent sizes which is easier to implement and more intuitive, and yet more efficient. The organisation of the rest of this chapter is as follows. Section 3.1 extends the mathematical background of *random grouping* and shows that the performance of this technique drops rapidly as the number of interacting variables increases. Section 3.2 describes how removing the *adaptive weighting* will allow to save the computational cost for doing more frequent random grouping. Section 3.3 proposes a new technique for self-adaptation of the subcomponent sizes. Finally, Section 3.4 summarises the important points of the chapter.

3.1 Mathematical Analysis

In their paper Yang et al. (2008a) proved that random grouping of decision variables and grouping them together at the beginning of each cycle increases the probability of placing two interacting variables in the same subcomponent. They used 50 cycles and have shown that the probability of grouping two parameters together at least for two iterations would be 96.62% in a CC setting with 10 subcomponents making up a total of 1000 variables (Yang et al., 2008a). However, in the theoretical analysis of their algorithm, it is assumed that there are only two interactive variables in the decision vector, and there is no analysis about the behaviour of the algorithm when the number of interacting variables are greater than two.

In this thesis we further generalised their theorem to be applicable to any number of interacting variables. Equation (9) calculates the probability of grouping v interacting variables in the same subcomponent for at least k cycles.

Theorem 1. *Given N cycles, the probability of assigning v interacting variables $x_1, x_2, ..., x_v$ into one subcomponent for at least k cycles is:*

$$P(X \geq k) = \sum_{r=k}^{N} \binom{N}{r} \left(\frac{1}{m^{v-1}}\right)^{r} \left(1 - \frac{1}{m^{v-1}}\right)^{N-r} \tag{9}$$

where N is the number of cycles, v is the total number of interacting variables, m is the number of subcomponents, and the random variable X is the number of times that v interacting variables are grouped in one subcomponent. Since we are interested in the probability of grouping v interacting variables for at least k cycles, X takes on the values greater than or equal to k. k is also subject to the following condition $k \leq N$.

Proof 1. *A variable can be assigned to a subcomponent in m different ways, and since there are v interacting variables, the probability of assigning all of the interacting variables into one subcomponent is:*

$$p_{sub} = \underbrace{\frac{1}{m} \times ... \times \frac{1}{m}}_{v \text{ times}} = \frac{1}{m^v}$$

Since there are m different subcomponents, the probability of assigning all v variables to any of the subcomponents is:

$$p = m \times p_{sub} = \frac{m}{m^v} = \frac{1}{m^{v-1}}$$

There are a total of N independent random decompositions of the variables into m subcomponents, so using a binomial distribution the probability of assigning v interacting variables into one subcomponent for exactly r times is:

$$\begin{aligned} P(X=r) &= \binom{N}{r} p^r (1-p)^{N-r} \\ &= \binom{N}{r} \left(\frac{1}{m^{v-1}}\right)^r \left(1-\frac{1}{m^{v-1}}\right)^{N-r} \end{aligned}$$

Thus,

$$P(X \geq k) = \sum_{r=k}^{N} \binom{N}{r} \left(\frac{1}{m^{v-1}}\right)^r \left(1-\frac{1}{m^{v-1}}\right)^{N-r}$$

□

Given $n = 1000$, $m = 10$, $N = 50$ and $v = 4$, we have:

$$P(X \geq 1) = 1 - P(X = 0) = 1 - \left(1 - \frac{1}{10^3}\right)^{50} = 0.0488$$

which means that over 50 cycles, the probability of assigning 4 interacting variables into one subcomponent for at least one cycle is only 0.0488. As we can see, this probability is very small, and it will be even less if there are more interacting variables. Figure 2 shows how the probability of grouping interacting variables for at least one cycle drops significantly as the number of interacting variables increases. The solid line shows how the probability will change by v for 50 cycles and the dashed line shows how the probability will change when there are 10000 cycles instead. We can see from the figure that the probability of grouping 5 variables for at least once using 50 cycles is close to zero whereas the probability of grouping the same number of interacting variables for at least once will increase to approximately 60% using 10000 cycles. Later in this section we show how to increase the number of cycles without increasing the total number of fitness evaluations.

Figure 3 shows the effect of increasing the frequency of random grouping (N) for different number of interacting variables. It is evident that higher frequency of random grouping will increase the probability of grouping interacting variables, regardless of the number of them. So more frequent random grouping will not only results in higher probability in grouping interacting variables, but also increases the efficiency of the algorithms in dealing with more than two interacting variables. Our studies on DECC-G and MLCC revealed that the frequency of random grouping could be significantly increased without increasing the total number of fitness evaluations. In order to maximise the frequency of random grouping the subcomponent optimisers should run for only one iteration which is equivalent to $N_{popsize}$ number of fitness evaluations, where $N_{popsize}$ is the population size of the EA.

In their experiments, Yang et al. (2008a) used only 50 cycles with $5e6$ FEs while there is a potential for approximately 3800 cycles using the same number of FEs by running the subcomponent optimiser for only *one* iteration. Using only 50 cycles is counter-intuitive because according to the above discussion, a higher frequency of running random grouping will result in higher probability in grouping any two interacting variables in one subcomponent.

3.2 Removing Adaptive Weighting

Further investigation on DECC-G have shown that *adaptive weighting* is not effective, and the computational resource can be spent on using more frequent random grouping instead.

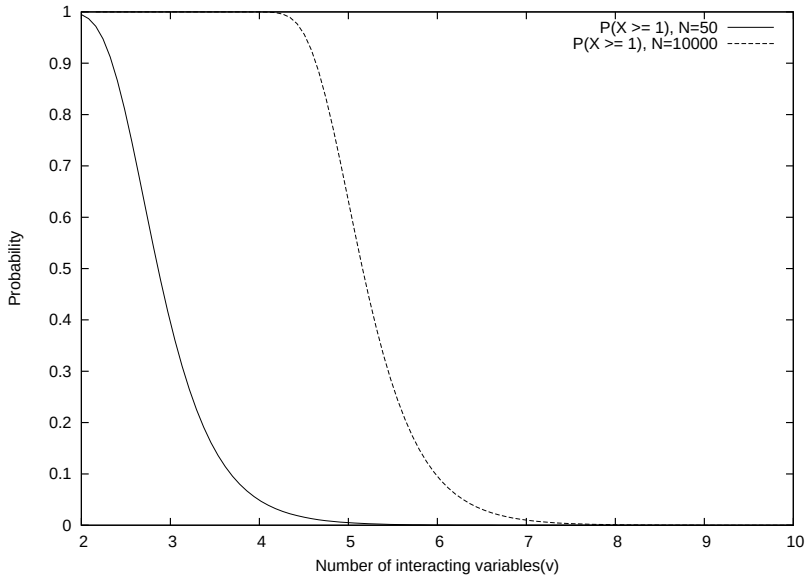


Figure 2: Increasing v, the number of interacting variables will significantly decrease the probability of grouping them in one subcomponent, given $n = 1000$ and $m = 10$.

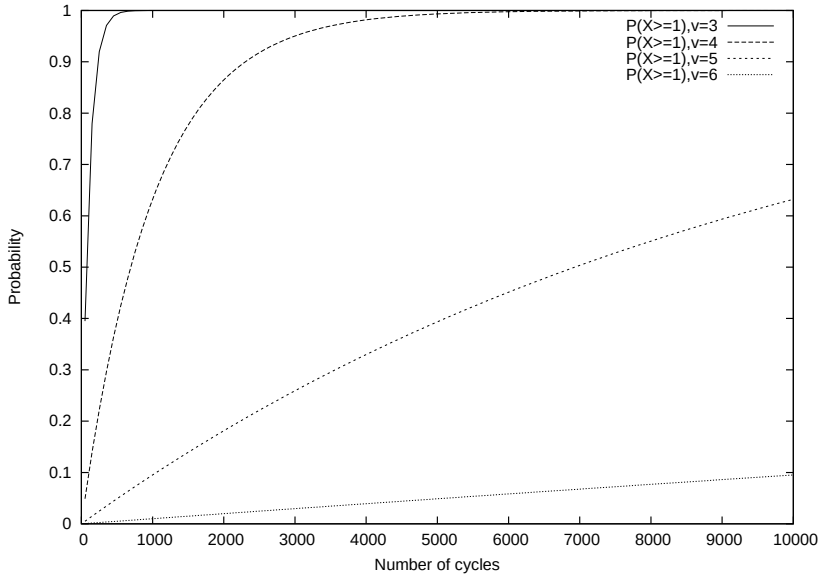


Figure 3: Increasing N, the number of cycle increases the probability of grouping v number of interacting variables in one subcomponent.

Although in theory it seems to be a good way of reducing the dimensionality of the original problem, in practice its role is insignificant in the whole optimisation process. In order to demonstrate this we counted the number of times that adaptive weighting feature manages to find a solution, and based on our experiments we realised that in most cases adaptive weighting fails to find a better solution. This essentially means that a significant number of fitness evaluations are wasted by adaptive weighting. In order to demonstrate this difference we modified DECC-G and disabled the adaptive weighting feature and conducted an experiment with the same settings as DECC-G with a total of 50 cycles. We called this new variant DECC-NW. We have also developed another variant of DECC-NW called DECC that performs more frequent random grouping by running the subcomponent optimiser for only one iteration. Section 5.1.2 summarises the results of running these two algorithms over the same benchmark functions.

3.3 Self-adaptation of subcomponent sizes

As we mentioned earlier, we also proposed a simpler technique for self-adapting the subcomponent sizes. In our method we preserved the same decomposer set $\mathbb{S}$ as it was used in MLCC, but instead of using the complicated formula (Equation (8)) for probability calculation in choosing a decomposer we simply use a uniform random number generator for selecting a decomposer from the set $\mathbb{S}$. The subcomponent size is only changed when there is no improvement in the fitness value between the previous and the current cycles. This new techniques has been integrated into DECC. We called this new algorithm DECC-ML. This is shown in lines 7-10 of Algorithm 1.

3.4 Chapter Summary

In Section 3.1 we mathematically proved that increasing the number of interacting variables will significantly reduces the probability of grouping interacting variables in one subcomponent using *random grouping*. In Section 3.2 we have shown how the frequency of *random grouping* could be increased without increasing the total number of fitness evaluations. Finally in section 3.3 we proposed a more efficient alternative to the self-adaptation technique used in MLCC (Yang et al., 2008b). In Chapter 5 we will show how the proposed techniques will significantly improve the performance of the new algorithm.

We have also shown that even more frequent random grouping will lose its efficiency as the number of interacting variables increases (Figure 3). In the next chapter we propose a new technique called *delta grouping* that uses a more intelligent and systematic approach for identification of interacting variables. The new algorithms has been compared with previous algorithms and the experimental results confirmed its improved performance. The detailed analysis of the results is provided in Chapter 5.

4 The Delta Grouping

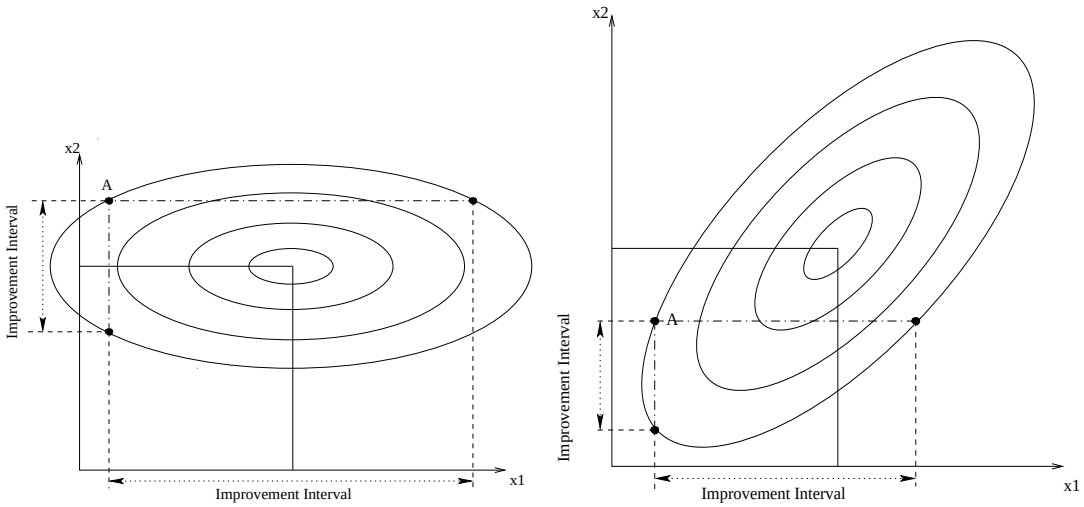
In chapter 2 we reviewed two evolutionary approaches for global optimisation of non-separable functions. *Random grouping* techniques such as DECC-G (Yang et al., 2008a), MLCC (Yang et al., 2008b), and DECC-ML (Chapter 3) rely on random shuffling of decision variables with the hope of grouping a subset of the interacting variables in a common subcomponent for a few cycles. Although it has been successful on a set of benchmark functions, we have shown that this blind approach does not scale properly when the number of interacting variables increases (Section 3.1). Correlation based techniques such as CCEA-AVP rely on calculation of correlation coefficient for identification of interacting variables. As it was mentioned in Section 2.2, CCEA-AVP divides the decision variables into two subcomponents based on the calculated correlation coefficients from the population. Splitting-in-half strategy does not scale properly as the number of dimensions increases. This is similar to scalability problem of the technique proposed by Shi et al. (2005). Another major issue of CCEA-AVP is the use of correlation coefficients as a measurement for degree of separability of variables. Statistical correlation coefficients such as Pearson's product-moment coefficient, only measures the *linear* dependency of the variables and does not give any information about the orientation of the points in relation to the coordinate system. In general, correlation coefficient is not a proper measurement for degree of separability (or non-separability) of variables in the context of large scale optimisation.

It can be envisaged that a more intelligent and systematic approach will do a better job than the rather blind random grouping or CCEA-AVP methods. In this chapter we propose a new technique called *delta grouping* which is capable of capturing interacting variables in a more intelligent and systematic way. The organisation of the rest of this chapter is as follows. Section 4.1 briefly introduces the *delta grouping* and the main intuition behind the technique. Section 4.2 describes the idea of improvement interval and explains the reasons behind the success of *delta grouping*. Section 4.3 describes how delta is implemented in a more formal language. Finally Section 4.4 summarises the important points about the *delta grouping*.

4.1 The Basic Model

As it has been shown by Salomon (1995) the improvement interval of a function shrinks significantly when it has high eccentricity and is not aligned with the coordinate axes (in other words, when the variables are interacting with each other). Improvement interval of a variable is the interval in which the fitness value could be improved while all the other variables are kept constant. This is further explained in Section 4.2. In non-separable functions, when two interacting variables are grouped in separate subcomponents, there would be a limit to the extent each variable can be improved towards its optimal value. We speculated that measuring the amount of change in each of the decision variables in every iteration can lead to identification of interacting variables. We called this amount of change *delta value* (see Section 4.3). As we mentioned earlier the improvement interval will shrink significantly on a non-separable function so we expect smaller delta values for those variables that have interaction amongst themselves. Based on this idea we proposed a new algorithm that sorts the decision variables based on the magnitude of their delta values and groups those variables with smaller delta values into one subcomponent. Experimental observations revealed that this new technique is very effective in capturing interacting variables compared to previous techniques.

As it was fully described earlier in Chapter 3 the probability of grouping interacting variables in one subcomponent will decrease rapidly as the number of interacting variables increases (Figure 2). Here we propose a new technique that is capable of grouping interacting variables in a more systematic way for more cycles than what is possible with random grouping.



(a) Quadratic Function (b) Rotated Quadratic Function

Figure 4: Coordinate rotation causes the improvement interval to shrink. Figures (b) is the rotated version of function in Figure (a). Point A on the level curves represent the same point before and after rotation. Note that in (b) the global optimum is outside of the improvement window, which makes it much harder for an algorithm to locate the global optimum.

4.2 Why Delta Grouping Works

The main idea behind *delta grouping* is based on a major property of most of non-separable problems. In cooperative co-evolution, the decision vector is decomposed into several smaller subcomponents and in order to optimise and evaluate the individuals within each of the subcomponents, the variables of other subcomponents must be kept fixed. In separable problems each variable can be optimised one at a time and since there is no interaction between the variables it is possible to get as close as possible to the optimal value for that specific variable. Whereas in a non-separable problem, variable interaction, imposes some limitation to the extent a variable can be optimised close to its optimal value. In other words, variable interaction reduces the improvement interval of variables especially when they are grouped in different subcomponents. Coordinate rotation is one way of turning a separable problems into a non-separable one (Salomon, 1995). Salomon used this technique to show how the performance of GA drops significantly when applied to rotated functions, especially those with high eccentricity (Salomon, 1995).

Figure 4 shows how coordinate rotation reduces the improvement interval of interacting variables. Figure 4(a) shows a quadratic function with its principal axis aligned with the coordinate system. As it can be seen the improvement window is relatively large and with some luck it is possible for the CCEA to find the optimal value of each variable before optimising the other variable. Figure 4(b) is identical to Figure 4(a) except that the function's fitness landscape is rotated. Point A on the level curve stays in the same location relative to the shape of the function, but as it is depicted in Figure 4(b) coordinate rotation causes the improvement interval of point A to shrink considerably. Moreover, when the interacting variables are grouped in different subcomponents, it becomes increasingly difficult to optimise the variables to their optimal value. Figure 4(b) clearly shows if one of the variables is kept fixed the other variable is confined to reaching only a suboptimal value.

We can use this information of small improvement intervals to identify possible interacting variables in a non-separable problem. When the improvement interval of a variable gets smaller, it creeps towards its optimal value in small successions. So whenever we observe such behaviour we can conclude that there might be another variable with a small improvement window. This is not necessarily true all the times, but using this simple heuristic we can significantly increase the probability of grouping interacting variables into one subcomponent. Another advantage of the new technique is its ability to adapt itself to the fitness landscape. It is clear that the degree of non-separability might change over time depending which area of the fitness landscape is being explored by the individuals in the population. This flexibility allows the algorithm to use more suitable decomposition strategy depending on the shape of fitness landscape.

4.3 How Delta Grouping Works

In the algorithm proposed in this chapter we calculate the amount of change in each of the dimensions between two consecutive cycles for all of the individuals in the population. We then construct the vector $\Delta = \{\bar{\delta}_1, \bar{\delta}_2, ..., \bar{\delta}_n\}$ where n is the number of dimensions and the elements of the vector are calculated using the following equation:

$$\bar{\delta}_i = \frac{\sum_{j=1}^{N_{popsize}} \delta_{i,j}}{N_{popsize}}, i \in \{1, 2, ..., n\} \tag{10}$$

where $N_{popsize}$ is the population size, and i is the index of the dimension, and j is the j^{th} individual in the population. This is a rough estimation of the improvement interval in every dimension which is used for capturing the interacting variables.

We can then sort the decision variables based on the magnitude of their corresponding delta values in descending order. Finally the sorted variables are divided into predetermined equally-sized subcomponents. Based on the idea of a small improvement interval, when a small delta value is observed, there is a high probability that another variable exists with a relatively small delta value that has interaction with the former variable. This algorithm is called Differential Evolution with Cooperative Co-evolution using Delta-Grouping (DECC-D) which is outlined below:

1. set $i = 1$ to start a new *cycle*.
2. Initialize the Δ vector to zero. This means that the delta is not used for arrangement of variables in the first cycle.
3. Divide the decision variables into predefined subcomponents. In the first cycle Δ vector is initialised to zero so the variables will preserve their original order.
4. Optimise the i^{th} subcomponent with a certain EA for one only one iteration. Note that this differs from DECC-G (Yang et al., 2008a) where the subcomponent optimisers run for more than one iteration.
5. If $i < m$ then $i++$, and go to Step 4.
6. Construct the Δ vector using Equation (10).
7. Sort the decision variables based on the magnitude of their corresponding delta values.
8. Stop if the maximum number of FEs is reached or go to Step 3 for the next *cycle*. Note that a *cycle* in a co-evolutionary context is equivalent to an iteration in classic EAs.

Algorithm 1: DECC-DML($cycles, FEs$)

1: $pop[1 : N_{popsize}, 1 : n] \leftarrow$ *random population*
2: $(best, best\_val) \leftarrow min(evaluate(pop))$
3: $oldpop \leftarrow pop$
4: $prev\_best\_val \leftarrow best\_val$
5: $\mathbb{S} \leftarrow \{50, 100, 200, 250\}$
6: **for** $i \leftarrow 1$ to $cycles$ **do**
7: **if** $best\_val = prev\_best\_val$ **then**
8: $decomposer\_index \leftarrow rand_U(0, size(\mathbb{S}))$
9: $s \leftarrow \mathbb{S}[decomposer\_index]$
10: **end if**
11: $\Delta \leftarrow mean(|oldpop - pop|)$
12: $(\Delta, indicies) \leftarrow sort(\Delta)$
13: $oldpop \leftarrow pop$
14: **for** $j \leftarrow 1$ to $\frac{n}{s}$ **do**
15: $l \leftarrow (j - 1) \times s + 1$
16: $u \leftarrow j \times s$
17: $subpop \leftarrow pop[:, indicies[l : u]]$
18: $subpop \leftarrow SaNSDE(best, subpop, FEs)$
19: $pop[:, indicies[:, l : u]] \leftarrow subpop$
20: $(best, best\_val) \leftarrow min(evaluate(pop))$
21: **end for**
22: **end for**

We also developed a variant of DECC-D called DECC-DML (See Algorithm 1) that self-adapts the subcomponent sizes using the techniques used in DECC-ML (Omidvar et al., 2010). DECC-DML differs from DECC-D in the way that it decomposes the decision vector. Instead of using a fixed subcomponent size in step 3, DECC-DML checks the fitness of the best individual and if there has been no improvement it will choose a different decomposer from a set of predetermined decomposers is called $\mathbb{S}$. For this research we used the following decomposers, $\mathbb{S} = \{50, 100, 200, 250\}$.

4.4 Chapter Summary

In this chapter we proposed a new technique called *delta grouping*. Delta values measure the average amount of change in one variable across the whole population. According to Salomon (1995) the improvement interval shrinks due to variable interaction between a subset of decision variables. Based on this intuition we hypothesised that the delta values should also shrink in size and there is high probability for two variables with small delta values to interact with each other. So in *delta grouping* the variables are grouped based on the magnitude of their delta values in the previous evolutionary step. In the next chapter we provide empirical evidence about the ability of the new technique in capturing variable interactions. In general, the first part of the next chapter is dedicated to the analysis of the ideas presented in Chapter 3 about more frequent random grouping, and the second part of the chapter focuses on the analysis of the delta grouping presented in the current chapter.

5 Results and Analysis

In this chapter we present the experimental results and the analysis of the algorithms that we proposed in Chapters 3, 4. For our experiments we mainly relied on two major benchmark functions which are widely used by the research community for global optimisation. The first test suite is the IEEE CEC'2008 (Tang et al., 2007) benchmark functions that consists of seven functions[1]. The other test suite that we used is the IEEE CEC'2010 (Tang et al., 2009) benchmark functions which have been proposed specifically for evaluating the algorithms designed for handling large scale global optimisation problems[2]. In Section 5.1 we first demonstrate and analyse the empirical results on DECC-ML, and in Section 5.2 we present the experimental results on DECC-DML and compare its performance with other major algorithms. For better clarity all the algorithms that are used for experimental analysis are summarised in Table 1.

Algorithm	**Description**
DECC-G (Yang et al., 2008a)	Cooperative Co-evolutionary DE with random grouping and adaptive weighting.
DECC-NW (Section 3.2)	The same as DECC-G with the adaptive weighting feature disabled.
DECC (Section 3.2)	Cooperative Co-evolutionary DE without adaptive weighting and running the subcomponent optimisers for only one iteration.
DECC-ML (Section 3.3)	The same as DECC, but uses a uniform selection for self-adapting the subcomponent sizes.
MLCC (Yang et al., 2008b)	The same as DECC-G, but it self-adapts the subcomponent sizes using historical performance of different decomposers.
DECC-D (Section 4.3)	This algorithm uses the delta grouping for capturing the interacting variables, however it relies on fixed subcomponent sizes through evolution.
DECC-DML (Section 4.3)	The same as DECC-D, but it self-adapts the subcomponent sizes using uniform selection of a decomposer from a predefined set.

Table 1: Summary of algorithms described in this thesis. Those without citations are proposed in this thesis.

5.1 Empirical Results on DECC-ML (More Frequent Random Grouping)

In this section we briefly describe the experiment setup (Section 5.1.1), then the experimental results are provided, and finally results are analysed and compared to other algorithms (Section 5.1.2).

5.1.1 Experiment Setup

We evaluated the DECC-ML algorithm on the CEC'08 test suite which was proposed in CEC'2008 Special Session and Competition on Large Scale Global Optimisation (Tang et al., 2007). This test

[1]The Java™and MATLAB® source code of these benchmark functions can be downloaded from `http://nical.ustc.edu.cn/cec08ss.php`

[2]The Java™and MATLAB® source code of these benchmark functions can be downloaded from `http://nical.ustc.edu.cn/cec10ss.php`

Function	**Description**	**Modality**	**Separability**
f_1	Shifted Sphere Function	Unimodal	Separable
f_2	Shifted Schwefel's Problem 2.21	Unimodal	Non-separable
f_3	Shifted Rosenbrock's Function	Multimodal	Non-separable
f_4	Shifted Rastrigin's Function	Multimodal	Separable
f_5	Shifted Griewank's Function	Multimodal	Non-separable
f_6	Shifted Ackley's Function	Multimodal	Separable
f_7	FastFractal "DoubleDip" Function	Multimodal	Non-separable

Table 2: Summary of the 7 CEC'08 Test Functions

Parameter	**Value**
Dimensions	$D = \{100, 500, 1000\}$
FEs	$5000 \times D = \{5e+5, 2.5e+6, 5e+6\}$
Population size	$N_{popsize} = 50$
Subpopulation sizes	$\mathbb{S} = \{5, 10, 25, 50, 100\}$
Number of runs	25

Table 3: Experimental Setup Parameters

suite consists of seven functions which are summarised in Table 2. The details of this function set is provided in Appendix A (Section 7.1).

We ran each algorithm for 25 independent runs on 100, 500, and 1000 dimensions and the mean and standard deviation of the best fitness values over 25 runs was recorded. The population size was set to 50, and the maximum number of fitness evaluations (FEs) was calculated by the following formula, $FEs = 5000 \times D$, where D is the number of dimensions. This is based on the guidelines for CEC'2008 Competition on Large Scale Global Optimisation (Tang et al., 2007). For the subpopulation sizes we used the same set as it was used in Yang et al. (2008b), $\mathbb{S} = \{5, 10, 25, 50, 100\}$. The experimental setup is summarised in Table 3

5.1.2 Analysis of Results

The result of 25 independent runs for DECC-ML is listed in the Table 4 for 1000 dimensions. This table show the progress of the algorithm and is based on the provided template for CEC'08 competition for Large Scale Global Optimisation (Tang et al., 2007). The summary Tables 5, 6, and 7 show the comparison for all of the five algorithms on 100, 500, 1000 dimensions respectively.

According to Table 4, DECC-ML shows faster convergence than MLCC on 6 out of 7 functions. A deeper analysis of the 25 runs revealed that existence of two outliers in the results of DECC-ML increased the mean significantly on f_5. Running Wilcoxon rank-sum test using 99% confidence interval has shown that DECC-ML is also significantly better than MLCC on f_5. The p-values of Wilcoxon rank-sum test is recorded in Table 7.

The global optimum point for f_7 is unknown, but from the Table 4 it is clear that the algorithm has a steady progress in the improvement of the fitness value. This shows the ability of DECC-ML to converge even on a difficult function such as f_7 which is highly multi-modal and has a rugged surface. It is noteworthy that DECC-ML found the absolute global point in all 25 runs for f_4 and in at least 19 out of 25 runs for f_1.

	f_1	f_2	f_3	f_4	f_5	f_6	f_7
At $5e+4$ Fitness Evaluations							
1^{st}	1.7161e+05	1.1887e+02	6.6224e+09	4.0066e+03	9.9964e+02	1.2131e+01	-1.1674e+04
7^{th}	4.1466e+05	1.3322e+02	3.4286e+10	5.4659e+03	2.9989e+03	1.4462e+01	-1.1328e+04
13^{th}	4.7111e+05	1.3836e+02	4.0710e+10	5.6821e+03	3.1484e+03	1.6667e+01	-1.0720e+04
19^{th}	4.8698e+05	1.6499e+02	9.6494e+10	7.1419e+03	4.1706e+03	1.6810e+01	-9.7556e+03
25^{th}	5.1115e+05	1.8057e+02	1.0925e+11	8.3134e+03	4.5463e+03	1.7210e+01	-9.6350e+03
M	4.3797e+05	1.4607e+02	5.9240e+10	6.1375e+03	3.3631e+03	1.5605e+01	-1.0589e+04
Std	7.7341e+04	1.8315e+01	3.5704e+10	1.3938e+03	9.4624e+02	1.7201e+00	7.7443e+02
At $5e+5$ Fitness Evaluations							
1^{st}	9.1964e-01	3.0613e+01	1.0463e+04	3.0303e+02	2.1304e-01	1.5612e-01	-1.3881e+04
7^{th}	2.2217e+01	3.5070e+01	3.2238e+04	3.5180e+02	1.1769e+00	3.9539e-01	-1.3812e+04
13^{th}	3.0578e+01	5.5145e+01	4.4875e+04	8.6533e+02	1.6448e+00	4.1639e-01	-1.3445e+04
19^{th}	3.2747e+01	9.6366e+01	1.2383e+05	8.9242e+02	2.1568e+00	7.7794e-01	-1.2909e+04
25^{th}	1.3197e+02	1.2751e+02	1.4550e+05	9.1424e+02	2.2286e+00	9.7411e-01	-1.2785e+04
M	4.1338e+01	6.4509e+01	6.9043e+04	6.8650e+02	1.5547e+00	5.2513e-01	-1.3362e+04
Std	4.0586e+01	3.2558e+01	4.7066e+04	2.6892e+02	6.0200e-01	2.8627e-01	4.1907e+02
At $5e+6$ Fitness Evaluations							
1^{st}	0.0000e+00	6.9941e-02	9.2453e+02	0.0000e+00	1.3323e-15	2.0606e-13	-1.4780e+04
7^{th}	0.0000e+00	9.0952e-01	9.8941e+02	0.0000e+00	1.4433e-15	2.4158e-13	-1.4771e+04
13^{th}	0.0000e+00	2.0454e+00	1.0675e+03	0.0000e+00	1.4433e-15	2.5224e-13	-1.4758e+04
19^{th}	0.0000e+00	2.8289e+00	1.1714e+03	0.0000e+00	1.5543e-15	2.7711e-13	-1.4743e+04
25^{th}	1.2937e-26	1.5571e+01	1.4235e+03	0.0000e+00	1.7226e-02	2.8777e-13	-1.4731e+04
M	5.1750e-28	3.4272e+00	1.0990e+03	0.0000e+00	9.8489e-04	2.5295e-13	-1.4757e+04
Std	2.5875e-27	4.4636e+00	1.3217e+02	0.0000e+00	3.6923e-03	2.3677e-14	1.4880e+01

Table 4: Experimental results of 25 independent runs from different stages of evolution ($\frac{FEs}{100}$, $\frac{FEs}{10}$, FEs).

By comparing DECC and DECC-G we can see that DECC found a better solution using same number of fitness evaluations in 6 out of 7 benchmark functions which shows that more frequent random grouping of variables yields better performance. This trend is almost the same for 100, 500, and 1000 dimensions.

The summary Tables 5, 6, and 7 also confirm our speculation about high failure rate of adaptive weighting feature in DECC-G. We can see from these Tables that a significant number of fitness evaluations cab be saved by removing adaptive weighting to be later used for more frequent random grouping. We can conclude that using the saved fitness evaluations for increasing the frequency of random grouping will improve the performance of the algorithm even further. We incorporated both of these changes into DECC-G,and created another version called DECC. Experimental results shows that DECC outperforms DECC-G over 6 out of 7 benchmark functions.

DECC-NW and DECC-G are identical except that in DECC-NW the adaptive weighting feature is disabled. We tested both algorithms for 50 cycles and equal number of fitness evaluations. By looking at the Tables 5, 6, and 7 we can observe that DECC-NW converged faster than DECC-G in 6 out of the 7 functions, confirming that *adaptive weighting* is not very effective.

Tables 5, 6, 7 show that DECC-ML substantially outperforms MLCC in 6 out of 7 test functions. The same trend exists on all tested dimensions which shows a better scalability of DECC-ML. Another observation is DECC-ML's faster convergence behaviour than MLCC for most of the test functions. This behaviour is especially clear from Figures 5(a), 7(b), 6(b). For functions f_1, f_5, f_6 a solution is found with almost the same quality with only half of the total FEs. This is a very valuable property specially in real-world problems where evaluation of the fitness function is very costly. DECC-ML also shows a quicker convergence over the rest of the test functions, but the trend is less significant compared to those of f_1, f_5, f_6. From the non-separability point of view, DECC-ML performed better

Function	DECC	DECC-G	DECC-ML	DECC-NW	MLCC
f_1	**2.7263e-29**	1.1903e-08	5.7254e-28	8.9824e-28	6.8212e-14
f_2	5.4471e+01	6.0946e+01	**2.7974e-04**	5.9949e+01	2.5262e+01
f_3	1.4244e+02	4.6272e+02	1.8871e+02	**1.2959e+02**	1.4984e+02
f_4	5.3370e+01	1.1783e+02	**0**	4.5768e+00	4.3883e-13
f_5	2.7589e-03	3.7328e-03	3.6415e-03	7.3233e-03	**3.4106e-14**
f_6	2.3646e-01	1.0365e+00	**3.3822e-14**	1.2224e+00	1.1141e-13
f_7	-9.9413e+02	-1.2666e+03	**-1.5476e+03**	-1.3967e+03	-1.5439e+03

Table 5: Results of different algorithms over 100 dimensions (Average over 25 runs). Best results are highlighted in bold.

Function	DECC	DECC-G	DECC-ML	DECC-NW	MLCC
f_1	**8.0779e-30**	1.0326e-25	1.6688e-27	2.4479e-27	4.2974e-13
f_2	4.0904e+01	7.6080e+01	**1.3396e+00**	7.2776e+01	6.6663e+01
f_3	6.6822e+02	1.4295e+03	**5.9341e+02**	1.2499e+03	9.2466e+02
f_4	1.3114e+02	5.6116e+00	**0**	5.2932e+00	1.7933e-11
f_5	2.9584e-04	4.1332e-03	1.4788e-03	2.7584e-02	**2.1259e-13**
f_6	**6.6507e-14**	1.8778e+00	1.2818e-13	1.0634e+00	5.3433e-13
f_7	-5.5707e+03	-6.0972e+03	**-7.4582e+03**	-6.9403e+03	-7.4350e+03

Table 6: Results of different algorithms over 500 dimensions (Average over 25 runs). Best results are highlighted in bold.

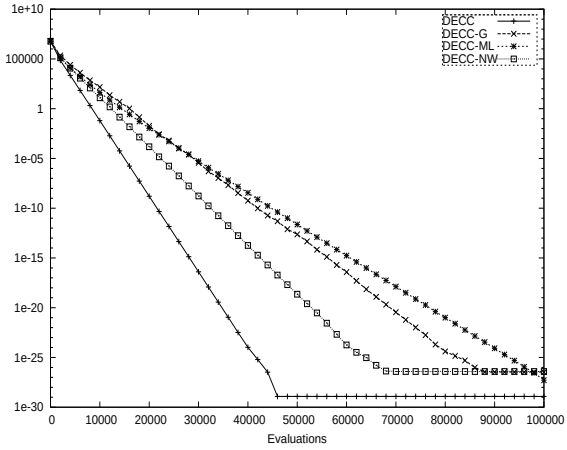
than MLCC on 2 out of 4 non-separable functions when the dimension was set to 100. It is interesting that the relative performance of DECC-ML increases when the number of dimensions increases. For example when the dimension is set to 500 and 1000, DECC-ML outperforms MLCC on 3 out of 4 non-separable functions. This clearly shows the advantage of using DECC-ML on large scale non-separable problems.

5.2 Empirical Results of the Delta Grouping

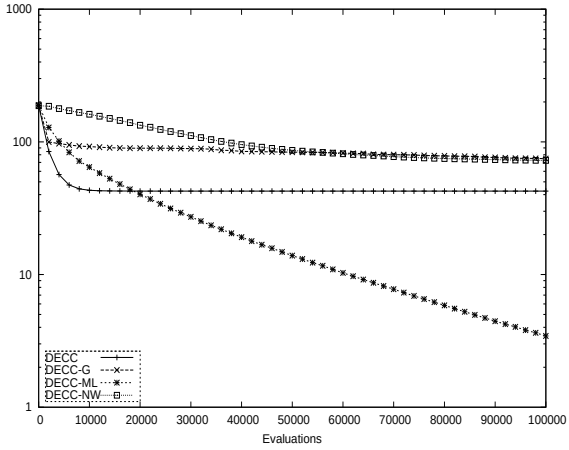
We first evaluated both DECC-D and DECC-DML on CEC'2008 benchmark functions, the detailed description of which can be found in (Tang et al., 2007). In order to further validate the results we also tested both DECC-D and DECC-DML on the new CEC'2010 benchmark functions (Tang et al., 2009). The results of experiments on both set of benchmark functions are provided in Sections 5.2.1 and 5.2.2 respectively. We ran each algorithm for 25 independent runs for every function and the mean and standard deviations were recorded. The population size is set to 50 for all of the experiments.

5.2.1 Results of Delta Grouping on CEC'2008 Benchmark Functions

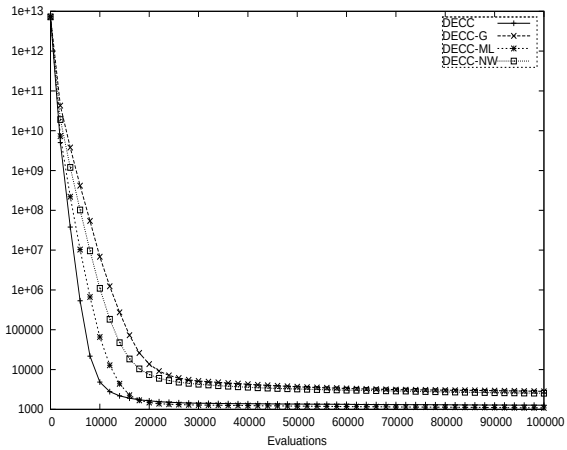
We tested our proposed algorithms with CEC'2008 function on 100, 500, and 1000 dimensions and the mean of the best fitness value over 25 runs was recorded. The performance of DECC-D and DECC-DML is compared with other algorithms in Tables 8, 9, 10 on 100, 500, and 1000 dimensions respectively. The maximum number of fitness evaluations (FEs) was calculated by the following



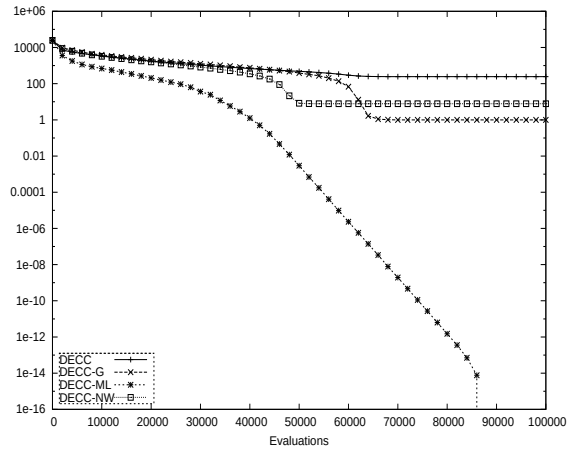
(a) f_1



(b) f_2

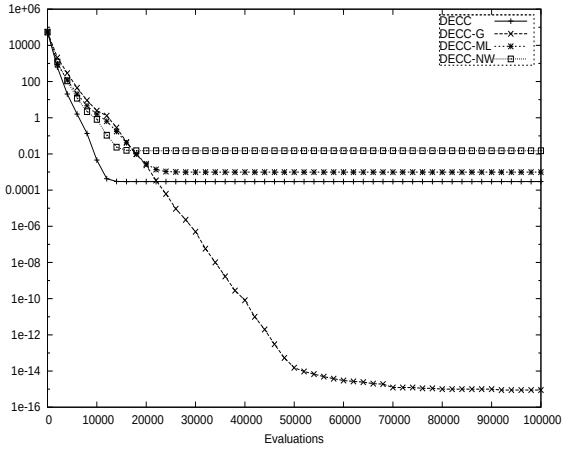


(c) f_3

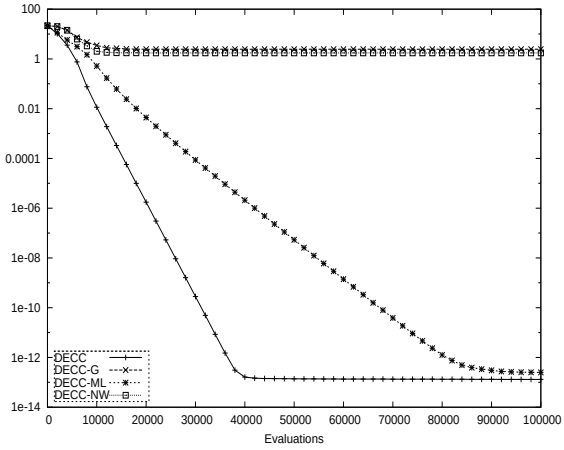


(d) f_4

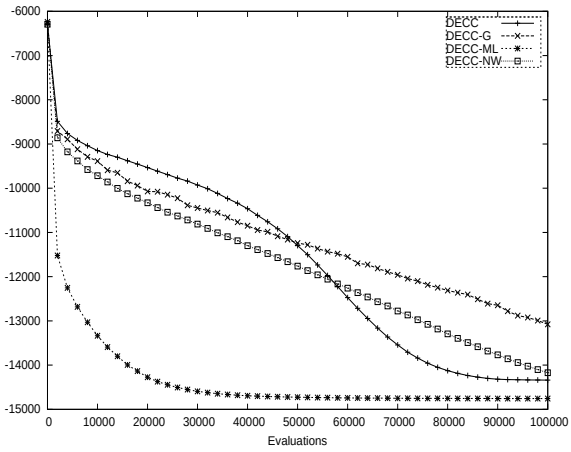
Figure 5: Convergence plots for $f_1 - f_4$ with 1000 dimensions



(a) f_5



(b) f_6



(c) f_7

Figure 6: Convergence plots for $f_5 - f_7$ with 1000 dimensions

Function	**DECC**	**DECC-G**	**DECC-ML**
f_1	**1.2117e-29**	3.8745e-27	5.1750e-28
f_2	4.2729e+01	7.4234e+01	**3.4272e+00**
f_3	1.2673e+03	2.6306e+03	**1.0990e+03**
f_4	2.4498e+02	1.0666e+01	**0**
f_5	2.9584e-04	7.8435e-03	9.8489e-04
f_6	**1.3117e-13**	2.3475e+00	2.5295e-13
f_7	-1.4339e+04	-1.3015e+04	**-1.4757e+04**
Function	**DECC-NW**	**MLCC**	**DECC-ML,MLCC MWW rank-sum test**
f_1	3.8145e-27	8.4583e-13	2.851e-06
f_2	7.2561e+01	1.0871e+02	6.535e-06
f_3	2.5411e+03	1.7986e+03	6.535e-06
f_4	7.7209e+00	1.3744e-10	6.482e-06
f_5	1.4976e-02	**4.1837e-13**	1.161e-03
f_6	1.7613e+00	1.0607e-12	6.506e-06
f_7	-1.41679e+04	-1.4703e+04	6.530e-06

Table 7: Results of different algorithms over 1000 dimensions (Average over 25 runs). Best results are highlighted in bold.

formula, $FEs = 5000 \times D$, where D is the number of dimensions. In DECC-D the subcomponent size is set to 50, and for DECC-DML the following set of decomposers are used, $\mathbb{S} = \{50, 100, 200, 250\}$.

As it can be seen from Tables 8, and 9, DECC-DML outperforms MLCC on 6 out of 7 functions with 100, and 500 dimensions, and on 1000 dimensions it outperforms MLCC on all the 7 functions (Table 10). It is interesting to see that DECC-ML which completely relies on more frequent random grouping has the best performance, but it should be noted that several of CEC'2008 benchmark functions are separable and for those that are non-separable the degree of non-separability is unknown. By a closer look at Table 10 we can see that despite the better performance of DECC-ML the final mean fitness value is indeed very close to what is achieved by DECC-DML. Since the aim of this research is to demonstrate the delta grouping as a new technique for grouping interacting variables, we leave further analysis of DECC-ML to a future study.

5.2.2 Results of Delta Grouping on CEC'2010 Benchmark Functions

In Table 12 the best, worst, median, mean, and standard deviation are recorded. These information are recorded at different stages of evolution to demonstrate the convergence behaviour of the algorithm. Table 12 is based on the template provided for CEC'2010 Competition on Large-Scale Optimisation (Tang et al., 2009). Figures 7, 8 show the convergence plot of DECC-D and DECC-DML for all the variants of Rastrigin and Rosenbrock functions. The choice of functions for the convergence plots is also based on the guidelines of CEC'2010 Competition on Large-Scale Optimisation (Tang et al., 2009). It can be seen from the convergence plots that the performance of DECC-DML is better than DECC-G. However the true performance of the DECC-DML and DECC-D is not reflected form the selected convergence plots, and the Table 13 should be consulted for a fair comparison. In addition to the competition requirements we have also compared the performance of DECC-D and DECC-DML with other algorithms such as DECC-G and MLCC as shown in Table 13. It can be seen that

Function	DECC	DECC-ML	DECC-D	DECC-DML	MLCC
f_1	**2.7263e-29**	5.7254e-28	2.9283e-29	4.7379e-28	6.8212e-14
f_2	5.4471e+01	2.7974e-04	5.2479e+01	**2.4811e-04**	2.5262e+01
f_3	1.4244e+02	1.8871e+02	**1.4077e+02**	1.9233e+02	1.4984e+02
f_4	5.3370e+01	**0.0000e+00**	5.4444e+01	**0.0000e+00**	4.3883e-13
f_5	2.7589e-03	3.6415e-03	8.8753e-04	7.8858e-04	**3.4106e-14**
f_6	2.3646e-01	3.3822e-14	1.2270e-01	**3.1548e-14**	1.1141e-13
f_7	-9.9413e+02	-1.5476e+03	-9.8976e+02	**-1.5480e+03**	-1.5439e+03

Table 8: Results of different algorithms over 100 dimensions(averaged over 25 runs). Best results are highlighted in bold.

Function	DECC	DECC-ML	DECC-D	DECC-DML	MLCC
f_1	**8.0779e-30**	1.6688e-27	3.8370e-29	1.7117e-27	4.2974e-13
f_2	4.0904e+01	1.3396e+00	3.8009e+01	**1.0232e+00**	6.6663e+01
f_3	6.6822e+02	5.9341e+02	**5.6941e+02**	6.8292e+02	9.2466e+02
f_4	1.3114e+02	**0.0000e+00**	1.4631e+02	**0.0000e+00**	1.7933e-11
f_5	2.9584e-04	1.4788e-03	2.9584e-04	2.9584e-04	**2.1259e-13**
f_6	6.6507e-14	1.2818e-13	**5.9828e-14**	1.2051e-13	5.3433e-13
f_7	-5.5707e+03	**-7.4582e+03**	-4.7796e+03	-7.4579e+03	-7.4350e+03

Table 9: Results of different algorithms over 500 dimensions (Averaged over 25 runs). Best results are highlighted in bold.

DECC-DML outperforms DECC-G on 14 out of 20 functions and outperforms MLCC on 12 out of 20 functions. This shows that the new delta method for grouping interacting variables is performing reasonably well. From a non-separability point of view, DECC-DML performed better than MLCC on 11 out of 17 non-separable functions ($f_4 - f_{20}$).

In order to further investigate the performance of delta grouping we counted the number of interacting variables that DECC-DML managed to group in the first 50 variables regardless of the subcomponent sizes. Table 11 shows the maximum number of interacting variables that were grouped for more than or equal to 2 cycles. Since functions $f_1 - f_3$ are separable it is meaningless to count the number of captured interacting variables for them. Functions f_{19}, f_{20} are completely non-separable so there is interaction between all of the variables so the number of captured interacting variables is not recorded for them. Let's consider f_4 as an example. The numbers in the table show that 46 interacting variables were captured by the delta grouping for 6 cycles. Since in DECC-DML the subcomponent size is self-adapted, the maximum number of cycles varies from function to function, but it suffices to say that in our experiments this is always less than 7000 cycles for all of the benchmark function. Even by using 7000 cycles which is slightly higher than what is really used in our experiments, the probability of grouping 46 variables for at least two cycles will be virtually zero in case of random grouping. The entries in Table 11 shows that the number of interacting variables that was captured by the delta grouping is reasonably high compared to random grouping, specially for functions $f_4 - f_8$. The entries for functions f_4,f_5,f_6,f_8 in Table 13 confirm that the high success rate of delta grouping in capturing interacting variables has a direct impact on its better performance compared to MLCC. The last column of Table 11 also shows the success rate for grouping at least 5 interacting variables.

Function	DECC	DECC-ML	DECC-D	DECC-DML	MLCC
f_1	1.2117e-29	5.1750e-28	**1.0097e-29**	3.3391e-27	8.4583e-13
f_2	4.2729e+01	**3.4272e+00**	3.8673e+01	5.81133e+00	1.0871e+02
f_3	1.2673e+03	**1.0990e+03**	1.1597e+03	1.22537e+03	1.7986e+03
f_4	2.4498e+02	**0.0000e+00**	2.7406e+02	**0.0000e+00**	1.3744e-10
f_5	2.9584e-04	9.8489e-04	**1.0392e-15**	1.4611e-15	4.1837e-13
f_6	1.3117e-13	2.5295e-13	**1.1866e-13**	2.2908e-13	1.0607e-12
f_7	-1.4339e+04	**-1.4757e+04**	-1.1035e+04	-1.4750e+04	-1.4703e+04

Table 10: Results of different algorithms over 1000 dimensions (Averaged over 25 runs). Best results are highlighted in bold.

Function	# Captured Interacting Vars	# Cycles	Success Rate
f_4	46	6	99.87%
f_5	32	3	3.45%
f_6	50	506	99.76%
f_7	31	4	97.92%
f_8	49	4	99.84%
f_9	7	7	6.98%
f_{10}	9	5	84.89%
f_{11}	13	4	53.85%
f_{12}	11	2	22.04%
f_{13}	10	3	21.71%
f_{14}	9	2	7.42%
f_{15}	9	2	9.07%
f_{16}	13	2	14.92%
f_{17}	8	9	10.97%
f_{18}	9	5	18.53%

Table 11: Maximum number of interacting variables captured by delta method for at least 2 cycles. The success rate of delta method on grouping at least 5 interacting variable.

For example the success rate of f_{11} is approximately 53% which means in 53 out of 100 cycles at least 5 variables are captured amongst the first 50 variables. In order to compare this results with random grouping we calculated the probability of grouping 5 variables for more than 1 cycle using Equation (9). We use 7000 cycles for this example because none of the experiments with DECC-DML used more than 7000 cycles. It is also assumed that there are on average 10 different subcomponents so given $n = 1000$, $m = 10$, $N = 7000$ and $v = 5$, using Equation (9) we have:

$$P(X \geq 1) = 1 - P(X = 0) = 1 - \left(1 - \frac{1}{10^{5-1}}\right)^{7000} = 0.5034$$

which means that the probability of grouping 5 variables for at least two cycle is approximately 0.5. In the worst case of f_5 more than 5 interacting variables are grouped for exactly 182 cycles which is considerably higher than what is achievable by only using random grouping. It is also noteworthy that the results of Tables 11 are based on the number of interacting variables captured *only* in the

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
At $1.2e5$ Fitness Evaluations										
Best	2.28e+08	5.51e+03	8.22e+00	3.80e+13	1.43e+08	1.25e+06	2.65e+09	2.23e+09	4.09e+09	1.32e+04
Median	2.85e+08	5.76e+03	9.71e+00	6.40e+13	2.85e+08	1.96e+06	5.50e+09	4.92e+09	4.91e+09	1.39e+04
Worst	7.02e+08	5.96e+03	1.01e+01	1.20e+14	5.21e+08	2.00e+07	1.17e+10	1.35e+10	5.54e+09	1.45e+04
Mean	4.09e+08	5.75e+03	9.51e+00	6.76e+13	3.00e+08	2.70e+06	5.97e+09	5.57e+09	4.89e+09	1.38e+04
StDev	1.75e+08	1.35e+02	5.55e-01	2.02e+13	9.31e+07	3.62e+06	2.49e+09	2.56e+09	3.77e+08	3.24e+02
At $6.0e5$ Fitness Evaluations										
Best	6.95e+01	2.51e+03	1.06e-02	7.92e+12	1.42e+08	4.59e+01	3.14e+08	4.19e+07	2.82e+08	1.25e+04
Median	4.63e+02	2.64e+03	1.83e-02	1.51e+13	2.85e+08	1.09e+02	5.42e+08	1.15e+08	3.85e+08	1.30e+04
Worst	1.22e+03	2.78e+03	2.20e-02	3.29e+13	5.20e+08	1.98e+07	9.17e+08	2.38e+08	4.21e+08	1.36e+04
Mean	6.02e+02	2.64e+03	1.81e-02	1.61e+13	2.99e+08	7.94e+05	5.84e+08	1.24e+08	3.73e+08	1.30e+04
StDev	4.11e+02	5.88e+01	3.08e-03	6.19e+12	9.31e+07	3.97e+06	1.68e+08	5.40e+07	3.13e+07	2.93e+02
At $3.0e6$ Fitness Evaluations										
Best	9.05e-27	1.62e+02	1.10e-13	1.38e+12	1.42e+08	3.55e-09	7.09e+07	7.34e+05	4.51e+07	1.21e+04
Median	1.22e-25	2.12e+02	1.14e-13	3.32e+12	2.85e+08	7.11e-09	1.23e+08	1.57e+07	5.97e+07	1.24e+04
Worst	7.12e-25	2.94e+02	1.35e-13	6.89e+12	5.20e+08	1.98e+07	4.82e+08	1.21e+08	7.09e+07	1.30e+04
Mean	1.93e-25	2.17e+02	1.18e-13	3.58e+12	2.99e+08	7.93e+05	1.39e+08	3.46e+07	5.92e+07	1.25e+04
StDev	1.86e-25	2.98e+01	8.22e-15	1.54e+12	9.31e+07	3.97e+06	7.72e+07	3.56e+07	4.71e+06	2.66e+02
	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}
At $1.2e5$ Fitness Evaluations										
Best	1.02e+02	4.07e+06	1.09e+08	1.26e+10	1.58e+04	3.22e+02	7.48e+06	1.56e+09	1.77e+07	2.04e+09
Median	1.22e+02	4.68e+06	1.82e+08	1.37e+10	1.65e+04	3.73e+02	8.77e+06	3.30e+09	2.23e+07	3.93e+09
Worst	1.70e+02	5.35e+06	3.72e+08	1.51e+10	1.73e+04	4.28e+02	1.01e+07	4.06e+09	2.72e+07	5.09e+09
Mean	1.24e+02	4.70e+06	2.11e+08	1.37e+10	1.65e+04	3.75e+02	8.81e+06	3.08e+09	2.20e+07	3.84e+09
StDev	1.38e+01	2.99e+05	9.68e+07	6.86e+08	3.61e+02	3.60e+01	6.86e+05	7.84e+08	2.36e+06	7.72e+08
At $6.0e5$ Fitness Evaluations										
Best	4.00e-01	3.64e+06	8.29e+02	9.82e+08	1.53e+04	3.46e+00	6.50e+06	5.64e+03	1.54e+07	1.43e+03
Median	7.09e-01	4.22e+06	1.71e+03	1.18e+09	1.59e+04	8.65e+00	7.29e+06	1.48e+04	1.84e+07	1.67e+03
Worst	1.72e+00	4.65e+06	1.47e+04	1.29e+09	1.67e+04	4.28e+02	7.99e+06	3.96e+04	2.46e+07	2.02e+03
Mean	7.66e-01	4.19e+06	3.15e+03	1.17e+09	1.59e+04	4.47e+01	7.27e+06	1.74e+04	1.87e+07	1.69e+03
StDev	2.81e-01	2.18e+05	3.09e+03	8.20e+07	3.63e+02	1.16e+02	3.77e+05	8.26e+03	1.99e+06	1.58e+02
At $3.0e6$ Fitness Evaluations										
Best	1.63e-13	3.46e+06	6.19e+02	1.54e+08	1.48e+04	2.74e-13	5.65e+06	1.64e+03	1.30e+07	9.69e+02
Median	1.78e-13	3.81e+06	1.06e+03	1.89e+08	1.53e+04	3.20e-13	6.55e+06	2.21e+03	1.59e+07	9.75e+02
Worst	2.03e-13	4.11e+06	2.09e+03	2.22e+08	1.62e+04	1.27e+00	7.63e+06	7.52e+03	2.16e+07	1.10e+03
Mean	1.80e-13	3.80e+06	1.14e+03	1.89e+08	1.54e+04	5.08e-02	6.54e+06	2.47e+03	1.59e+07	9.91e+02
StDev	9.88e-15	1.50e+05	4.31e+02	1.49e+07	3.59e+02	2.54e-01	4.63e+05	1.18e+03	1.72e+06	3.51e+01

Table 12: Experiment results of CEC'2010 functions for 25 independent runs with 1000 dimensions.

first 50 variables. It is highly possible that more interacting variables are grouped within the next subcomponents. This is especially true for functions f_9-f_{18} where there are more than one group of interacting variables (Tang et al., 2009). Another interesting pattern that can be seen from Table 11 is that delta grouping is highly effective for those set of non-separable functions that has only one group of interacting variables. Functions $f_4 - f_8$ have this property.

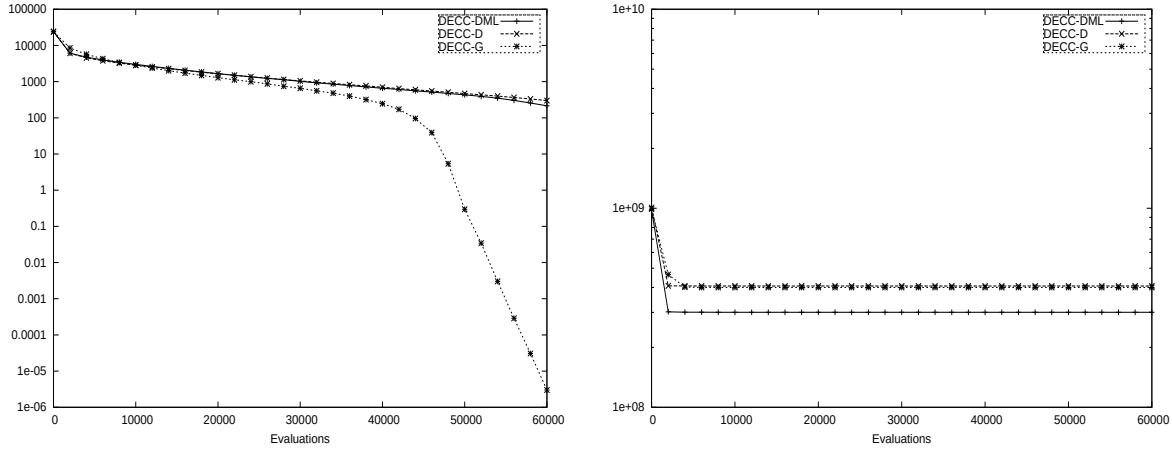
5.3 Chapter Summary

In the first part of this chapter (Section 5.1.2) we demonstrated that more frequent random grouping and removing the adaptive weighting are beneficial and the performance of the algorithm will increase dramatically. Running Wilcoxon rank-sum test on the results of DECC-ML and MLCC revealed that DECC-ML is significantly better than MLCC on all of the CEC'2008 benchmark functions. In Section 5.2 we analysed the performance of the *delta grouping*. A comparative study with other algorithms revealed that DECC-DML managed to capture interacting variable with a reasonable success rate. For example for one class of CEC'2010 benchmark functions a success rate of 80% has been achieved

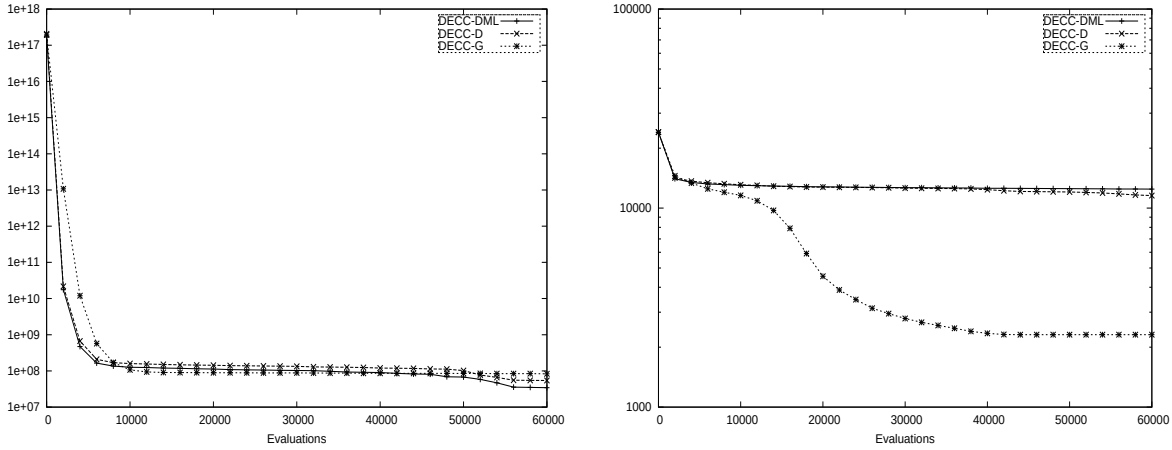
Functions	DECC-DML	DECC-G	DECC-D	MLCC
f_1	1.925263e-25	2.93e-07	1.013417e-24	**1.53e-27**
f_2	2.169774e+02	1.31e+03	2.995242e+02	**5.57e-01**
f_3	**1.180922e-13**	1.39e+00	1.813305e-13	9.88e-13
f_4	**3.580284e+12**	1.70e+13	3.994117e+12	9.61e+12
f_5	2.985220e+08	**2.63e+08**	4.162337e+08	3.84e+08
f_6	**7.932774e+05**	4.96e+06	1.356873e+07	1.62e+07
f_7	1.387946e+08	1.63e+08	6.578934e+07	**6.89e+05**
f_8	**3.463122e+07**	6.44e+07	5.392069e+07	4.38e+07
f_9	**5.918405e+07**	3.21e+08	6.187354e+07	1.23e+08
f_{10}	1.246898e+04	1.06e+04	1.156625e+04	**3.43e+03**
f_{11}	**1.800515e-13**	2.34e+01	4.764118e+01	1.98e+02
f_{12}	3.795382e+06	8.93e+04	1.527193e+05	**3.49e+04**
f_{13}	1.144516e+03	5.12e+03	**9.867780e+02**	2.08e+03
f_{14}	**1.890322e+08**	8.08e+08	1.983536e+08	3.16e+08
f_{15}	1.540041e+04	1.22e+04	1.531490e+04	**7.11e+03**
f_{16}	**5.078991e-02**	7.66e+01	1.880495e+02	3.76e+02
f_{17}	6.536997e+06	2.87e+05	9.030164e+05	**1.59e+05**
f_{18}	2.472471e+03	2.46e+04	**2.123339e+03**	7.09e+03
f_{19}	1.586111e+07	**1.11e+06**	1.332509e+07	1.36e+06
f_{20}	**9.906186e+02**	4.06e+03	9.912724e+02	2.05e+03

Table 13: Comparison of different algorithms on CEC'2010 functions with 1000 dimensions. Numbers show the mean of the best fitness over 25 runs. Best results are shown in bold.

which is significantly beyond the capabilities of random grouping. Table 13 also shows that DECC-DML outperformed MLCC on 12 out of 20 CEC'2010 benchmark functions. In next chapter we conclude this thesis and briefly summarise the answers to the research questions. Finally we provide some directions for future extensions of the current work.

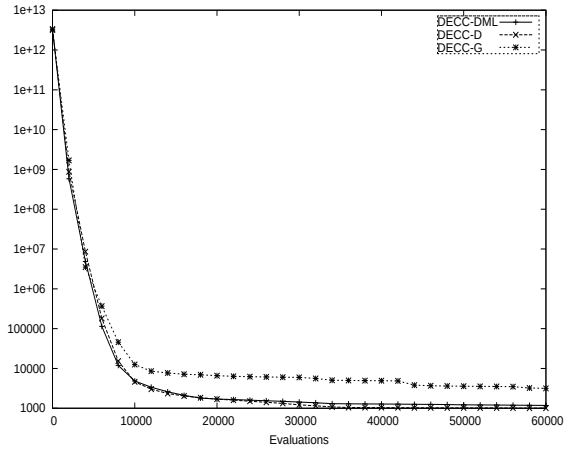


(a) f_2 (b) f_5

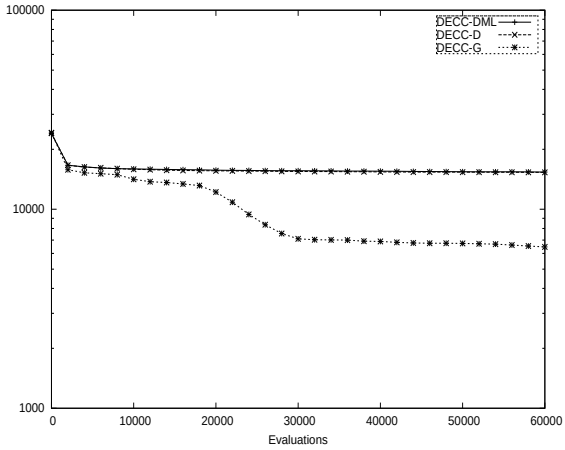


(c) f_8 (d) f_{10}

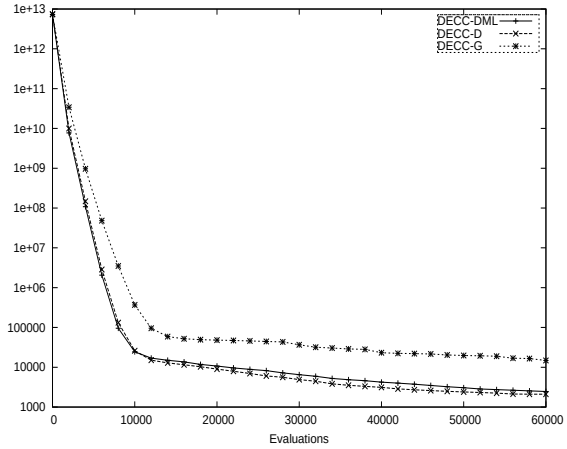
Figure 7: Convergence plots of f_2, f_5, f_8, and f_{10}(All variants of Rastrigin and Rosenbrock functions) for DECC-D and DECC-DML. Each point on the graph is the average over 25 independent runs.



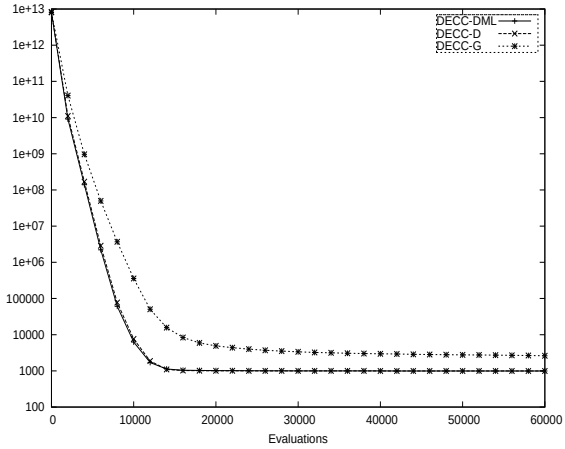
(a) f_{13}



(b) f_{15}



(c) f_{18}



(d) f_{20}

Figure 8: Convergence plots of f_{13}, f_{15}, f_{18}, and f_{20}(All variants of Rastrigin and Rosenbrock functions) for DECC-D and DECC-DML. Each point on the graph is the average over 25 independent runs.

6 Conclusion

In this thesis we have proposed several techniques to enhance an existing cooperative co-evolutionary algorithm aiming to further improve its performance and to substantially reduce its computational cost. We have shown that more frequent *random grouping* will result in faster convergence without sacrificing solution quality due to increased probability in grouping interacting variables in a subcomponent. More frequent random grouping will also increase the efficiency of the algorithms in dealing with problems with higher number of interacting variables. We have shown that *adaptive weighting* is not as effective as it was initially thought in reducing the dimensionality of the problem and in most cases fails to improve the performance. This may waste a considerable amount of fitness evaluations which might be better used for more effective random grouping and co-evolution of subcomponents.

We have also proposed an alternative technique for self-adapting the subcomponent sizes in CC framework and have shown that this simple technique is very effective. Finally we proposed an algorithm called DECC-ML which uses a uniform selection of subcomponent sizes together with more frequent random grouping. Experimental results show that this new algorithm outperforms MLCC, with a considrable difference, over all of CEC'2008 benchmark functions with up to 1000 dimensions (Section 5.1.2). Fast convergence of DECC-ML allows for saving a significant amount of fitness evaluations for most of the functions. This suggests that the algorithm may have the potential to save considerable amount of CPU time specially in real-world problems with costly objective functions.

Another significant contribution of this thesis is the development of a new technique called *delta grouping*. This new technique uses a more systematic way of grouping interacting variables of a non-separable problem based on their sorted delta values. Delta values measure the averaged difference in a certain variable across the entire population. Experimental results confirmed that this new technique is capable of grouping interacting variables more effectively. For the class of non-separable problems with a single group of interacting variables, delta grouping has shown an improved performance compared to the more blind approach, i.e., *random grouping*.

6.1 Answers to Research Questions

In this section it is tried to summarise our answers to research questions.

Does the *random grouping* scheme scales properly when the number of interacting variables increases?

In Section 3.1 we mathematically proved that random grouping does not scale well when the number of interacting variables increases. Although increasing the frequency of random grouping is proven to be beneficial (Section 5.1.2), the probability of grouping more than two interacting variables drops so fast that the increased frequency is negligible. This can be seen from Figure 2.

Is there a simpler and more effective alternative to MLCC for self-adaptation of the subcomponents sizes in Cooperative Co-evolution?

In Section 3.3 we have shown that using a uniform random number generator to select a decomposer from a set of predetermined decomposers, boosts the performance of DECC-ML as compared to MLCC. The technique which is used in MLCC relies on some empirical constants that are problem dependent. The new technique is simpler and more generalisable to different types of problems. Section 5.1.2 provides the experimental results on DECC-ML and the comparison reveals that DECC-ML is superior to MLCC.

Is it possible to develop a new technique, capable of capturing the interacting variables in a more systematic way?

Chapter 4 is dedicated to a new technique called *delta grouping*. Unlike random grouping, delta grouping, relies on a more systematic way of capturing interacting variables. In Section 4.2 we explain, in details, that why the *delta grouping* works. The main reason behind the success of this new technique is the decrease in the size of the improvement interval due to rotated fitness landscape. This behaviour is clearly shown in Figure 4. In Section 4.3 we describe how the *delta grouping* works. Empirical results in Section 5.2.1 shows that DECC-DML performs significantly better than MLCC on the CEC'2008 benchmark functions.

How effective does the new technique scales up when the number of interacting variables increases?

In order to further investigate the performance of *delta grouping* we tested our newly proposed algorithm called DECC-DML on a new set of benchmark functions which was proposed in CEC'2010 Special Session and Competition on Large-Scale Global Optimisation (Tang et al., 2009). The focus of the new test suite is on non-separable problems. The empirical results on the new function set which was provided in Section 5.2.2 shows that the new technique has a good performance on the new test set. Table 11 shows that the number of interacting variables that are successfully captured using the *delta grouping* is far more that the capabilities of *random grouping* even with increased frequency of random shuffling of decision variables.

6.2 Future Works

Delta grouping method seems to be a promising technique for large scale non-separable optimisation problems, but this new technique is still in its infancy and further research is required in order to fully understand its underlying mechanics. For example it is not clear why DECC-DML is less efficient on non-separable functions with more than one group of interacting variables. Further investigation is also required in order to understand the reason for good performance of DECC-ML on many of CEC'2008 benchmark functions. As it was briefly mentioned earlier this might be due to the fact that most CEC'2008 benchmark functions are separable and DECC-DML might pay off when applied to CEC'2010 functions. This calls for a separate study for comparing the performance of DECC-DML and DECC-ML on CEC'2010 test suite.

In the current study we relied on equally-sized subcomponents, however in real-world problems this is not necessarily the case and different groups of interacting variables might have different sizes. Applying some clustering techniques on the delta values in order to group the interacting variables in a more natural way is another way that the current work could be extended.

Using other types of subcomponent optimisers in a delta grouping framework might also show good performance. Specially the techniques which are rotationally invariant. For example CMA-ES (Hansen and Ostermeier, 2001) is a robust, rotationally invariant, optimiser for non-linear optimisation problem however it looses its efficiency in higher dimensions. Combination of CMA-ES as the subcomponent optimiser with delta grouping could produce promising results.

Less greedy EA technique has shown better performance on large scale optimisation problems. This is natural because in large scale problems more exploration is needed before finding a potential global optimum. Analytical study of DE's selection schemes (Price et al., 2005) in order to calculate its takeover time could result in discovery of better selection schemes with a better balance between exploration and exploitation suited for large scale optimisation problems.

7 Appendix

7.1 Appendix A: IEEE CEC'2008 Benchmark Functions

The following benchmark test functions, which were proposed in the IEEE CEC'2008 Special Session and Competition on Large Scale Global Optimisation (Tang et al., 2007) has been used in this thesis. The Java™and MATLAB® source of the test suite can be downloaded from `http://nical.ustc.edu.cn/cec08ss.php`.

f_1: Shifted Sphere Function

$$f_1(x) = \sum_{i=1}^{D} z_i^2 + f_{bias1}, z = x - o$$

$$-100 \leq x_i \leq 100, min(f_1) = f_1(o) = f_{bias1} = -450,$$

where o is the shifted optimum[1].

f_2: Shifted Schwefel's Problem 2.21

$$f_2(x) = max_i\{|z_i|, 1 \leq i \leq D\} + f_{bias2}, z = x - o$$

$$-100 \leq x_i \leq 100, min(f_2) = f_2(o) = f_{bias2} = -450$$

f_3: Shifted Rosenbrock's Function

$$f_3(x) = \sum_{i=1}^{D-1} ((z_i^2 - z_{i+1})^2 + (z_i - 1)^2) + f_{bias3}, z = x - o$$

$$-100 \leq x_i \leq 100, min(f_3) = f_3(o) = f_{bias3} = 390$$

f_4: Shifted Rastrigin's Function

$$f_4(x) = \sum_{i=1}^{D} (z_i^2 - 10cos(2\pi z_i) + 10) + f_{bias4}, z = x - o$$

$$-5 \leq x_i \leq 5, min(f_4) = f_4(o) = f_{bias4} = -330$$

f_5: Shifted Griewank's Function

$$f_5(x) = \sum_{i=1}^{D} \frac{z_i^2}{4000} - \prod_{i=1}^{D} cos(\frac{z_i}{\sqrt{i}}) + 1 + f_{bias5}, z = x - o$$

$$-600 \leq x_i \leq 600, min(f_5) = f_5(o) = f_{bias5} = -180$$

[1]This is consistently used for functions $f_1 - f_6$

f_6: Shifted Ackley's Function

$$f_6(x) = -20exp\Big(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} z_i^2}\Big) - exp\Big(\frac{1}{D}\sum_{i=1}^{D} cos(2\pi z_i)\Big) + 20 + e + f_{bias6}, z = x - o$$
$$-32 \leq x_i \leq 32, min(f_6) = f_6(o) = f_{bias6} = -140$$

f_7: FastFractal "DoubleDip" Function

$$f_7(x) = \sum fractal1D\Big(x_i + twist(x_{(i \bmod D)+1})\Big)$$

$$twist(y) = 4(y^4 - 2y^3 + y^2)$$

$$fractal1D(x) \approx \sum_{k=1}^{3}\sum_{1}^{2^{k-1}}\sum_{1}^{rand2(o)} doubledip\Big(x, ran1(o), \frac{1}{2^{k-1}\big(2 - ran1(o)\big)}\Big)$$

$$doubledip(x,c,s) = \begin{cases} \big(-6144(x-)^6 + 3088(x-c)^4 - 392(x-c)^2 + 1\big)s, -0.5 < x < 0.5 \\ 0, otherwise \end{cases}$$

$$-1 \leq x_i \leq 1, min(f_7) = \text{Unknown}$$

7.2 Appendix B: IEEE CEC'2010 Benchmark Functions

This appendix contains a summary of benchmark functions which were proposed in the IEEE CEC'2010 Special Session and Competition on Large Scale Global Optimisation (Tang et al., 2009). The Java™ and MATLAB® source of the test suite can be downloaded from `http://nical.ustc.edu.cn/cec10ss.php`. This new test suite contains the following major categories of functions:

1. Separable functions.
2. Partially separable functions, in which a small number of variables are dependent while all the remaining ones are independent.
3. Partially separable functions that consist of multiple independent subcomponents, each of which is m-nonseparable.
4. Fully non-separable functions.

In the above list, separable functions are the easiest to optimise and fully non-separable functions are the most difficult to optimise. Partially separable functions are design to closely mimic the situation in most of the real-world problems. These benchmark functions are shifted and rotated versions of the following base functions:

1. Elliptic Function
2. Rastrigin Function
3. Ackley
4. Schwefel Problem 1.2
5. Rosenbrock Function

The Table 14 shows the function numbers, the category that they belong to as well as their base functions.

	Base Functions				
Categories	Elliptic	Rastrigin	Ackley	Schwefel 1.2	Rosenbrock
Separable Functions	f_1	f_2	f_3		
Single-group m-nonseparable Functions	f_4	f_5	f_6	f_7	f_8
$\frac{D}{2m}$-group m-nonseparable Functions	f_9	f_{10}	f_{11}	f_{12}	f_{13}
$\frac{D}{m}$-group m-nonseparable Functions	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}
Non-separable Functions	f_{19}	f_{20}			

Table 14: List of IEEE CEC'2010 benchmark functions, the category that they belong to and their corresponding base functions. D is the dimensionality of the function that could be varied by the user. m controls the degree of separability and indeed controls the number of variables in each group. This variable is set to 50 based on (Tang et al., 2009) and can be changed by users based on their purpose.

References

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. ser. Dover Books on Mathematics. Oxford University Press.

Bäck, T., Fogel, D. B., and Michalewicz, Z., editors (1997). *Handbook of Evolutionary Computation*. Institute of Physics Publishing, Bristol, and Oxford University Press, New York.

Bellman, R. E. (1957). *Dynamic Programming*. ser. Dover Books on Mathematics. Princeton University Press.

Gamperle, R., Muller, S., and Koumoutsakos, P. (2002). A parameter study for differential evolution. In *Proceedings WSEAS International Conference on Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, pages 293–298.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

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

Hasenjäger, M., Sendhoff, B., Sonoda, T., and Arima, T. (2005). Three dimensional evolutionary aerodynamic design optimization with cma-es. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 2173–2180.

Hestenes, M. R. and Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49:409–436.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, USA.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948.

Li, X. and Yao, X. (2009). Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms. In *Proceedings of Congress on Evolutionary Computation*, pages 1546–1553.

Liu, Y., Yao, X., Zhao, Q., and Higuchi, T. (2001). Scaling up fast evolutionary programming with cooperative coevolution. In *Proceedings of Congress on Evolutionary Computation*, pages 1101–1108.

Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer series in operations research and financial engineering. Springer, New York, NY, 2. ed. edition.

Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, Cambridge, MA.

Olhofer, M., Jin, Y., and Sendhoff, B. (2001). Adaptive encoding for aerodynamic shape optimization using evolution strategies. In *Proceedings of Congress on Evolutionary Computation*, volume 2, pages 576–583. IEEE Press.

Omidvar, M. N., Li, X., Yang, Z., and Yao, X. (2010). Cooperative co-evolution for large scale optimization through more frequent random grouping. In *Proc. of IEEE World Congress on Computational Intelligence(under review)*.

Potter, M. A. and De Jong, K. A. (1994). A cooperative coevolutionary approach to function optimization. In *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, volume 2, pages 249–257.

Price, K., Storn, R., and Lampinen, J. (2005). *Differential Evolution: A Practical Approach to Global Optimization*. Natural Computing Series. Springer.

Qin, A. and Suganthan, P. (2005). Self-adaptive differential evolution algorithm for numerical optimization. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 2, pages 1785–1791.

Ray, T. and Yao, X. (2009). A cooperative coevolutionary algorithm with correlation based adaptive variable partitioning. In *Proc. of IEEE Congress on Evolutionary Computation*, pages 983–989.

Rosenbrock, H. H. (1960). An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184.

Salomon, R. (1995). Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions - a survey of some theoretical and practical aspects of genetic algorithms. *BioSystems*, 39:263–278.

Shi, Y., Teng, H., , and Li, Z. (2005). Cooperative co-evolutionary differential evolution for function optimization. In *Proc. of the First International Conference on Natural Computation*, pages 1080–1088.

Sobieszczanski-Sobieski, J. and Haftka, R. T. (1997). Multidisciplinary aerospace design optimization: Survey of recent developments. *Structural Optimization*, 14:1–23.

Sofge, D., Jong, K. D., and Schultz, A. (2002). A blended population approach to cooperative coevolution fordecomposition of complex problems. In *Proc. of IEEE World Congress on Computational Intelligence*, pages 413–418.

Storn, R. (1996). On the usage of differential evolution for function optimization. In *Proceedings of the 1996 biennial conference of the North American fuzzy information processing society*, pages 519–523.

Storn, R. and Price, K. (1995). Differential evolution . a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization 11 (4)*, pages 341–359.

Tang, K., Li, X., Suganthan, P. N., Yang, Z., and Weise, T. (2009). Benchmark functions for the cec'2010 special session and competition on large-scale global optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC, China. http://nical.ustc.edu.cn/cec10ss.php.

Tang, K., Yao, X., Suganthan, P. N., MacNish, C., Chen, Y. P., Chen, C. M., , and Yang, Z. (2007). Benchmark functions for the cec'2008 special session and competition on large scale global optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC, China. http://nical.ustc.edu.cn/cec08ss.php.

van den Bergh, F. and Engelbrecht, A. P. (2004). A cooperative approach to particle swarm optimization. *IEEE Transactions on Evolutionary Computation 8 (3)*, 2:225–239.

Vesterstrom, J. and Thomsen, R. (2004). A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical. In *Proceedings of Congress on Evolutionary Computation*, pages 1980–1987.

Yang, Z., Tang, K., and Yao, X. (2008a). Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, 178:2986–2999.

Yang, Z., Tang, K., and Yao, X. (2008b). Multilevel cooperative coevolution for large scale optimization. In *Proc. of IEEE World Congress on Computational Intelligence*, pages 1663–1670.

Yang, Z., Tang, K., and Yao, X. (2008c). Self-adaptive differential evolution with neighborhood search. In *Proc. of IEEE World Congress on Computational Intelligence*, pages 1110–1116.