LARGE-SCALE OPTIMIZATION: COMBINING CO-OPERATIVE COEVOLUTION AND FITNESS INHERITANCE

by

Aboubakar Hameed Ali Hameed

Submitted for the degree of
Doctor of Philosophy

Department of Computer Science
School of Mathematical and Computer Sciences
Heriot-Watt University

February, 2019

Abstract

Large-scale optimization, here referring mainly to problems with many design parameters remains a serious challenge for optimization algorithms. When the problem at hand does not succumb to analytical treatment (an overwhelmingly commonplace situation), the engineering and adaptation of stochastic black box optimization methods tends to be a favoured approach, particularly the use of Evolutionary Algorithms (EAs). In this context, many approaches are currently under investigation for accelerating performance on large-scale problems, and we focus on two of those in this research.

The first is co-operative co-evolution (CC), where the strategy is to successively optimize only subsets of the design parameters at a time, keeping the remainder fixed, with an organized approach to managing and reconciling these subspace optimization.

The second is fitness inheritance (FI), which is essentially a very simple surrogate model strategy, in which, with some probability, the fitness of a solution is simply guessed to be a simple function of the finesses of that solution's parents. Both CC and FI have been found successful on nontrivial and multiple test cases, and they use fundamentally distinct strategies.

In this thesis, we explored the extent to which both of these strategies can be used to provide additional benefits. In addition to combining CC and FI, this thesis also introduces a new FI scheme which further improves the performance of CC-FI. We show that the new algorithm CC-FI is highly effective for solving problems, especially when the new FI scheme is used.

In the thesis, we also explored two basic adaptive parameter setting strategies for the FI component. We found that engineering FI (and CC, where it was otherwise not present) into these algorithms led to good performance and results.

Acknowledgements

I would like to thank my supervisor prof David Corne for his continued encouragement, guidance and advice in every stage of my PhD. Without his help this thesis could not finished.Also I would like to thank my wife and children for their support and love!

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Large-scale problems are a set of challenges that still face EA researchers. The usual meaning of large-scale, in this context, is an optimization problem with a reasonably (or unreasonably) large number of decision parameters [12]. Another common meaning of large scale, and a closely-related challenge, refers to problems where the processor time requirement for evaluating a single solution is very high.

The challenge for the algorithm design community in large scale optimization is to find search strategies that provide sufficiently good solutions in as small as possible a number of fitness evaluations. In those cases where the computational time-complexity of the fitness function is high, the need to make progress in reduced numbers of evaluations is obvious. On the other hand, when the number of parameters is high, the issue tends to be different: in these circumstances, standard algorithms tend to converge prematurely [3, 55], long before effecting a suitably extensive exploration of the parameter space; the challenge therefore becomes that of making the most of the available fitness evaluations, to achieve a better level of exploration in the available time.

Evolutionary Algorithms (EAs) are one of the well-known optimization approaches [91] which are very successful in general, but will face difficulties in solving these kind of problems.

Large scale problems naturally tend to require high processing speed and significant memory requirements. In general, for hard optimization problems, there is no guarantee for finding the optimal solution in a given or finite amount of time;

for large scale problems, this is even more true, and we are that much more likely to return poor solutions; in solving complex optimization problems, if the diversity mechanism does not work properly, for example, EAs often prematurely converge into local optima [83].

Many approaches are under investigation in the research domain to solve large scale problems. Two of the most effective of these approaches were adopted in this thesis: co-operative co-evolution (CC) and fitness inheritance (FI). In particular, the work described in this thesis was the first to combine these two approaches.

1.1 Optimization

Optimization problems are tasks in which there can be a huge number of potential solutions, and our job is to search this set of possible solutions to find the best one, or to find one as good as possible, in a reasonable time. It is a common task in many disciplines such as science, agriculture, engineering and even in our daily life. When we are thinking to enhance our way to go to work, select a line at the supermarket, or deciding our holiday traveling package, we are facing optimization problems.

As an example of a more topical type of optimization problem faced in industry, we can point to agriculture, and the task of finding the most economical and effective schedule for irrigation and fertilization that maximises the farmer's profits [31]. Or, when mechanical engineers try to design a new engine with high performance and low cost, this is an optimisation problem. Also when engineers trying to maximize the load of a robot which can lift a heavy load, this is an optimisation problem [114].

In most optimization problems we cannot solve the problem in one step. Therefore, we should follow some process or model in order to guide us through the problem solving process. The process of the solution is divided into many stages [15]. These stages are usually defining and recognising the problem, formulating solution models, and then evaluating and implementing solutions.

There are many common optimization problems that can be solved by using EAs. The traveling salesman problem is one of the most well known of these common problems. It is a classical problem known to be an NP-hard problem. It can be

described as follows: given a list of cities with their distance between each pair of the cities, there are hundreds of paths to visit these cities but the problem is finding the shortest path or round trip that involves all of cities in it by starting and returning to the same city, at the same time ensuring that each city will be visited only once [53].

For Example: See the following set of cities as in figure 1.1: The problem is to find a minimal route by passing all cities in turn. For example the first route is {ABCDEA} and the second route is {ABCEDA}. We pass through all the cities with a total length of 18 in the first route, and a total length of 27 by the second route. A possible simple EA for solving the TSP is the Hillclimbing algorithm; this EA starts with an initial random solution route, for instance {ABCEDA}, then we find out its fitness value, and this becomes our current solution. The Hillclimbing algorithm then repeatedly mutates the current solution to a mutant solution which represents a potential new path. Whenever the mutant is better than the current solution, it becomes the new current solution.

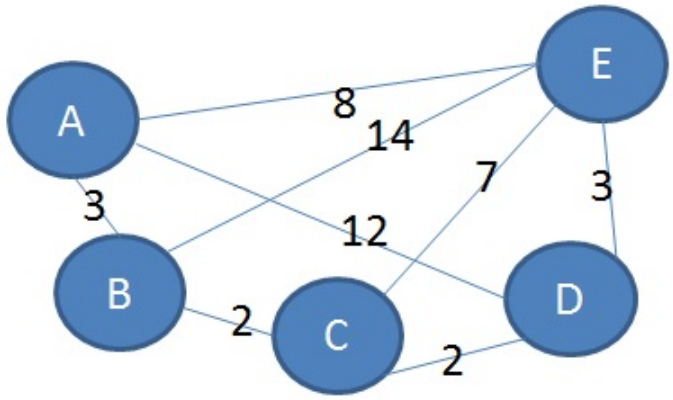

Figure 1.1: Travel salesman problem.

***Bin packing problem*:** The bin-packing problem is another common example of an optimization problem. The bin packing problem (BPP) is a class of NP-hard problem [34] and the standard BPP is as follows: given a set of items with sizes

between 0 and 1, these items should be packed into a minimum number of bins. Each bin must not contain items with total size more than 1 [27]. In the early 1970s, Johnson created the problem definition of bin packing problem together with Granham [39, 40]. They explored solving problems in this area by using approximation algorithms and heuristics. Among these algorithms in the early works is the first fit (FF) algorithm which is still often used for solving bin packing problems.

The (FF) algorithm steps are: According to a given ordered list of the items, each item is packed into the first existing bin where it can fit. If the item does not fit any existing bin, a new bin will be opened for this item. Another optimisation algorithm is the Best Fit (BF), it works the same as FF but the different in the BF algorithm is the item is packed into the most full bin first if it fits. If the items in the list are ordered by decreasing sizes, both algorithms (FF and BF) are named as first fit decreasing (FFD) and best fit decreasing (BFD). The time required for the algorithms is O(n log n) [13].

Here we consider the following problem definition : "There are a set of N items, and each item has a given weight. Also, each item has a given type (there are T different types of item). The items have to be arranged into C containers, in such a way that the total weight of each container is as similar as possible. However there are constraints involving the types. The fitness function is (heaviest container - lightest container) + (pairs $\times$ TW)" [22]. The experimental results of our algorithm CCEA-FI on this problem are presented in section 3.3.2.

Also, optimization problems can be large scale which become difficult for a simple evolutionary algorithm. An example of such problem is a very-large Neural Networks due to the high dimension of their input space (tens of thousands of weights), these large nets become infeasible using direct encoding that map genes one to one to network components [51]. On the other hand, sometimes the problem does not have particularly many parameters, but takes a long time simply to evaluate the fitness of a single solution. For instance, if we have an optimization task in which computational fluid dynamics code must be run to evaluate a single solution [107,

130], or indeed any type of fine-grained simulation.

Despite these difficulties, research in this area continues to develop better methods, especially for large scale problems. Several interesting and effective approaches are now available for large-scale optimizaton, such as Cooperative coevolution (CC) which can be integrated into other algorithms in order to improve the performance, and which seems to work well by helping the algorithm better guide its parameter changes in response to fitness values, it makes better use of each fitness evaluation. Another such method is fitness inheritance (FI), which works in a completely different way, by simply estimating fitness in some cases (rather than, for example, running the full fitness function which may be computationally expensive). In this research we focus on combining CC and FI, and see to what extent this improves on the individual components.

1.2 Summary

EAs have been successfully and extensively used to solve many optimization problems in recent years [91]. However, we still have immense amounts to learn about how to engineer an EA to do well on a given problem class, and one of the more urgent challenges for EAs is that of large-scale problems. By large-scale, we mainly refer to optimization problems with a relatively large number of decision parameters [12]. A related challenge is that of problems where the time complexity of evaluating a single solution is high. In both cases (and in cases which combine the two), the challenge is to find strategies for evolutionary search that enable good enough solutions to be found in a smaller number of fitness evaluations than would be needed by a conventional EA. When the time-complexity of the fitness function is high, the need for progress in fewer evaluations is obvious. When the number of parameters is high, the issue tends to be that conventional EAs converge long before achieving a suitably rich exploration of the parameter space, and we must somehow achieve a better level of exploration in the available time.

In this thesis we look in particular at two quite different strategies that are both effective. The first of these, currently under investigation within the EA community,

is the strategy of co-operative co-evolution (CC).

Meanwhile, second strategy is fitness inheritance (FI) is a quite different strategy aimed at reducing need for expensive fitness function evaluations. The thesis also investigations the difficulty of FI and why in some functions (FI) is not really effective, and often led to very poor performance in some fitness inheritance proportioncyan for example (70, 80, 90%) [2, 93]. A new model and scheme are designed based on analysis in this thesis for FI to overcome these difficulties. Some popular benchmark functions are chosen to demonstrate our new algorithm CC-FI.

1.3 Thesis Contributions

The main contributions are as follows:

1. A combination of Cooperative Coevolution framework CC and Fitness Inheritance FI, called CC-FI, is developed, and a simple EA is used here as subcomponents optimizer. The performance of our new algorithm CC-FI tested on well-known 4 functions Rastrigin, Schwefel, Rosenbrock and Ackley. The raw findings indeed suggest that CC-FI generally achieves significantly better performance than either a CC-based EA without FI (CCEA), or an EA with FI but without CC (EA-FI).

2. We explore the high-performance techniques Self-Adaptive Neighbourhood Search Differential Evolution (SaNSDE) with our (CC-FI) algorithm in the field of large-scale optimization instead of EA. We implemented SaNSDE from the description in the literature, and engineered CC+FI into it, using the same CC+FI framework as in our algorithm in the previous contribution, but with one key improvement on FI as it loses its performance at high levels of inheritance, especially on high dimensional problems. The results conclude that the new approach to combining CC and FI involving the key improvement in FI (nFI) represents a recommended algorithm-enhancement strategy.

3. Two simple adaptive schemes for FI are investigated in the CC-FI context, and engineering these new schemes (CC-aFI) into DECC-DML algorithm [71]. We

noticed that the mechanisms in the original DECC-DML source code associated with calculating and updating the delta value, incorporated some calls to function evaluations that were not accounted for in the total which counted towards algorithm termination. Therefore, we corrected the version of DECC-DML andcyan it was then compared with our algorithm DECC-DML-aFI. The comparison was over the CEC 2010 large scale global optimization test suite of 20 test functions on 1000 dimensions.

1.4 Thesis Structure

This thesis consists of six chapters, including this introduction chapter. The remaining five chapters are organised as follows:

Chapter 2 presents the background required to develop an efficient combination of CC and FI for large-scale optimisation problems, outlining the evolutionary algorithm and co-evolution in general and co-operative coevolution in particular. In addition, it describes why evaluation is sometimes expensive and how FI could be an alternative approach to reduce this expense. It also discusses differential evaluation and its various applications. The chapter ends with a review of the related work on large-scale optimisation.

Chapter 3 explains in detail a combination of co-operative coevolution (CC) and fitness inheritance (FI) and how the use of both of these strategies can provide additional benefits. We combined CC and FI into a straightforward algorithm over the basic EA and called the result CCEA-FI. This chapter describes the pseudo-code of the CCEA-FI algorithm. It also reports the results of evaluating the CCEA-FI algorithm on the well-known functions Rastringin, Schwefel, Ackley and Rosenbrock. The second part of this chapter discusses the testing of our algorithm on numerical problems (bin packing problems).

Chapter 4 presents the extent to which CC and FI provide added value when engineered together in the context of more sophisticated, the so-called state-of-the-art, algorithms instead of using a basic EA as discussed in Chapter 3. Self-adaptive neighbourhood search differential evolution (SaNSDE) with our (CC-FI) algorithm

was explored in the field of large-scale optimisation with a key improvement on FI, as it often loses its efficiency when increasing its proportion.

Chapter 5 discusses the design of two simple adaptation schemes were designed for the FI key parameters in the context of a further investigation into engineering CC and FI to another sophisticated state-of-the-art algorithm. In this case, the algorithm of choice was DECC-DML [71].

Chapter 6 concludes our work in this thesis. First, we summarise the work that we have done on our combination of CC and FI. Second, it presents a list of the contributions that we have achieved during the development of the CC-FI algorithm. Finally, we suggest future work directions.

1.5 Thesis Publications

1. A. Hameed, D. Corne, D. Morgan, and A. Waldock. Large-scale optimization: Are co-operative co-evolution and fitness inheritance additive? In Computational Intelligence (UKCI), 2013 13th UK Workshop on, pages 104111, Sept 2013.

2. Hameed, A.; Kononova, A.; Corne, D., Engineering Fitness Inheritance and Co-operative Evolution Into State-of-the-Art Optimizers, in Computational Intelligence, 2015 IEEE Symposium Series on , vol., no., pp.1695-1702, 7-10 Dec. 2015.

Chapter 2

Background

In this chapter we outline the basic idea of evolutionary algorithms, and then illustrate the concept of Co-evolution in general terms. Also in this chapter we introduce the key concept of Cooperative Coevolution, and discuss its variants and successful applications as a promising technique to solve large-scale optimization problems. Among these introductory descriptions, we highlight the issue of fitness evaluation, since this is a key issue and motivator for the techniques explored in this thesis; hence we explain how sometimes evaluation is expensive and the time required to evaluate a single solution becomes too high. Such situations motivate the need to explore alternative approaches, which reduce this expense, and this leads us to a discussion of fitness inheritance, which has a different approach (from CC) to speeding up black box optimization and reducing the need for expensive fitness evaluations. In this chapter we also introduce differential evolution and its variants, which is another algorithm that plays a part in later chapters. At the end of this chapter, we cover related work in the general field of methods to solve large-scale optimization problems.

2.1 Evolutionary Algorithms

In the biological world, evolution is a process by which individuals will develop and change in the population, generation by generation, according to the principle of survival of the fittest [28, 49]. Evolutionary algorithms (EA) can be regarded as

loosely simulating the process of natural evolution, and they have been successfully applied to numerous optimization problems. Evolutionary algorithms are a collection of techniques that is also often generally called Evolutionary Computation (EC). EC's family of algorithms has four well-known classes: evolutionary strategies [1], evolutionary programming [19], genetic algorithms [116] and genetic programming [19].

2.1.1 Outline of evolutionary algorithm

EAs normally start with a population of individuals, each individual in the population referred to as a solution. During the operation of an EA, each individual will be updated, with the overal aim of improving the solutions of the entire population via a combination of operations known as recombination, mutation, selection and replacement. These operations will repeat in each generation until a terminating criterion is met, also the quality of each individual in the population will be evaluated by applying the fitness function to that individual. When the algorithm has terminated, the process typically returns the best solution (in terms of the values returned by the fitness function) that was found during the process. A standard schema for an EA algorithm is shown here in pseudocode 1.

The most important operations in EA as shown in Algorithm 1 are:

- Initialization
- Evaluate function (fitness function)
- Select parents
- Generate offspring (Mutation and Recombination)
- Select survivors
- Termination

- ***Initialisation*:** In this operation, the algorithm will initialise a collection of individuals, called the population, by following a predefined encoding scheme.

Algorithm 1 Evolutionary algorithm

```
1: Initialize population of individuals
2: Evaluate population
3: repeat
4:     Select parents from population
5:     Generate offspring from parents
6:     Evaluate offspring
7:     Select survivors for new population
8: until Terminating criteria is met.
```

Initialisation might be randomly generated or by using heuristic strategies [19]. According to the problem, the number of the parameters in each individual is set manually and called the dimension of the problem. Moreover, we manually set the number of the individuals to be generated which is often called population size.

- ***Evaluation*:** Each problem has a fitness function designed for building a solution (by interpreting the encoded form of the solution) and then evaluating the quality of the solution. In general, the encoded form of the solutions is called a genotype, in this thesis, where we mainly deal with numerical optimization, the genotype is simply a vector of real numbers. In many applications, the genotype is itself a convenient encoded form of the real solution being represented (the phenotype). The fitness function therefore needs to begin by decoding the genotype to form the phenotype, before the phenotype can be evaluated [116]. After that the fitness function will provide an actual quality value for each individual in order to make an accurate assessment of each individual and help the algorithm to rank the population from the best to the worst candidates. An example of genotype and phenotype mapping in Table 2.1.

Genotype			$\Longrightarrow$	Phenotype		
100	101	110		4	5	6

Table 2.1: Genotype and phenotype mapping

- ***Select parents*:** This is an important operation in evolutionary algorithms. To generate potentially high quality new solutions (offspring) from so-called

parent individuals, it is useful to prefer that higher-quality individuals have a better a chance of being parents; this is achieved by a selection mechanism. There are different types of selection mechanisms in EA used to select those parents in order to generate offspring, such as, roulette wheel selection [6], tournament selections [65], rank selection [118], and truncation selection [18]. more details of these selections are in the next section.

- ***Generate offspring (mutation and recombination)*:** Mutation is the most popular genetic operation in evolutionary algorithms, which changes the data (genes) of the vector to produce a new vector. For example, if the data of the vector is binary, mutation can simply choose a random element of the solution and change it from zero to one or from one to zero. In the case of real-valued solution vectors, the change could be by a randomly generated new parameter value from the problem domain. This random value could come from a uniform or non-uniform distribution, both of which are widely used. In this thesis, we typically use mutation on real-valued vectors, and we generate a new gene for the parent by adding a random value from a Gaussian distribution to the parent gene to produce a new child as in equation 2.1.

$$X_i' = X_i + \sigma.N(0,1) \tag{2.1}$$

  Recombination or crossover is also another common genetic operation which occurs between two or more vectors in order to create a new vector or candidate solution. There are different types of crossover techniques such as one point crossover and two or more points crossover. Basically, the crossover role is to take the traits from parents and pass it to the next generation. This is in contrast to mutation, which essentially randomly generates new traits, or new variants of existing traits that their parents do not currently have. More details about mutation and crossover techniques are illustrated in section 2.1.3

- ***Select survivors* :** Selection of survivors is also called the replacement strategy. This refers to the process by which individuals with higher fitness will

be preferred to take part in the next generation. In some EAs, this process simply involves ranking the previous population and new individuals by fitness, and choosing the best P of these (where P is population size) to be the next generation. In other EAs, this process could involve using the parent selection mechanism P times (on the combination of previous population and new individuals) to choose the P individuals for the next generation. There are several other mechanisms.

- ***Termination* :** In most EAs, termination occurs after a predefined maximum number of fitness evaluations, or a predefined maximum number of generations. Other termination conditions are possible. For example, there may be a target value of fitness quality, or a known fitness quality value which cannot be bettered. Termination can be set to happen when such a fitness value is reached; however with such a stopping condition there is no guarantee that this solution will be found and the algorithm may never stop. Therefore, the most common termination conditions usedcyan in experiments are a predefined number of function evaluations to terminate the evolutionary cycle.

- ***Limitations of evolutionary algorithms*:** Many benefits and successes of EAs have been recorded in the general enterprise of applying them to solving optimization problems. Typically we see that EAs provide better solutions than previous methods. However, these improvements and successes could still be quite suboptimal, and considerably worse than the best solutions achievable. This is especially the case when we consider problems with a high number of dimensions. When EAs produce poor solutions, this is cyan often because of "premature convergence"; this occurs when the evolution process has stagnated, in the sense that all solutions in the population have become the same, or very similar to each other, and the operators are unable to discover better solutions in the vicinity of the current population. Such issues can be controlled to some extent by choosing good parameters and appropriate selection methods, and so on. However, these issues are also the motivation for a wide range of new EA mechanisms and operations.

2.1.2 Parents selection methods

In genetic algorithms, selection is a very important part of the process since their convergence is affected significantly. The basic idea behind the selection methods is: the better an individual's fitness value, the larger chance of its mating and survival to generate offspring [56]. Broadly speaking, the role of the selection method is to ensure that new offspring, which might become members of the next generation, are produced by good quality parents [89]. The most common and well-known selection methods in EAs include roulette wheel, rank, steady state and tournament selection.

***Fitness proportionate selection*:**

Fitness Proportionate Selection is one of the most common methods of parent selection. In this method, each individual of the population can become a parent with a probability proportional to its fitness value. Therefore, fitter individuals have a greater chance of mating and propagating their features to the next generation. Therefore, such a selection technique applies a selection weight to the more fit individuals in the population, introducing better individuals over time. Roulette wheel selection (described next) is the most common selection method used for implementing Fitness Proportionate [66].

***Roulette wheel selection*:**

In this method parents are selected based on their fitness. Therefore, the roulette wheel is divided into sectors. Every sector represents a chromosome with size proportional to its fitness value. The probability P_i of select an individual is depend on its fitness value as in equation 2.2

$$P_i = \frac{f_i}{\sum_{i=1}^{N} f_i} \tag{2.2}$$

Where f_i is the fitness values of the individuals and i=(1,2,3..N) where $f_i > 0$ and N is the population size [56]. This method has a problem when there is a big difference between the fitness values of the chromosomes. For example, if one chromosome has 90% and others have 5% or less the last one has a very limited chance to be selected as shown in figure 2.1.

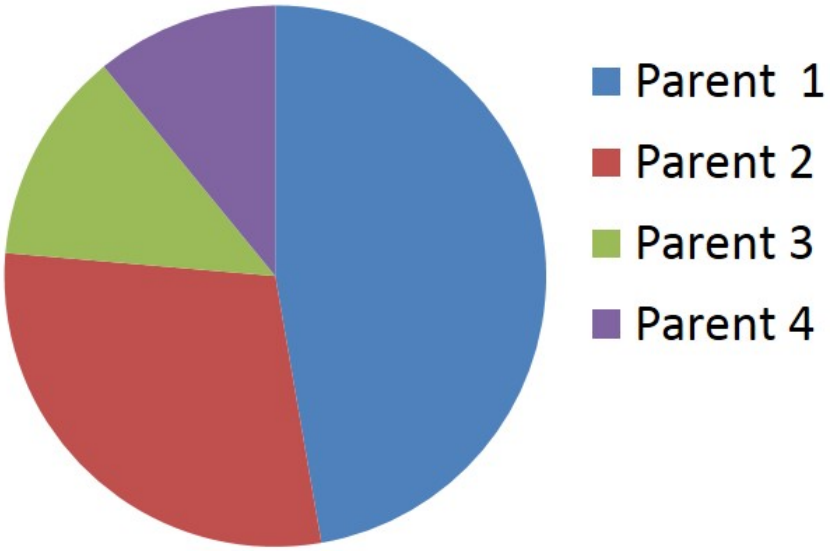

Figure 2.1: Roulette wheel selection.

***Rank selection*:**

This straightforward idea was introduced by Baker [8]. The idea is to sort the population from the best to the worst, then a rank number is given to each chromosome based on their ranking position. The worst fitness values takes rank 1 and the next one takes rank 2 and so on, until the best fitness takes rank N where N is the number of chromosomes in the population. All parents get a different probability to be selected even if they have the same fitness value [16]. By using this selection method, we can solve a number of problems faced by straightforward Roulette Wheel Selection in certain scenarios; figure 2.2 illustrates this by showing the difference before and after rank selection.

***Stochastic universal sampling (SUS)*:**

Stochastic Universal Sampling is a method which is similar to Roulette wheel selection; it is a sampling algorithm with zero bias and minimum spread which was introduced by [9]. It uses many equally-spaced pointers and spins the wheel once, instead of using just one pointer and spinning the wheel multiple times. The distance between each pointer is 1/N, where N is the number of pointers [74]. By having N pointers then each single spin has N winners instead of one.

***Steady state selection*:**

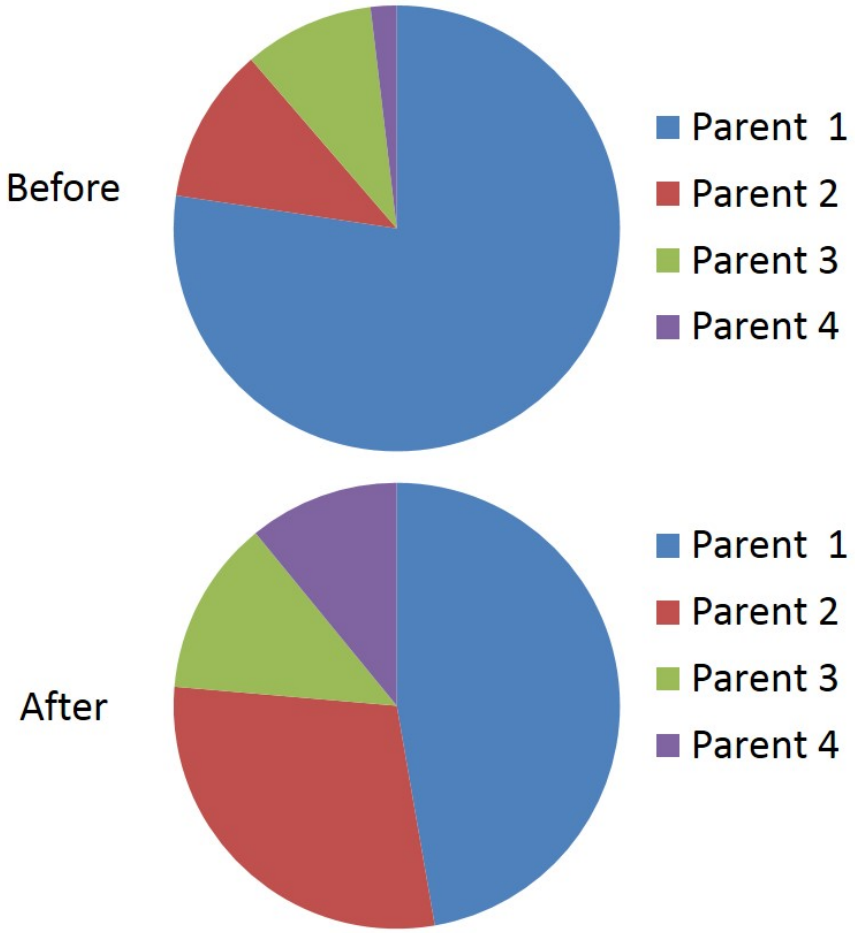

Figure 2.2: Rank selection.

Steady-state selection can be seen as a combined parent selection and survivor selection strategy, where the majority of solutions stay in the population from generation to generation, with only a small number being introduced and removed. In a steady-state approach, a small number of individual solutions in each generation are selected for creating new offspring, and at the same time a small number are chosen for replacement by the newly created offspring [110].

***Tournament selection*:**

Tournament selection is a simple and effective approach widely used as a GA selection method [64]. In tournament selection, K individuals from the population are randomly chosen, and then, the best individual from these N (the one that has the best fitness value) will become a parent. K is the selection pressure, or tournament size, that can be easily adjusted. For example, if the tournament size is 1, then this method corresponds to random selection; if the tournament size is high, then there

is a strong pressure towards selecting only the fittest individuals.

figure 2.3 shows an example of the process of tournament selection. Here, K= 3; therefore, A, E, and T were randomly selected with the values of 4, 5, and 7, respectively. Then, the one with the best fitness value was selected as the winner, which was A in this example (with the best fitness value) in the case of minimisation.

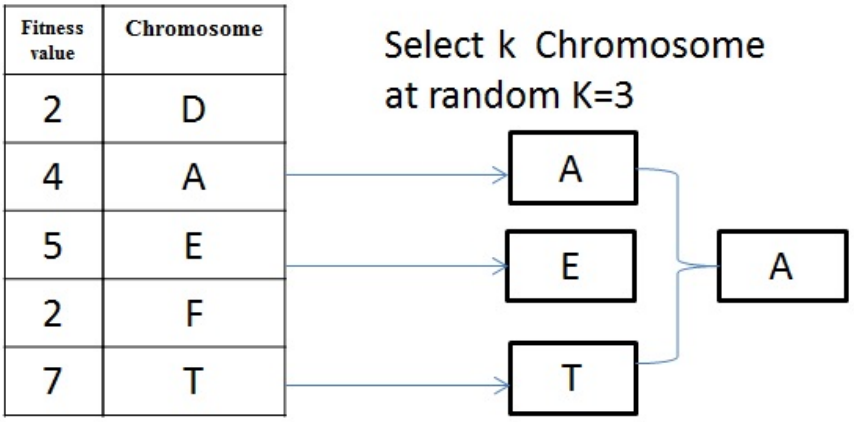

Figure 2.3: Tournament selection.

***Elitism*:**

The combined effect of selection and the other operators tended to lead to improved fitness from generation to generation. However, despite this, depending on the survivor selection scheme, it was possible that the best-so-far candidates in the population get lost. If elitism were used, the best chromosomes would have always been retained to stay in the next generation. Elitism has generally been found to speed up the improvement process and increase the performance of GAs [26, 133].

2.1.3 Generate offspring

In GAs, the contents of the population from generation to generation are strongly influenced by each of the operators. Broadly speaking, we can say that the selection mechanism influences the speed of improvement in average fitness, while the mutation and crossover mechanisms influence the balance between exploration and exploitation. In GAs, crossover generating offspring with more than one parent is

the primary tool for generating new offspring, while mutation plays a secondary role. Meanwhile, in evolutionary strategies, mutation is the primary mechanism for generating new offspring offspring [104].

Crossover operators are analogous to reproduction and biological mating. In a crossover operation, more than one parent is selected and one or more offspring are generated by using the genetic material of the parents. There are a variety of well-known and very generic schemes for the crossover operator, and the EA designer often chooses to implement a problem-specific crossover operator according to the requirements of the problem being addressed, and the genotype-to-phenotype encoding in use. In the following, we explain some of the generic approaches [17].

One Point Crossover **:** In this technique, a random crossover point is chosen to divide the chromosome into two parts, and then, the two parts of its two parents are swapped to get new offspring, as shown in Figure 2.4 [17].

In this technique, a random crossover point is chosen to divide the chromosome into two parts and then the two parts of its two parents are swapped to get new offspring as shown in figure 2.4 [17].

Figure 2.4: One point crossover.

Multi Point Crossover**:** In a multi-point crossover, many points are chosen to divide the chromosome into several segments, and these segments are swapped to produce the new offspring, as shown in Figure 2.5 [17].

Uniform Crossover**:** In a uniform crossover, chromosomes are not divided into segments; rather, each gene is treated separately. In this method, a coin is flipped

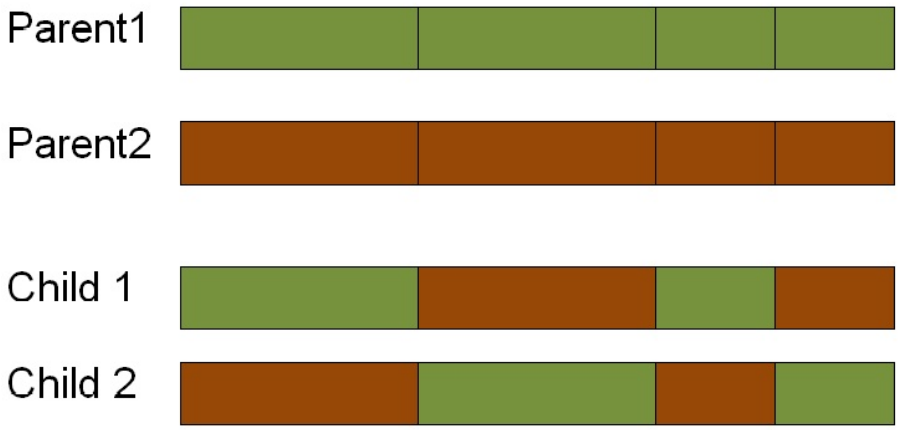

Figure 2.5: Multi point crossover.

(metaphorically) for each gene, to determine which parent will be used to provide the value for that gene in the offspring. We can also bias the coin towards one of the parents in order to have more genetic material in the child from that parent. The below figure in 2.6 depicts a typical example of this operation [17].

Figure 2.6: Uniform crossover.

***Arithmetic Crossover*:** In the arithmetic crossover operation, two parents are selected and linearly combined to produce two new offspring; we usually use this crossover method in the case of real-value encodings, according to the following equations:

$Child_1 = (a \times Parent_1) + ((1 - a) \times Parent_2)$

$Child_2 = ((1 - a) \times Parent_1) + (a \times Parent_2)$

where a is a random weighting number in the range [0,1] [119, 102].

***Three Parent Crossover*:** There are many ways in which crossover operators can be defined to use more than two parents. One such common method is as follows. Three parents are selected, and then, each gene from the first parent is compared with the same gene (same index) from the second parent. If both these genes are the same, then the gene is copied to the offspring in the same position. Otherwise, the

equivalent gene from the third parent is copied into the offspring at this position. This crossover is often used for binary-encoded problems [102].

***Order Crossover (OX)*:** Thus far, we have presented examples of operators that are suitable for k-ary encoding; this is where a solution is represented by a vector of numbers, and each gene in the solution can be any number from a pre-specified range. However, in many applications it is more suitable for the solution vector to be a permutation, or some other structure which is not k-ary. In such cases, we need to design different operators, which preserve the correctness of solutions. Here is an example of a crossover operator that is commonly used for permutation representations. This is Order Crossover (OX), first proposed by [24], extending the modified crossover of Davis [97]. First, we generate two random crossover points to be used in both the parents. Then, the offspring is created by copying the segment between these two points from the first parent directly to the first offspring. Then, the remaining positions in the offspring are filled by starting from the second crossover point in the second parent and copying the remaining unused numbers from the second parent to the first child. When we reach the end of the parent, we continue from position 1. See example below:

$Parent_1$: 9 0 | 5 3 4 | 3 8 7

$Parent_2$: 1 9 | 2 3 6 | 5 7 8

$stage_1$: . . 5 3 4 . . .

$stage_2$: 2 6 '5 3 4' 7 8 1

***Mutation*:** This is a vital operator, used in all forms of EA. Its chief role is accepted to be the maintenance of genetic diversity from one generation to the next. A mutation operator involves a single parent solution and usually involves making a small change (e.g. a change to just one of the genes) [68]. Some commonly used mutation operators are as follows:

***Bit-Flip Mutation*:** Bit-flip mutation is normally used when the solutions are encoded as binary strings. This operator simply works by choosing a gene at random and then flipping it (from 0 to 1 or from 1 to 0) to produce a new child [103].

***Swap mutation*:** Swap mutation is commonly used when the encoding is a permutation. In this operator, two different gene positions are randomly chosen, and then, the values at these two positions are swapped [103].

***Uniform Mutation*:** This mutation strategy works by changing the chosen gene value with a new uniform random value from the boundary specified by the user. This mutation operator is normally used in the case of integer- or real-value-encoded chromosomes [103].

***Reversing Mutation*:** In this operation, we choose a starting position at random, and then, the sequence of genes following this position is reversed. This is more commonly used for binary-encoded chromosomes [103].

***Inversion Mutation*:** In this mutation operation, two positions are chosen at random, and the sequence of genes between them is reversed to create a new offspring. This operator is used in the case of permutation encodings [103].

***Creep Mutation*:** This operator is suitable for real-valued solution vectors. We select a gene at random, and change its value by adding a random new value, usually small, so that the gene value creeps either upwards or downwards, but respecting the upper and lower bounds [103].

2.2 Approaches To Improve Evolutionary Algorithms

EAs have been used successfully in many applications. However, this success is counterbalanced by some limitations when they are applied to large and complex problems. One of these limitations is that the process becomes very time-consuming, since each generation has to implement fitness evaluations on every member in the population; when the fitness evaluation process is costly in time, this means that it may not be possible to reach good solutions on larger-scale problems in reasonable time. Premature convergence is another well-known example of the limitations of EAs; at the early stage of the traditional EA process, the population diversity may decrease, and become stagnant, leading to little or no further improvement despite the time available. Research have attempted to address these issues by focusing

on implementation of new methods in order to improve the traditional EA. More specifically, below we list some of these weak points in EAs, indicating the attempts that have been made to address them:

***Convergence speed*:** One of the greatest challenges associated with genetic algorithm design is accomplishing efficiency with regards to the space and time required for devising solutions of an adequate standard. With real-world applications, the function evaluation aspect of ancyan algorithm can represent a substantial drain on time. For instance, those tasked with developing contemporary engineering systems normally make use of costly computer analysis and simulation programs involving execution times that last anywhere between a few hours and several days just for one function evaluation alone [41]. These programs include but are not limited to finite element analysis (FEA), computational fluid dynamics (CFD), heat transfer and vehicle dynamic simulations. The speed of a genetic algorithm search can be increased [37] when using the following: hybridization in addition to parallelization [20], time utilization [36], and evaluation relaxation (function approximation). Observable increases in the search speed of genetic algorithms can be achieved through the use of local search techniques as well, which employ domain-specific knowledge [111], [52]. With hybrid optimization, it is possible to attain the best of both schemes [59], which explains why the popularity of genetic hybrids is growing in relation to resolving real-world issues. Varying search techniques have been combined with genetic algorithms in order to do so [78, 120, 128].

***The issue of diversity*:** The issue of diversity is particularly pertinent in Evolutionary Computation given that a premature diversity loss has the potential to result in premature convergence of the algorithm into parts of the search space where it is hard or impossible for the operators to find any escape towards better solutions. Further, a number of algorithms may not be capable of generating new genotypes even though diversity is high, meaning that they subsequently stagnate. One of many approaches to improving the management of diversity is the use of Memetic Algorithms. Memetic algorithms are essentially hybrid search techniques that are either population-based [29] or neighborhood-based (LS) [46]. Commonly

used population-based techniques are Genetic Algorithms and other Evolutionary Algorithms. Popular local search methods used in memetic algorithms are Tabu Search and Simulated Annealing (SA). The fundamental logic underpinning an MA is the combination of population-based and local-based search techniques in a way that helps get the benefits of both methods. That is, the exploration power of a population based method, coupled with the exploitation power of a local search method.

***Fitness function estimation*:** If the fitness evaluation function is costly in terms of time, or costly in other cyanways, these issues can potentially be resolved through the use of approximate function evaluation methods. This means replacing the fitness function with a faster and/or cheaper alternative. This alternative will almost certainly provide estimates of the real fitness. However, empirical experience has shown that this approach can be used effectively to make the search faster, simultaneously avoiding any detrimental impacts on the overall effectiveness of the search. The reason for this is that genetic algorithms are characterized by a large degree of robustness, in that they can accomplish convergence even in the presence of noise resulting from approximation. Cheaper approximate fitness assignment takes the place of more expensive accurate fitness evaluation; this involves a chromosome's fitness being guessed, for example, based on the fitness of its parents. Alternatively, function approximation can be used, which involves an alternate or simpler function taking the place of the full fitness function. Jin [48] conducted a broad survey on fitness approximation techniques. The choice of the right approximation model to precede the real function is paramount if the optimization issue at hand is to be resolved both effectively and efficiently. Neural network models have popularly been used in function approximation [54].

***Population size*:** With traditional EAs, population size is determined by users to a defined value at the outset of the search and does not change at any point during the run. The need to set this parameter value from the beginning raises some issues. On one hand, if the population size is not large enough, the algorithm can prematurely converge, and fail to achieve solutions that are of an adequate quality. On the other

hand, if the population size is excessively large, then the algorithm uses too many computational resources and will progress too slowly. Unfortunately, setting this parameter appropriately is quite challenging. Experts have demonstrated, in both empirical and theoretical terms, that a one-size-fits-all approach does not work in this respect. Rather, the best population size is determined on a case-by-case basis, influenced by the individual problem to be resolved. Additionally, many researchers and practitioners suggest that the best approach is setting the parameters differently at varying stages of the run. A common proposition in this respect is that the parameters should match the size and complexity of the problem at hand. Yet, making such a judgement is challenging in itself for real-world issues, which simply makes defining population size even more difficult. In order to sidestep this, some users turn to standard settings (50-100 individuals), using estimated numbers, or a trial-and-error approach testing different population sizes and choosing the one that is optimal. A substantial part of this is down to chance, and the probability that a user will select an inappropriate population size is high. For this reason, Smith and Smuda [101] made an attempt to apply the population sizing theory developed by Goldberg *et al.* [38] in a practical setting, combining the equation associated with their theory in the genetic algorithm. The rationale behind this was to take away the parameters, instead beginning with the population size in the first instance, given that evidence demonstrates that the genetic algorithm had no issues in relation to other parameters. The researchers thus proposed an algorithm for autonomously adjusting the population size progressively throughout the run.

***Optimizing the controlparameters*:** With EAs, one of the primary components in working out the balance between exploration and exploitation is the appropriate setting of control parameters. Alternative methods can be employed to monitor the way in which an EA is currently operating (e.g. the rate of improvement in average fitness from generation to generation) so as to change control parameters accordingly to optimise search performance. Fuzzy logic is capable of representing knowledge in non-exact ways, meaning that it can be employed for the purpose of reasoning on knowledge that is ambiguous or is not fully understood. This means that fuzzy

logic is a good option for adapting control parameters. It has permitted studies on methods of improving performance and the quality of solutions [88]. Moreover, it has been employed to combine the various heuristics and methods of seasoned experts into fuzzy logic models so as to adapt control parameters. The objective of this is usually to sidestep detrimental effects, including premature convergence, and to make the convergence of the genetic algorithm faster [44]. The optimization of a neural network can be achieved using a genetic algorithm in a multitude of ways. Further, it is capable of being applied to adjust the neural network weights [11, 67], topology [5], and learning rules [33].

***Large-scale optimization*:** A range of real-world issues originating in different disciplines represent large-scale optimization tasks. Several different evolutionary algorithms have been designed to address such challenges, but the performance of these systems usually worsens as the size of the challenge becomes more significant. Cooperative Coevolution (CC), which takes a divide-and conquer approach, has been incorporated into EAs to cope with larger dimensional problems. CC, first developed by Potter and De Jong [77], breaks a problem into smaller segments, each tackled by a separate EA. The researchers first incorporated CC into GA by breaking down an n-dimensional problem into n 1-D mini-problems for function optimization, the results of which demonstrated better performance than the GA without CC. Liu et al. [58] later attempted to tackle large-scale optimization issues using CC.

Approaches such as adaptive operators, adaptive population size, memetic algorithms, and other hybrid algorithms, all have their contribution in improving the performance of EAs in some cases. However, they all continue to face restrictions and limitations especially with large scale optimization problems. Therefore, research continues to develop and test variants of these approaches and new approaches to try to address the challenge of large-scale problems. One of the more promising threads of this reseach seems to be the use of CC, mentioned above. Incorporating CC into EAs seems to be reliably more effective than using the basic EA alone. Another, but quite different technique that seems to be reliably effective when used, is the use of fitness inheritance. We discuss both of these next in further

detail.

2.3 Co-evolution

As discussed above, since traditional EAs have some limitations when coping with high-dimensional problems, many researchers have started to give co-evolution more attention, since studies show far suggest that it can often offer performance advantages. Basically, coevolution involves combining two or more populations which evolve separately, but in which individuals from one population have some influence on the evolution in the other population [80, 112]. The way that evolution in two (or more) co-evolving populations affect each other is normally by way of the fitness function. If we have two evolving populations, A and B, then the fitness of an individual in population A will be affected by population B because population B provides aspects of the environment in which the individuals in population A are measured (and vice versa). For example, an individual in population A may represent some of the parameters of a function that needs to be minimized, while population B may influence other parameters of that function. There are two different type of coevolution, competitive and cooperative, which handle this influence in different ways. More details of these methods are provided next.

2.3.1 Competitive coevolution

Hillis [45] was the first to propose the idea of competitive coevolution. The idea of competitive coevolution is to approach a problem by using two populations. One population is a population of candidate solutions, in the normal way; the other population, called the test population, provides tests and challenges to the candidates in the solution population. The idea is that, as solutions evolve to be better, the tests evolve alongside to provide harder challenges. Several models have been designed for the detailed interaction between the solution and test populations [25, 4, 129, 90, 99]. In standard competitive coevolution, each individual in the solution population represents a complete solution; meanwhile, each solution in the test population is a standalone test. To calculate the fitness of an arbitrary individual in the solution

population, a number of tests from the test population are randomly chosen, and the fitness of the solution is calculated on the basis of its performance across those tests. After every individual in the solution population is evaluated, we can now evaluate the fitness of ancyan individual in the test population. This is done by considering the average fitnesses of the solution individuals that encountered each test. Individual tests that tended to provide a greater challenge (and hence reduced the average fitness of solutions) will be higher fitness in the test population.

2.3.2 Cooperative coevolution

Potter and DeJong [77] introduced the concept of cooperative co-evolution (CC) within the EA community. Their initial work found the idea promising, particularly for separable problems with independent components. However, the success of the original approach on non-separable problems is somewhat limited [122]. The basic idea of CC is to solve a large-scale (many dimensions) optimization problem by first decomposing it into several smaller sub-problems; second solving each of the sub-problems; third constructing a solution to the larger problem by combining the solutions to the smaller ones from other sub-components see figure 2.7 and algorithm 2.

Algorithm 2 The original CC framework

1: Decompose the decision vector into m lower dimensional subcomponents.
2: Set i = 1 to start a new cycle.
3: Optimize the i^{th} subcomponent with a certain EA for a predefined number of fitness evaluations (FEs).
4: **if** i < m **then**
5: i++, and go to Step 3.
6: **end if**
7: Stop if halting criteria are satisfied; otherwise go to Step 2 for the next cycle

A common-sense arrangement was made to periodically update these fixed values, making use of best-so-far solutions from other sub problems. Many variations on this idea have since been explored. In general, any CC approach tends to be configured to evolve solutions to sub-problems one after another, each time using the best parameter values for the previous sub-problem when moving on to the next. In a sensibly configured CCEA, there are mechanisms for more sophisticated interac-

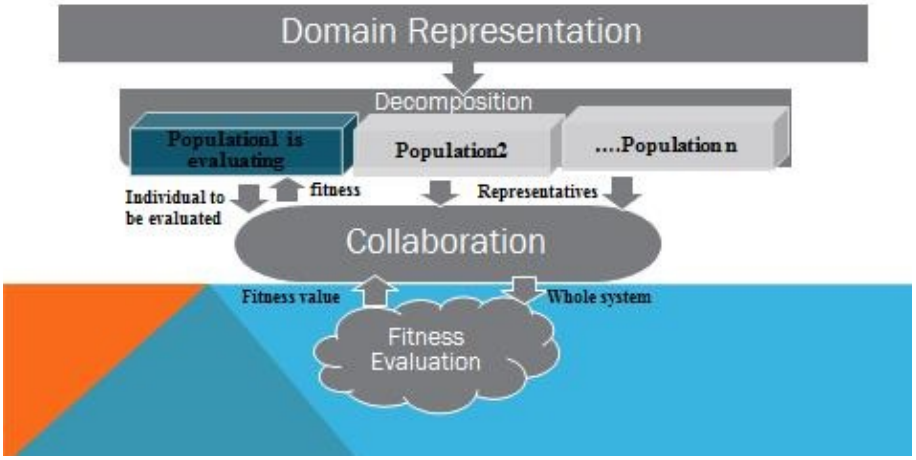

Figure 2.7: Processing in CCEA.

tion between sub-problems. For example, sub-problems might be solved in parallel, and there might then be brief spells of whole-problem evolution, or hill-climbing, rooted in the best parameters found from the sub-problems so far.

In the original framework [77] (and in the majority of subsequent research), the decomposition into sub-problems is cyansimply affected by a partitioning of the decision variable space. For example, if we need to optimize a function of 1,000 variables, v1,v2,...,v1000, then we can decompose this into two smaller problems: the first sub problem only considers variables 1—500; and the second only considers variables 501—1000. In the above example, when optimizing the first sub problem, the quality of solutions can only be estimated by assuming a reference set of fixed values for variables 501—1000. Similarly, when optimizing the second sub-problem, there needs to be a reference fixed set of values for variables 1—500. Typically, these reference sets may start at random, but be updated as the algorithm runs via the occasional evaluation of a judiciously composed full solution. The initialization and updating of these reference sets is a key dimension of variation among the literature of CC variants, along with other key aspects such as the details of decomposition and its adaptation. In this thesis, three strategies are used in the CC component

as the decompose strategy: contiguous, random and delta grouping. More details about each of these are provided in later sections.

2.4 Evaluation Is Expensive

Evaluation of fitness is a central aspect of all algorithms in evolutionary computation; evaluation is used to find the quality of a solution, and/or to compare the quality of two solutions, for example to distinguish a new solution (child) from a current one (its parent) in order to decide who will continue for the next generation. In some situations, and especially in large-scale problems, the evaluation process might be very expensive; in other words, simply to evaluate the quality of a single solution might take seconds or minutes of time on a processor.

For example, in co-operative co-evolutionary algorithms, when the problem is decomposed to sub problems, each sub problem is tackled via a dedicated population, and each single candidate solution in every population needs to have its fitness assessed. The standard way to do this is to form temporary complete solutions by combining an individual solution to a subproblem with other parts from other subpopulations, and then evaluating the complete solution. The fitness of a subproblem individual is a function of the fitness of the complete solutions that it has contributed to in this way. In this way, co-operative coevolution certainly does not reduce, and probably increases, the number of fitness evaluations required to perform well on the problem at hand. The challenge therefore becomes that of making the most of the available fitness evaluations. Also, another challenge for the algorithm design community is to find search strategies that provide sufficiently good solutions in as small as possible a number of fitness evaluations in general.

2.5 Fitness Inheritance

Fitness Inheritance (FI) is an entirely different approach to speeding up black box optimization, introduced by Smith, Dike and Stegmann [100]. FI tries to cut down on the need for real function evaluations, by making a proportion of evaluations

estimates, based on parents' fitnesses. The approach can be seen as a variation on a surrogate model based EA. In a surrogate model approach, a model of fitness (the surrogate is learned while the algorithm is running, and used some proportion of the time in place of the full fitness function, to provide a fast estimate. In fitness inheritance, this surrogate model approximation to fitness is formed by inheriting fitness values from one or more of an individual's parents [100, 2].

Fitness inheritance (FI) refers to the simple idea of making a fast estimate of an individual's fitness, rather than performing an accurate evaluation. FI can be potentially be deployed in any 'black box' algorithm context. Where c is an individual in the population, and eval(c) represents the evaluation of its fitness in the context of pseudocode, then we can affect the deployment of FI simply by replacing eval(c) with: With prob. PFI, run inherit(c) Otherwise run eval(c). The outline of fitness inheritance is as in algorithm 3.

However, sometimes care needs to be taken about the context of the evaluation. In the majority of cases, this means that we should ensure two things: (i) the initial population is always evaluated with eval, and (ii) updating of the 'best so far' solution, and similar book-keeping, are done on the basis of fitness values returned by eval. With the latter protections in place, we can then replace a proportion PFI of evaluations with fast inherited estimates. There are three commonly used approaches to producing an inherited fitness [32]. Where ch is the child of parents p1 and p2, and where fx represents the stored value of the previous evaluation of x (whether real or inherited), these are:

- ***Averaged inheritance*** **:** in which the estimate of the child's fitness is simplycyan the mean of its parents fitnesses:

  $$inherit(ch) = \frac{(f_{p1} + f_{p2})}{2}$$

- ***Weighted inheritance*** **:** this is in the spirit of averaged inheritance, but introduces weights according to the child's different similarities to its two parents:

  $$inherit(ch) = \left\{ \frac{(w_{p1}.f_{p1} + w_{p2}.f_{p2})}{w_{p1} + w_{p1}} \quad Where, \right.$$

$w_{pi} = w(x, x^{pi}) = 1 - \frac{d(x^h, x^{pi})}{(d(x^U, x^L)} and \quad d(x^i, x^j)$ is the Euclidean.

- ***Parental inheritance*** **:** this is an extreme version of weighted inheritance, in which the entire current population P is used to estimate the fitness of ch: $inherit(ch) = \frac{\sum w_p.f_p}{\sum w.p}$

How weights are calculated in the latter two approaches depends on the algorithm and encoding context, but the overall idea is to guess the child's fitness by appealing to a very simple linear approximation to the fitness landscape that is either based solely on the parents, or based on the entire population. Naturally, also, the obvious variations can be made to calculate inherited fitness for children of just one parent, or of more than two parents.

Algorithm 3 Pseudo code of fitness inheritance

```
p_i refers to inheritance proportion.
N_G refers to number of generation.
Initialize Population (P).
Evaluate (P).
t=0.
while t < N_G do
    Select s* individuals from P at random to be inheritance,according to p_i.
    Evaluate (P − s*) individuals in P by eval(c).
    Evaluate s* individuals in P by inherit(c).
    t= t + 1.
end while
```

As previously indicated, Smith, Dike and Stegmann [100] originated the concept of fitness inheritance (FI), at the same time introducing the first of the two simple inheritance schemes above. In the original work,cyan Smith *et al.* evaluated the concept on two problems. The first of these was the well-known (and these days very little used) ONEMAX test problem, in which the objective is simply to maximize the total number of 1s in a binary chromosome. Their second choice of test case was somewhat more interesting, being a realistic aircraft routing problem. On both problems they found impressive performance from FI. In [93], Sastry, Pelikan and Goldberg made some headway into a theoretical understanding of FI, untangling the relationship between population size and the fitness inheritance proportion on the ONEMAX landscape. Their conclusions, as regards separable landscapes, were that

FI could lead to significant efficiency enhancements, halving the required number of function evaluations, and yielding speed-ups of between 1.75-fold and 2.25-fold. Pelikan and Sastry [73] went on to find even more impressive speedups (e.g. 30-fold) when using FI in other cyanapplication and this has also been found in many other studies [87, 63, 92, 132, 10].

Although little work so far has combined FI with CC (we only know of [42]), FI is increasingly being explored outside the region of standard EAs, and being applied to nonstandard applications. For example in [35], FI is used as an essential tool for reducing a number of fitness evaluations when solving a problem of constructing robust continuous multi-objective test functions with various noise-induced features capable of uncovering truly capacity of the tested algorithms. Meanwhile, a modification of FI where fitness of a solution is based on its positional relationship with other particles has also demonstrated its benefits when used in PSO [76], typically suffering from overwhelmingly high number of fitness evaluations needed to find acceptable solution.

2.6 Evolution Strategies

Evolution strategies (ES), introduced by Rechenberg and Schwefel in the middle of the 20th Century [86, 96, 14], is an important subfamily of EC algorithms. We briefly review and describe ES here because it is a core algorithm at the heart of certain parts of the research in this thesis, specifically the algorithms we use for comparison. The paramount aspect setting ES apart from all other types of EC is the fact that the main strategy parameters have the capability of self-adaptation. The power of self-adaptation is such that the parameters which determine the evolutionary performance are not fixed and are instead subject to variation during the time the algorithm runs. Essentially, when an ES is operating, the strategy parameters and the solution parameters (i.e. the components of solutions) are subject to co-evolution. Prior to moving onto any more detailed discussion of this, it is prudent to first consider the details of a simple two member ES for the optimisation issue in relation to decreasing an n-dimensional function. The essence of this ES is offered

below as Algorithm 4:

Algorithm 4 Pseudo code of evolution strategy

1: t = 0.
2: Produce an initial point $({x_1}^t, ..., {x_n}^t) \in R$
3: **repeat**
4: Draw z_i from a normal distribution $N(0, \sigma)$, for all i $\in$ (1, . . . , n).
5: $y_i = x_i + z_i$ for all i $\in (1, ..., n)$.
6: **if** $(f(x^t) \leq f(y^t))$ **then**
7: $x^{t+1} = x^t$
8: **else**
9: $x^{t+1} = y^t$.
10: **end if**.
11: Set t = t + 1.
12: **until** Termination condition is satisfied.

Thus, it is possible that by looking at this basic algorithm, the basic principles and components of ES can be identified. To start with, on the whole, ES is employed for the purposes of real parameter optimisation. It works directly on the phenotype area that represents the real valued vectors. It is possible to describe the issue here as an objective function; namely, $R^n \rightarrow R$. Further to this, the mutation operator is the primary operation used to generate a new child. Because a present solution $\overline{x}^t$ manifesting as a vector with n length, the another new candidate $\overline{x}^{t+1}$ is introduced through adding a new random number z_i for i $\in (1, ..., n)$ to every one of the n elements. A distribution, whether it is Gaussian or normal, is employed with zero mean and standard deviation σ for the purposes of producing the random numbers. σ is also referred to as the mutation step size.

2.6.1 Self-adaptation

As was noted previously, the key aspect of ES is the capability of self-adaptation, which is shown in two ways. For the representation, each component of ES is made up of two pieces, the first of which is the object parameters $(x_1, ..., x_n)$ which stand for the component itself, and the second of which is the strategy parameters which consist of two lots of values σ and α. The former of these are the mutation step sizes and normally their number nσ are 1 or n, and α are interactions between the step sizes employed for all variables. Thus, the common depiction of components in

ES is as follows:

$x_1, ..., x_n, \sigma_1, ..., \sigma_n, \alpha_1, ..., \alpha_n$

The mutation operator is the other element where the self-adaptation capability can be depicted. This is the conversion of the strategy parameters for the mutation occurring throughout the runs of ES. On the whole, in the research to date, there have been two key methods of executing this self-adaptation. The first of these is the covariance matrix adaptation, which ascertains the mutation probability distribution. Second is the application of self-adaptive control parameters methods [86, 95]. The strategy parameters are clearly coded at the same time as the decision variables and they are updated by making use of an already established framework of rules in relation to updates applicable to individual generations. There are essentially three varying implementations, which are as follows:

1. ***Uncorrelated mutation with one step size, (Isotropic Self-Adaptation)*:** In this instance of uncorrelated mutation with one step size, homogeneous distribution is employed in order to mutate each x_i, meaning that there is a lone strategy parameter σ in each element. This σ is mutated every time step by increasing its number by a term of $(e^{\Gamma} * \Gamma)$, Γ is a random variable drawn on each occasion from a standard distribution with mean 0 and standard deviation τ. The mutation mechanism is therefore represented by these formulas:

$$\sigma^{'} = \sigma.e^{\Gamma.N(0,1)} \tag{2.3}$$

$$x_i^{'} = x_i + \sigma^{'} Ni(0,1) \tag{2.4}$$

Equation 2.3, N(0, 1) shows a draw from the standard distribution, but Equation 2.4 Ni(0, 1) shows an individual draw from the standard distribution for all variables i. Users can set the proportionality constant Γ, which is an external parameter. Normally, it is inversely proportional to the square root of the problem size, $\Gamma\ \alpha\ 1/\sqrt{}$ n. The parameter Γ can be viewed as a sort of learning rate [7], The justifications for mutating σ by multiplying with a variable with

a log-normal distribution are: there should be more modifications of a smaller size than a larger size; standard deviations must be higher than 0.0; the median should be 1.0; on average, mutation should be neutral, which necessitates the same probability of pulling a particular value and its reciprocal value for all values. Using this implementation, the representation for each component takes the form $x_1, ..., x_n$,σ.

2. ***Uncorrelated mutation with n step sizes,(Non-Isotropic Self-Adaptation))*:**

   The justification for employing n step sizes is the desire to handle dimensions in a non-homogeneous way. Specifically, it is anticipated that varying step sizes are utilised for individual dimensions i ∈ (1, . . . , n). This is due to the challenge presented if the fitness landscape does not have the same slopes for all directions on every axis. So, every basic chromosome $x_1, ..., x_n$ is extended with n varying step sizes, one per dimension. The altered mutation mechanism now looks like this:

$$\sigma_i^{'} = \sigma_i.e^{\tau^{'}.N(0,1)+\tau.Ni(0,1)} \tag{2.5}$$

$$x_i^{'} = x_i + \sigma_i^{'}.Ni(0,1) \tag{2.6}$$

   where $\tau\ \alpha\ 1/\sqrt{2n}$, and $\tau^{'}\ \alpha\ 1/\sqrt{2\sqrt{n}}$, the sum of two normally distributed variables is also normally distributed, meaning that the resulting distribution is still lognormal. The justification for this, based on concepts, is that the common base mutation leaves the potential open for a complete shift in the mutability, ensuring the protection of the degrees of freedom, at the same time as the coordinate-specific offers the choice to employ varying mutation strategies in varying directions. In this implementation, the component is shown as $(x_1, ..., x_n, \sigma_1, ..., \sigma_n)$

3. ***Correlated mutations*:** The justification for correlated mutations is to

permit the variable vector to take whichever direction by rotating them with a rotation covariance matrix C. It is the mutation step sizes and the angles between the dimensions which determine the entries of this matrix. Thus, the entry of the covariance matrix is $c_{ij(i\neq j)} = 1/2(\sigma_i^2\text{-}\sigma_j^2)$ tan $(2\alpha_{ij})$, if there is link between the i and j dimensions. The mechanism is now as follows:

$$\sigma_i^{'} = \sigma_i.e^{\tau^{'}.N(0,1)+\tau.Ni(0,1)} \tag{2.7}$$

$$\alpha_j^{'} = \alpha_j + \beta.N(0,1) \tag{2.8}$$

$$x^{'} = x + N(0,1) \tag{2.9}$$

Where τ α $1/\sqrt{2n}$, and $\tau^{'}$ α $1/\sqrt{2\sqrt{n}}$. The parameter β is fixed as 0.0873 (or 5°). The σ_i are mutated in exactly the same fashion as they were previously in Equation 2.5, the α_j are mutated with a supplement, usually distribution variation, which is not that dissimilar to the mutation of object variables. The mutation of object variables x now occurs by joining the variance taken from an n-dimensional normal distribution with covariance matrix C.

In correlated self-adaptation, as well as n mutation strengths, at best n.(n-1)/2 covariance α are part of every stand-alone solution. Therefore, altogether there are n.(n+1)/2 strategy parameters which have to be updated for the individual solutions. This means that this form of self-adaptive ES has the capability to acclimatize to problems characterised by correlated decision variables x. With correlated problems, in order for there to be a disconnect between the objective function and the new coordinate system, it is important to locate all pair-wise coordinate rotations in addition to the spread of solutions in all rotated coordinate systems. In this way, the component takes its general form. In the past, no crossover operators are used in ES, but all types of real coded crossover operators can be used to kit out ES. Parent selection in ES is unaffected by the fitness values. If a parent is required, then it

is selected at random alongside uniform distribution from the entire population at present. This selection is a key influence bettering the average quality of the entire present population. It seems that this selection is unable to perform this function itself, and this is instead completed by the survival selection in ES.

2.6.2 Survival selection

In ES, two survivor selection schemes exist, following creating λ products and working out their fitness values, the optimum μ of them are selected in a deterministic way, from only the products, known as (μ, λ) selection, or from the combination of parents and products, known as (μ+λ) selection. Both the (μ, λ) and (μ +λ) selection methods operate deterministically and are founded on rank as opposed to the absolute fitness values. The selection method normally employed in evolution strategies is (μ, λ) selection, which is viewed as preferable for a number of reasons: The (μ,λ) selection gets rid of all parents and (small) local optima can therefore be left, meaning that it is beneficial is the instance of multimodal landscapes. If there is no fixed fitness function, but alterations in time, the (μ + λ) selection maintains outdated solutions, so its ability to follow the moving optimum to a high standard is compromised. (μ + λ) selection negatively impacts the self-adaptation mechanism in terms of strategy parameters to operate well, because misadapted strategy parameters can remain for many generations when a component is characterised by good object variables and bad strategy parameters.

With ES, the pressure present in relation to selection is usually very elevated, which can be explained by the fact that theλ value for offspring is far greater than the μ value for parents. Normally, 1/7 ratio is the optimal.

2.7 Differential Evolution (DE)

2.7.1 Classical differential evolution:

DE was proposed by Storn and Price [106]; it is simple but has been effective and efficient in both real-world problems and benchmark functions [105]. The idea be-

hind DE was to develop a new mutation method by adding a weighted difference between two individuals into the third. Then, the crossover operation and selection will work between these mutated individuals and the corresponding individuals from the previous generation x_i to generate a new trial child x'_i. The classical DE has two important control parameters, namely the scaling factor (F) and the crossover rate (CR), which were always constant. Therefore, considerable research has been conducted in an attempt to improve the classical DE by introducing a self-adaptive strategy to adapt these parameters (CR and F). According to [106], DE can be described as follows:

1. ***Mutations*:**

$$v_i = x_{i1} + F.(x_{i2} - x_{i3}) \tag{2.10}$$

   where the vectors x_i are individuals and the candidate solutions in DE's population (NP) and each vector consists of the D-dimensional parameter, i = 1, 2, ..., NP, and in each index i, we need three other indexes, i_1, i_2, i_3, which are randomly different chosen and different from i. The scale factor F is a real number and is often set to 0.5.

2. ***Crossover*:** The crossover here will build a new trial candidate by the equation 2.11 and pass it to the selection method to decide whether it should become a part of the new generation.

$$Tr_i(j) = \begin{cases} v_i(j), & if \quad U_i(0,1) \leq CR \quad or \quad j = rand_j \\ x_i(j), & \text{otherwise} \end{cases} \tag{2.11}$$

   $rand_j$ is a randomly chosen index from v_i(j) to ensure that the trial vector does not duplicate the original vector x_i(j). U_i (0,1) refers to a uniform random number generator between [0, 1].

3. ***Selection*:** By using this equation, DE can provide a successful offspring for

the next generation.

$$x_i' = \begin{cases} Tr_i, & if \quad f(Tr_i) < f(x_i) \\ x_i \quad , & \text{otherwise} \end{cases} \tag{2.12}$$

In DE, there are five mutations schemes, but only two of them are used often (2.13), (2.15) because of their greater effectiveness and performance [79, 123].

$$v_i = x_{i1} + F.(x_{i2} - x_{i3}) \tag{2.13}$$

$$v_i = x_{best} + F.(x_{i1} - x_{i2}) \tag{2.14}$$

$$v_i = x_i + F.(x_{best} - x_i) + F.(x_{i1} - x_{i2}) \tag{2.15}$$

$$v_i = x_{best} + F.(x_{i1} - x_{i2}) + F.(x_{i3} - x_{i4}) \tag{2.16}$$

$$v_i = x_{i1} + F.(x_{i2} - x_{i3}) + F.(x_{i4} - x_{i5}) \tag{2.17}$$

2.7.2 Differential evolution with neighbourhood search (NSDE)

Neighbourhood search (NS) has been used successfully in combination with evolutionary programming (EP) to improve performance [126]. As the evolutionary processing in both EP and DE might be similar, in [124], the researchers proposed a new idea by combining NS with DE (NSDE). The experimental results in [124] showed that NSDE is better than and superior to DE. NSDE is similar to DE, but instead of the scale factor F remaining constant in the DE mutation, in NSDE, F will be replaced by the following equation (2.18):

$$F_i = \begin{cases} N_i(0.5, 0.5), & if \quad U_i(0,1) < fp \\ \delta_i \quad , & \text{otherwise} \end{cases} \tag{2.18}$$

where N_i (0.5,0.5) refers to a Gaussian random with a mean of 0.5 and a standard deviation of 0.5 and δ_i is a Cauchy random number with a scale parameter of 1. Often in NSDE, fp is set to 0.5.

2.7.3 Self-adaptive differential evolution (SADE)

SaDE was introduced in [82] to evolve and adopt the two best mutation strategies in DE equations (2.13), (2.15) by using the self-adapted control parameter (P) as in equation (2.19). During the evolution in SaDE, the scale factor F and the crossover rate CR are adjusted automatically. SaDE has been shown to be more effective than DE in terms of performance [82].

$$V_i = \begin{cases} Eq_4 & ,if \quad U_i(0,1) < fp \\ Eq_6 & , \quad \text{otherwise} \end{cases} \tag{2.19}$$

Here, in equation (2.19), P will be initially set to 0.5. Then, in the evaluation of all the offspring that go to the next generation by using both equations (2.13), (2.15), are recorded as ns_1 and ns_2. $nf1$ and $nf2$ record the number of candidates rejected by both the equations, respectively. After 50 generations in SaDE, P is updated according to equation (2.20) and the values of ns_1, ns_2, $nf1$, and $nf2$ are reset.

$$P = \frac{Ns1.(ns2 + nf2)}{Ns2.(ns1 + nf1) + Ns1.(ns2 + nf2)} \tag{2.20}$$

In the SaDE the scale factor F will be a Gaussian random number of 0.5 mean and, standard deviation 0.3, as (2.21):

$$F = N_i(0.5, 0.3) \tag{2.21}$$

SaDE also evolves and self-adapts the important value of the crossover rate CR rather than using a constant value of 0.5, as in equation (2.14). Each individual in the population assigns a value of CR, as in (2.22).

$$CR_i = N_i(CR_m, 0.3) \tag{2.22}$$

After every five generations in SaDE, a new value of CR_i in each individual in the population is updated according to equation (2.22). However, in each generation, the best value of CR that allows the offspring to enter the new generation in a array is recorded as CR_{Rec}. Here, CR_m is initially equal to 0.5. After 25 generations, the value of CR_m is updated according to (2.23).

$$CR_m = \frac{1}{CR_{Rec}} \sum_{i=1}^{CR_{Rec}} CR_{Rec}(i) \tag{2.23}$$

2.7.4 Self-adaptive differential evolution with neighbourhood search (SaNSDE)

If we compare the performance of Sa and NS separately over DE, we find that both have good effects and success with DE (SaDE and NSDE); therefore, in [123], to benefit from these advantages, the researchers combined these two algorithms as SaNSDE. SaNSDE is similar to SaDE except for some of the steps to update the values of F and CR_m. Therefore, first, the scale factor F or the equation (2.21) should be replaced with (2.24).

$$F_i = \begin{cases} N_i(0.5, 0.3) & , if \quad U_i(0,1) < fp \\ \delta_i & , \quad \text{otherwise} \end{cases} \tag{2.24}$$

fp here would be self-adapted as in SaDE in equation (2.20). Second, the crossover rate CR is similar to the SaDE strategy, but we also record the different weighted fitness values between the new offspring and its parent in the array (F_{Rec}) by the successfully recorded CR rate.

Hence, the equation (2.23) is changed to 2.25:

$$CR_m = \frac{1}{CR_{Rec}} \sum_{i=1}^{CR_{Rec}} W_i \times CR_{Rec}(i) \tag{2.25}$$

$$W_i = \frac{F_{Rec}(i)}{\sum_{i=1}^{F_{Rec}} F_{Rec}(i)} \tag{2.26}$$

2.8 Related Work

A considerable amount of research has been devoted to the challenge of large-scale optimisation, particularly in the promising area of co-operative coevolution. Many researchers have investigated the variations on the basic CC framework. Here, we review the prominent and relevant previous work on the following:

1. ***Non-separable problems:*** The original CC framework on non-separable problems is somewhat limited [122], because parameters interact with each other. In natural genetics [81], when two genes represent a feature at phenotype level then they are non-separable genes and they interact with each other. Also, if when the activation of one gene affects the activation of another gene, then these two genes are again said to interact with each other. Also [117, 50, 23] if two genes interact, the term epistasis is used to refer to that interaction. In genetic algorithms, this interaction between variables is also referred to as linkage between them [116, 117] Therefore, much work has been done trying to investigate interacting parameters in CC, and in particular aiming to ensure that interacting parameters are placed together in the same subcomponent in CC to share information. Yang, Tang and Yao [122] proposed their variant, DECC-G, in which the groups of parameters within a sub-problem adapted over time, in attempt to maximize the extent to which interacting parameters were present in the same sub-problem. This led to improved performance on non-separable problems, when compared to the standard framework.

   Cao *et al.* [21], also focus on non-separable problems in large scale optimization problems. In their work they tried to shed light on cases where the variables are fully non-separable, since the classical CC algorithms are ineffective in this scenario. They propose a new effective CC framework named CC-GLS with a new decomposition method called Sequential Sliding Window. They used this decomposition strategy when integrating global search algorithm Self adaptive Differential Evolution with Neighborhood Search (SaNSDE) and local search traditional Solis and Wets' algorithm (SW).the results of their algorithm

CC-GLS when compared with other algorithms used different decomposition method such as random grouping suggests it is a promising framework, especially for large scale fully non-separable problems.

Also correlation identification grouping was used in CC in [108] in order to improve CC 's ability to deal with non-separable problems; they used correlation identification grouping to capture the interactions between variables, and group closely correlated variables together into co-evolving subsets; these subsets were naturally self-adapting in size as the interactions between variables changed during the course of an algorithm run.

2. ***Large scale optimization:***

   Ye *et al.* [127], focus on the subcomponent optimizer in the CC framework in order to improve performance on large scale optimization problems. They proposed a new Hybrid adaptive optimization method by taking two very efficient differential evolution algorithms, JADE and SaNSDE, and making a hybrid of these two algorithms which they named HACC-D. At the begin of the evaluation process, the initial population is evolved with these two algorithms (JADE and SaNSDE) separately with an appropriate number of fitness function evaluations, then the algorithn which performs more effectively in this initial phase will be chosen to be the optimizer in this subcomponent for the rest of the evaluation process. Their results confirmed that the HACC-D algorithm is able to benefit from the advantages of the two algorithms SaNSDE and JADE. Regarding convergence speed and diversity in the population on large scale optimization, X. Zhang, W. N. Chen and J. Zhang [131] proposed a dynamic competitive swarm optimizer (DCSO) based on entropy of the population. In their work they divided the population into two sub-populations, one with the best fitness values and the other with the worst fitness values, dynamically at an early stagy of the evaluation process. The idea is that the worse sub-group will learn from the best sub-group. Their results for DCSO showed better and faster convergence speed when compared with competitive swarm optimizer (CSO). Min Han and Jianchao Fan [43] also tried to improve

the performance of particle swarm optimization (PSO) on large scale optimization by combining it with dynamic neighbourhood topology (DNT). Their algorithms PSO-DNT will divide the whole swarm into sub-swarms adaptively without decomposing. The dynamic neighbourhood topology DNT will assist the particles share the information from the neighbourhood particles, their results suggested that the proposed algorithm PSO-DNT has a good performance compared with PSO alone in solving a benchmark test functions. S. Mahdavi, S. Rahnamayan and K. Deb [60] tried to improve CC's performance on large scale non-separable problems by looking at the influence of the population initialization. Three strategies are used to initialize the population in this work, centre-based, hybrid random-centre normal distribution and central golden region. These strategies provide better solutions when using a centre point-based sampling method. The centre-point method was proposed by Rahnamayan and Wang [84], which is showing that the probability of closeness to an unknown solution for the centre point is significantly higher than any other points in the search space. Therefore, their results confirm that the new algorithm improves the performance of CC on the majority of the non-separable benchmark functions they tested. S. Mahdavi, M. E. Shiri and S. Rahnamayan [61] proposed a new decomposition method to recognise those variables. Their decomposition method was based on High Dimensional Model Representation (HDMR) to discover and group separable and non-separable variables before applying the optimization. They used their RBF-HDMR model in [98] to see the effect of the two variables. Their algorithm DM-HDMR (decomposition method based on High Dimensional Model Representation) results suggested, when applied on CEC'2010 benchmarks functions and compared with state-of-the-art algorithms, that DM-HDMR can efficiently solve large-scale optimization problems.

Yang *et al.* [121], proposed a new technique for Differential evolution called multiple parents guided (MPGDE) algorithms, instead of using just one parent as in the original DE variants. In their work they made an archive to reserve

the failed tried individuals during the evaluation process, because failed tried parents may nevertheless contain useful information; these parents are updated with a method called niching to preserve diversity. In order to direct individuals from the populations and the archive, a multiple top ranked selection strategy is used, where both individuals from the archive and the population are able to participate. They found that their algorithm was statistically superior to the compared state-of-the-art algorithms on most of the 20 CEC'2010 benchmark functions on large scale optimization.

3. ***Decomposition strategies:***

   Omidvar *et al.* [69] introduced a new decomposition strategy called Differential Grouping. This method aimed to group any interacting variables into the same sub-component. The algorithm applies an interaction test between all pairs of genes in the chromosome, and then places all interacting variables together in the same subcomponent. In contrast, if no interacting variables are detected then they will treat the variables as a separable problem. Their results using CEC'2010 benchmark functions shows that this automatic way of decomposing an optimization problem has the ability of grouping interacting variables with great accuracy on the majority of functions. In further work on differential grouping C. Peng and Q. Hui [75], they made some comparison between two decomposition strategies, random and differential grouping methods in CC. Their results show that after some improvement with differential grouping the speed of the approach is doubled, in terms of the number of evaluations required to meet the same fitness level. An example of CC in use in the context of Differential Evolution is provided by Trunfio [113], who investigated the use of several variants of search space decomposition in parallel during short learning phases, allowing adapting the size of subcomponents during the CC search. Meanwhile, Sayed *et al.* [94] proposed a technique for the identification of variable interaction aimed at limiting the number of interdependent variables among decomposed problems, a common theme in CC research also echoed in [71] in which they try to group interacting variables

into several subcomponents, especially for non-separable problems by Sayed et al particular technique, related to the concept of separability and based on random resampling and propagation of variable partitions, is reported to improve the ability of the decomposition-based optimization models in scaling up to 1000 dimensions.

Another effective decomposition scheme is proposed in Mei, Li and Yao [62], which is designed especially for the large-scale capacitated arc routing problem. In this scheme, route information about the best-so-far solution is actively employed in constructing the following decomposition to guarantee the non-decreasing quality of the decomposition. Differentiation between a vast number of possible decomposition is made based on special distances between the solutions which allow efficient identification of promising regions of the search. Liu *et al.* [57], tried to explore the relationships between search algorithms and grouping strategies in CC. From their comprehensive experiments, they aimed to shed light on the balance between the contribution of the algorithm itself and the contribution of the decomposition strategy (e.g. maybe a good decomposition strategy could make up for a poor algorithm). Overall they found that the quality of the algorithm was more important e.g. some good decomposition strategies failed to achieve better results when combined with less effective search algorithms.

Van den Bergh and Engelbrecht [115], who explored CC under the framework of particle swarm optimization (PSO). In their case, parameters were randomly assigned to sub-problems (this is also the case in [85]). In such random assignment, each sub-problem comprises a random selection of parameters scattered across the decision parameter vector, and this random assignment may change in every cycle of the algorithm. Among other key algorithm strategy choices in CC is the question of how, for any given sub-problem, to choose the reference set of fixed parameters those are not included in that sub-problem. The most common approach, naturally enough, is to use the best-so-far complete solution, updating these reference sets whenever a new best-so-far is found.

M. El-Abd [30] started from [69], and proposed hybrid cooperative co-evolution (hCC). Since we can group the interacting variables in one subcomponent and also group the non-interacting variables together in a different subcomponent, as in [69], he chose from the literature two distinct algorithms to be the optimizers for the groups, one for separable groups (artificial bee colony (ABC)), and Self-adaptive differential evolution with neighbourhood search (SaNSDE) for the non-separable groups. The results on CEC'10 benchmarks functions show that hCC has promising performance on different classes of the functions.

2.9 Summary

In this chapter, the key concepts related to the cooperative coevolution framework were introduced and explored. The chapter began by explaining evolutionary algorithms in general and its importance and success in solving optimization problems, and also discussed the challenges when facing large scale problems. The main steps of the standard co-evolution approach were presented. Moreover, we discussed the topic of expensive fitness evaluation, and approaches to address this, especially fitness inheritance. New approaches involving co-operative coevolution (CC) and fitness inheritace (FI) are the focus of the research described in later chapters in this thesis. However, our experiments and comparisons also involve other algorithms, particularly evolution strategies and differential evolution. Therefore this chapter also introduced and described the latter algorithms.

Chapter 3

Co-operative Coevolution with Fitness Inheritance for Large-Scale Optimization

In this chapter, we introduce a combination of Cooperative Coevolution (CC) and Fitness Inheritance (FI) and investigate its performance. Both CC and FI have been found to be successful on nontrivial and multiple test cases, and they use fundamentally distinct strategies. In this chapter, we explore the extent to which using both of these strategies at once provides an additional benefit. We combined CC and FI into a straightforward algorithm that we called CCEA-FI, which incorporates those design aspects of both of these strategies that seem effective (from the available literature so far), while being relatively free of additional parameters or complexity. CCEA-FI uses FI with averaged inheritance and CC with random parameter grouping and non-random grouping (contiguous grouping) schemes. The results of CCEA-FI were compared with those of both CCEA alone and the basic EA, on 20D, 50D, 100D, 500D, and 1000D. Moreover, the three FI approaches were compared on the Rastrigin function with 50D and 100D. The second part of this chapter discusses the testing of our algorithm on non-function optimisation; therefore, we chose bin packing problems as a candidate. A Mann-Whitney U test was used to test for significant differences between the algorithms considered in this chapter.

3.1 CCEA-FI Algorithm

The proposed algorithm and its testing, discussed in the next section, were aimed at accelerating performance on large-scale problems; comparisons with other algorithms were also made. In this section, we will describe our algorithm CCEA-FI step by step.

Step (0): parameter settings: set values for population size, set f_e which refers to the number of function evaluations, set p_i (refers to inheritance proportion). Set K (indicate the number of subpopulations).

Step (1): Generate a population of candidates on D dimensions with population size in the space and evaluate this population. Each candidate parameters Xi is initialized with a random value depend on the range of the function (from x to y).

Step (2): Update B which is the best or fittest member of the population.

Step (3): Terminate if a total of function evaluations have been done.

Step (4): partition the D dimensions into k subpopulation (P1,.., PK) where each subpopulation will evolve D/K parameters.

Step (5): for each sub population, j in 1,.., K.

- Initialize the sub population pj and
- Run select s^* individuals from pj sub population at random to be inheritance according to inheritance proportion p_i.
- Evolve sub population Pj and evaluate $Pj - s^*$ individuals in Pj by simulation and evaluate s^* individuals in Pj by inheritance.
- Until Ns real evaluations have elapsed.
- Update best-so-far B.

Step (6) Return to 4.

Our algorithm basically is as in Algorithm 5, in each subpopulation randomly will choose individuals to be inheritance in each generation this means that every individual has the same chance in the subpopulation to be evaluated by fitness function or inheritance by surrogate modelling fitness inheritance. The number of

Algorithm 5 The basic structure of CCEA-FI

```
1: Initialize Population P.
2: Evaluate all N members of P, and let B be the best member of the population
   (breaking ties randomly)
3: repeat
4:     Partition the D dimensions randomly into K sub-populations, P1,.., PD
   where each sub-population P_j will evolve D/K parameters
5:     for j in 1, .., K do
6:         repeat
7:             initialize sub-population P_j as N
8:             evolve sub-population P_j using FI with inheritance P_i
9:         until NS real evaluations have elapsed
10:    end for
11:    Update best-so-far B
12: return to 4
13: until a total of FE function evaluations have been done
```

the individuals chosen for their fitness to be inherited is controlled by the inheritance proportion P_i.

3.2 Fitness Inheritance Approaches

In some preliminary investigation, a comparison has been done on fitness inheritance approaches to decide which method should be used in our work. From the literature [32, 10] they found that averaged inheritance is more effective when compared with other methods. With this in mind, we decided to make some simple comparative tests between this and two other common methods. The average, weighted and parental fitness inheritance methods were there implemented and tested with basic CCEA-FI with non-random grouping scheme on 100 dimensions on the Rastrigin function.

1. ***Parameter settings*:**

   We test the three methods average, weighted and parental fitness inheritance on Rastrigin function, We used a population size of 100, and ran experiments 20 times independently, continuing for 50,000 function evaluations. We fixed K (number of subpopulations) at 2, and experimented with a range of FI values from 0 to 80%. The EA used in this part is a steady-state, replace-worst with binary tournament selection.

2. ***Summary*:** From figure 3.1 we can confirm the results of [32, 10] which found average fitness inheritance to perform well. Our test was on the Rastrigin function; it is clear that the averaged inheritance is better than weighted and parental inheritance on this problem. Therefore, we decided to use this method for testing our algorithm on the 4 functions which are commonly used in CCEAs, separable problems Rastringin and Schwefel and non-separable problems Ackley and Rosenbrock, to see the effect of our algorithm CCEA-FI.

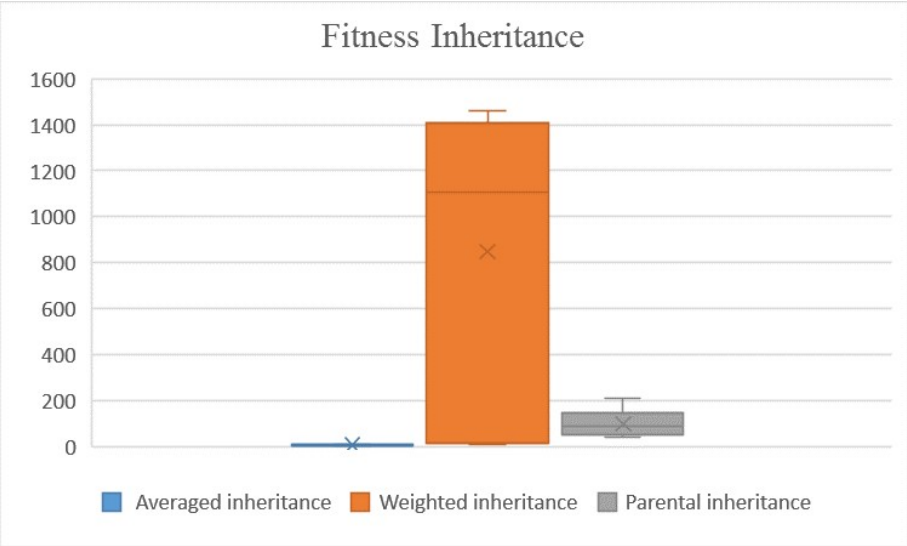

Figure 3.1: Testing different methods of FI on RASTRIGIN, 100D, 5×10^4 evaluations.

3.3 Evaluating CCEA-FI

3.3.1 Experiment study 1

This section describes the experimental results of our algorithm CCEA-FI with comparison between EA, FI and CCEA over CCEA-FI. We first combined FI with the basic performance of CC framework scheme so that we could sample the degree to which CCEA-FI performance, therefore, random scheme technique were also used and compared with continuous grouping method to overcome the drawbacks of CCEA-FI.

1. ***Test functions*:** In order to test CCEA-FI, we used the same four test functions that were used in [85], and we use (as in [85]) their 50-dimensional and 100-dimensional variants, and perform further experiments on their 20D, 500D and 1000D variants. For convenience these functions are defined below, where n is the number of parameters as in table 3.1 and landscapes of these functions are designed in [109] as in figure 3.2 .

Name	Functions	Type	parameters x_i in the range
RASTRIGIN	$f(x) = \sum_{i=1}^{n}[x_i^2 - 10\cos(2\pi x_i) + 10]$	Separable	$-5.12 \leq x_i \geq 5.12$
SCHWEFEL	$f(x) = 418.9829n - \sum_{i=1}^{n}(x_i \sin(\sqrt{\lvert x_i \rvert}))$	Separable	$-500 \leq x_i \geq 500$
ROSENBROCK	$f(x) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	non separable	$-30 \leq x_i \geq 30$
ACKLEY	$f(x) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) -$ $\exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)) + 20 - e$	non separable	$-32 \leq x_i \geq 32$

Table 3.1: Functions

2. ***Further details of algorithm and baseline experiments*:** We used a population size of 100, and ran experiments (all repeated 20 times independently) on each of the four functions at 20D, 50D, 100D, 500D and 1000D, continuing for 10^5 evaluations (20D, 50D and 100D), 2.5×10^6 evaluations (500D) and 5×10^6 evaluations (1000D). In all of the experiments reported here, we fixed K (number of subpopulations) at 2, and experiment with a range of FI values from 0 (plain CCEA) to 90 %.

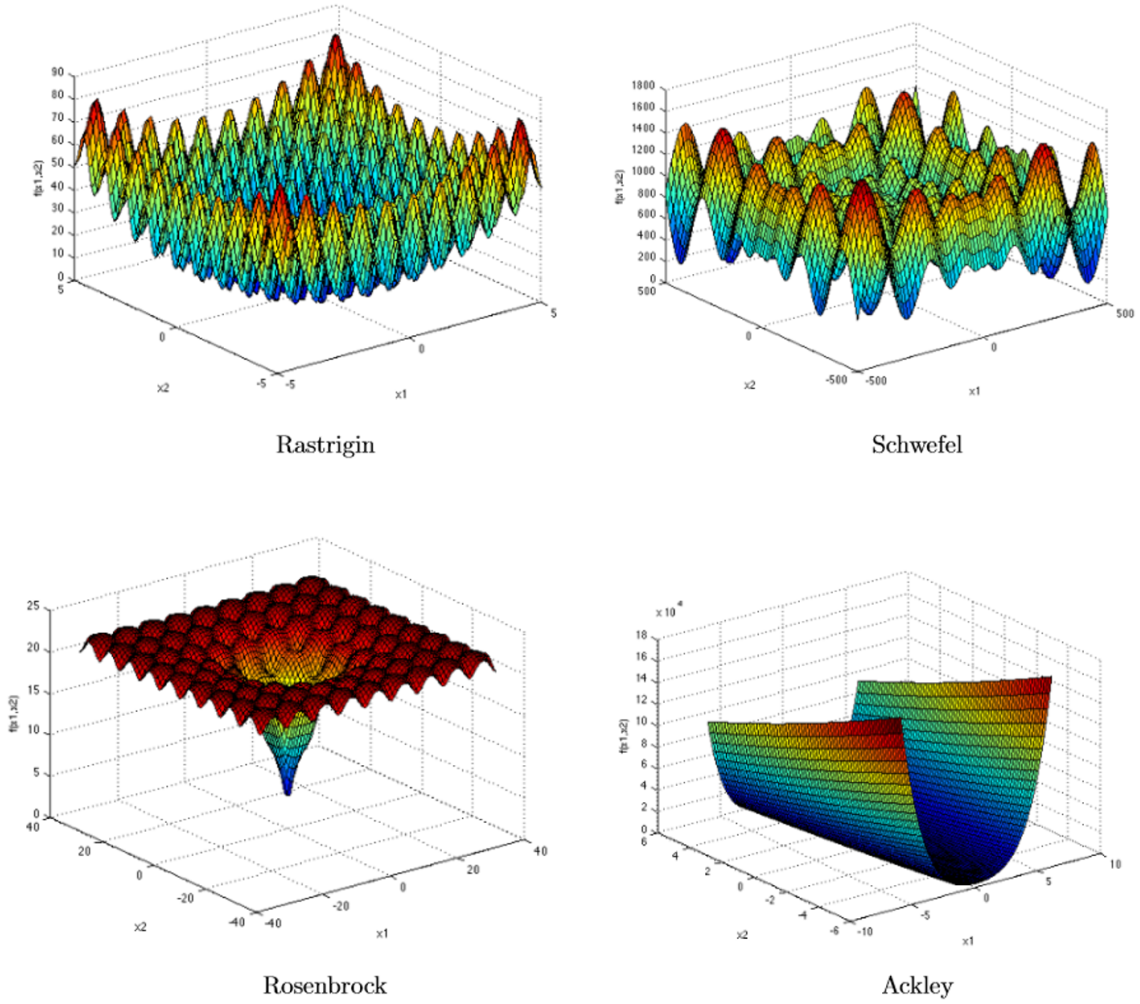

Figure 3.2: Landscapes [109].

Functions	EA	CCEA	CCEA-FI
Rastrigin	36.7854	1.5050	**0.1425**
Schwefel	169.4754	11.7503	**0.8203**
Rosenbrock	331.0421	283.5972	**171.5593**
Ackley	2.8829	0.5720	**0.1201**

Table 3.2: Mean results on 4 functions, 50D, 10^5 evaluation CC methods use contiguous grouping.

3. ***Results*:** We first focus on visualizing the benefits of both CCEA-FI over both CCEA alone and the basic EA, by looking at experiments comparing EA, CCEA, and CCEA-FI on 50D and 100D versions of the functions at 10^5 (total, real) fitness evaluations. These experiments use basic contiguous grouping in the CC cases i.e. the parameters 1-50 comprised one sub-population and parameters 51-100 comprised the other. In the tables and figures below we also will see the difference between the random and contiguous grouping schemes on 50D and 100D. The results of the basic contiguous grouping are on the 4 functions, as in table 3.2 provide the mean of 20 runs for Rastrigin, Schwefel, Rosenbrock and Ackley respectively.

Table 3.2 shows the results of the four functions with the 50D cases, in which CCEA without FI seems a considerable improvement on the EA alone; while CCEA-FI provides further significant improvement on the all functions (we only show the single parameter that gave the best mean result in our CCEA-FI). Our full results are detailed at appendix A. We can see that CCEA-FI variants provide better results than CCEA alone. However the Rosenbrock function continues to show distinctly unusual behaviour, and we see that clearly when we increase the dimension for example 100D, 500D and 1000D.

We now show results using random grouping in the CC methods on 20D and 50D, also we making a comparison between random and contiguous grouping results from our algorithm CCEA-FI on the 4 functions on 50D with the same experiment setting. Figures 3.3 to 3.6, show that CCEA-FI combination can provided a better results comparing to either CCEA or EA-FI alone on 4 functions even of lower dimension 20D.

In the table 3.3 we only show the single parameter that gave the best mean result

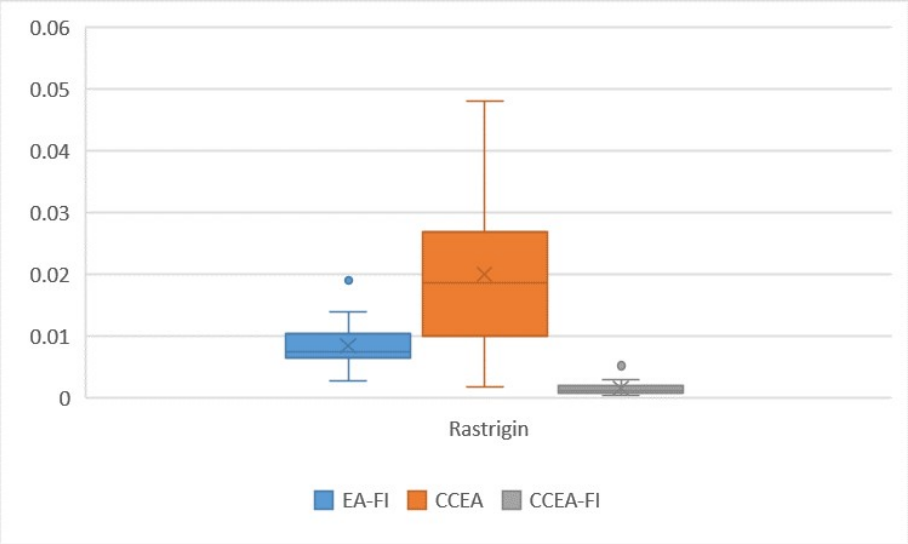

Figure 3.3: Results on Rastrigin, 20D, 10^5 evaluations, and CC methods use random grouping.

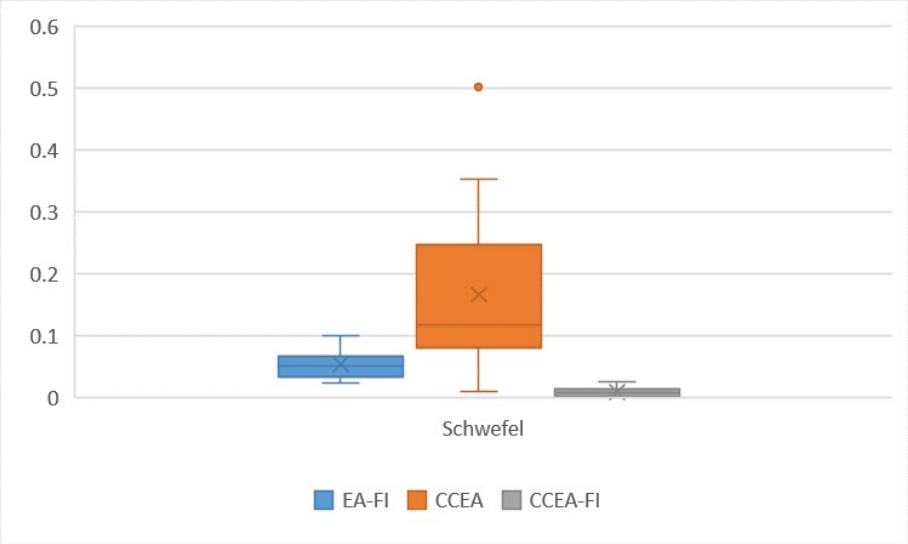

Figure 3.4: Results on Schwefel, 20D, 10^5 evaluations, and CC methods use random grouping.

Functions	CCEA-FI Contiguous G	CCEA-FI Random G
Rastrigin	0.142510258	**0.015302448**
Schwefel	0.820364663	**0.075102988**
Rosenbrock	171.5593012	**87.06613618**
Ackley	0.120186142	**0.035319179**

Table 3.3: Comparisons between random and contiguous grouping using CCEA-FI on 4 functions 50D, 10^5 evaluations.

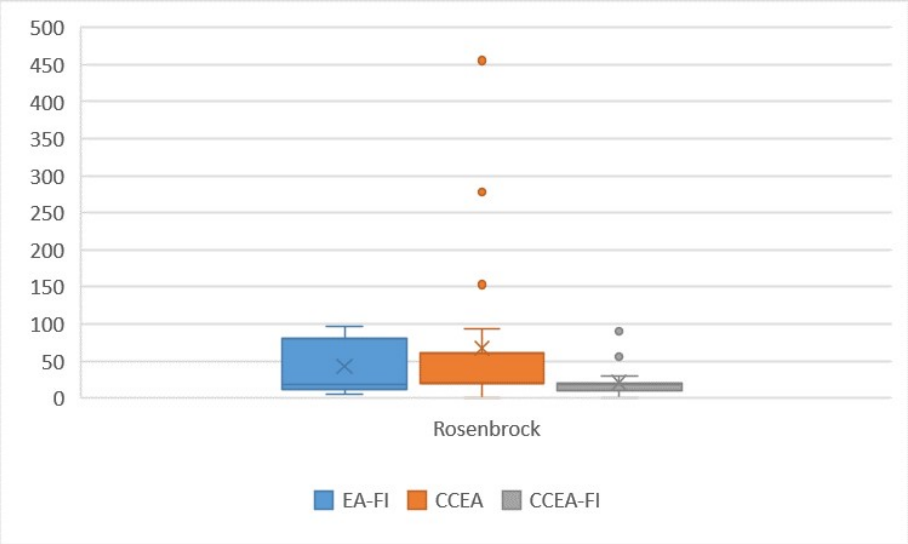

Figure 3.5: Results on Rosenbrock, 20D, 10^5 evaluations, and CC methods use random grouping.

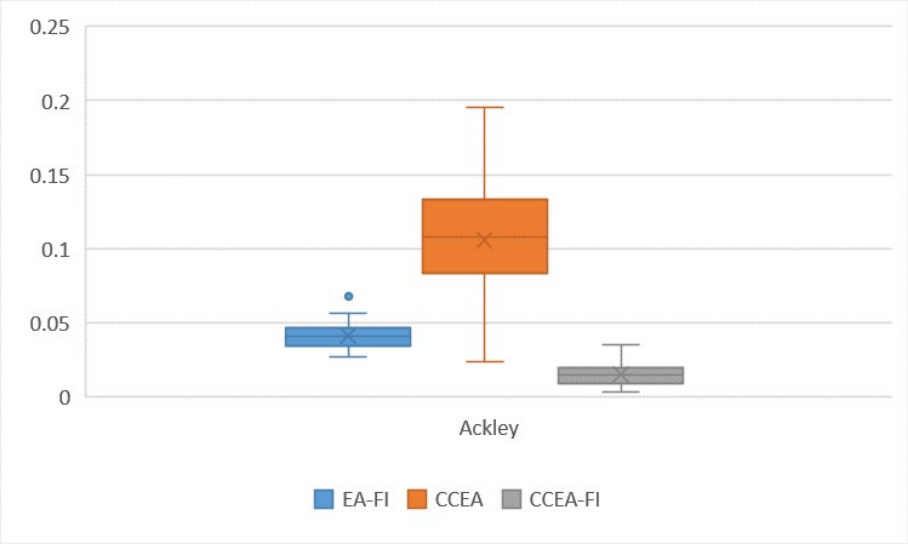

Figure 3.6: Results on Ackley, 20D, 10^5 evaluations, and CC methods use random grouping.

Functions	EA	CCEA	CCEA-FI
Rastrigin	70.79572794	9.261418907	**1.075074564**
Schwefel	1154.523561	92.64353015	**7.426763263**
Rosenbrock	22242.34955	1167.323001	**516.4006409**
Ackley	4.440522724	1.415020322	**0.318114575**

Table 3.4: Mean results on 4 functions, 100D, 10^5 evaluation CC methods use contiguous grouping.

in our CCEA-FI with either contiguous grouping or random grouping. We find from table 3.3 and figures 3.3 to 3.6 that CCEA-FI seems to be highly effective, especially when a random grouping scheme is used in the CC component in both 20D and 50D on the 4 functions.

Table 3.4 also provides the mean of 20 runs for Rastrigin, Schwefel, Rosenbrock and Ackley respectively, but now for 100D with 10^5 evaluations. In the case using contiguous grouping, again we only show the single parameter of FI percentage that gave the best mean result in our CCEA-FI.

From figures and tables on dimensions 20D, 50D and 100D show the results in which CCEA without FI seems a considerable improvement on the EA alone, while CCEA-FI provides further significant improvement, especially at higher rates of fitness inheritance.

The main exception to this pattern is the results on the Rosenbrock function, which we visualize in figures, again for the 100D, 10^5 evaluations case. Again we see the significant improvement of CCEA over EA, and further improvement as we start to introduce FI. However on the Rosenbrock problem, FI values above 50% led to very poor performance on 100D.

We now show results using random grouping in the CC methods on the 100D functions, this time omitting the EA results, which were always considerably worse, but including the results for EA-FI (i.e. using fitness-inheritance, but not involving co-evolution). As before, we show the CCEA-FI results for a variety of FI parameters ranging between 10% and 90%, but for EA-FI we only show the single parameter that gave the best mean result in our EA-FI experiments, our full results are detailed at appendix A. Hence, these plots indicate, for the 100D cases, the relative performances of CC, FI, and the CC-FI combination, with the basic EA omitted but

clearly understood to be considerably worse than any of the combinations shown.

As we can see in figures 3.7 - 3.10, the CCEA-FI combination can always yield results that are superior to either CC or FI alone. As ever, the Rosenbrock function provides some anomalous signals, but still showing CCEA-FI better than either CC or FI alone for 7 of the 9 CCEA-FI parameterisations.

We can now consider table 3.5 which summarises the previous figures by taking the best mean results from each algorithm (we only show the single parameter that gave the best mean result in CCEA-FI or EA-FI). It is clear that the CCEA-FI

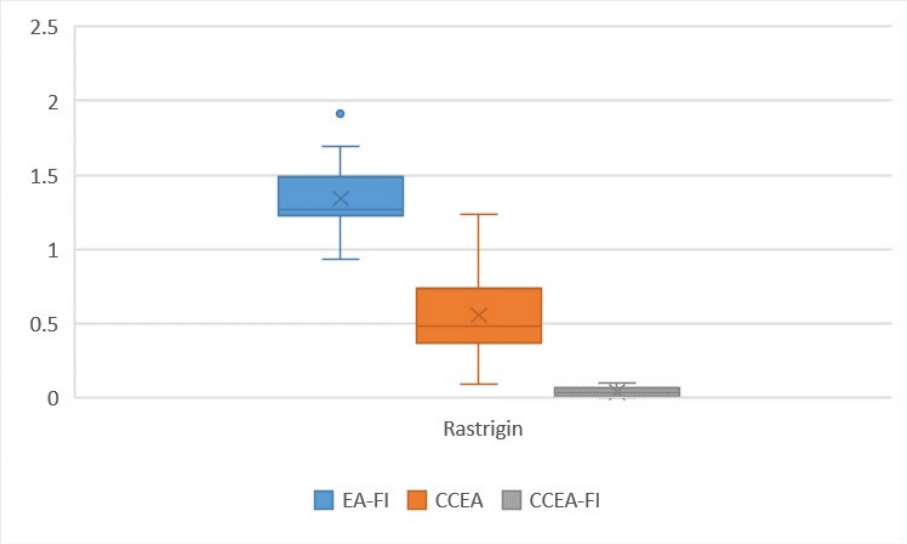

Figure 3.7: Results on Rastrigin, 100D, 10^5 evaluations, and CC methods use random grouping.

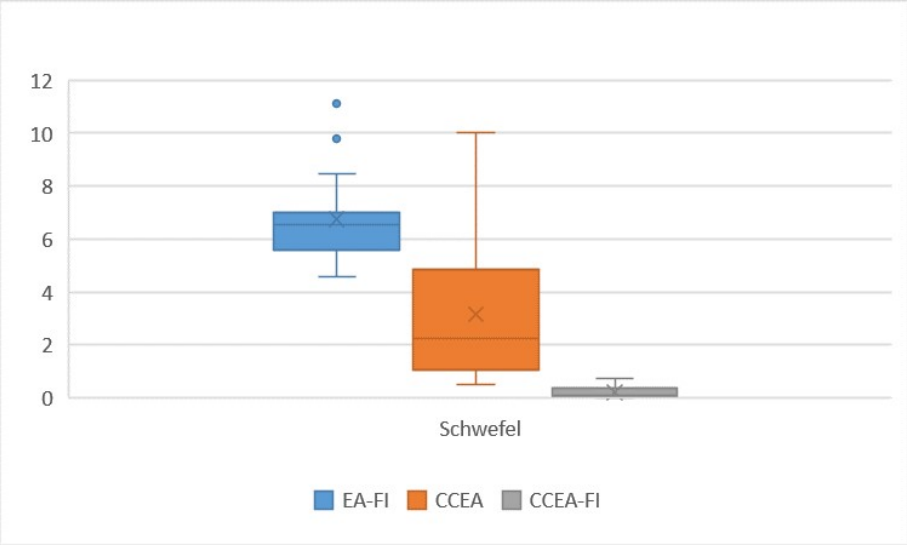

Figure 3.8: Results on Schwefel, 100D, 10^5 evaluations, CC methods use random grouping.

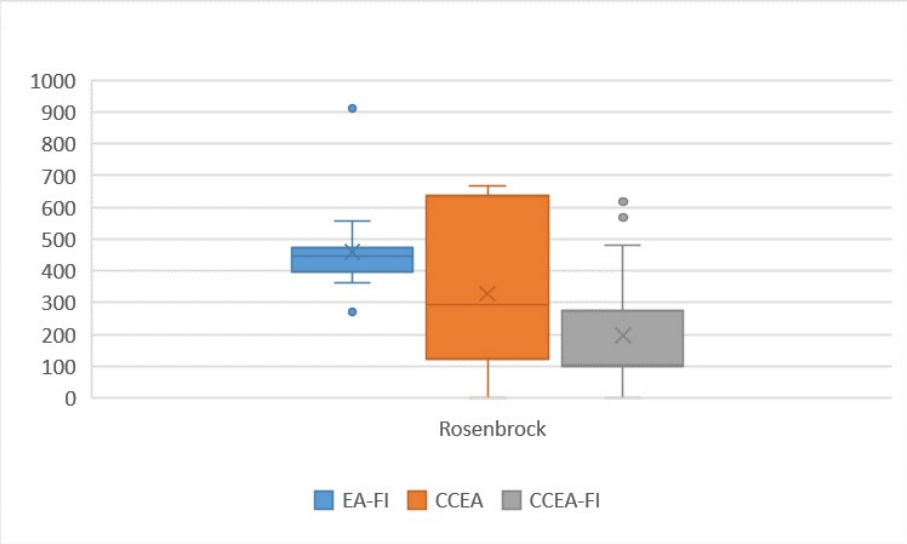

Figure 3.9: Results on Rosenbrock, 100D, 10^5 evaluations, CC methods use random grouping.

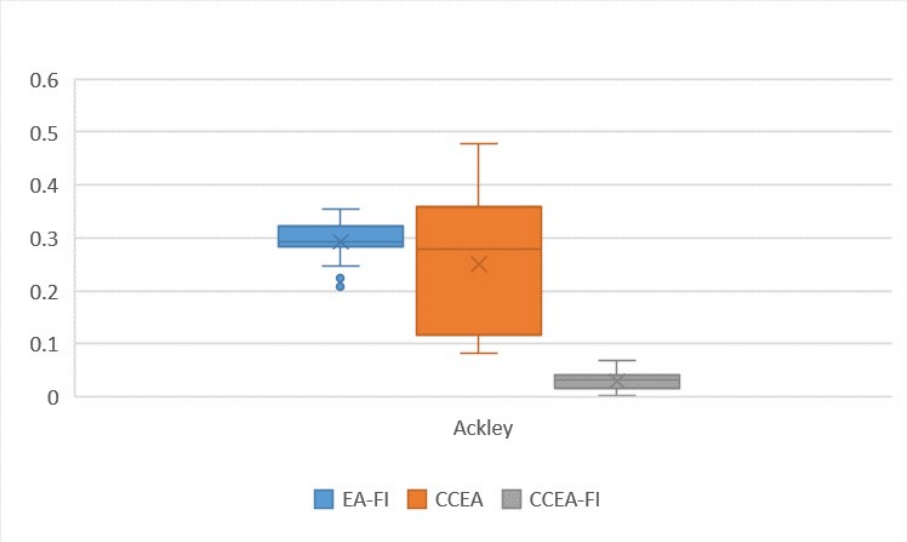

Figure 3.10: Results on Ackley, 100D, 10^5 evaluations, CC variants use random grouping.

Functions	EA-FI	CCEA-FI Contiguous G	CCEA Random G	CCEA-FI Random G
Rastrigin	1.279083727	1.075074564	1.004464035	**0.030195445**
Schwefel	6.743000336	7.426763263	4.381846259	**0.177473264**
Rosenbrock	459.6672477	516.4006409	331.9520982	**215.0991237**
Ackley	0.293446274	0.318114575	0.247710572	**0.031163218**

Table 3.5: Summarise previous figures by the best mean results so far by the 4 algorithms on 4 functions by 100D, 10^5 evaluations by random and contiguous grouping.

combination can always yield results that are superior by using random grouping in CC instead of contiguous grouping. In the table 3.5 the letter G in for example (CCEA-Random G) indicates grouping.

We now turn our attention to the 500D and 1000D test cases. In these cases, the better CCEA-FI results were obtained for lower percentages of fitness inheritance. So we take this opportunity to limit the presentation of CCEA-FI variants to 10%–50%, and are therefore able to fit in the best and worst of 20 runs in addition to the mean results. In all of these cases, the results for EA-FI (i.e. without CC) were always significantly worse than the CCEA-FI variants, and so are omitted from these plots.

As with the 1000D cases, we can see that CCEA-FI variants provide better results than CCEA alone. However the Rosenbrock function continues to show distinctly unusual behaviour, and we see that only the 10% case shows a better mean for CCEA-FI over CCEA, although all of the CCEA-FI cases for Rosenbrock show a better best of 20 result than CCEA alone. Note that the Rosenbrock plot shows the log values of the results.

Now we can consider table 3.6 which summarises the previous figures by looking the best mean results from each algorithm on 500D (we only show the single parameter that gave the best mean result in CCEA-FI). It is clear that CCEA has worse performance than CCEA-FI on the 4 functions.

Finally we show the results on 1000D versions of the test functions. Our display of results in these cases matches the display for 500D cases, i.e. we show only CCEA and CCEAFI, with log values shown for the Rosenbrock function.

Again, particularly by inspection of Figures 3.7 - 3.18, we can see that CCEA-FI

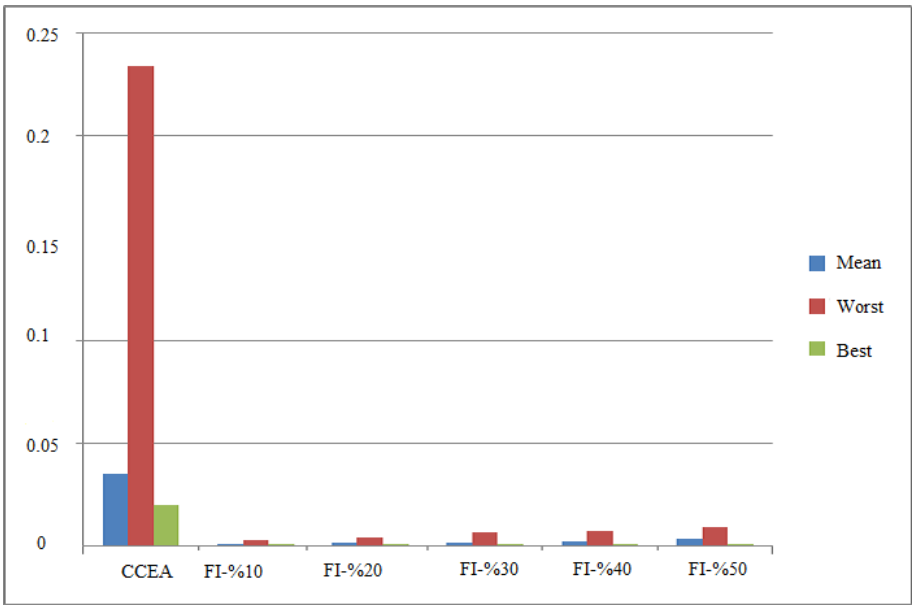

Figure 3.11: Best, mean and worst (of 20) results on Rastrigin, 500D, 2.5×10^6 fitness evaluations, showing CCEA and CCEA-FI results; CC variants use random grouping.

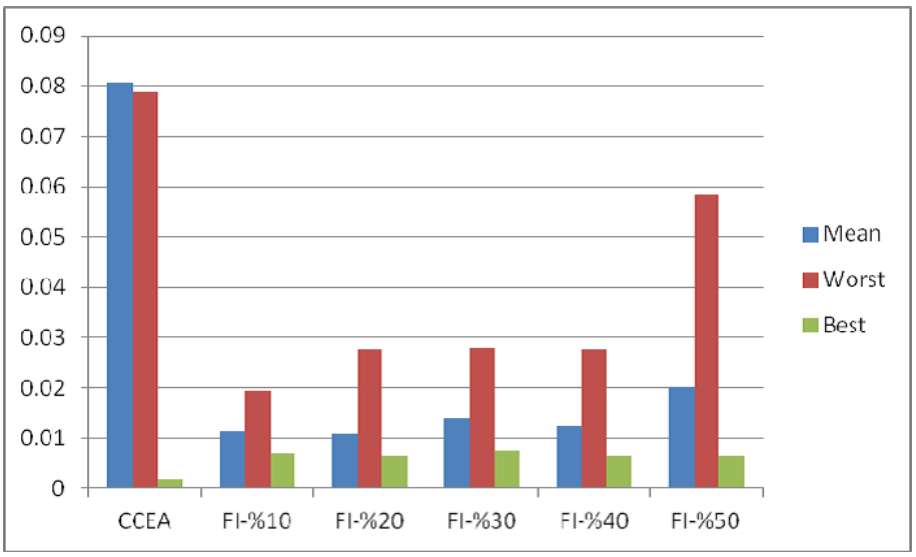

Figure 3.12: Best, mean and worst (of 20) results on Schwefel,500D, 2.5×10^6 fitness evaluations, showing CCEA and CCEA-FI results; CC variants use random grouping.

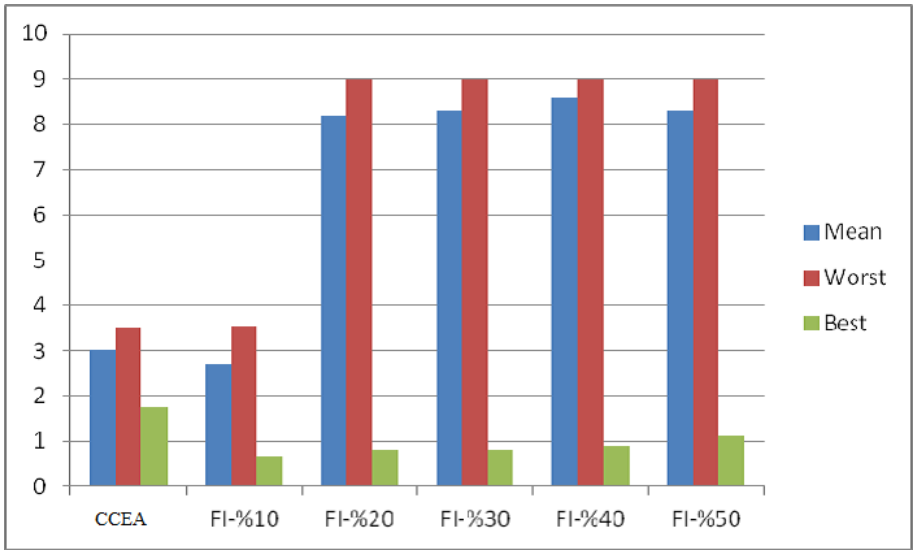

Figure 3.13: Log values of best, mean and worst (of 20) results on Rosenbrock, 500D, 2.5×10^6 fitness evaluations, showing CCEA and CCEA-FI results; CC uses random grouping

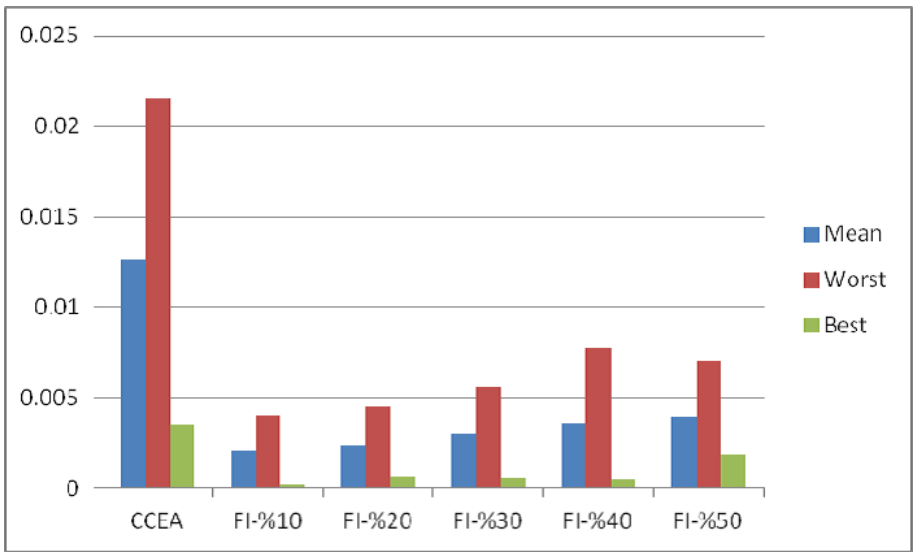

Figure 3.14: Best, mean and worst (of 20) results on Ackley, 500D, 2.5×10^6 fitness evaluations, showing CCEA and CCEA-FI results; CC variants use random grouping.

Functions	CCEA	CCEA-FI
Rastrigin	0.03484	**0.0009383**
Schwefel	0.0807805	**0.011004**
Rosenbrock	1011.82936	**491.0903324**
Ackley	0.01263579	**0.002049139**

Table 3.6: Summarise previous figures by the best mean Results of CCEA over CCEA-FI on 4 functions by 500D, 2.5×10^6 fitness evaluations, CC variants use random grouping.

provides considerable gains in solution quality over CCEA alone. In the Rosenbrock case, Figure 3.13 and 3.17, this is clearly the case when we look at the best of 20 results, indicating that CCEA-FI enables the EA to access superior results more readily than CCEA alone, but clearly with a high variance which dampens the quality of the mean performance of the CCEA-FI variants. The CCEA-FI variants with best performance on the 500D and 1000D tests are invariably those with FI percentages at 10%–30%, with 10% usually leading to the best mean result, but best result often appearing at 20% or 30%. This is in contrast to the results for 100D, where (with random grouping), better results were obtained for 30%–60% FI. We can also report that on 50D cases the best results were in the region of 60%–80% FI.

We now consider table 3.7, which summarises the previous figures by taking the

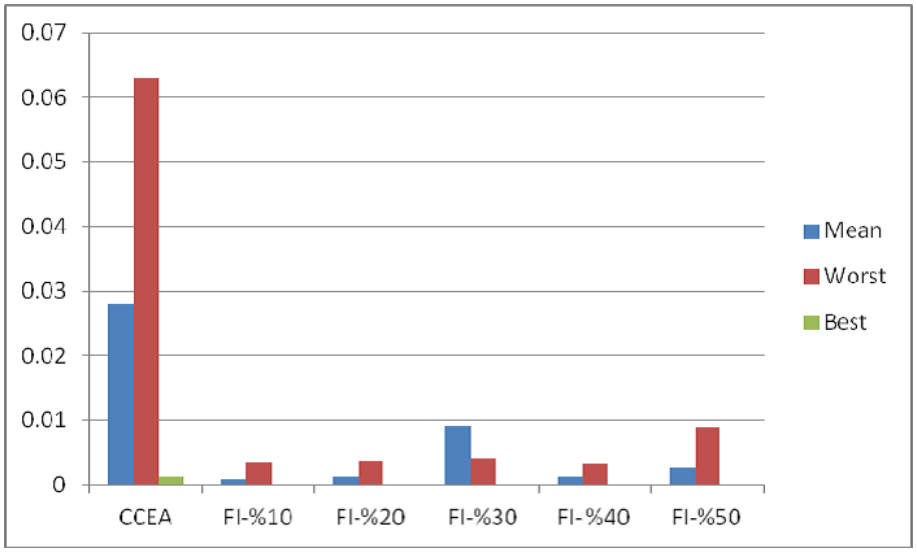

Figure 3.15: Best, mean and worst (of 20) results on Rastrigin, 1000D, 5×10^6 fitness evaluations, showing CCEA and CCEA-FI results; CC variants use random grouping.

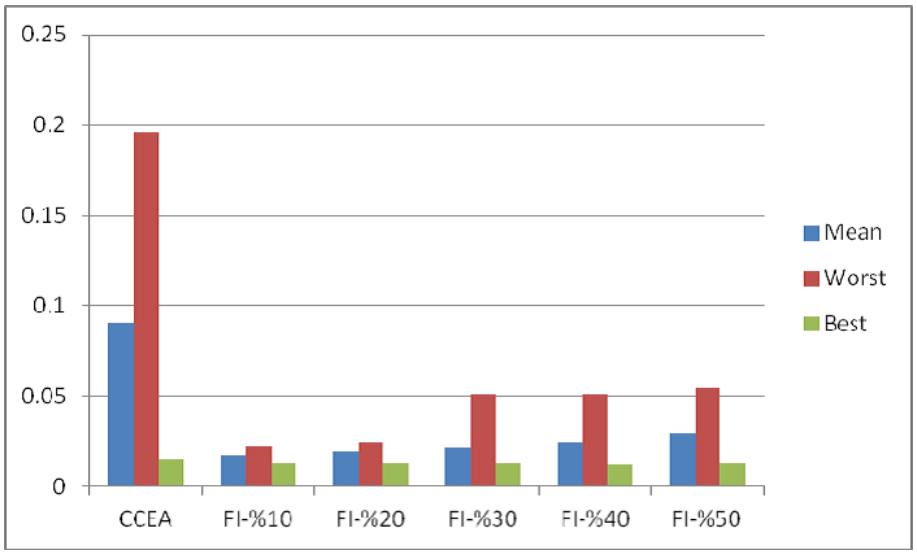

Figure 3.16: Best, mean and worst (of 20) results on Schwefel, 1000D, 5×10^6 fitness evaluations, showing CCEA and CCEA-FI results; CC variants use random grouping.

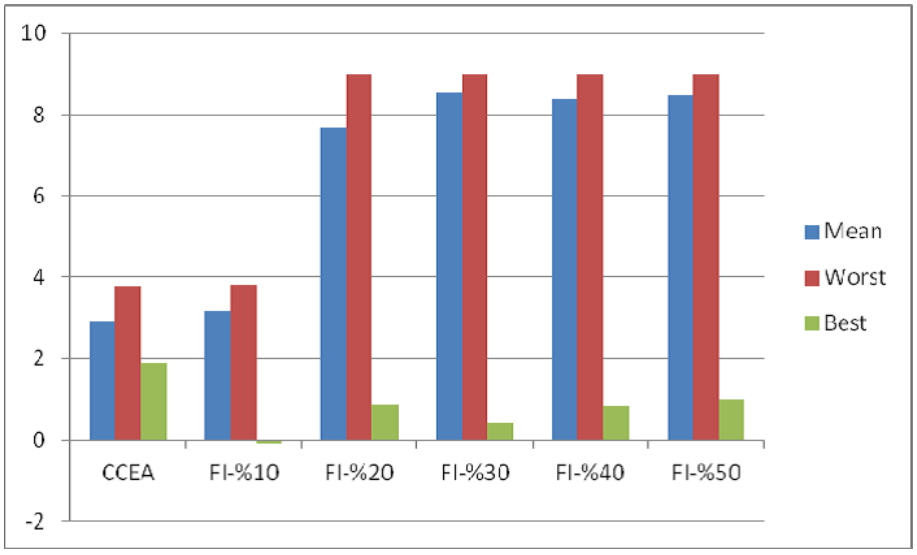

Figure 3.17: Log values of best, mean and worst (of 20) results on Rosenbrock, 1000D, 5×10^6 fitness evaluations, showing CCEA and CCEA-FI results; CC variants use random grouping.

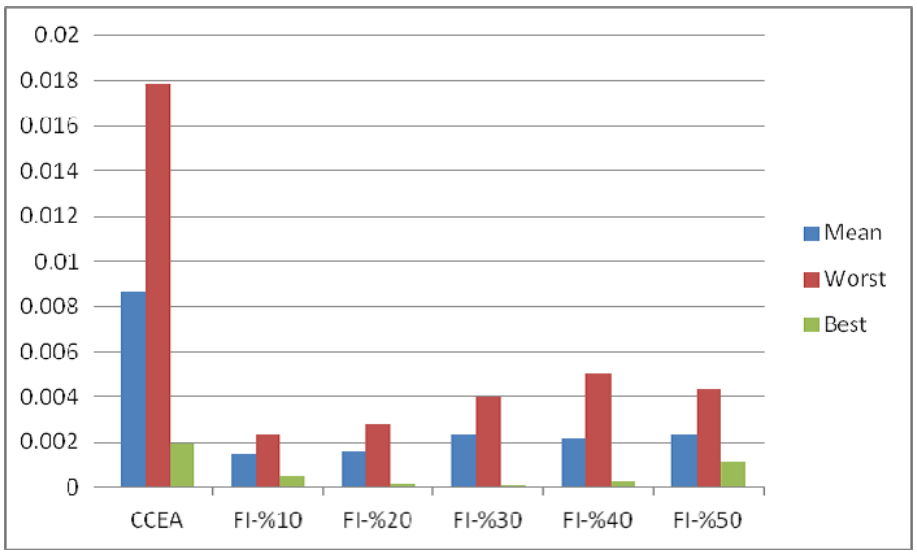

Figure 3.18: Best, mean and worst (of 20) results on Ackley, 1000D, 5×10^6 fitness evaluations, showing CCEA and CCEA-FI results; CC variants use random grouping.

best mean results from each algorithm on 1000D (we only show the single parameter that gave the best mean result in CCEA-FI). CCEA-FI gives better fitness values than CCEA alone for the most part. However on the Rosenbrock problem, FI values above 10% led to very poor performance on 1000D. In the next chapter we will investigate why FI provides worse performance when increasing the percentage especially on high dimension.

Functions	CCEA	CCEA-FI
Rastrigin	0.02804	**0.0008791**
Schwefel	0.0904649	**0.01710602**
Rosenbrock	**848.396831**	1476.675499
Ackley	0.008642917	**0.001462057**

Table 3.7: Summarise previous figures by the best mean Results of CCEA over CCEA-FI on 4 functions by 1000D, 5×10^6 fitness evaluations, CC variants use random grouping

Finally, we summarise the results of statistical tests associated with the experiments on numerical optimization in the bulk of this chapter. For all statistical tests, we use the well-known Mann-Whitney U test based on 20 runs of each algorithm compared; we set in most cases, a confidence level of 95%. For independent pairwise comparisons, we ran 20 runs each on a pair of problems from Rastrigin, Schwefel, Rosenbrock and Ackley respectively, at 100D and with a population size of 100,

continuing for 100,000 evaluations; in all of the experiments reported here, we fixed K (number of subpopulations) at 2.

One-tailed Mann-Whitney U test were applied on these results to determine whether there were any significant differences in the means. For all problems, we found that the means were significantly different at the 95% confidence level ($p<0.05$), indicating that CCEA-FI performs significantly better than EA-FI and CCEA.

3.3.2 Experiment study 2

This section again describes the experimental results of our algorithm CCEA-FI with comparison between EA, FI and CCEA over CCEA-FI. However this time we test on a combinatorial problem, therefore providing a very different context to numerical function optimization, and therefore providing a different context to validate the effectiveness of combining CC and FI. The combinatorial problem we look at is the standard bin-packing problem.

1. ***Test functions and problem definition*:**

   There are a set of N items, and each item has a given weight. Also, each item has a given type (there are T different types of item). The items have to be arranged into C containers, in such a way that the total weight of each container is as similar as possible. However there are constraints involving the types. In this assignment, there will always be 5 types, and the constraints are: items of type i and i+1 cannot be in the same container. The fitness function is (heaviest container - lightest container) + (pairs $\times$ TW). In this work, TW set to 10.

   We get the data from [22], in which column 1 just gives an item ID, column 2 gives that item's weight, and column 3 gives its type. (These data are for a 500-item problem with 5 types, and they need to be packed into a varied number of containers. In all cases we need to minimise the fitness function described previously i.e. the difference between the heaviest and lightest containers, with penalties added for any invalid pair of items. In this experiment study, we compare EA, EA with FI, CCEA and our algorithm CCEA-FI

2. ***Further details of algorithm and baseline experiments*:**

   We implement a steady-state EA with a single-gene new-allele mutation using binary tournament selection and replace-worst. We run this EA for 10,000 evaluations by 10 times independently and the result of a set of 10 runs will be a set of 3 numbers: the best, worst, and mean. We set the number of containers to be 50. 2 subpopulations were used in CC with population size 100. The proportion of fitness inheritance has been used between 0-80%. Also we use contiguous grouping in CC as the decomposition strategy.

3. ***Results*:** The experimental results are summarized and are listed in figure 3.19. From this experiment results we can see that CCEA-FI outperforms the normal EA and also CCEA. From this experimental study the combination can provided a better results that are superior to either CCEA or EAFI alone.

Mann-Whitney U test (one-tailed) with confidence level 95% (significant level by $p<0.05$) was applied on the Bin-packing problem results. The results of this test suggested that the results of CCEA-FI is significantly better than EA alone, EA-FI and CCEA with ($p<0.05$).

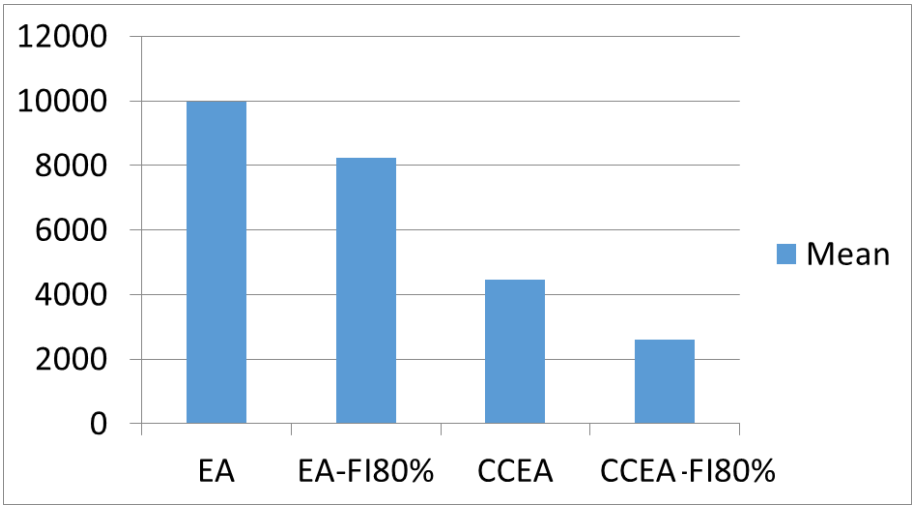

Figure 3.19: Mean results on bin-packing problem, showing EA, EA-FI, CCEA and CCEA-FI, 10^4 evaluations, CC methods use contiguous grouping

3.4 Discussion and Conclusions

The idea we explore in this chapter is to combine the two distinct strategies involved in co-operative co-evolution and fitness inheritance. Our approach has been to design a simple algorithm that combines the two strategies (CCEA-FI) and evaluate them on the same set of test functions that were explored in a fairly recent contribution to the co-evolution literature [85]. In addition we examined performance on 500D and 1000D versions of these test functions. Also in this chapter we examined our algorithm on Bin-packing problems and the raw findings indeed suggest that CCEA-FI generally achieves significantly better performance than either a CC-based EA without FI, or an EA with FI but without CC. In almost every cases Bin-Packing problem and benchmark function, in the benchmark function the best values over 20 runs for best, worst and mean were obtained with CCEA-FI. Only the Rosenbrock function presented anomalous results, showing higher sensitivity to the fitness inheritance percentage parameter, but still generally showing better results than CCEA alone for one or more CCEA-FI parameterisation at 50D-500D, and better best of 20 results at 1000D.

Considering the speedup obtainable by using CCEA-FI compared to CCEA, a general inspection of the results tables (table 3.8 example) suggests that, on both the 50D and 100D versions, the best CCEA-FI results at 50,000 function evaluations were close to the results of CCEA alone at 100,000 evaluations on the corresponding problem. For example, the best mean result recorded by CCEA on the 50D Rastrigin function at 100,000 evaluations is bested by CCEA-FI at 10 or 20% inheritance at 50,000 evaluations.

No. fitness function evaluations		CCEA	CCEA-FI-10%
10^5	Best	1.178716287	0.148440435
	Worst	2.036865564	0.529611689
	Mean	1.505050576	0.305271656
5×10^4	Best	2.299901201	1.013566337
	Worst	7.042936060	2.697007697
	Mean	4.47448432	1.649593208

Table 3.8: Rastrigin function 50D, CCEA-FI over CCEA for speedup obtainable

Finally, in this chapter we have investigated the additionality of fitness inheritance in the context of a co-operative coevolution algorithm, and demonstrated that this approach certainly promises further investigation. We note that state of the art results in this context seem at the moment to arise from the use of sophisticated and self-adaptive versions of differential evolution in combination with a CC approach (e.g. [122]). Such strategies are entirely amenable to the incorporation of FI, and in the next chapters we explore this idea, as well as investigate why FI reduced its performance at high levels of inheritance especially on high dimensions.

Chapter 4

Engineering Fitness Inheritance and Co-operative Evolution into SaNSDE with one key improvement on FI

4.1 Overview

The previous chapter has shown that combining CC with fitness inheritance (FI), a technique heretofore rarely explored in the context of large-scale optimization, can reliably lead to better performance. However that work was done in the context of a simple underlying EA (allowing us to be more confident that the benefits were due primarily to the combination of CC and FI). Here we explore the extent to which CC and FI provides added value when engineered together in the context of more sophisticated, so-called state of the art underlying algorithms, pre-adorned with a variety of additional enhancements. In this chapter we explore the high-performance techniques Self-Adaptive Neighbourhood Search Differential Evolution (SaNSDE) with our (CC-FI) algorithm in the field of large-scale optimization.

SaNSDE is a sophisticated black box optimization algorithm with a strong performance profile, which, arguably can still be considered among the state of the art, and we took it as a candidate for the engineering into it of both CC and FI. It turns out, as it happens, that it is not easy to find a state of the art algorithm that does not already have CC installed, so SaNSDE was a good choice for us in this respect.

We implemented SaNSDE from the description in the literature, and engineered CC+FI into it, using the same CC+FI framework as in our algorithm in the previous chapter, but with one key improvement as below on FI.

4.2 The key improvement on FI

The improvement was simply to ensure that in the steps of CC where a best from each subpopulation is chosen to populate new reference sets, these were constrained to be chosen on the basis of real evaluations (although not actually requiring additional real evaluations). Preliminary tests showed that this led to significant improvement in results at high levels of the fitness inheritance proportion (as is intuitively reasonable). The lack of this strategy in FI is presumably an explanation for reduced performance at high levels of inheritance especially on high dimension as seen in the previous chapter, [32], and also in [42], in which, although not involving CC, use us made occasionally of the best so far irrespective of how its fitness was calculated. We denote our engineered version of SaNSDE as CCDE-FI and with the key improvement on FI as CCDE-nFI as we make comparison between them (CCDE-FI and CCDE-nFI) before starting compare CCDE-nFI with DECC-G and JACC-G algorithms.

4.3 Evaluating CCDE-FI and CCDE-nFI

1. ***Test Functions*** **:** The choice of test functions is the suite used in Yang Tang and Yao [122], which enables the maximal comparisons we can make with previously published results for DECC-G and JACC-G. We use (as in [122]) their 500-dimensional and 1000-dimensional variants. For convenience these functions are defined below in table 4.1.

2. ***Further Details of Algorithm and Baseline Experiments*****:** Matching the key experimental variables with those reported in association with the comparative results, The results in each case(CCDE-FI and CCDE-nFI) are the means of 20 independent runs and for 2,500,000 function evaluations (real,

Name	Functions	F_{Min}	parameters x_i in the range
F1	$f(x) = \sum_{i=1}^{n} x_i^2$	0	$-100 \leq x_i \geq 100$
F2	$f(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	0	$-10 \leq x_i \geq 10$
F3	$f(x) = \sum_{i=1}^{n} (\sum_{j=1}^{n} x_i)^2$	0	$-100 \leq x_i \geq 100$
F4	$f(x) = \max\{\lvert x_i \rvert, 1 \leq i \geq n\}$	0	$-100 \leq x_i \geq 100$
F5	$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	0	$-30 \leq x_i \geq 30$
F6	$f(x) = \sum_{i=1}^{n} (x_i + 0.5)^2$	0	$-100 \leq x_i \geq 100$
F7	$f(x) = \sum_{i=1}^{n} i x_i^4 + random[0, 1]$	0	$-128 \leq x_i \geq 128$
F8	$f(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{\lvert x_i \rvert})$	-12569.5	$-500 \leq x_i \geq 500$
F9	$f(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$	0	$-5.12 \leq x_i \geq 5.12$
F10	$f(x) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)) + 20 - e$	0	$-32 \leq x_i \geq 32$

Table 4.1: Functions

not inherited) for 500D cases, and for 5,000,000 function evaluations in the 1000D cases. The population size was 100, and the sub-problems (in the context of CC) were always of dimension 100 (hence, 5 sub-problems in the 500D cases, and 10 sub-problems in the 1000D cases) and experiment with a range of FI values from 0 to 90%.

3. ***Results*:** The first of the comparative algorithms was CC-FI over CC-nFI with SaNSDE on 500D and 1000D in this context, and the results are presented in Tables (4.2 and 4.3).

   The second of the comparative algorithms was DECC-G [122], which supersedes SaNSDE in this context, being a version of SaNSDE that has been independently configured with CC. The other was JACC-G [125], which further improves on DECC-G via developments in the area of its embedded DE algorithm; the results of this comparison are presented in Table 4.4.

   The first comparison has been done to see how the new (FI) strategy impacts on the results from 10-90% results graphed in appendix B, we here present only 90% of FI of both algorithms.

   We compare the 90% of FI and nFI on SaNSDE algorithm, the means results of these two algorithms (CCDE-FI and CCDE-nFI) as shown in the tables (4.2 and 4.3). The tables (4.2 and 4.3) are shown the influence of the new

Functions	CCDE-FI	CCDE-nFI
F1	3.48E-160	**7.6E-313**
F2	1.54E-66	**2.15E-297**
F3	8.74644E+12	**0.00E+00**
F4	77.41334984	**39.8203829**
F5	587238.3914	**296.503831**
F6	0.00E+00	**0.00E+00**
F7	5767.215732	**1.56E-03**
F10	1.15E-11	**5.07E-14**

Table 4.2: Mean results of CCDE-FI over CCDE-nFI on 10 functions by 500D, 2.5×10^6 fitness evaluations, CC variants use random grouping and FI 90%.

Functions	CCDE-FI	CCDE-nFI
F1	3.34E-120	**8.73E-123**
F3	1.43218E+14	**0.00E+00**
F4	90.036168	**81.09686156**
F5	302315.1696	**923.1656446**
F6	0.00E+00	0.00E+00
F7	47087.25953	**4.62E-03**
F10	3.49E-13	**1.05E-13**

Table 4.3: Mean results of CCDE-FI over CCDE-nFI on 10 functions by 1000D, 5×10^6 fitness evaluations, CC variants use random grouping and FI 90%.

scheme in FI with CC (CCDE-nFI) over (CCDE-FI) on the dimension D500 and D1000 more details of the best, worst and mean are present in appendix B.

We first note that for functions F8 and F9 (results not displayed here), we can reliably obtain the optimum solution without FI, in other words with only CC and SaNSDE, on both dimensions D500 and D1000.

From these tables (4.2 and 4.3) it is clear that CCDE-nFI provides considerable gains in solution quality over CCDE-FI. We can see the effect particularly on F5 (Rosenbrock function). It is also clear that the best mean results were generally obtained from CCDE-nFI, which led to significantly improved results on both dimensions (D500, D1000). It is worth noting that even with standard non-improved FI with CCDE (CCDE-FI), we see improved results with increasing levels of FI, however these are outperformed by the use of nFI in CCDE-nFI.

In our second set of comparative tests, we compare our CCFI engineered version of SaNSDE with two other state of the art algorithms. The first of these comparative

Functions	D	DECC-G	JACC-G	CCDE-nFI
F1	500	6.33 e-27	-	**7.6e-313**
	1000	2.17e-25	2.7e-80	**8.73e-123**
F2	500	5.95e-15	-	**2.15e-297**
	1000	5.37e-14	**2.3e-20**	INF
F3	500	6.17e-25	-	**0.00e+00**
	1000	3.71e-23	2.4e-10	**0.00e+00**
F4	500	**4.58e-05**	-	3.9e+01
	1000	1.01e-01	**8.0e-05**	8.11e+01
F5	500	4.92e+02	-	**2.97e+02**
	1000	9.87e+02	9.83e+02	**9.23e+02**
F6	500	0.00e00	-	0.00e+00
	1000	0.00e00	0.00e00	0.00e00
F7	500	**1.50e-03**	-	1.56e-03
	1000	8.40e-03	**1.2e-03**	4.62e-03
F8	500	-209491	-	-209491
	1000	-418983	-418983	-418983
F9	500	0.00e00	-	0.00e00
	1000	3.55e-16	0.00e00	0.00e00
F10	500	9.13e-14	-	**5.07e-14**
	1000	2.22e-13	**1.4e-14**	1.05e-13

Table 4.4: Comparing CCDE-NFI (with FI at 90%)with descendants of SANSDE

algorithms is DECC-G [122], which can be regarded as superseding SaNSDE in this context, being a version of SaNSDE that has independently been configured with CC. The second is JACC-G [125], which further improves on the DECC-G algorithm in certain contexts. The results of comparing these with our CCDE-nFI algorithm with FI at 90% are presented in table 4.4

We show and compare only the results for CCDEnFI using 90% fitness inheritance. This achieves better or equivalent mean performance than DECC-G on 8 of the 10 500D cases (winning 5, losing 2), and 8 of the 10 1000D cases (winning 6, losing 2). Meanwhile, it performs better or equivalently to JACC-G on 7 of the 10 1000D test cases (winning 4, losing 3). Comparative 500D results were not available for JACC-G.

There is not enough evidence comparing with single runs of both algorithms JACC-G and DECC-G to claim superiority of CCDEnFI at a high level of confidence with statistical tests. However, we can conclude that simply engineering CC and FI into a sophisticated algorithm (such as SaNSDE) in a straightforward way, is able to provide potentially superior results, and at least competitive results, to

Functions		CCDE-FI Vs CCDE-nFI
F1	P-value	0.49202
F2	P-value	0.49202
F3	P-value	0.00212
F4	P-value	0.00187
F5	P-value	<0.00001
F6	P-value	-
F7	P-value	<0.00001
F8	P-value	-
F9	P-value	-
F10	P-value	0.49202

Table 4.5: The P-values results of, CCDE-FI Vs CCDE-nFI on ten functions by Mann-Whitney U Test at 500D, 2.5×10^6 fitness evaluations, FI=90%.

those available via a suite of alternative sophistications, such as those incorporated beyond SaNSDE in each of DECC-G and JACC-G.

From the previous experiment results on 10 functions by our algorithms CCDE-FI and CCDE-nFI, we took the results of each single ran at 90% FI of both algorithms on 500D and used here in statistic test (Mann-Whitney U Test) as presented in table 4.5.

Mann-Whitney U Test (one-tailed) with confidence level 95% (significant level by $p<0.05$) was applied on the results for each of the ten functions to determine whether there is a statistically significant difference between the results of the two algorithms on the same function.

In table 4.5, we can see the results of the statistical tests comparing the two algorithms on each function; the key result in each case is the p-value. The obtained p-values suggest that there is a statistically significant difference between the two algorithms on four functions, (F3, F4, F5 and F7) which corresponds to the new proposed method on FI having significantly improved performance over CCDE-FI on these functions. Meanwhile, for the three other functions (F1, F2 and F10) there is no statistically significant difference between these three algorithms because p-value >0.05. The '-'symbol indicates the results of both algorithms on the functions are exactly the same which prevents us from make the test as in functions F6, F8 and F9.

4.4 Discussion and Conclusions

The previous chapter had suggested promise for using both cooperative coevolution (CC) and fitness inheritance (FI) in the design of a black box optimization algorithm (especially with large-scale problems in mind). Essentially, that work had shown that the strategy of combining CC and FI works well, providing independent and additional improvements, in the case where the underlying EA being engineered was otherwise a straightforward algorithm.

In this chapter, however, we further explored the strategy of combining CC and FI by jumping in at the deep end by seeing if combining these strategies could lead to any advancement for algorithms that were already state of the art, in the sense that they already comprise a variety of sophisticated mechanisms that boost their performance.

The question (perhaps rationalized a little after the fact) is not Can CC+FI improve a state of the art algorithm (which does not already use both)?, we would a priori not hold out high expectations that this is possible, because state of the art algorithms tend to incorporate a complex cookbook of mechanisms that would likely be perturbed by significant additional mechanisms such as either CC, FI or both. Instead, the question is, given that we already know CC+FI can improve a straightforward algorithm: Can CC and FI be engineered into arbitrary black box optimization algorithms, without doing harm? (Alternatively: if the algorithm contains either CC or FI already, can we engineer the other one into it without doing harm?). If the answer to that was positive, it would suggest that adding CC and FI to simpler algorithms would lead to improvements, and adding them to state of the art algorithms would leave their overall performance profile unchanged. This is precisely what you want if, beforehand, you do not actually know where your algorithm sits in the virtual algorithm league table.

In that sense, we would tentatively conclude that the new approach to combining CC and FI explored here (involving the key improvement in FI (nFI)) represents a recommended algorithm-enhancement strategy and turns out that CCDE-nFI outperformed our implementation of unadorned SaNSDE with high statistical confi-

dence at most levels of the FI proportion

Finally, the performance of CCDE-nFI with lower levels of fitness inheritance was generally similar to that shown for 90%, although lower levels (10% – 40%) occasionally showed reduced (and highly problem-dependent) performance. Obviously, to achieve candidacy as a usable algorithm, CCDEnFI (or, the engineering of CC and FI into an algorithm in general) needs to either come along with good guidelines for setting the fitness inheritance parameter, or there needs to be a viable adaptive approach for FI. The latter notion is investigated in the next chapter.

Chapter 5

Engineering Adaptive CC AND FI INTO State-of-the-Art Optimizers DECC-DML

5.1 Overview

So far, we have investigated the combination of CC and FI in the last chapters and herein, without the intention of declaring a new algorithm, but with the intention of exploring their potential as adornments to arbitrary black box algorithms. Nevertheless, to be deployable and proposable an algorithm that incorporates both CC and FI must come along with either a good set of fixed parameters, or with an adaptation scheme for its key parameters. The single key parameter that looms large in this respect is the fitness inheritance proportion (often abbreviated as FI, at the risk of overloading that term). In this chapter we take a first step at investigating adaptive schemes for it in the CC-FI context.

Two simple first-cut adaptive approaches were designed as follows:

Method A: In this approach, the performance of 10%, 20%, and so on, up to 90% FI are sampled in the first batch of evaluations, with each being the sole setting for a duration of FI test evaluations (hence 9xFItest evaluations are devoted to this sampling process). The value of FI that performed best in this early sampling is

then used for the remainder of the run.

Method B: This begins in the same way as method A. However the sampling process is repeated every EPOCH evaluations. After a set of EPOCH evaluations is completed, the sampling process is repeated (consuming 9xFItest evaluations), and the best FI from that most recent sampling is then used for a further (EPOCH -9xFItest) evaluations; this process is repeated until termination.

Our adaptive schemes were tested in the context of a further investigation into engineering CC and FI to a sophisticated state of the art algorithm. In this case, the algorithm of choice was DECC-DML [71]; DECC-DML is a further co-operative co-evolution approach based on DE, which improves on the previous random grouping method for constructing CC sub problems, aimed at being more effective at identifying interacting variables, which are then best treated within the same sub-problem.

DECC-DML and its available results provide a further opportunity for us to test the engineering of combined CC and FI. However, a slight drawback which has turned out not easy to avoid in the context of current large-scale optimization research is the fact that DECC-DML already incorporates CC. Nevertheless, the spirit of investigation remains in that we are exploring the capability of combinations of CC and FI, especially in the context of already-sophisticated methods. We therefore soldiered on, and proceeded by retaining DECC-DML's existing (and quite sophisticated) variant of the CC mechanism, and engineered our adaptive versions of FI into the existing DECC-DML code (which the authors had made available). Then we a developed a variant of DECC-DML called DECC-DML-aFI.

In DECC-DML, subcomponents sizes are also self-adaptive, by checking the fitness of the individuals if no improvement then will choose different decomposer from S which is, a set of predetermined decomposers. S = 50, 100, 200, 250, instead of using fixed size. Also, DECC-DML [71] used delta grouping as a decomposition strategy which was shown to outperform random grouping, they used delta grouping by trying to improve the interval of non-separate problems. In delta grouping variables are sorted based on the average dimension-wise displacement of the sample points over the entire population between two consecutive cycles. Then, after the

decision variables are sorted, they are grouped into k sub groups of size s.

Algorithm 6 The basic structure of DECC-DML-aFI

1: Set i=1 to start a new cycle.
2: Initialize the Δ vector to zero. This means only for the first cycle. So the variables will preserve their original order.
3: **repeat**
4: Divide the decision variables into subcomponent here self-adapts subcomponent sizes, using the techniques used in DECC-ML [70].
5: Optimize the i^{th} subcomponent with a certain EA using FI as it also here self-adaptive as Method A or B.
6: **if** i < m **then**
7: i++, and go to Step 5.
8: **else**
9: Construct the Δ vector using Equation (5.1).
10: Sort the decision variables based on the magnitude of their corresponding delta value.
11: **end if**.
12: Go to Step 4 for the next cycle .
13: **until** a total of FE function evaluations have been done

The Δ vector represented as $\Delta = \{\bar{\delta_1}, \bar{\delta_2}..\bar{\delta_n}\}$ where n is referring to the number of dimensions and each element in this vector is calculated as the following equation (5.1):

$$\bar{\delta_i} = \frac{\sum_{i=1}^{p_z} \delta_{ij}}{p_z}, i \in (1,.., n). \tag{5.1}$$

Where P_z is the population size, and δ_i,j referring the delta value of the jth individual on the ith dimension.

5.2 Evaluating DECC-DML-aFI

1. ***Test Functions*:** The latter was done, and the resulting DECC-DML-aFI algorithm was compared with DECC-DML over the CEC 2010 large scale global optimization test suite of 20 test functions 1000D [71]. Following preliminary investigation, the method A and B parameters FI test and EPOCH (only in method B) were set to 100 and 300,000 respectively.

2. ***Further Details of Algorithm and Baseline Experiments*:**

   Tables 5.1, 5.2 summarizes the results. The results in each case are the best,

median, worst, mean, p-value and standard deviation of 25 independent runs, each of which continued for a maximum of 3,000,000 real evaluations. The population size is set to 50, the size of subcomponents S = {50, 100, 200, 250}, and fitness inheritance FI= 10%,..90%. DECC-DML source code in Matlab, as presumably deployed in [71] was obtained from Xiaodong Li's website [72]. We engineered the inclusion of FI directly into this version. However, we noticed that the mechanisms in the original DECC-DML source code associated with calculating and updating the delta value, incorporated some calls to function evaluations that were not accounted for in the total which counted towards algorithm termination.

3. ***Results*:** The results for the DECC-DML algorithm shown in Table 5.1 and 5.2 are therefore from our own runs with the corrected version of the source code, ensuring comparison on an equal basis in terms of real function evaluations. In the tables 5.1 and 5.2, the best mean result for any given function is highlighted in bold, while underline is used to indicate with of the two adaptation methods achieved the best mean (independent of whether either achieved overall best mean for that function). Analysis of Table 5.1 shows that the laurels are shared quite equally between the three approaches. Original DECC-DML has 7 wins, compared with 7 wins for DECC-DML-aFI Method B, and 6 wins for the Method A version. Similarly, both Method A and Method B show a complementary performance profile over the 20 functions.

Mann-Whitney U Test (one-tailed) with confidence level 95% (significant level by $p<0.05$) was applied to determine if these algorithms results are significantly different from each other. According to the P-values from table 5.3 by the algorithms DECC-DML vs (DECC-aFI method A and method B), these differences are significant on 17 functions of the 20 functions while these differences are not significant on 3 functions F5, F8 and F13 with ($P>0.05$).

Functions		Corrected, DECC-DML	DECC-DML-aFI Method A	DECC-DML-aFI Method B
F1	Best	2.10E-08	1.91E-09	1.45E-09
	Median	7.88E-08	2.79E-08	1.70E-08
	Worst	1.63E-07	9.57E-08	4.37E-08
	Mean	8.68625e-08	2.94803e-08	**1.876923e-08**
	St.Dve	4.67534e-08	2.39913e-08	1.13908e-08
F2	Best	1.02E+03	1.02E+03	1.03E+03
	Median	1.10E+03	1.13E+03	1.13E+03
	Worst	1.18E+03	1.19E+03	1.19E+03
	Mean	**1.091e+3**	1.1209e+03	1.12580e+03
	St.Dve	2.8947e+01	4.501e+01	4.326e+01
F3	Best	1.14E-07	2.06E-08	5.68E-08
	Median	2.02E-07	1.23E-07	8.59E-08
	Worst	2.47E-07	2.50E-07	1.36E-07
	Mean	1.94621e-07	1.23452e-07	**9.007184e-08**
	St.Dve	2.97274e-08	6.69694E-08	2.24766e-08
F4	Best	3.36E+12	1.70E+12	2.93E+12
	Median	6.79E+12	5.38E+12	5.56E+12
	Worst	9.41E+12	1.18E+13	8.78E+12
	Mean	6.68762e+12	**5.56665e+12**	5.78655e+12
	St.Dve	1.61611e+12	2.32989e+12	1.98951e+12
F5	Best	1.43E+08	1.89E+08	1.54E+08
	Median	2.38E+08	2.71E+08	2.82E+08
	Worst	5.47E+08	5.11E+08	4.10E+08
	Mean	**2.82387e+08**	2.97985e+08	2.84147e+08
	St.Dve	1.0612e+08	8.0744e+07	6.4545e+07
F6	Best	1.70E-05	2.08E-06	1.75E-06
	Median	5.66E-05	1.56E+06	4.80E-06
	Worst	2.32E+06	2.94E+06	3.26E+06
	Mean	**9.2674e+04**	1.09202e+06	6.37554e+05
	St.Dve	4.54008e+05	1.093741e+06	1.051391e+06
F7	Best	1.22E+08	7.43E+07	1.30E+08
	Median	2.32E+08	1.64E+08	3.70E+08
	Worst	4.30E+08	3.10E+08	1.13E+09
	Mean	2.31131e+08	**1.65121e+08**	4.10521e+08
	St.Dve	7.32581e+07	5.61666e+07	1.99737e+08
F8	Best	3.13E+07	2.44E+07	3.35E+07
	Median	9.91E+07	1.04E+08	1.32E+08
	Worst	1.78E+08	1.66E+08	1.75E+08
	Mean	**9.34754e+07**	9.36069e+07	1.16837e+08
	St.Dve	4.81367e+07	4.45266e+07	4.38386e+07
F9	Best	1.14E+08	1.00E+08	1.06E+08
	Median	1.33E+08	1.26E+08	1.27E+08
	Worst	1.58E+08	1.61E+08	1.56E+08
	Mean	1.3458e+08	1.27753e+08	**1.25674e+08**
	St.Dve	1.1019e+07	1.5981e+07	1.1204e+07
F10	Best	1.22E+04	1.21E+04	1.27E+04
	Median	1.25E+04	1.26E+04	1.32E+04
	Worst	1.38E+04	1.32E+04	1.38E+04
	Mean	1.2646e+04	**1.26387e+04**	1.32528e+04
	St.Dve	3.93e+02	2.83e+02	2.44e+02

Table 5.1: Engineering simple adaptive FI into DECC-DML

Functions		Corrected, DECC-DML	DECC-DML-aFI Method A	DECC-DML-aFI Method B
F11	Best	3.98E-06	9.28E-07	1.57E-06
	Median	5.91E-06	3.35E-06	2.34E-06
	Worst	1.15E+02	9.00E-05	1.46E-05
	Mean	4.61e+00	6.86595e-06	**3.226461e-06**
	St.Dve	2.25e+01	1.71E-05	2.77971E-06
F12	Best	3.51E+06	3.85E+06	4.33E+06
	Median	4.00E+06	4.09E+06	4.83E+06
	Worst	4.33E+06	4.41E+06	5.84E+06
	Mean	**3.97037e+06**	4.09242e+06	4.86637e+06
	St.Dve	2.42e+05	1.50e+05	3.46e+05
F13	Best	7.08E+02	5.49E+02	7.37E+02
	Median	1.08E+03	1.06E+03	1.08E+03
	Worst	4.48E+03	2.28E+03	2.45E+03
	Mean	1.320e+03	1.15158e+03	**1.27881e+03**
	St.Dve	7.83e+02	4.23e+02	4.54e+02
F14	Best	3.49E+08	3.76E+08	3.55E+08
	Median	4.35E+08	4.15E+08	4.15E+08
	Worst	4.96E+08	4.59E+08	5.05E+08
	Mean	4.31999e+08	**4.18162e+08**	4.1924e+08
	St.Dve	3.40e+07	2.42e+07	3.28e+07
F15	Best	1.49E+04	1.47E+04	1.56E+04
	Median	1.57E+04	1.55E+04	1.62E+04
	Worst	1.64E+04	1.61E+04	1.69E+04
	Mean	1.5586e+04	**1.55049e+04**	1.62337e+04
	St.Dve	3.81e+02	3.77e+02	3.67e+02
F16	Best	2.59E-05	3.15E-06	1.17E-05
	Median	5.25E-05	6.07E-05	3.68E-05
	Worst	4.27E+02	4.26E+02	2.61E-04
	Mean	3.41033e+01	1.71398e+01	**5.378018e-05**
	St.Dve	1.15e+02	8.34e+01	5.26922E-05
F17	Best	6.35E+06	6.31E+06	7.41E+06
	Median	7.00E+06	7.08E+06	8.41E+06
	Worst	7.65E+06	8.11E+06	9.21E+06
	Mean	**7.04423e+06**	7.14235e+06	8.36006e+06
	St.Dve	3.72e+05	4.27e+05	5.14e+05
F18	Best	1.46E+03	1.45E+03	1.65E+03
	Median	7.47E+03	2.38E+03	2.54E+03
	Worst	1.89E+04	1.62E+04	1.58E+04
	Mean	7.95526e+03	3.97663e+03	**3.90853e+03**
	St.Dve	5.362e+03	3.651e+03	3.81e+03
F19	Best	1.36E+07	1.47E+07	1.77E+07
	Median	1.70E+07	1.71E+07	2.13E+07
	Worst	1.93E+07	2.04E+07	2.56E+07
	Mean	1.783e+07	**1.73677e+07**	2.16407e+07
	St.Dve	1.52e+06	1.25e+06	1.96e+06
F20	Best	9.82E+02	9.82E+02	9.81E+02
	Median	9.84E+02	1.04E+03	9.83E+02
	Worst	1.04E+03	1.14E+03	1.14E+03
	Mean	**9.944e+02**	1.0201e+03	1.00183e+03
	St.Dve	2.21e+01	4.25e+01	3.71e+01

Table 5.2: Engineering simple adaptive FI into DECC-DML

Functions		DECC-DML Vs Method A	DECC-DML Vs Method B
F1	P-value	<0.00001	<0.00001
F2	P-value	0.00714	0.00048
F3	P-value	0.0001	<0.00001
F4	P-value	0.01923	0.05262
F5	P-value	0.14007	0.2451
F6	P-value	0.46812	0.00317
F7	P-value	0.00097	<0.00001
F8	P-value	0.46812	0.0548
F9	P-value	0.0505	0.00402
F10	P-value	0.33724	<0.00001
F11	P-value	<0.00001	<0.00001
F12	P-value	0.04746	<0.00001
F13	P-value	0.25463	0.25463
F14	P-value	0.04551	0.06552
F15	P-value	0.30854	<0.00001
F16	P-value	0.13786	0.00427
F17	P-value	0.26435	<0.00001
F18	P-value	0.00289	0.00187
F19	P-value	0.15625	<0.00001
F20	P-value	0.02743	0.20897

Table 5.3: The P-values results of, DECC-DML Vs DECC-DML-aFI method (A and B) on 20 functions by Mann-Whitney U Test at 1000D.

5.3 Discussion and Conclusions

Our results suggest that, at best, we can say that the engineering of FI into DECC-DML (via either of the two adaptive methods) has adjusted its overall performance profile, while not providing an overall improvement to its performance, as estimated over this particular function suite. It is also clear that this engineering has not provided any overall detriment to performance.

Isolating each DECC-DML-aFI method and comparing that with DECC-DML, we note that Method A achieves a better mean value on 13 of the 20 functions, while Method B achieves a better mean on 9 of the 20 functions. If we consider these success rates against expectations according to cumulative binomial probability (assuming null hypothesis of a success rate of 0.5 in each case), we note that 13 or more wins can be expected to occur with probability 0.132, while 9 wins (or equivalently, 11 wins for DECC-DML) can be expected with probability 0.411. This further confirms that the engineering of FI into the algorithm has not caused any overall loss in performance, but has clearly changed the performance profile in a way we have yet to fully understand.

Chapter 6

Conclusions and Future Work

6.1 Summary

This research proposed a combination of the cooperative coevolution framework CC and fitness inheritance FI for large scale optimization. Experimental results showed that the developed algorithm CC-FI explored here (involving the amended and adapted FI) represents a recommended algorithm-enhancement strategy, this technique or the combination could be useful for many algorithms. Even if the algorithm contains either CC or FI already, we can engineer the other one into it without doing harm. In other words, if this technique did not improve the overall performance profile, we expect that at worst it would not harm performance.

Initially, we combined CC and FI into a straightforward algorithm that we call CC-FI. Both CC and FI have been found successful on nontrivial and multiple test cases, and they use fundamentally distinct strategies. We explore the extent to which employing both of these strategies at once provides additional benefit. We first focused on visualising the benefit of our algorithms CC-FI over both CCEA alone and the basic EA. These comparison were done on well-known functions Rastrigin, Schwefel, Rosenbrock and Ackley function on several dimension. The raw findings results and statistical evidence indeed suggest that CCEA-FI generally achieves significantly better performance than either a CC-based EA without FI, or an EA with FI but without CC especially when we swap the basic contiguous grouping method with random grouping that is used in the CC cases. Only the Rosenbrock function

presented anomalous results, showing higher sensitivity to the fitness inheritance percentage parameter and with 10% FI usually leading to the best mean result, however, still generally showing better results than CCEA alone. This initial work was done with a simple EA, to allow us to be more confident that the benefits were due primarily to the combination of CC and FI.

The second developed idea is to replace the simple EA with more sophisticated, so called state of the art algorithm in order to explore the extent to which CC and FI provided added value when engineered together in this context. To that end, we explore SaNSDE, which is a sophisticated black box optimization algorithm with a strong performance profile, which, arguably can still be considered among the state of the art, and we took it as a candidate for the engineering into it of both CC and FI. It turns out, as it happens, that it is not easy to find a state of the art algorithm that does not already have CC installed, so SaNSDE was a good choice for us in this respect. Before we implemented SaNSDE from the description in the literature, and engineered CC+FI into it, we make one key improvement on FI as anomalous results came from the Rosenbrock function from our first developed idea, which showed high sensitivity to the fitness inheritance proportion parameter. The key improvement was simply to ensure that in the steps of CC where a best from each subpopulation is chosen to populate new reference sets, these were constrained to be chosen on the basis of real evaluations of the fitness value from candidates; this prevented choosing a best candidate from among those solutions who had inherited their fitness values during the evaluation process. The results showed that the key improvement on FI led to significant improvement in results at high levels of the fitness inheritance proportion. We denote our engineered version of SaNSDE and CC-nFI as CCDEnFI and tested it on test functions is the suite used in Yang Tang and Yao [122], on 500-dimensional and 1000-dimensional variants. The results indicate that the new approach to combining CC and FI involving the key improvement in FI (nFI)) represents a recommended algorithm-enhancement strategy and turns out that CCDE-nFI outperformed our implementation of unadorned SaNSDE with high statistical confidence at most levels of the FI proportion.

Our last chapter investigated how to improve our algorithm in terms of adapting the CC-FI parameters, especially here the FI parameters as these were set manually (fitness inheritance proportion) in the previous chapters. This investigation was aimed at making our algorithms more readily deployable, reducing the need for preliminary parameter studies. To be more deployable, an algorithm must come with either a robust set of fixed parameter that work well over a very wide range of cases, or with an adaptation scheme for its key parameters. In this context, we investigate adaptive schemes for FI in CC-FI. Two simple adaptive methods were designed and tested in the context as well as a further investigation into engineering CC and FI to a sophisticated state of the art algorithm. In this case, the algorithm of choice was DECC-DML [28] as its all parameters are self-adaptive including the number of subcomponents in each evaluation. We engineered our adaptive versions of FI into the existing DECC-DML which already have CC installed. In fact, the available code and results from Xiaodong Li's website [85] provide a further opportunity for us to test the engineering of combined CC and FI into an existing algorithm. During our experiments we noticed that the mechanisms in the original DECC-DML source code associated with calculating and updating the delta value, incorporated some calls to function evaluations that were not accounted for in the total which counted towards algorithm termination. Therefore, we corrected the version of the MATLAB source code ensuring comparison on an equal basis in terms of real function evaluations. The corrected results of DECC-DML were compared with our denote algorithm DECC-DML-aFI. The comparison was over the CEC 2010 large scale global optimization test suite of 20 test functions on 1000 dimensions. The results suggests that, engineering of FI into DECC-DML (via either of the two adaptive methods) has adjusted its overall performance profile, while not providing an overall improvement to its performance, as estimated over this particular function suite. However, this also provided another confirmation that the engineering of FI into the algorithm has not caused any overall loss in performance.

Overall we can conclude that:

- Engineering CC and FI into a good algorithm may well lead to improved

performance, and is unlikely to lead to reduced performance.

- Engineering CC and FI into a sophisticated, state of the art algorithm (or engineering FI into a state of the art algorithm that already includes CC) may also lead to improvement, and will likely not lead to detriment. However, the more state of the art the original algorithm is, the more the exercise may provide diminishing returns.

6.2 Contributions

The contribution of this thesis can be restated again with some explanation as stated in chapter 1 section 1.2:

- **Contribution 1** was explored in chapter 3, where a combination of Cooperative Coevolution framework CC and Fitness Inheritance FI, called CC-FI, is developed, and a simple EA is used here as subcomponents optimizer. The performance of our new algorithm CC-FI tested on well-known 4 functions Rastrigin, Schwefel, Rosenbrock and Ackley. The experiment indeed suggest that CC-FI generally achieves significantly better performance over either a CC-based EA without FI (CCEA), or an EA with FI but without CC (EA-FI), especially when a random grouping scheme is used in the CC component.

- **Contribution 2** was explored in chapter 4, where Self-Adaptive Neighbourhood Search Differential Evolution (SaNSDE) was described on the beginning of the chapter, we explore the high-performance techniques (SaNSDE) with our (CC-FI) algorithm in the field of large-scale optimization instead of EA. We implemented SaNSDE from the description in the literature, and engineered CC+FI into it, using the same CC-FI framework as in our algorithm in the previous contribution, but with one key improvement on FI as it was losing its performance at high levels of inheritance specially on high dimension. The results conclude that the new approach to combining CC and FI involving the key improvement in FI (nFI) represents a recommended algorithm-enhancement strategy with SaNsDE on the ten CEC'2005 benchmark functions.

- **Contribution 3** Two simple adaptive schemes for FI are investigated in the CC-FI context in chapter 5; this done engineering these new schemes (CC-aFI) into a sophisticated state of the art algorithm. In this case, the algorithm of choice was DECC-DML [71]. We noticed that the mechanisms in the original DECC-DML source code associated with calculating and updating the delta value, incorporated some calls to function evaluations that were not accounted for in the total which counted towards algorithm termination. Therefore, we corrected the version of DECC-DML and compared it with our algorithm DECC-DML-aFI. The comparison was over the CEC 2010 large scale global optimization test suite of 20 test functions on 1000 dimensions. This suggests that, at best, we can say that the engineering of FI into DECC-DML (via either of the two adaptive methods) has adjusted its overall performance profile, while not providing an overall improvement (and certainly not providing any overall detriment) to its performance, as estimated over this particular function suite.

6.3 Future Work

Based on what we have done in this thesis, the future work may include the following ideas:

- More experiments and explorations that could be tried, but were not in the thesis. For instance, the number of sub-populations itself could be adaptive, using strategies like in our methods A and B. There are also alternative approaches to CC+FI that could be tried, for example: N generations of CC, followed by N generations of FI (with 1 population all dimensions).

- Investigate the usefulness of using different sorts of fitness inheritance such as weighted inheritance and parental inheritance with Cooperative coevolution framework CC on different type of functions. furthermore, these strategies of fitness inheritance could be self adaptive, using strategies also like in our methods A and B so we could switch between these strategies until we find the best (inheritance strategy) for the problem.

- More investigation to understand the relationship between CC+FI parameters and performance, by using meta algorithms as in algorithm configuration studies [47], by running tuning studies on different problems, and different groups of problems, we can get an idea of the relationship between the CC+FI configuration and its performance on different types of problems.

- Exploring the CC+FI idea in problems with different representations (not a parameter list). FI can be applied in any problem, but for CC it needs to be possible to group it into lower-dimension subpopulations. This can not obviously be done when it is a permutation-based representation, for example. However, for such problems one could use the sort-order representation. This is any list of N real values between 0 and 1, which encode a permutation.

Bibliography

[1] Roux C (1995) Registration of non-segmented images using a genetic algorithm. Lecture notes in computer science, vol. 905, pp. 205211, author = D. Goldberg, booktitle = Eiben AE, editor = Smith, J. E., publisher = Introduction to Evolutionary Computing. Springer. Jacq J, title = Genetic algorithms in optimization, search and machine learning, "Addison Wesley, New York, year = 1989,.

[2] S. C. Agrawal. Metamodeling: a study of approximations in queueing models. MIT Press, 1985.

[3] J. Andre, P. Siarry, and T. Dognon. An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization. *Advances in engineering software*, 32(1):49–60, 2001.

[4] P. J. Angeline and J. B. Pollack. *Competitive Environments Evolve Better Solutions for Complex Tasks.* in Proceedings of the 5th International Conference on Genetic Algorithms, Urbana-Champaign, USA pp. 264-270, 1993.

[5] P. Arena, R. Caponetto, L. Fortuna, and M. Xibilia. Mlp optimal topology via genetic algorithms. In *Artificial Neural Nets and Genetic Algorithms*, pages 670–674. Springer, 1993.

[6] T. Back. Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. *Oxford University press Oxford*, 996, 1996.

[7] T. Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms.* Oxford university press, 1996.

[8] J. E. Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and their applications*, pages 101–111. Hillsdale, New Jersey, 1985.

[9] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the second international conference on genetic algorithms*, pages 14–21, 1987.

[10] R. Barbour, D. Corne, and J. McCall. Accelerated optimisation of chemotherapy dose schedules using fitness inheritance. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[11] R. K. Belew, J. McInerney, and N. N. Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. In *In*. Citeseer, 1990.

[12] R. E. Bellman. Dynamic programming. Ser. Dover Books on Mathematics. Princeton University Press, 1957.

[13] J. O. Berkey and P. Y. Wang. Two-dimensional finite bin-packing algorithms. *Journal of the operational research society*, pages 423–429, 1987.

[14] H.-G. Beyer and H.-P. Schwefel. Evolution strategies–a comprehensive introduction. *Natural computing*, 1(1):3–52, 2002.

[15] M. A. Bhatti. Optimization problem formulation. In *Practical Optimization Methods*, pages 1–45. Springer, 2000.

[16] T. Blickle and L. Thiele. A comparison of selection schemes used in genetic algorithms, 1995.

[17] T. Blickle and L. Thiele. A comparison of selection schemes used in genetic algorithms, 1995.

[18] T. Blickle and L. Thiele. a comparison of selection schemes used in evolutionary algorithms," evolutionary computation, vol. 4. *pp*, 361, 1997.

[19] E. K. Burke, J. P. Newall, and R. F. Weare. Initialization strategies and diversity in evolutionary timetabling. 6:1, 1998.

[20] E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis*, 10(2):141–171, 1998.

[21] Z. Cao, L. Wang, Y. Shi, X. Hei, X. Rong, Q. Jiang, and H. Li. An effective cooperative coevolution framework integrating global and local search for large scale optimization problems. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 1986–1993. IEEE, 2015.

[22] D. Corne. Bin-packing problem, data. `https://www.macs.hw.ac.uk/~dwcorne/Teaching/data.txt`. Accessed: 2013-02-30.

[23] Y. Davidor. Epistasis variance: Suitability of a representation to genetic algorithms. *Complex Systems*, 4(4):369–383, 1990.

[24] L. Davis. Handbook of genetic algorithms. 1991.

[25] E. D. de Jong. *The Incremental Pareto-Coevolution Archive.* in Proceedings of Genetic and Evolutionary Computation Conference pp. 525-536, 2004.

[26] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[27] G. Dosa. *First Fit Algorithm for Bin Packing*, pages 1–5. Springer US, Boston, MA, 2008.

[28] K. L. Downing. pp. 381406, title = Tantrix: A minute to learn, 100 (genetic algorithm) generations to master," Genetic Programming and Evolvable Machines, vol. 6, no. 4, year = 2005,.

[29] A. E. Eiben, J. E. Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[30] M. El-Abd. Hybrid cooperative co-evolution for large scale optimization. In *Swarm Intelligence (SIS), 2014 IEEE Symposium on*, pages 1–6. IEEE, 2014.

[31] E. Feinerman and M. S. Falkovitz. Optimal scheduling of nitrogen fertilization and irrigation. *Water resources management*, 11(2):101–117, 1997.

[32] L. G. Fonseca, A. C. Lemonge, and H. J. Barbosa. A study on fitness inheritance for enhanced efficiency in real-coded genetic algorithms. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8. IEEE, 2012.

[33] J. Fontanari and R. Meir. Evolving a learning algorithm for the binary perceptron. *Network: Computation in Neural Systems*, 2(4):353–359, 1991.

[34] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[35] C. K. Goh, K. C. Tan, C. Y. Cheong, and Y.-S. Ong. An investigation on noise-induced features in robust evolutionary multi-objective optimization. *Expert Systems with Applications*, 37(8):5960–5980, 2010.

[36] D. E. Goldberg. Using time efficiently: Genetic-evolutionary algorithms and the continuation problem. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*, pages 212–219. Morgan Kaufmann Publishers Inc., 1999.

[37] D. E. Goldberg. Foreward. *EURASIP Journal on Applied Signal Processing*, 8:731–732, 2003.

[38] D. E. Goldberg, K. Deb, and J. H. Clark. Genetic algorithms, noise, and the sizing of populations. *Urbana*, 51:61801, 1991.

[39] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell Labs Technical Journal*, 45(9):1563–1581, 1966.

[40] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

[41] K. A. Hacker, J. Eddy, and K. E. Lewis. Efficient global optimization using hybrid genetic algorithms. In *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, pages 4–6, 2002.

[42] A. Hameed, D. Corne, D. Morgan, and A. Waldock. Large-scale optimization: Are co-operative co-evolution and fitness inheritance additive? In *Computational Intelligence (UKCI), 2013 13th UK Workshop on*, pages 104–111. IEEE, 2013.

[43] M. Han and J. Fan. Particle swarm optimization using dynamic neighborhood topology for large scale optimization. In *Intelligent Control and Automation (WCICA), 2010 8th World Congress on*, pages 3138–3142. IEEE, 2010.

[44] F. Herrera and M. Lozano. Adaptive genetic operators based on coevolution with fuzzy behaviors. *IEEE Transactions on Evolutionary Computation*, 5(2):149–165, 2001.

[45] W. D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D: Nonlinear Phenomena*, 42:228–234, 1990.

[46] H. H. Hoos and T. Stützle. *Stochastic local search: Foundations and applications*. Elsevier, 2004.

[47] F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *AAAI*, volume 7, pages 1152–1157, 2007.

[48] Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 9(1):3–12, 2005.

[49] H. k. Tsai, J. m. Yang, Y. f. Tsai, and C. y. Kao. pp. 17181729, title = An evolutionary algorithm for large traveling salesman problems," Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, vol. 34, no. 4, year = 2004,.

[50] W. S. Klug, M. R. Cummings, C. Spencer, C. A. Spencer, and M. A. Palladino. Concepts of genetics. In N. Y. a. Pearson, editor, *9th ed.* USA.2008.

[51] J. Koutní'k, G. Cuccu, J. Schmidhuber, and F. Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of*

the 15th annual conference on Genetic and evolutionary computation, pages 1061–1068. ACM, 2013.

[52] N. Krasnogor and J. Smith. A memetic algorithm with self-adaptive local search: Tsp as a case study. In *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, pages 987–994. Morgan Kaufmann Publishers Inc., 2000.

[53] V. Kureichick, A. Melikhov, V. Miagkikh, O. Savelev, and A. Topchy. Some new features in genetic solution of the travelling salesman problem. In *Adaptive Computing in Engineering Design and Control*, volume 96, 1996.

[54] S. Lawrence, A. C. Tsoi, and A. D. Back. Function approximation with neural networks and local methods: Bias, variance and smoothness. In *Australian conference on neural networks*, volume 1621. Australian National University, 1996.

[55] X. Li and X. Yao. Cooperatively coevolving particle swarms for large scale optimization. *IEEE Transactions on Evolutionary Computation*, 16(2):210–224, 2012.

[56] A. Lipowski and D. Lipowska. Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391(6):2193–2196, 2012.

[57] H. Liu, Y. Wang, X. Liu, and S. Guan. Empirical study of effect of grouping strategies for large scale optimization. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 3433–3439. IEEE, 2016.

[58] Y. Liu, X. Yao, Q. Zhao, and T. Higuchi. Scaling up fast evolutionary programming with cooperative coevolution. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 2, pages 1101–1108. IEEE, 2001.

[59] F. G. Lobo and D. E. Goldberg. Decision making in a hybrid genetic algorithm. In *Evolutionary Computation, 1997., IEEE International Conference on*, pages 121–125. IEEE, 1997.

[60] S. Mahdavi, S. Rahnamayan, and K. Deb. Center-based initialization of cooperative co-evolutionary algorithm for large-scale optimization. In *Evolutionary Computation (CEC), 2016 IEEE Congress on*, pages 3557–3565. IEEE, 2016.

[61] S. Mahdavi, M. E. Shiri, and S. Rahnamayan. Cooperative co-evolution with a new decomposition method for large-scale optimization. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 1285–1292. IEEE, 2014.

[62] Y. Mei, X. Li, and X. Yao. Cooperative coevolution with route distance grouping for large-scale capacitated arc routing problems. *IEEE Transactions on Evolutionary Computation*, 18(3):435–449, 2014.

[63] E. Mezura-Montes, L. Muñoz-Dávila, and C. A. C. Coello. A preliminary study of fitness inheritance in evolutionary constrained optimization. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)*, pages 1–14. Springer, 2008.

[64] B. L. Miller, D. E. Goldberg, et al. Genetic algorithms, tournament selection, and the effects of noise. *Complex systems*, 9(3):193–212, 1995.

[65] B. L. Miller, B. L. Miller, D. E. Goldberg, and D. E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise," complex systems, vol. 9. *pp*, 193, 1995.

[66] M. Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[67] D. J. Montana. Neural network weight selection using genetic algorithms. *Intelligent Hybrid Systems*, 8(6):12–19, 1995.

[68] P. Moscato et al. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989, 1989.

[69] M. N. Omidvar, X. Li, Y. Mei, and X. Yao. Cooperative co-evolution with differential grouping for large scale optimization. *IEEE Transactions on Evolutionary Computation*, 18(3):378–393, 2014.

[70] M. N. Omidvar, X. Li, Z. Yang, and X. Yao. Cooperative co-evolution for large scale optimization through more frequent random grouping. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[71] M. N. Omidvar, X. Li, and X. Yao. Cooperative co-evolution with delta grouping for large scale non-separable function optimization. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[72] N. Omidvar. DECC-DML source code . `https://bitbucket.org/mno/`. Accessed: 2019-01-30.

[73] M. Pelikan and K. Sastry. Fitness inheritance in the bayesian optimization algorithm. In *Genetic and Evolutionary Computation–GECCO 2004*, pages 48–59. Springer, 2004.

[74] T. Pencheva, K. Atanassov, and A. Shannon. Modelling of a stochastic universal sampling selection operator in genetic algorithms using generalized nets. In *Proceedings of the Tenth International Workshop on Generalized Nets, Sofia*, pages 1–7, 2009.

[75] C. Peng and Q. Hui. Comparison of differential grouping and random grouping methods on sccpso for large-scale constrained optimization. In *Evolutionary Computation (CEC), 2016 IEEE Congress on*, pages 2057–2063. IEEE, 2016.

[76] E. Popovici and K. De. Jong. In *"Understanding competitive co-evolutionary dynamics via fitness landscapes" Artificial Multiagent Symposium Part of the 2004 AAAI Fall Symposium on Artificial Intelligence*, 2004.

[77] M. A. Potter and K. A. D. Jong. a co-operative co-evolutionary approach to function optimization. in PPSN III: Proceedings of the 3rd International Conference on Parallel Problem Solving from Nature pp. 249 257, 1994.

[78] P. Preux and E.-G. Talbi. Towards hybrid evolutionary algorithms. *International transactions in operational research*, 6(6):557–570, 1999.

[79] K. Price, R. M. Storn, and J. A. Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[80] P. Price and B. Evolution. Saunders college publishing. 1998.

[81] M. Ptashne. How gene activators work. *Scientific American*, 260(1):40, 1989.

[82] A. K. Qin and P. N. Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1785–1791. IEEE, 2005.

[83] R. Qu. *Case-based reasoning for course timetabling problems*. PhD thesis, University of Nottingham, 2002.

[84] S. Rahnamayan and G. G. Wang. Center-based sampling for population-based algorithms. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 933–938. IEEE, 2009.

[85] T. Ray and X. Yao. A cooperative coevolutionary algorithm with correlation based adaptive variable partitioning. pages 983–989, (Trondheim,Norway),, 2009. Proceedings of the IEEE Congress on Evolutionary Computation.

[86] I. Rechenberg. Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution. *Frommann-Holzboog*, 39:40, 1973.

[87] M. Reyes-Sierra and C. A. Coello Coello. Dynamic fitness inheritance proportion for multi-objective particle swarm optimization. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 89–90. ACM, 2006.

[88] J. N. Richter and D. Peak. Fuzzy evolutionary cellular automata. *Proccedings of ANNIE*, 2002.

[89] M. Ridley. Evolution blackwell scientific publications. *Boston, MA*, 1993.

[90] C. D. Rosin and R. K. Belew. New methods for competitive co-evolution. In *Evolutionary Computation*, pages 1–29. vol. 5, 1997.

[91] R. Sarker, M. Mohammadian, X. Yao, E. Optimization, K. A. Publishers, and M. A. Norwell. Usa. 2002.

[92] K. Sastry, D. E. Goldberg, and M. Pelikan. Don't evaluate, inherit. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 551–558. Morgan Kaufmann Publishers Inc., 2001.

[93] K. Sastry, M. Pelikan, and D. E. Goldberg. efficiency enhancement of genetic algorithms via building-block-wise fitness estimation, in congress on evolutionary computation. 2004. CEC2004., pp. 720 727,2004.

[94] E. Sayed, D. Essam, R. Sarker, and S. Elsayed. Decomposition-based evolutionary algorithm for large scale constrained problems. *Information Sciences*, 316:457–486, 2015.

[95] H.-P. Schwefel. Collective intelligence in evolving systems. In *Ecodynamics*, pages 95–100. Springer, 1988.

[96] H.-P. Schwefel. Evolution and optimum seeking. sixth-generation computer technology series, 1995.

[97] M. Sehrawat and S. Singh. Modified order crossover(ox) operator. *International Journal on Computer Science and Engineering*, 3(5):2019–2023, 2011.

[98] S. Shan and G. G. Wang. Metamodeling for high dimensional simulation-based design problems. *Journal of Mechanical Design*, 132(5):051009, 2010.

[99] K. Sims. Evolving 3d morphology and behavior by competition. In *Artificial life*, pages 353–372. 1(4, 1994.

[100] R. E. Smith, B. A. Dike, and S. A. Stegmann. fitness inheritance in genetic algorithms. in SAC 95: Proceedings of the 1995 ACM symposium on applied computing, pp. 345 350, ACM Press, 1995.

[101] R. E. Smith and E. Smuda. Adaptively resizing populations: Algorithm, analysis, and first results. *Complex Systems*, 9(1):47–72, 1995.

[102] N. Soni and T. Kumar. Study of various crossover operators in genetic algorithms. *International Journal of Computer Science and Information Technologies*, 5(6):7235–7238, 2014.

[103] N. Soni and T. Kumar. Study of various mutation operators in genetic algorithms. *International Journal of Computer Science and Information Technologies*, 5(6):4519–4521, 2014.

[104] M. Srinivas and L. M. Patnaik. Genetic algorithms: a survey. *Computer*, 27(6):17–26, June 1994.

[105] R. Storn. System design by constraint adaptation and differential evolution. *IEEE Transactions on Evolutionary Computation*, 3(1):22–34, 1999.

[106] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[107] C. Sun, Y. Jin, J. Zeng, and Y. Yu. A two-layer surrogate-assisted particle swarm optimization algorithm. *Soft computing*, 19(6):1461–1475, 2015.

[108] J. Sun and H. Dong. Cooperative co-evolution with correlation identification grouping for large scale function optimization. In *Information Science and Technology (ICIST), 2013 International Conference on*, pages 889–893. IEEE, 2013.

[109] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved January 29, 2019, from `http://www.sfu.ca/~ssurjano`.

[110] G. Syswerda. A study of reproduction in generational and steady state genetic algorithms. *Foundations of genetic algorithms*, 2:94–101, 1991.

[111] E.-G. Talbi. A taxonomy of hybrid metaheuristics. *Journal of heuristics*, 8(5):541–564, 2002.

[112] J. N. Thompson. The coevolutionary process. *Chicago: University of Chicago Press*, 226:2009–07, 1994.

[113] G. A. Trunfio and A. Cooperative. Coevolutionary differential evolution algorithm with adaptive subcomponents, procedia computer science. 51:834–844, 2015.

[114] T. Ueno, N. Sunaga, and H. Asada. Mechanism and control of a dynamic lifting robot. In *International Symposium on Automation and Robotics in Construction and Mining*, pages 95–102, 1996.

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

[116] D. A. V. Veldhuizen. Multiobjective evolutionary algorithms: classifications, analyses, and new innovations. Technical report, DTIC Document, 1999.

[117] T. Weise, R. Chiong, and K. Tang. Evolutionary optimization: Pitfalls and booby traps. *Journal of Computer Science and Technology*, 27(5):907–936, 2012.

[118] D. Whitley. In *in Proceedings of the Third International Conference on Genetic Algorithms. Morgan Kaufmann pp. 116121, title = The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best, year = 1989,.*

[119] T. Yalcinoz, H. Altun, and M. Uzam. Economic dispatch solution using a genetic algorithm based on arithmetic crossover. In *2001 IEEE Porto Power Tech Proceedings (Cat. No.01EX502)*, volume 2, pages 4 pp. vol.2–, 2001.

[120] T. Yamada and C. Reevesm. Solving the c/sub sum/permutation flowshop scheduling problem by genetic local search. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 230–234. IEEE, 1998.

[121] Q. Yang, H.-Y. Xie, W.-N. Chen, and J. Zhang. Multiple parents guided differential evolution for large scale optimization. In *Evolutionary Computation (CEC), 2016 IEEE Congress on*, pages 3549–3556. IEEE, 2016.

[122] Z. Yang, K. Tang, and X. Yao. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, 178:2986–2999, August 2008.

[123] Z. Yang, K. Tang, and X. Yao. Self-adaptive differential evolution with neighborhood search. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1110–1116. IEEE, 2008.

[124] Z. Yang, X. Yao, and J. He. Making a difference to differential evolution. In *Advances in metaheuristics for hard optimization*, pages 397–414. Springer, 2007.

[125] Z. Yang, J. Zhang, K. Tang, X. Yao, and A. C. Sanderson. An adaptive coevolutionary differential evolution algorithm for large-scale optimization. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 102–109. IEEE, 2009.

[126] X. Yao, Y. Liu, and G. Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary computation*, 3(2):82–102, 1999.

[127] S. Ye, G. Dai, L. Peng, and M. Wang. A hybrid adaptive coevolutionary differential evolution algorithm for large-scale optimization. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 1277–1284. IEEE, 2014.

[128] J. Yen, J. C. Liao, B. Lee, and D. Randolph. A hybrid approach to modeling metabolic systems using a genetic algorithm and simplex method. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 28(2):173–191, 1998.

[129] T.-S. Yo and E. D. de Jong. A comparison of evaluation methods in coevolution. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 479–487, England, United Kingdom, 2007. London.

[130] Q. Zhang, W. Liu, E. Tsang, and B. Virginas. Expensive multiobjective optimization by moea/d with gaussian process model. *IEEE Transactions on Evolutionary Computation*, 14(3):456–474, 2010.

[131] W.-X. Zhang, W.-N. Chen, and J. Zhang. A dynamic competitive swarm optimizer based-on entropy for large scale optimization. In *Advanced Computational Intelligence (ICACI), 2016 Eighth International Conference on*, pages 365–371. IEEE, 2016.

[132] L. Zhao, W.-K. Cao, and Y.-T. He. Fitness inheritance-based evolutionary algorithm and its application in hybrid electric vehicle design. *International Journal of Wireless and Mobile Computing*, 7(2):180–186, 2014.

[133] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.