Improving Results of Differential Evolution Algorithm

Rushikesh Sawant

A dissertation submitted for the degree of

Msc in Dependable Software Systems

Department of Computer Science

Faculty of Science and Engineering

National University of Ireland Maynooth

September 2015.

Head of Department: Dr. Adam Winstanley

Supervisor: Dr. Diarmuid O'Donoghue

Word Count: 20439

Contents

List of Figures

List of Tables

Abstract

Optimisation problems are of prime importance in scientific and engineering communities. Many day-to-day tasks in these fields can be classified as optimisation problems. Due to their enormous solution spaces, optimisation problems frequently lie in class NP. In such cases, engineers and researchers have to rely on algorithms and techniques that can find sub-optimal solutions to these problems. One of the most dependable algorithms for numerical optimisation problems is Differential Evolution (DE). Since its introduction in the mid 1990's, DE has been on the fore front when it comes to applicability of optimisation algorithms to variety of real-parameter optimisation problems. This popularity of DE has driven intensive research to further improve its capability to find optimal solutions.

In this thesis we present a variant of DE to produce improved solutions with greater reliability. In doing so, we introduce a novel strategy to incorporate ancestral vectors into the optimisation process. We show that a controlled introduction of ancestral vectors into the optimisation process has a generally positive influence on convergence rate of the algorithm. Evaluation of the proposed algorithm forms a major part of this work, as an empirical evidence serves to demonstrate the performance of stochastic algorithms. The resulting implementation of the algorithm is made available as an open source software along with its reference manual.

Acknowledgements

I would like to express my deep gratitude to Dr Diarmuid O'Donoghue for being a very kind and patient supervisor. Without his constructive guidance this thesis would not have been completed. His numerous suggestions while reviewing this work have shaped this thesis to its current state. I am in a great debt to him for all the efforts he took for helping me in this work. I would also like to offer my special thanks to Dr Rosemary Monahan for all the support, technical ideas, and suggestions regarding writing this thesis. Finally, I am thankful to all the people I met at MU for making my stay here so enjoyable.

Contributions

Publications arising from Thesis:

An Ancestor based Extension to Differential Evolution (AncDE) for Single-Objective Computationally Expensive Numerical Optimization, T*he annual IEEE Congress on Evolutionary Computation (IEEE CEC)*, *Special Session and Competition on Bound Constrained Single-Objective Computationally Expensive Numerical Optimization* Sawant R, Hatton D, O'Donoghue D.P, Sendai, Japan, May 2015.

Acknowledgment arising from Thesis:

Susan J. Lolle and Diarmuid P. O'Donoghue, "**The RNA cache hypothesis: Mechanistic considerations and evolutionary implications**," *Frontiers in Plant Science*, 2015 (submitted).

CHAPTER 1

Introduction

Optimization is everywhere. Engineers and researchers are often confronted with various optimisation problems in their day-to-day work. Application areas range from Computer Science, Telecommunications, Finance, to Civil Engineering and so on. Application servers in distributed information systems are tuned to achieve a required level of performance in order to attain business goals[1]. Airlines try to optimise their schedules to minimise resource consumptions, even civil engineers run simulations to find optimal values before proceeding to physical layout for structures, such as reservoirs [2]. NASA's use of an optimisation algorithm for the design of an X-Band antenna for Space Technology 5 Mission [3] adds one more example to depict the ubiquity of optimisation problems. Reliability is an important aspect of such systems; which essentially traces back to how these systems are designed and built. That is, whether trade-off between different properties of the system has been optimally balanced or not.

In general, the objective is to find the best way to optimise certain properties of the system without violating any imposed constraints. However, large number of such problems remain in the class of NP [4] problems. One of the reasons for this is the enormous search space of these problems. In this case, a compromise is to try to obtain a sub-optimal solution (i.e. what is called as "good enough" solution). Hence, the problem at hand reduces to finding a solution that is plausible for a concerned system.

The frequent and ubiquitous nature of such problems have increased our dependence on algorithms that can target optimisation problems efficiently with currently available technology. It has driven intensive study to find and, or, improve optimisation techniques and algorithms to achieve better results in a feasible time using available computational resources[5].

For this work we are concerned with one specific class of optimisation problems, called numerical optimisation[1]. Input parameters to these problems are in real space, $\mathbb{R}$. Specifically, numeric optimisation is of prime concern in engineering and scientific fields.

[1] Also called continuous optimisation

1.1 Objective

Differential Evolution (DE) is one such stochastic algorithm to solve numerical optimisation problems[6]. All of the examples discussed in the outset can be solved by this algorithm efficiently. This thesis set out to improve the quality and reliability of optimisation results produced by standard DE algorithm. The specific goal of this thesis was to improve the convergence rate[2] by introducing *an ancestral cache* of recently discarded solutions produced during some previous generation(s). The scope of this work also included an investigation of the influence of ancestral vectors[3] on the optimisation process and the proposal of a novel technique to improve results obtained with the standard DE algorithm. Empirical investigation of apposite value ranges for new control parameters for the proposed algorithm inevitably became central part of this work.

1.2 Contributions and Publication

This thesis proposes a new strategy for improving the convergence rate of differential evolution. We present an analytical study of the influence of *introducing* ancestral vectors into the process of evolutionary optimisation. Proposed strategy can be seen as an improvement to the original DE algorithm. Hence, recently proposed methods in DE community, that improve on DE using techniques, such as local search or "learning from experience", for example SaDE[7], can arguably incorporate the strategy proposed here as part of their technique to further improve their results.

The proposed algorithm, called *AncDE* (Ancestral Differential Evolution), primarily improves on standard DE algorithm and also on a recently proposed DE variant, *ArpDE*, by Hatton and O'Donoghue (*submitted, decision pending*), as observed through evaluation results. AncDE is made available as an open source implementation with a public API for using it in client environments. A "user guide" is also provided for using the API and this includes information on other technical details. This user manual provides essential information regarding the value ranges for control parameters of the algorithm; that should be useful to practitioners.

However, we do not intend to investigate the "universal" parameter set in this work. Finding the best possible parameter set has long been the subject of research on its own, and can be seen as a considerable future work. Studies show that finding such parameter set is not a trivial, if not infeasible, task since DE is very sensitive to its controlling parameters and partly the properties of an objective function it is trying to optimise[8, 9, 10]. AncDE essentially inherits these properties from the original DE algorithm. Hence, for this work our approach is to find one "acceptable" parameter set with which "good quality" results are achieved for expensive functions in a benchmark test suite[11]. This parameter set generally varies based on dimensionality of the problem function. For this work we include one such general parameter set for 10 dimensional problems and one for 30 dimensional version of these problems. A reference manual also provides acceptable ranges for newly

[2]The rate at which an evolutionary algorithm moves towards the given target

[3]list of input parameters

proposed AncDE parameters; as they also vary based on properties of the objective function being optimised.

Authors have published a paper on AncDE algorithm at The annual IEEE congress on Evolutions Computation (IEEE CEC) May, 2015 held in Sendai, Japan[12][4], along with performance results generated for a set of benchmark problems provided by CEC 2015 ("*Bound Constrained Single-Objective Computationally Expensive Numerical Optimisation*"). These results are also presented in this thesis for evaluation purposes and for comparison with the original DE algorithm. Also, results produced by AncDE have been used in an evolutionary theory paper submitted to "Frontiers in Plant Science" by Lolle *et al.* (2015).

1.2.1 Evaluation

We evaluate our work using the method provided by IEEE CEC along with some other statistical methods. The evaluation includes comparative analysis against DE and AncDE. Proposed algorithm is applicable in all problem instances where DE is applicable, without employing any additional constraints, hence primary comparative study is performed with DE's performance, with the best achieved parameter configuration. A public API made available to the clients has been thoroughly tested for its robustness against its functional specification documented in AncDE reference manual and source code documentation.

1.3 Thesis Outline

The structure of the thesis is as follows. Chapter 2 introduces Differential Evolution and Numerical Optimisation. And the current state-of-the-art in the DE community is also presented. AncDE and its implementation details are presented in Chapter 3. Chapter 4 enumerates all functions included in IEEE CEC 2015 benchmark suite and the experimental setup. In Chapter 5 we present software engineering aspect of this project. Chapter 6 then presents the evaluation of AncDE including results obtained with AncDE and a comparison with DE using the CEC method and some additional statistical methods. In chapter 7 we conclude our work with discussion on some probable future work that could be based on the work presented in this thesis.

[4]The paper was *in press* at the time of writing this thesis

CHAPTER 2

Differential Evolution

This chapter introduces the evolutionary approach to numerical optimisation and also introduces Differential Evolution (DE) algorithm. These topics are discussed to the level of depth necessary to understand subsequent discussion in this thesis. The remaining part of the chapter presents a discussion on some related work in the field.

2.1 Numerical Optimisation

What is optimisation? In general, optimisation is when one tries to find the best way to perform some task or use available resources, without violating any constraints that are imposed on a system. Formally, optimisation is the process of minimising or maximising a function which is subjected to constraints on its input parameters.

Optimisation involves maximising or minimising some given *objective* problem[13]. It is a quantitative measure of the performance of the system under study, such as profit, time or any combination of quantities that can be presented by a single number. The objective depends on problem *variables* (also called *unknowns*). These correspond to certain characteristics of a system. The goal of optimisation is to find the values of these variables that will optimise the objective. Often these variables are constrained in some way, for example, a non-negative loan value.

2.1.1 Mathematical Formulation

We give mathematical formulation similar to the one given by Quing[5] to precisely define optimisation as follows.

Definition 2.1.1. Find $x^* = [x_1^* \; x_2^* \; ... \; x_N^*] \; \in \; D^N = D_1 \cap D_2 \cap ... D_N$
where,

$$f_i^{min}(x^*) \leq f_i^{min}(x), \forall x = [x_1\ x_2\ ...\ x_N]\ \in\ D^N,\quad 1 \leq i \leq N_{f^{min}},$$
$$f_i^{max}(x^*) \geq f_i^{max}(x), \forall x\ \in\ D^N,\quad 1 \leq i \leq N_{f^{min}},$$
$$c_i^{=}(x^*) = 0,\quad 1 \leq i \leq N_{c^{=}},$$
$$c_i^{+}(x^*) > 0,\quad 1 \leq i \leq N_{c^{+}},$$
$$c_i^{-}(x^*) < 0,\quad 1 \leq i \leq N_{c^{-}}.$$

Here, x is a vector of variables corresponding to a list of parameters and x_i indicates the ith component of this vector. Here, x^* is the optimal solution that we seek to identify in an N-dimensional search space D^N. f_i is the ith objective function of x that we want to maximise, $f_i^{max}(x^*)$, or minimise, $f_i^{min}(x^*)$. c_i are ith constraint functions, where, $c_i^{=}(x^*)$ is the equality constraint function, $c_i^{+}(x^*)$ is the positive constraint function, and $c_i^{-}(x^*)$ is the negative constraint function. In the case of numerical optimisation, the search space D_i is *continuous* (i.e. $x \in \mathbb{R}^n$, where n is the dimension of the problem).

An optimisation problem is thus made up of *optimisation parameters x*, *objective functions f*, and *constraint functions c*. An objective function is what we want to optimise lies at the heart of the problem. Frequently the objective is to minimise some "cost", and hence most of the optimising algorithms are targeted to find a global minimum, also called global optimum. Figure 2.1 shows sample objective function with global maximum and minimum. It also has multiple local optima. In our case the global optimum always refers to global minimum of such functions.

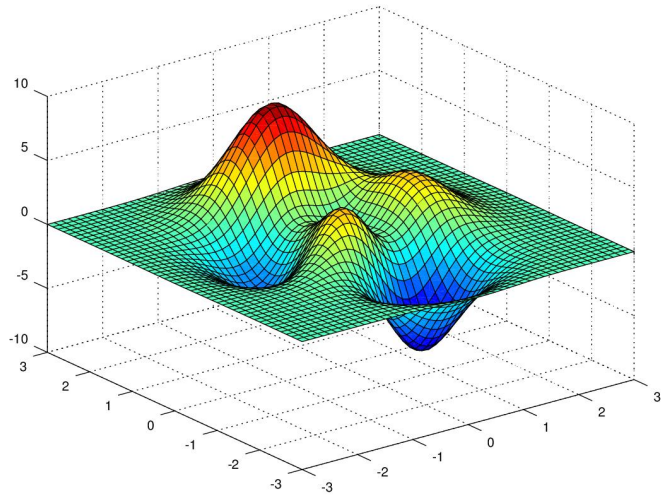


Figure 2.1: Function with multiple optima

2.2 Differential Evolution

From the large list of algorithms targeting optimisation, one class of algorithms, called Evolutionary algorithms (EA), uses mechanism inspired by the biological evolution. The idea behind EAs is to use Drawinian Principles of "Survival of the Fittest" for automated problem solving[14]. EAs are mathematically less complex than their deterministic counter parts. However, EAs require relatively more function evaluations, and due to their non-deterministic nature their results are not clonable.

Despite this fact, EAs are much more preferred in practice as they are applicable to wider class of optimisation problems and impose less restrictions[5]. EAs are population based stochastic algorithms being composed of the basic steps: *reproduction*, *mutation*, *recombination*, and *selection*.

Of these four steps, reproduction, mutation, and recombination together are responsible for producing new individuals with mixed properties from randomly chosen individuals from the current population. In selection stage then old individuals are replaced by new individuals, if the new ones managed to produce better results. This process mimics the biological phenomenon called "survival of the fittest."

One such algorithm we are concerned with is Differential Evolution (DE), introduced by Storn and Price [6] for numerical optimisation. DE has been subject of intensive study since its inception in 1997 due to its mathematical elegance, applicability to problems in various fields, and ease of use[1].

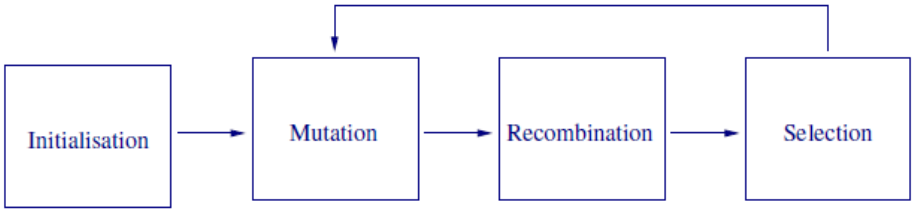


Figure 2.2: Basic step in Differential Evolution

Figure 2.2 shows four main steps of the DE algorithm and we shall discuss each in turn. Basically, DE employs an *evolution loop*, consisting of mutation, recombination, and selection, until the specified criteria to stop this loop is reached. As discussed earlier, optimisation problems, in general, have multiple input parameters. An array of such parameters is called a *vector*, x, in DE terminology. DE operates on population containing NP solutions each being a D-dimensional vector. One complete generation, G, is utilised in one pass of the algorithm. A vector in a current generation is denoted as follows.

$$x_{i,G} \quad \text{where, } i = 1,2,...,NP \tag{2.1}$$

[1] 9200+ Google Scholar citations were recorded at the time of writing this thesis.

2.2.1 Initialisation

DE requires an initialisation step to fill the first generation of vectors with with random values subject to the the given constraint of the objective function. From this point onwards DE employs a *self-referential* population reproduction scheme; it mutates and recombines vectors from within the population to produce a new (and improved) population of the same size, NP.

2.2.2 Mutation

In its mutation step, DE generates a new vector called *mutant vector*, which is generated by adding weighted difference of two random vectors, producing a temporary vector called *difference vector*, to a third vector within current population. Figure 2.3 shows a snapshot of difference vector generation.

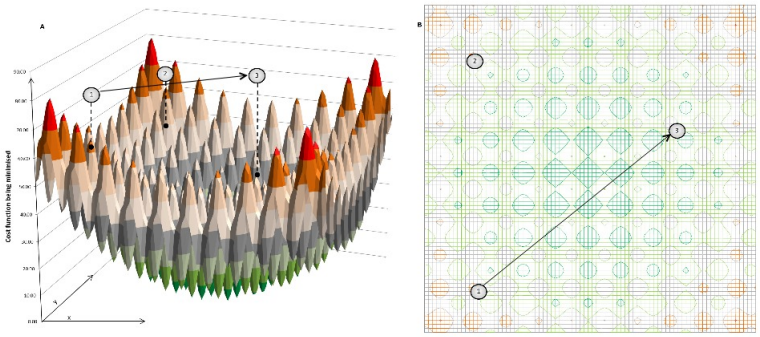


Figure 2.3: Difference vector generated with weighted difference of two random vectors, here vector 3 - vector 1. Show optimisation landscape is of Rastrigin optimisation problem.

The whole process to produce mutant vector is called as *differential mutation*. For mutation there are many strategies proposed by Price *et al.*[15]. In order to discuss details of mutation we present an equation from one of these strategies as shown in 2.2.

$$v_{i,G+1} = x_{r1,G} + F.(x_{r2,G} - x_{r3,G}) \tag{2.2}$$

Generally, mutant vector, $v_{i,G+1}$, is generated from the current population, where $G+1$ denotes that it is a candidate for subsequent generation. $r_1, r_2, r_3 \in 1...NP$ are randomly chosen indices in the current population, and are mutually different. The scaling factor, $F \in [0,1)$ scales a difference vector produced from $x_{r2,G}$ and $x_{r3,G}$. Vector $x_{r1,G}$ is called as *base vector* and produces the mutant vector when a scaled difference vector is added to it. Mutant vectors are generated for each individual vector in the current population. The current vector, for which a mutant is produced, is called as *target vector*, $x_{i,G}$.

2.2.3 Crossover

In its third step, DE employs *uniform crossover*, also called, *discrete recombination*. Crossover generates a new trial vector (also called *offspring*), $u_{i,G+1}$, by randomly copying input parameters either from a target vector or a mutant vector.

$$u_{j,i,G+1} = \begin{cases} v_{j,i,G+1} & \text{if } rand_j[0,1] < CR \text{ or } j = k, \\ x_{j,i,G} & \text{otherwise} \end{cases} \tag{2.3}$$

In equation 2.3, $j \in 1...D$ and $CR \in [0,1]$ is a crossover constant that controls the fraction of parameters that a trial vector, u, gets from the mutant vector. $rand_j[0,1]$ in 2.3 indicates the j-th evaluation of pseudo random number generator (PRNG).

2.2.4 Selection

Finally, the selection step decides who survives; the new trial vector or the existing vector target vector in the current generation. The trial vector is evaluated to obtain new "test cost" which is then compared against old available cost produced by target vector. If $u_{i,G+1}$ yields smaller objective value than the target vector $x_{i,G}$, then $u_{i,G+1}$ replaces $x_{i,G}$.

This process of mutation, recombination, and selection continues until the desired target value is reached or specified number of evaluations are completed. Parameters *NP, F,* and *CR* are user defined, and together they are referred to as *control parameters* of DE.

2.3 Related Work

Since 1997, numerous techniques and strategies have been proposed to improve DE or to attack a specific class of numeric optimisation problems. A recent survey published by Das *et al.*[16] gives a concise snapshot of advancements in DE community. Applying standard DE to a given problem (to obtain acceptable results) requires finding the right strategy[2] to be chosen for the problem under study. This involves intensive trial-and-error search efforts with available strategies. Also, tuning control parameters of DE requires similar search with experimental runs of the algorithm. The goal here is to find out whether there is another optimal setting for the algorithm that can give better results than the current setting in place. As this can expend huge amount of computation costs, techniques to automatically *learn and adapt* are being investigated (e.g., EPSDE[17], CoDE[18], ESADE[19], etc). One such popular algorithm, SaDE[7], maintains a pool of different strategies and during execution it gradually learns which strategies to use and the values of control parameters for them based on earlier experience. Another approach by Brest *et al.*[20] is to encode control parameters, F and CR, into individual target vectors. However, these values are then in-turn controlled by two newly introduced parameters τ_1 and τ_2. AncDE does not use any learning or internal search technique to improve the convergence rate, hence required computational resources are equivalent to original DE. Solution quality is improved by

[2]Strategy is an umbrella term used for denoting process of mutation and cross over together.

the virtue of right amount of *inter-generational* mutation of ancestral vectors with target vectors.

Zaharie[21] proposed another adaptive strategy, called ADE, to improve the convergence rate. Zaharie's idea is based on controlling population diversity to achieve balance between search space exploration and exploitation[3]. Similarly, Das *et al.*[22] proposed a topological neighbourhood based mutation scheme also to achieve a balance between exploration and exploitation. AncDE, on the other hand, introduces diversity by occasionally re-introducing ancestral vectors during the mutation step. The age of an ancestral population and their introduction frequency is controlled by two new control parameters proposed for AncDE.

Finally, Zheng *et al.*[23] proposed JADE which uses an optional external archive of the old population to provide information to direct the progress in solution space. Mathematical formulation of their strategy can be represented as follows:

$$v_{i,G+1} = x_{i,G)} + F_i.(x^p_{best,G} - x_{i,G}) + F_i.(x_{r1,G} - X'_{r2,G}) \tag{2.4}$$

In equation 2.4, $p \in (0,1]$ and $x^p_{best,G}$ is a randomly sampled vector from the 100% vectors of the current population. $X'_{r2,G}$ is, however, randomly sampled from $P \cup A$. Where, P denotes current population and A is an external archive of old vectors discarded at selection step. In a case when archive size exceeds, some vectors are randomly discarded to keep archive at NP.

AncDE is similar to JADE in that AncDE also maintains archive of old vectors. However, archive maintained by AncDE is an *ancestral* archive; the i-th target vector from current population relates to i-th vector from ancestral population. For the same reason, AncDE does not run into ancestral archive overflow condition. Also, AncDE's strategy is different from that of JADE, as shall be discussed in Chapter 3. Finally, as implementations of these algorithms were not readily available, we did not benchmark against them in this thesis.

2.3.1 Earlier Work Employing Ancestors

The idea of introducing an ancestral population (or archive) comes from the recent study published by O'Donoghue *et al.*[24]. This study performs investigations into genetic restorations from non-parental ancestors. O'Donoghue *et al.* proposed an ancestral repair strategy within an evolutionary algorithms to solve constraint based optimisation problems. The same group has incorporated this strategy into standard DE, and the variant is called as *ArpDE*[4]. This study shows that ancestral vector can have a positive influence on the optimisation process. The idea of occasionally using ancestral vectors during differential mutation (making broad movements in search space) seems to produce competitive results to standard DE.

Their algorithm modifies DE's differential mutation process to incorporate ancestral vectors. Inclusion of ancestors is controlled by two parameters– 1. ancestor replacement

[3]Here, exploration refers to broad movements through the search space seeking new regions. While exploitation refers to small steps around the region found during exploration to find optimum

[4]Paper on ArpDE is submitted; decision is pending.

probability (ARP) 2. and ancestor usage probability (AUP). During differential mutation AUP controls the rate of producing *inter-generational* difference vectors. ARP, on the other hand, controls the rate at which ancestors are replaced by vectors from the current population. That is, with ARP one controls how old the ancestors should be. Authors note that good results were obtained with ARP value of 0.05 and AUP value of 0.15.

The efficiency of evolutionary algorithms is generally characterised by their ability to perform exploration and exploitation during the search process[25, 16]. Hence the optimisation process of DE, and its variants, can also be divided into two aspects– 1. exploration 2. exploitation. Exploration refers to the ability of an algorithm to make broad movements through the search space to quickly explore promising new regions. DE, and its variants, generally start with exploration as the initial population is relatively dispersive and randomly distributed in the search space. As the optimisation process matures, the population starts to converge gradually and clusters around a global or local optima. This state of the algorithm is called as exploitation, making smaller movements around the search space that has been already explored. This aspect can be seen as small refinements applied to an already discovered good solution and is based on the information gathered during exploration.

The exploration power of these algorithms comes from the diversity of the population. Due to diverse population in exploration state, the magnitude of difference vectors in differential mutation step is relatively high, and it consequently results in broad movements through the search space. ArpDE uses an ancestral cache to introduce additional diversity. Its ability to utilise this diversity and produce inter-generational difference vectors between the current and ancestral population broadens the search.

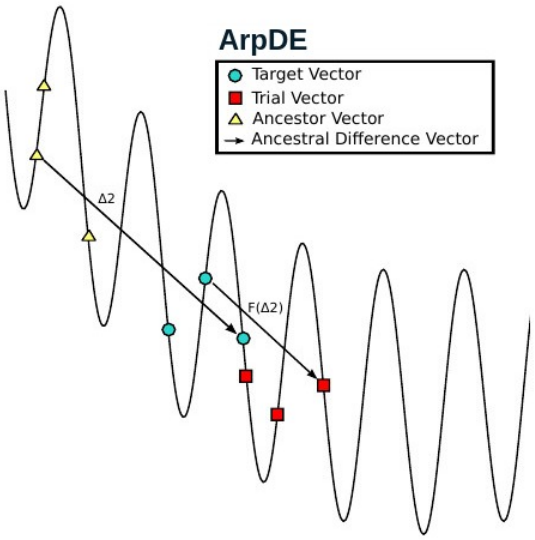


Figure 2.4: ArpDE search space exploration

Figure 2.4 shows ArpDE's solution space exploration mechanism with inter-generational difference vector Δ_2. In this case the magnitude of a vector is increased due to the diversity between two populations. Note that JADE is different from ArpDE, as the external archive used by JADE does not necessarily maintain any relation between two populations.

Building on previous work [24], this thesis attempts to improve the quality and reliability of results produced by an ancestral extension to DE. In doing so, we propose a new criteria for ancestor replacement and a novel strategy for differential mutation to improve over DE. We also observe that results produced by AncDE are better in quality than ArpDE. The challenging part of the work here was to incorporate an ancestral population into differential mutation and to decide the right proportion of such *inter-generational* mutation. The difficulty in investigation arises due to two primary reasons– 1. The sensitivity of these algorithms to control parameters and 2. the properties of objective functions being optimised[5].

2.4 Conclusion

This chapter presented overview of numerical optimisation and Differential Evolution algorithm. We also discussed related work in the field and briefly outlined how it compares with our approach. We also focused on ArpDE which further extended by AncDE. In the next chapter we shall look into more details how AncDE differs from ArpDE.

[5]Objective function properties are discussed in Chapter 5.

CHAPTER 3

Ancestral DE

This chapter details AncDE (Ancestral DE) algorithm discussing its mutation strategy and its ancestral cache maintenance mechanism. The algorithm is presented at a pseudo code level of abstraction to further discuss its computational details. The chapter concludes with a formal description of the AncDE strategy and the role of two new control parameters used by AncDE.

3.1 Ancestral Extension to DE

The DE community is primarily focused on improving the efficiency of DE and employing new techniques more reliably identify optimal solutions[16]. To this end, AncDE's approach is similar to ArpDE (*discussed in chapter 2*). AncDE aims to achieve a fundamental improvement over standard DE by utilising vectors that are getting discarded after failing the *survival test*. Essentially, these are the vectors that were produced during an earlier generations, but now a new generation is better than them.

Unlike ArpDE, AncDE caches these ancestors only when its new offspring vector is producing a good result. In doing so, AncDE requires only one additional *ancestral cache*, in addition to the main population. This results in an algorithm that requires resources (memory and time) equivalent to DE but produces better results.

3.1.1 AncDE Concept

The crux of the strategy is the controlled replacement and usage of vectors in ancestral cache. AncDE employs ancestral vectors to generate inter-generational difference vectors during mutation. Figure 3.1 shows a snapshot of AncDE's mechanism to generate inter-generational difference vector, rather than normal difference vector. We know that vectors in the current generation are always provably better than old generations. However, though

the new population is progressing towards global optimum, it has tendency to get misdirection and getting caught in local optima or stagnate[1]. Apart from helping in making big jumps to explore search space rapidly, ancestral vectors help direct new offspring vectors in the right direction towards the global optimum. In figure 3.1, for example, vector 6 denotes an ancestral vector utilised to generate a new offspring vector, 1', that is now progressing in the right direction. AncDE employs a newly proposed strategy to achieve this result, as we shall see in the next section.

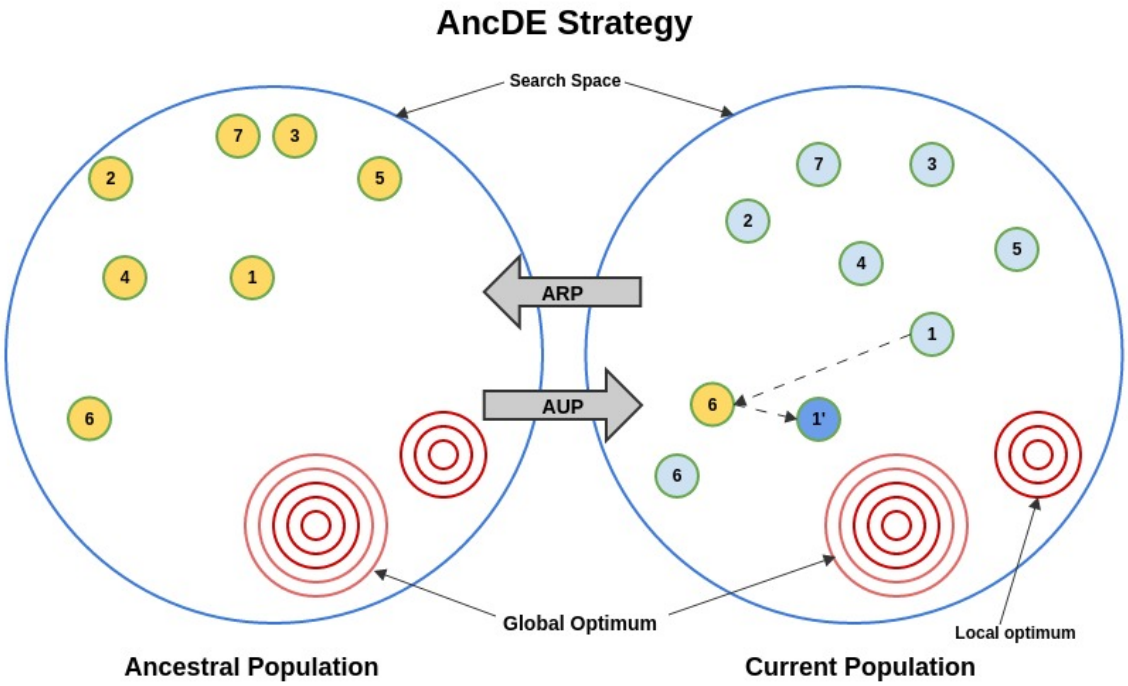


Figure 3.1: AncDE strategy involving random antecedent from ancestral cache to generate new trial vector

At the beginning, both ancestral and current population are identical, and this is the only time in the evolution process they are provably same. After that point, whenever a new low-cost vector is found, AncDE consults to the *arp* (ancestor replacement probability) value to stochastically decide the replacement of a related ancestor in the ancestral cache. For example, consider vector `1` in the current population as shown in figure 3.1. If the newly produced vector, `1'`, is found to be better than `1`, the former one will be discarded and the latter one will take its place in the current population; as per the standard DE evolution process. However, there is an additional step in AncDE, which will (stochastically) store a copy of the (discarded) vector to the *ancestral cache*. This mechanism allow AncDE to keep (stochastically) updating the ancestral cache without running into a cache overflow situation.

[1]Stagnation is a state in the evolution where no new better vector is found even though the current generation is not clustered around local optima

We have to control the use of ancestors, as using ancestors all the time will impede the convergence process. Controlled introduction of ancestral vectors is enabled through user defined *aup* (ancestor usage probability) value. For example, the ancestral vector 6 shown in figure 3.1, in this case, is introduced in the mutation process only because AncDE's *aup* control (stochastically) allowed this instance of mutation to generate inter-generational difference vector.

Both, *arp* and *aup* values has to be determined empirically, and in general are directly related to the properties of optimisation problem and other control parameter values provided to the algorithm. However, we later suggest overall "good" general purpose values based on results of our experiments.

3.1.2 AncDE Algorithm

As discussed earlier, AncDE introduces second population of ancestral vectors (also called *ancestral cache*) into the standard DE algorithm. The four steps of DE remain the same, however, mutation step is changed to allow an occasional usage of vectors from ancestral cache. When a vector from ancestral cache is selected, the strategy used for mutation is also changed to use an inter-generational difference vector between a solution from the main population and one from the ancestral cache. Pseudo code for AncDE algorithm is as follows:

```
Input:  D, NP, F, CR, Range, ARP, AUP, Number of Evaluations
Output:  Minimum cost and vector producing that cost
```

1 Read D, NP, F, CR, Range, and Evaluations

2 Randomly initialise NP vectors as G_0.

$$G_0 = x_{i,j,0} := rand_j[0,1].Range \quad \text{i=1,...,NP} \quad \text{j = 1,...,D}$$

Where, G_0 is initial population and $x_{i,j,0}$ is the i-th vector in G_0. And j denotes the parameter index of that vector.

3 Copy all vectors from current population G_0 to ancestral population *AncG*

4 for each vector in G, $x_{i,G}$, from 1 to NP

4.1 Evaluate objective function with $x_{i,G}$ as input vector.

4.2 Store fitness cost in $cost_i$

end for

5 While *evaluations* count $\leq$ specified Number of Evaluations do

5.1 for each vector in G, $x_{i,G}$, from 1 to NP

5.1.1 Generate new mutant
vector, $v_{i,G+1}$, either using inter-generational
difference vector or using normal difference
vector as follows:

if $rand_j[0,1] < aup$ then
  $v_{i,G+1} := x_{i,G} + F.(x_{r,AncG} - x_{i,G})$
else
  $v_{i,G+1} := best_G + F.(r_{1,i,G} - r_{2,i,G})$, such that $r_1 \neq r_2 \neq x_i$
end if

where, r denotes randomly selected vectors from
respective generations

5.1.2 Perform binomial crossover to generate new offspring vector,
$u_{i,G+1}$,
as follows:

for each j-th parameter, 1,...,D, in $x_{i,G}$
  if $rand_j[0,1] \leq CR \parallel j = D$ then
    $u_{i,j,G+1} := v_{i,j,G+1}$
  else
    $u_{i,j,G+1} := x_{i,j,G+1}$
  end if

5.1.3 Perform selection to decide the survival of the new vector,
$u_{i,G+1}$, as follows:

5.1.3.1 Evaluate objective function with $u_{i,G+1}$ as an input vector
and store the resulting *testcost*

5.1.3.2 if $testcost \leq cost_i$ then
  if $rand_j[0,1] \leq arp$ then
    copy contents of $x_{i,G}$ to $x_{i,AncG}$
  end if
  Generate $x_{i,G+1}$ by copying contents of vector $u_{i,G+1}$
  to vector $x_{i,G}$
else
  Generate $x_{i,G+1}$ by keeping contents vector $x_{i,G}$
  unchanged.
end if

end for

5.2 Increase generation count: $G := G+1$

end while

```
6 Output minimum cost obtained and the vector producing that cost.
```

Algorithm 3.1: AncDE Algorithm

Algorithm 3.1 works as follows. Control parameters *NP*, *F*, and, *CR* are externally set parameters to the algorithm. Note that optimisation problems have a vector of parameters as their input. AncDE operates on *population* of such vectors; feeding one vector at time to the problem to evaluate and obtain the new cost. Externally set control parameters influence how algorithm generates new trial vectors (new offspring) from the current vector population. *NP*, here, corresponds to the vector population size and *D* represents the dimension of each vector (i.e., number of parameters in each vector). One pass of the algorithm, beginning at `5.1`, is referred to as current *generation, G*. We say next generation, $G+1$, has begun when all the vectors from the population have been evaluated and generation counter is increased at `5.2`.

Additional two control parameters for AncDE are – 1. Ancestral replacement probability, $arp \in [0.0, 1.0]$, which controls the probability that an ancestor is replaced by a vector from the current population, 2. Ancestral usage probability, $aup \in [0.0, 1.0]$, which controls the probability that an ancestor will be used produce inter-generational difference vector with AncDE's mutation strategy. Essentially, we want to control how many generations old the ancestors should get and how often they should be used in a mutation process. Both *aup* and *arp* are user defined parameters.

Input parameter, *Range*, and input vector dimensions, *D*, are related to an objective function. *Range* puts constraint on the values of parameters in the input vector such that, $values \in [-Range, +Range]$. This constraint information is generally available from the optimisation problem itself. It is a practitioner in the field who decides the stopping criteria. This decision is generally based on optimisation problem, and available time and computational resources. Stopping criteria can be a number of evaluations or the algorithm can be to set to run until the objective function does not produce cost within an acceptable range[2].

3.1.2.1 Initialisation

In initialisation step, specified number of vectors, *NP*, are created and are initialised randomly within a given *Range*.

$$P_0 = X_{i,j,0} := rand_j[0,1].Range \qquad \text{i=1,...,NP} \qquad \text{j = 1,...,D} \tag{3.1}$$

In equation 3.1, $rand_j[0,1]$ denotes a *uniformly distributed random value* $\in$ [0.0,1.0] obtained for each *j*-th evaluation of such generator, *rand*. *Range* denotes a randomly selected value from a closed interval $[-Range, +Range]$, where bounds on this interval are user specified.

[2]AncDE can be easily modified for the second condition as well. Check user manual for configuration details

Figure 3.2 shows a snapshot of the initial state for the algorithm. Population for current generation is denoted by, G, and an ancestral population by, $AncG$. This population size, NP, does not change during the entire process as result of self-referential reproduction (explained in chapter 2). Note that we are not concerned about parameter values, denoted by p_1, p_2, etc., as their generation is completely controlled by stochastic initialisation (within the given constraints), and later by reproduction[3]. Hence, we simply show them as p with subscript index, and distinguish ancestral population with vectors in green color.

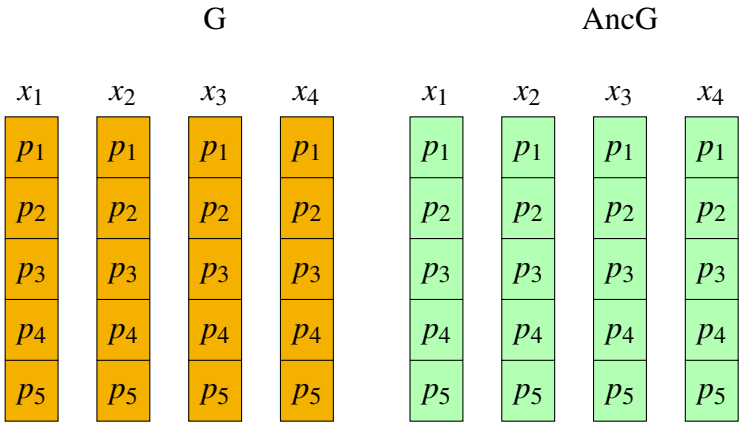


Figure 3.2: Initial vector population with NP=4 and D=5. Both generations G and AncG have same vectors

3.1.2.2 Mutation

After random initialisation, evolution process starts. AncDE performs *differential mutation*[4], where scaled vector difference of two random vectors is added to a third vector, to generate a mutant vector, $v_{i,G+1}$. In DE community there is naming convention for denoting different DE strategies applied for mutation and crossover.

$$DE/x/y/z \tag{3.2}$$

In equation 3.2, x denotes the vector to be mutated, also called as *base vector*. It can be any random vector (*rand*) or the best cost producing vector in the current generation (*best*). y denotes the number of vector differences to be used. Finally, z denotes the crossover method. Crossover method can be binomial (explained later in following section) or exponential. Using this notation, one of DE's well known strategies is called *DE/best/1/bin*. Similarly, AncDE's strategy is called as *AncDE/target/1/bin*, explained as follows.

AncDE Strategy. In AncDE's evolution loop each vector in the current population gets an opportunity to be a *target vector*, $x_{i,G}$. That means for each vector in the current population, a new mutant vector is generated. AncDE's strategy, however, differs from DE's (2.2) such that AncDE introduces *aup* in the process to produce *inter-generational*

[3]Only the values of final vector found by an algorithm at the end are of interest to users (practitioners).
[4]Explained in Chapter 2

difference vector as follows:

$$v_{i,G+1} := \begin{cases} x_{i,G} + F.(x_{r,AncG} - x_{i,G}) & \text{if } rand_j[0,1] < aup, \\ gen\_best_G + F.(x_{1,G} - x_{2,G}) & \text{otherwise} \end{cases} \tag{3.3}$$

In equation 3.3, $v_{i,G+1}$ denotes a new mutant vector generated with this strategy and subscripted r denotes a random vector. For generating mutant vectors, AncDE occasionally introduces one random ancestral vector, $x_{r,AncG}$, to generate an inter-generational difference vector. Whenever this is the case, the target vector, $x_{i,G}$, for whom a mutant is generated, is itself involved in the mutation process. Here, the target vector is used as a base vector and the inter-generational difference is computed between the ancestral vector and the target vector. Due to stochasticity, the mutation process switches between DE's *DE/best/1/bin* and AncDE's *AncDE/target/1/bin*. Note that this switching between two different methods of computation is also a part of AncDE's strategy. Hence, an *inter-generational mutation* is applied to a statistically controlled fraction of the population in each generation.

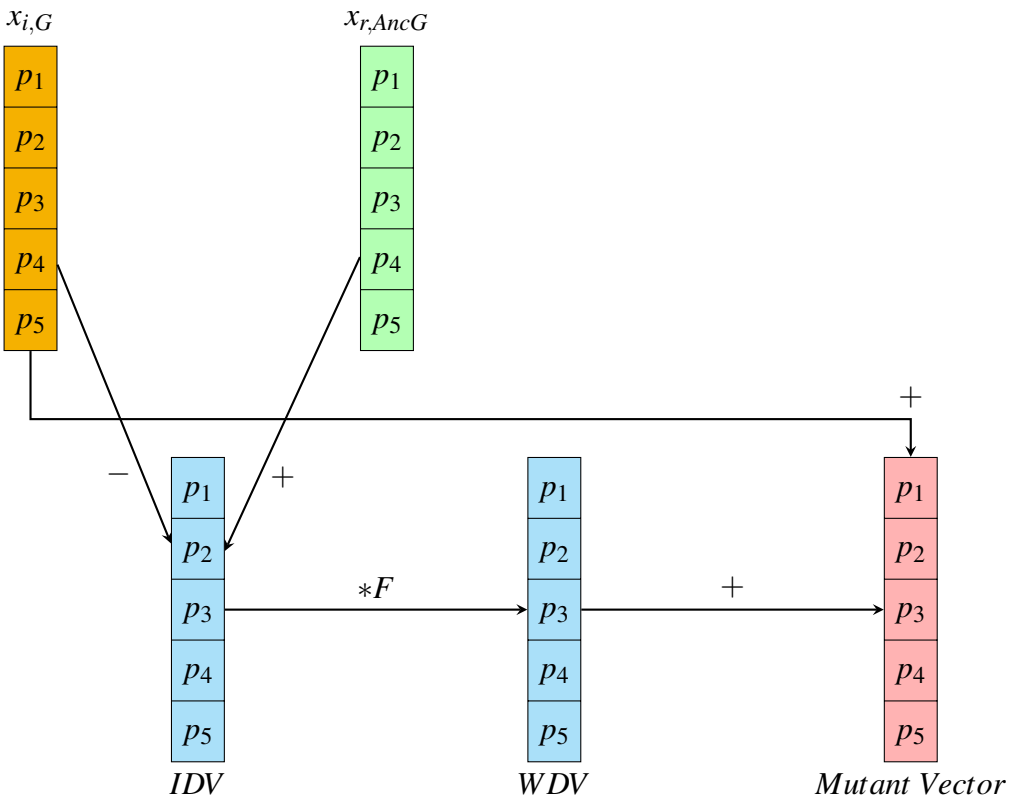


Figure 3.3: AncDE differential mutation when $rand_j[0,1] < aup$. (IDV= inter-generational difference vector and WDV = weighted difference vector).

Figure 3.3 shows the process of mutant vector generation using a random ancestor, as presented in equation 3.3. We use *DE/best/1/bin* in AncDE as this variant of DE is shown

to produce robust results in various studies[26, 10, 8]. Scaling factor, $F \in [0,1)$, controls the rate at which population evolves by scaling the influence of a difference vector. In case of AncDE, a good value for F is $F \in [0.55, 0.70]$, as shown in Chapter 6.

3.1.2.3 Crossover

AncDE does not make any changes to DE's crossover mechanism. A new trial vector, $u_{i,G+1}$, is generated by employing a *binomial crossover*, where the trial vector gets a fraction of parameter values from mutant vector, $v_{i,G+1}$, and the remaining fraction from the target vector, $x_{i,G}$.

$$u_{j,i,G+1} := \begin{cases} v_{j,i,G+1} & \text{if } rand_j[0,1] < CR \parallel j = k, \\ x_{j,i,G} & \text{otherwise} \end{cases} \tag{3.4}$$

In equation 3.4, the crossover probability, $CR \in [0,1]$, influences the proportion of parameters that a trial vector will get from mutant vector generated in mutation step. That is, crossover rate ultimately controls if and how much of the parameters from a mutant vector with ancestral influence are passed on to the new offspring vector.

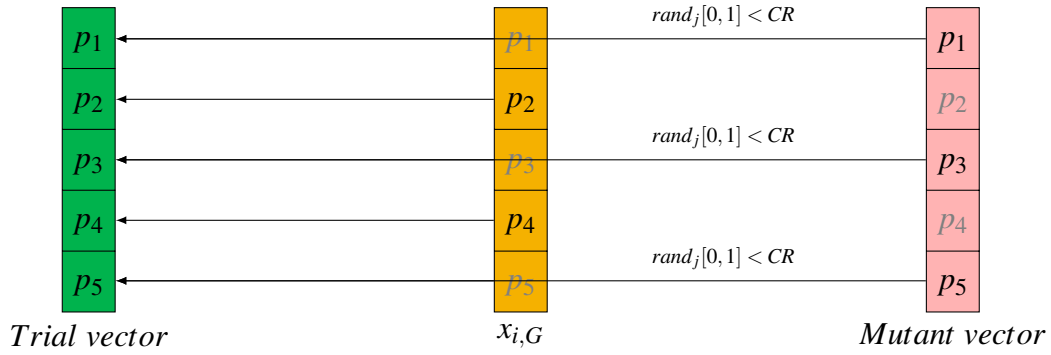


Figure 3.4: Binomial crossover. Parameters either from target vector or from mutant vector are copied to trial vector, based on CR. Parameters with gray color indicate they are *not* copied to the trial vector

Figure 3.4 shows a binomial crossover scheme explained for equation 3.4. Newly generated trial vector at this point is also termed as a mutated child or offspring in evolutionary algorithms terminology[5].

3.1.2.4 Selection

Survival of newly generated trial vector is decided in selection step. The objective function is evaluated using each trial vector as an input. The cost returned for the trial vector is then

compared with the cost of the target vector. If the new cost is better than the old one, the trial vector replaces the target vector in the current population.

$$x_{i,AncG+1} := \begin{cases} x_{i,AncG} & \text{if } rand_j[0,1] \leq arp \wedge newcost \leq mincost, \\ x_{i,G} & \text{otherwise} \end{cases} \tag{3.5}$$

As shown in equation 3.5, AncDE introduces an ancestor replacement probability in this process. Ancestor replacement is decided on two explicit checks– 1. ancestor replacement probability itself 2. replace ancestor by a target vector being discarded only if its corresponding trial vector has produced better cost, *newcost*. Thus, based on *arp*, we occasionally store the target vector that is being discarded. A high *arp* value yields very recent ancestors while a low *arp* value produces very ancient ancestors. Later we shall explore the "ideal" age for this ancestral cache.

The evolution loop of AncDE continues until the stopping criteria is met, for example, until the specified number of function evaluations have been performed. Note that with smaller *NP* more generations will pass quickly, that is, evolution is faster. And with higher *NP*, more function evaluations occur per generation and more population diversity as well. Hence, balanced value for *NP*, along with *F* and *CR*, is essential to improve the convergence rate of AncDE. In Chapter 6 we show that AncDE performs better on smaller *NP* values. That is, AncDE requires relatively *less* memory to produce competitive results. Also note that the usage of inter-generational difference vectors is user defined and can be tuned based on the properties of an objective function and population size. Hence, finding the right ranges of values for these parameters is essential and shall be discussed along with the results.

3.2 Conclusion

This chapter was primarily focused on the idea behind AncDE algorithm and its implementation. We discussed in detail all four steps involved in the evolution process of AncDE algorithm. We concluded with the claim that AncDE requires relatively less memory to produce strong results, which we show in Chapter 6.

CHAPTER 4

Software Engineering - AncDE

This chapter presents the software engineering aspect of AncDE: the algorithm implementation, design, documentation, and its testing are all presented in this chapter. The algorithm has been made available to be integrated in client environments, and thus its public application programming interface (API) is discussed. We also describe how to integrate AncDE in a client environment and how to use its logging mechanism. The chapter concludes with a discussion on functional testing of the public API and some maintenance aspects of AncDE.

4.1 Software Process and R&D

As test driven development was major part of this project, the software process model that we followed was *spiral*. The first step in the development was the integration of IEEE CEC benchmark suite into the design and code, and its validation. After successful integration of this benchmark, software design was finalised and steps were taken to implement the initial version of AncDE. Initial version implemented adapted version of the strategy used by ArpDE. From this point onwards numerous strategies were evaluated before coming up with *target/1/bin*.

Each time a new prototype was produced for experimenting a newly hypothesised strategy to utilise ancestral vectors. Of the various strategies experimented, we briefly discuss some of the interesting ones. *Stochastically initialised ancestors* – This strategy relied on two separate initialisations, one for regular generation and one for ancestral cache. With very low ARP, ancestral vectors can remain diverse (and distant) than the current population. This was the idea behind the strategy; more diverse ancestors should produce larger difference vector. Strategy showed good results for some of the simple functions, but failed to produce even acceptable results for most of the problems. Due to stochastic initialization and low ARP, ancestors tend to stay in initial "random directions". Therefore, most of the times they could not reliably direct new vectors in the right direction. *Scaling factor to*

change directions – Generated mutant vector is always scaled to get a new weighted difference vector. Since ancestral vectors that are too old (like in earlier discussed strategy) tend lead offspring in the wrong direction, idea was to mitigate this problem with stochastic use of scaling factor, which can vary in the given range. Thus, even if there is a misdirection, it is on smaller scale. This strategy produced good results on some of the difficult problems as well, and with a good consistency. However, the number of problems it could optimise were still less than half. It relied on the right combination of AUP and F (scaling factor). *best one to Anc one bin* – This strategy stochastically switched between ancestral strategy to generate difference vector and normal DE strategy utilising best cost producing vector. This strategy produced better results on 8 problems and marked the next step in our strategy exploration. The idea here was that DE's normal strategy should be able to mitigate the misdirection caused by "bad" ancestral vectors. Continuing in this direction and after trying other various variants, *target/1/bin* was decided to be the strategy of choice.

Once the strategy was finalised, the next step was to find overall "good" values for all the control parameters. This part of the project focused on experimenting with different value ranges and combinations of parameter values. Experiments were carried out in an attempt to exploit results documented in DE literature for AncDE. As AncDE introduced two new parameters, sensitivity analysis of the algorithm to these parameters was inherent part of this project.

Finally, we extended AncDE's strategy to introduce multiple ancestors in to the mutation process - a recent ancestor and an older ancestor. These should in theory support the calculation of stochastic second order difference vectors that might further improve convergence. The very initial work is briefly discussed in chapter 7.

4.2 Implementation Overview

We base our implementation of AncDE on the existing open source DE implementation made available by Storn *et al.* on Differential Evolution website[27]. AncDE has been implemented in both Java and C++ programming languages, and is made available as an open source code base. Two versions are aimed to provide a suitable implementation for practitioners' who wish to use AncDE. We focus on the Java version of the algorithm, since its structure is replicated in the C++ version.

Two main components of AncDE are its strategy module and the module which implements its main evolution loop. The strategy module is responsible for implementing our new strategy, while the latter introduces ancestor replacement scheme, both introduced in Chapter 3. Figure 4.1 shows an excerpt from the `AncDETarge1Bin` class which shows how *aup* is utilised in the new strategy, *AncDE/Target/1/Bin*.

```
public class AncDETarget1Bin extends AncDEStrategy {

    ...

        while (counter++ < dim) {
            if ((deRandom.nextDouble() < Cr) || (counter == dim)) {
                // ancestral usage criterion
                if (deRandom.nextDouble() < aup) {
                    x[i] = x[i] + F * (ancG[0][i] - x[i]); // inter-generational
                } else {                                    // difference vector
                    x[i] = gen_best[i] + F * (g0[0][i] - g0[1][i]); // x1 - x2
                }
            }
   ...
```

Figure 4.1: AncDE/target/1/bin

As one can notice in listing 4.1, mutation is performed *after* the crossover criteria is satisfied for the current trial vector. This, at first, seems somewhat contradicting to what is presented in Chapter 3. For runtime efficiency purpose, however, the mutation step is combined with the crossover step. From earlier discussions on trial vector generation in Chapter 2 and 3, it is clear that mutant is a temporary vector, v_i. Its goal is to provide genetic information to the final trial vector, u_i, produced as a result of binomial crossover. In real-world applications, list of input parameters required for optimisation problems can be large, resulting in large dimension vectors. Hence, rather than creating a large temporary mutant vector for each target vector, differential mutation is performed only when crossover is applicable for a j-th parameter. This optimisation avoids a need for creating such large temporary vectors. In figure 4.1, resultant vector, `x[i]`, is a trial vector generated once this process is complete. Note that the criteria, `counter == dim`, makes sure that trial vector gets at least one parameter from a mutant vector, irrespective of the crossover rate.

```
public double optimize() {

      ...

      /*---Apply the AncDE strategy-------------------*/

       targetBin.apply(F, Cr, dim, trial, genbest, rvec, randVecAnc, aup);

       testcost = cecProbs.eval(trial, dim, 1, this.current_func)[0];
       evaluation++;
       // Better solution than target vectors cost ?
       if (testcost <= cost[i]) {
         // replace ancestor by soon to be an ancestor using arp:
           if (deRandom.nextDouble() <= arp) {

              ancG[i] = g0[i];
           }
          // put trial vector in new population
          System.arraycopy(trial, 0, g1[i], 0, dim);

         // and save the new cost value
         cost[i] = testcost;

        ...
}
```

Figure 4.2: AncDE/target/1/bin

Figure 4.2 shows an excerpt from the AncDE kernel, controls the whole process of optimisation; it implements the main evolutionary loop. It invokes the AncDE strategy to produce new trial vectors for each (target) vector in the current population. For every trial vector, the objective function is evaluated producing *testcost* that is compared against the cost produced by the target vector, `cost[i]`. This is a prime criteria for deciding who survives; the target vector or trial vector. If the target vector is getting discarded, AncDE adds a *caching* criteria based on the *arp* replacement probability (as discussed in Chapter 3), to decides whether to cache or not that target vector. Figure 4.2 represents the implementation of equation 3.5 presented in Chapter 3. The AncDE kernel is also responsible for maintaining the evaluations count, total number of generations that have passed, monitoring if target has been reached or not, and the objective function instance being optimised.

4.3 Design and Public API

AncDE is designed to be programmatically integrable in client applications and thus it exposes a public API through that clients can configure and use for optimising their own objective functions. AncDE also integrates the IEEE CEC 2015 benchmark problem suite on "*bound constrained single-objective computationally expensive numerical optimisation*"[11] for evaluation purpose.

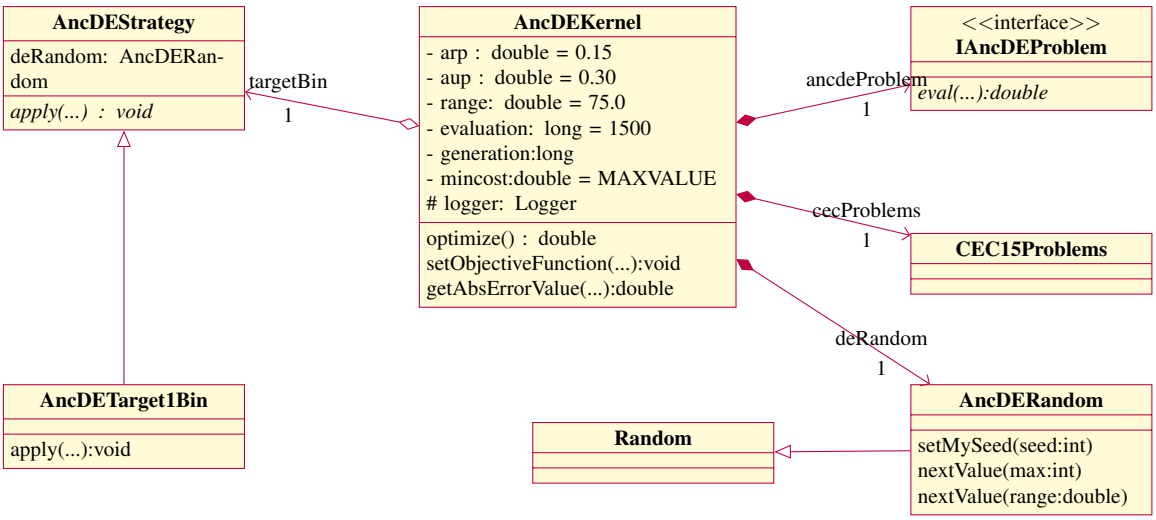


Figure 4.3: AncDE Class Diagram

Fiugre 4.3 shows the class diagram for AncDE. This design employs a structure that is similar to *builder pattern*[28]. With this structure a client can set its own objective function for optimisation without having to know anything about the internal composition of the other components. In order to use its own objective functions, a client needs to implement the `IAncDEProblem` interface and provide a definition of the `eval` method.

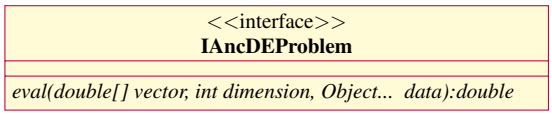


Figure 4.4: IAncDEProblem Interface

Figure 4.4 shows the method signature for `eval`. Its input parameter, *vector*, is a trial vector that should be used to evaluate a client objective function. Optional `data` is provided to meet unforeseen future requirements that client objective functions may have. Defining `IAncDEProblem` as an interface rather than an abstract class is a design decision adhering to *Liskov substitution principle*[29]. Client function need not be a behavioural subtype of AncDE problem. At most, what can be expected from the client function is that it should be able to perform the *role* of AncDE problem (by implementing the `eval` method).

AncDE also employs the *strategy pattern* to allow different embeddable and *interchangeable* strategies to be used with AncDE. The advantage of this design is that clients who wish to use their own evolution strategy while utilising the ancestral replacement and usage scheme provided by AncDE, can do so by extending the `AncDEStrategy` abstract class and overriding its `apply` method. Default strategy for AncDE is `AncDETarget1Bin`, and can be unset to a use newly defined strategy through the respective API.

The performance of AncDE has been evaluated on IEEE CEC 2015 optimisation benchmark. Hence, this benchmark has been integrated within AncDE application. This will also serve to compare AncDE purpose of evaluating new strategies proposed in the future that utilise ARP and AUP scheme of AncDE. Each instance of `AncDEKernel` can be configured

to run on CEC problems or *client specified* problems. For this purpose `AncDEKernel` has been built to provide multiple constructors each targeting a specific purpose.

Public API. Exposed public API allows a client to set its own objective function, the values of all the control parameters, the number of evaluations to be performed, and any known *value to reach*. Once configured for client problems, AncDE is designed to be used subsequently as many times as the client wishes for different objective functions and strategies. Configurations via setting and removing objective functions and strategies are also made available through public API. A small walk-through example on using public API is presented in the next section.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
  <date>2015-04-18T20:18:24</date>
  <millis>1429384704304</millis>
  <sequence>0</sequence>
  <logger>cs.nuim.ancde_logger</logger>
  <level>INFO</level>
  <class>cs.nuim.ancde.AncDEKernel</class>
  <method>&lt;init&gt;</method>
  <thread>1</thread>
  <message>IEEE CEC 2015, SS04 function: 7</message>
</record>
...
```

Figure 4.5: AncDE logging

Logging. AncDE maintains its own configuration trace for instantiation and configuration. Logging is employed using its own `java.util.Logger` instance with default `handler` specified to write to *ancde_log.log* file on disk. This file consists of *structured* (XML) trace at `java.util.logging.Level.INFO` level containing information related to current configuration of AncDE and `WARNING` to `SEVERE` level messages to indicate anything that has gone wrong while configuring AncDE prior to optimisation. A snapshot of AncDE loggin is shown in figure 4.5.

4.4 Documenting AncDE

Documenting AncDE has two aspects. First, is to document information on AncDE parameter tuning, and other details associated with AncDE strategy, ARP and AUP. And second is to document the software engineering aspect of AncDE. For parameter tuning and strategy details, we have developed a "user guide" to provide practitioners all the information required to use and tune AncDE. It also includes Design and implementation details necessary for integrating AncDE into the client environment. Listing 4.6 is a small example of integration walk-through that is also presented in the mentioned "user guide".

Source listing 4.6 shows a client that is using one instance of AncDE for optimising IEEE CEC 2015 function seven and another for its own `ChebychevT4` polynomial function. This listing also shows the use of important public API that client needs to interact with.

```
import cs.nuim.ancde.AncDEKernel;

/**
 * AncDE client
 * @author sawant
 *
 */
public class AncDEClient {

    public static void main(String[] args) throws Exception {

        // AncDEKernel for IEEE CEC 2015 problems
        new AncDEKernel(7, 30, 25, 0.6, 0.6).optimize();


        // give you flexibility of whatever type of file handler you want to add
        Handler handler = new FileHandler("ancde_log.log");
        SimpleFormatter formatter = new SimpleFormatter();
        handler.setFormatter(formatter);

        //AncDEKernel for client specified objective functions
        AncDEKernel optimizer = new AncDEKernel(55, 0.6, 06);
        optimizer.setAncDELogLevel(Level.INFO);
        optimizer.setAncDELogHandler(handler);
        optimizer.setARP(0.15);
        optimizer.setAUP(0.30);
        optimizer.setObjectiveFunction(new ChebychevT4(), 30);
        optimizer.setRange(100);
        optimizer.prepareToRun();

        //This check is important to see if we are ready to go or not
        if (optimizer.isPrepared()) {
            optimizer.optimize();
        }
    }
}
```

Figure 4.6: AncDE client in action

Source code documentation (Javadoc), provided along with the source code, documents all details about AncDE methods and constructors.

4.5 Testing AncDE

The main input to AncDE are its control parameters, objective function, its dimension and the maximum evaluation count. These are fed to AncDE either through constructors or through *setter* methods. To ensure robustness of public interface, it has been unit tested using *equivalence partitioning* technique. While *boundary value analysis* (BVA) would have been more appropriate, most input parameters for AncDE are in a *real* domain. Since boundary value analysis requires to identify next immediate number after a specified value range of a parameter, such a guess would have to be made in the case of real numbers when producing test cases. We prefer equivalence partitioning over BVA because any value can be selected within a specified range when using this technique. After this basic unit testing, we perform *state testing* to verify the correct behaviour of AncDE instance to client specified operations and also to ensure that instance state is not corrupted due to

wrong method invocations on the instance.

4.5.1 Input Parameter Analysis

Table 4.1 shows the *valid value ranges* for all input parameters for AncDE as mentioned in the available source code documentation for AncDE.

No.	**Parameter**	**Valid Range** (as per Javadoc specification)
1	`NP`	8...500
2	`F`	0.0...Double.MAX_VALUE
3	`CR`	0.0...1.0
4	`range`	0.0...Double.MAX_VALUE
5	`arp`	0.0...1.0
6	`aup`	0.0...1.0
7	`dimension`	1...150
8	`evaluation`	1...Double.MAX_VALUE
9	`objectiveFunction`	`IAncDEProblem` instance
10	`cecFunc_count`	1...15

Table 4.1: Valid value ranges for all input parameters

As metnioned earlier, clients are required to provide values within these ranges to AncDE through its programming interface. Hence, we employ equivalence partitioning technique such that for each *real* and discrete parameter there will be three test cases. First one providing a value from the *valid* input range, second providing a value from the *lower* invalid input range, and finally, one will provide a value from *upper* invalid input range. These partitions for parameter, *NP*, are shown in table 4.2.

Parameter	**Range**
`NP`	Integer.MIN_VALUE...7 8...500 501...Integer.MAX_VALUE

Table 4.2: Equivalence partitions for NP

4.5.2 JUnits

JUnit tests have been written for each input partition for individual parameters. To ensure that instantiation and instance configuration works as per specification, test cases have been written with different combinations of each parameter input partition values. The method `isPrepared` is used to test that the AncDE instance produces an expected outcome. That is, it should produce `false` for any invalid combination or configuration of input parameters.

This method returns `true` only if `AncDEKernel` instance is properly configured, either for the CEC problems or some client specified functions, and is ready to run optimisation. The optimiser has been implemented in such way that the optimisation will not proceed until all the configurations as checked by `isPrepared` has been satisfied. Sample test cases for parameter, NP, are as shown in table 4.3.

Case	Parameter	Range	Test
1*		Integer.MIN_VALUE...7	EP-2
2	NP	8...500	EP-1
3*		501...Integer.MAX_VALUE	EP-3

Table 4.3: Test cases for all public interfaces with NP as input parameter

In table 4.3, $*$ indicates error cases, and they are tested separately to avoid *error hiding*[30]. Sample test data for test cases generated for NP is shown in table 4.4.

ID	Test Cases Covered	Inputs	Expected Output
		NP	isPrepared
EP-1	2	55	true
EP-2	1*	-55	false
EP-3	3*	655	false

Table 4.4: Test data for test cases in table 4.3.

Test case and test data for all the other control parameters have been generated using this same method. For state testing we want to verify consistency of AncDE's *state* when invalid values are passed to it or invalid sequence of operations are invoked during its instance configuration. Again, `isPrepared` method, along with other getter methods, is used to verify the outcome of generated test cases. State testing is performed for all public setter methods that facilitate client specific objective function optimisation.

AncDE Correctness. `isPreperd` method just acts as a gateway checkpoint before the control is handed down to the `optimiser`. However, validity of `isPrepared` and the main method, `optimise`, that encompasses the AncDE algorithm, is not checked by the above tests cases. Also, as discussed in Chapter 2, results generated by stochastic algorithms are not cloneable. Hence, to test the correctness of AncDE algorithm we have to apply it to some optimisation problem whose global optimum is known and it *can* be reached by DE. Since results produced are *real* values, standard equality specified to allow error is `0.001`, advocated by IEEE CEC. With this approach we are sure that the global optimum of that specific problem can be reached.

We decided to implement *Matyas Function* [31] for testing AncDE's correctness. It is considered as one of the *artificial landscape*[1] created to test optimisation algorithms. To ensure AncDE's consistency, the test has been written to execute 50 times, while verifying that each time AncDE reaches to the global optimum or not. Therefore, if AncDE passes this test, it shows that AncDE implementation is correct.

[1] An artificial landscape term refers to the artificial optimisation problem created for testing purpose.

4.6 AncDE Application

Like other variants proposed to improve over DE, AncDE has also been evaluated over benchmark problems provided by IEEE Congress on Evolutionary Computing (CEC) – specifically, the "*Problem Definition and Evaluation Criteria for CEC 2015 Special Session and Competition on Bound Constrained Single-Objective Computationally Expensive Numerical Optimisation*"[11]. To facilitate our experiments and evaluation, the original DE application with graphical user interface (GUI) has been adopted and modified to implement AncDE. GUI for existing DE application provides an input panel for setting all required control parameter. We have restructured this control panel to include *arp* and *aup* values. It is a multi-threaded environment and has been updated to produce datasets as per IEEE CEC 2015 specification [11]. This update makes data produced from multiple runs available in a matrix format that is then used for subsequent data analyses. Required format for CEC is shown in table 4.5.

	0.01 x MaxFES	**0.02 x MaxFES**	**...**	**MaxFES**
Run 1				
Run 2				
...				
Run 20				

Table 4.5: Information matrix for IEEE CEC 2015 problems

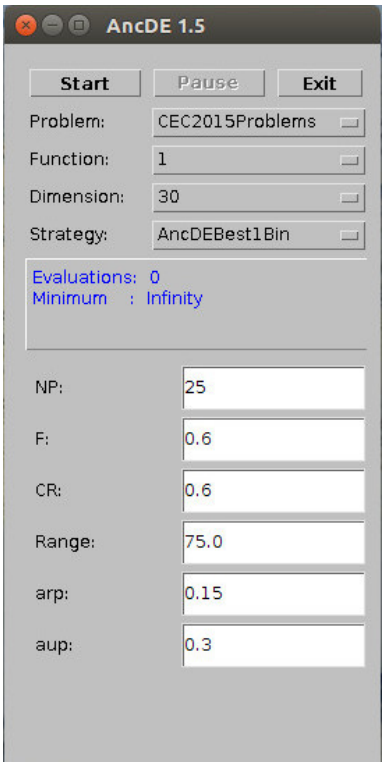


Figure 4.7: AncDE user interface

Figure 4.7 shows a snapshot of the AncDE application. Currently IEEE CEC 2015

benchmark suite is integrated with this application, and the 15 problems from this benchmark can be individually set as the objective problem through the GUI. We do not discuss updates and modifications to this application in further detail (such as, updated GUI or new dataset producing mechanism), even though they have been made for this thesis. This is due to the fact that they were developed as by-products to facilitate the main task: experimenting and evaluating AncDE. Also, original DE application (last updated in, 1999) is available on DE website[27], and this was the starting point for AncDE application.

4.7 Conclusion

This chapter covered software engineering aspect of the thesis. We discussed all software development related activities performed for this project. Of the important areas, we focused on the design, documentation and testing of AncDE. We also discussed essential contents of the user manual that will be made available to its users.

CHAPTER 5

Experimental Setup

This chapter presents an experimental setup used to evaluate AncDE. The benchmark test suite provided by IEEE CEC 2015 is represented here to discuss the method prescribed by CEC to generate and record results on the given benchmark. The chapter concludes with details on the execution environment used for performing numerical experiments in this thesis.

5.1 Objective Functions

All optimisation problems have at least one global optimum and may have multiple local optima.

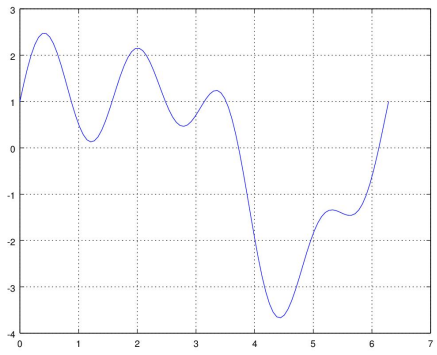


Figure 5.1: function with multiple local optima and one global optimum

Figure 5.1 shows a sample function with multiple local optima. Generally, $f(x^*)$, denotes the function value at the global optimum point[1]. These objective functions consist of

[1]Note that in our case global optimum refers to global minimum of the objective function – discussed in

various characteristic properties. Based on these properties, objective functions are divided into different categories. Of the many properties, *modality* is of prime importance to us; functions representing optimisation problems are categorised on this property.

Modality. A function is said to be unimodal if there is a path from each point x to its optimal solution point x^* along which the function is *monotonous*. In all other cases, function is *multimodal*. In general, from statistics point of view, *mode* refers to the value that tend to appear most of the times in a dataset. In case of multimodal, it is then said to be processing more than one modes. Figure 5.1 is an example of multimodal function, while figure 5.2 shows a unimodal function.

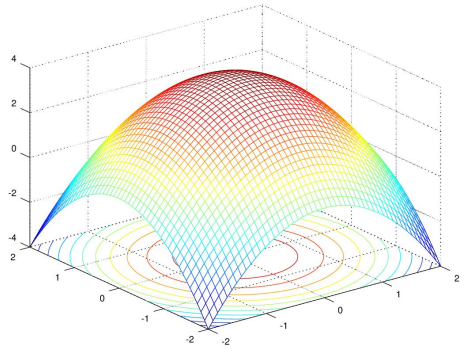


Figure 5.2: Simple continuous unimodal function

5.2 IEEE CEC 2015 Benchmark

Benchmark provided by IEEE CEC is considered as a prime source for evaluating variants of DE and other optimisation algorithms. It consists of expensive optimisation problems that try to simulate many real world problems. Generally, this benchmark tries to cover various kinds of optimisation problem properties that a real world problems tend to have. Hence, it consists of objective functions that are simple to optimise to various combinations of basic functions that make these functions increasingly difficult to optimise. At the time of development of AncDE, the 2015 version of the CEC benchmark[11] became available and AncDE has been evaluated on this latest benchmark.

Chapter 2

Categories	**No**	**Functions**	**Related basic functions**	F^*
Unimodal functions	1	Rotated Bent Cigar Function	Bent Cigar Function	100
	2	Rotated Discuss Function	Discuss Function	200
Simple Multimodal functions	3	Shifted and Rotated Weierstrass Function	Weierstrass Function	300
	4	Shifted and Rotated Schwefel's Function	Schwefel's Function	400
	5	Shifted and Rotated Katsuura Function	Katsuura Function	500
	6	Shifted and Rotated HappyCat Function	HappyCat Function	600
	7	Shifted and Rotated HGBat Function	HGBat Function	700
	8	Shifted and Rotated Expanded Griewank's plus Rosenbrock's Function	Griewank's Function Rosenbrock's Function	800
	9	Shifted and Rotated Expanded Scaffer's F6 Function	Expanded Scaffer's F6 Function	900
Hybrid functions	10	Hybrid Function 1 (N=3)	Schwefel's Function Rastrigin'sFunction High Conditioned Elliptic Function	1000
	11	Hybrid Function 2 (N=4)	Griewank's Function WeierstrassFunction Rosenbrock's Function Scaffer's F6 Function	1100
	12	Hybrid Function 3 (N=5)	KatsuuraFunction HappyCat Function Griewank's Function Rosenbrock's Function Schwefel's Function Ackley's Function	1200
Composition functions	13	Composition Function 1 (N=5)	Rosenbrock'sFunction High Conditioned Elliptic Function Bent Cigar Function Discus Function High Conditioned Elliptic Function	1300
	14	Composition Function 2 (N=3)	Schwefel'sFunction Rastrigin's Function High Conditioned Elliptic Function	1400
	15	Composition Function 3 (N=5)	HGBatFunction Rastrigin's Function Schwefel's Function Weierstrass Function High Conditioned Elliptic Function	1500

Table 5.1: IEEE CEC 2015 expensive optimisation test problems

Table 5.1 summarises all fifteen *continuous expensive optimisation problems* included in CEC 2015 benchmark. First eight problems are composed from fundamental optimisation problems by shifting their global optimum. Problems belonging to *hybrid* category simulate the real world situation where different subset of variables can have different properties. In this category, variables are randomly divided into various subsets and then basic functions that constitute to the hybrid function are used on individual subsets. On the other hand, *composition* functions merge the properties of all the sub-functions and maintain the continuity around the optima of a composed function. N, in case of hybrid and composition functions, denotes the number of basic functions used as sub-functions. As usual, F^* in the last column denotes the known global optimum for each function.

5.2.1 Experimental Setting

With the standard benchmark suite, IEEE CEC also specifies evaluation criteria and rules to produce datasets, under the *experimental settings* section of the benchmark specification. Benchmark problems are defined for 10D (10 dimensions) and 30D (30 dimensions) by CEC 2015 organisers. Hence, the algorithm is evaluated for both 10D and 30D individually. Result data is always to be collected from 20 independent runs performed for each problem. MaxFES (Maximum number of function evaluations) are restricted to 500 for problems with 10D runs and 1500 for functions with 30D runs.

This criteria essentially tests algorithm's *convergence ability*. If an algorithm has a faster convergence than DE, it should produce better results at the end. From an analytical perspective, how an algorithm progresses towards the convergence is also taken into account. That is, we want to know whether an algorithm gets caught in local optima or not. If it does, one may try to *tune* control parameters for that algorithm. With such trial-and-error experiments "overall good" values for all control parameters, for specific optimisation problem, are determined. Finding universal good values is a non-trivial task and has been subject of research on its own, as discussed in chapter 1 and 2.

For this thesis we provide two sets of values, one for 10D and one for 30D, that produce overall acceptable results for 15 optimisation problems in the benchmark. This indicates that if AncDE produces better results for a particular problem with these settings, then its performance can be further *improved* by tuning control parameters for that problem.

5.3 Recording Data

Current best function values. For analysis purpose, each algorithm's convergence rate is recorded after specific number of function evaluations, based on its dimensions. As specified by CEC, we record best function values obtained after 0.01 $\times$ **MaxFES**, 0.02 $\times$ **MaxFES**, ..., 0.1 $\times$ **MaxFES**, 0.2 $\times$ **MaxFES**, ..., **MaxFES** for each individual run (i.e. at 1%, 2%, 3%...of the total function evaluations). Finally all such data on 20 individual runs is collected together in a *local matrix*[2] for analysis. Collected dataset includes best, worst, mean, median, and standard deviation values for 20 runs.

Algorithm Complexity. Again, IEEE CEC advocates on how to judge the efficiency of algorithms for expensive optimisation problems. A pseudo evaluation function, shown in figure 5.3, is provided for benchmarking running time of an algorithm. One needs to run and record its running time within the same execution environment used for DE variant[3].

[2]By local matrix we mean the matrix generated for collecting data on 20 individual runs for a single optimisation problem.

[3]Note: although this competition was held in late May 2015, final results of the competition were not available at the time of submitting this thesis.

```
public class CecRtFunction {

        public static void main(String[] args) {

                compute();
        }

        private static void compute() {

                double x = 0;
                final long startTime = System.currentTimeMillis();
                for (int i = 0; i < 1000000; i++) {
                        x = 0.55 + ((double) i);
                        x = x + x;
                        x = x / 2;
                        x = x * x;
                        x = Math.sqrt(x);
                        x = Math.log(x);
                        x = Math.exp(x);
                        x = x / (x + 2);

                }
                final long endTime = System.currentTimeMillis();
                System.out.println("Total execution time: " + (endTime - startTime));
        }

}
```

Figure 5.3: CEC baseline for comparing runtime efficiency of algorithms

The complexity of the competing algorithm for each function is then measured by: $T1/T0$. Where, time taken by function in 5.3 is $T0$, and time taken by the algorithm for an individual problem is $T1$. We shall present all results in the next chapter.

5.4 Execution Environment

As mentioned in earlier chapters, evolutionary algorithms are non-deterministic, so their results not cloneable. Hence, the execution environment used for evaluation has to discussed for completeness.

All numerical experiments for this thesis were performed on a computer with Intel® Core™ i7-3520M CPU @ 2.90GHz × 4 and 16 GB RAM, under Ubuntu 14.04 LTS, 64-bit OS. The implementation of AncDE was primarily done in Java, using Java development kit (JDK) version 1.7. Note that the results presented in this work were generated using Java 1.7 and its native pseudo random number generator (PRNG). Because of changes to the PRNG algorithm in Java 1.8, the results generated using AncDE on newer versions of Java will be different.

5.5 Conclusion

This chapter presented the test-bed used for evaluating the algorithm. We enumerated all optimisation problems provided by CEC 2015 and discussed some important properties of these problems. We also discussed method prescribed by CEC to carry out experiments in

order to evaluate algorithms. We concluded this chapter with the description of execution environment used to carry out these experiments. In the next chapter we present results and the evaluation of AncDE.

CHAPTER 6

Algorithm Evaluation

This chapter presents evaluation of AncDE primarily through comparative analysis of results obtained from AncDE, DE, and ArpDE. Method prescribed by IEEE CEC is primarily used to evaluate these algorithms. We also use statical techniques to provide better a understanding of these results. The chapter concludes with sensitivity analysis of the algorithm to variant in ARP and AUP control parameters, and discuss benefits and limitations of AncDE.

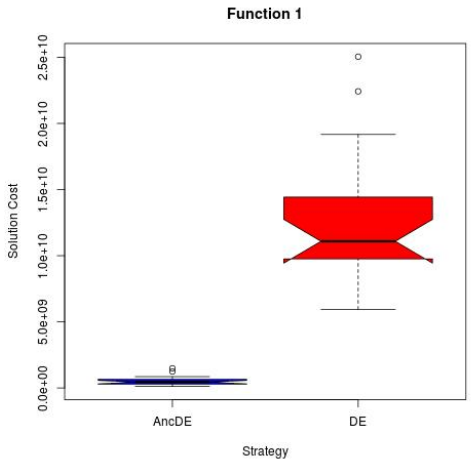
6.1 Evaluation

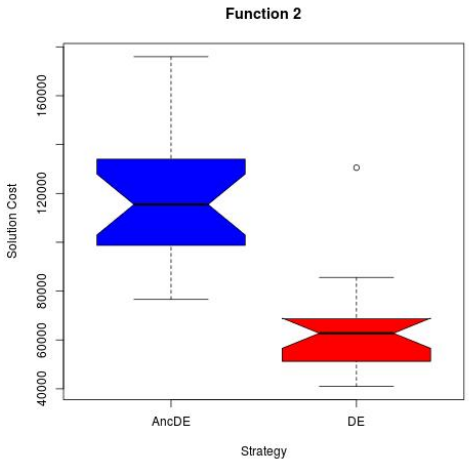
Evaluation is major part of this thesis as new algorithms should show that they produce better results than DE. Note that our goal for this thesis is not to win over other techniques discovered, but to improve on DE. We want to see fundamental improvement to the original algorithm, with as minimal changes as possible. Implication is that other techniques can utilise this change in DE to improve themselves further.

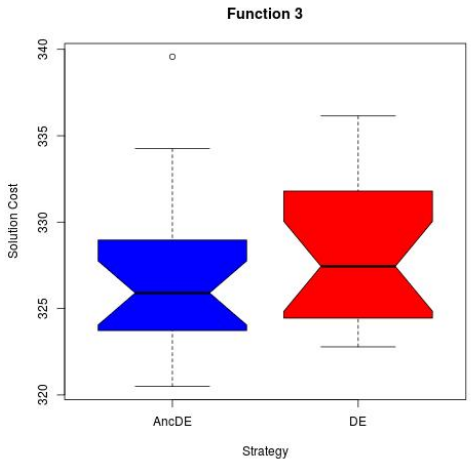
6.1.1 Comparative Analysis: AncDE vs DE

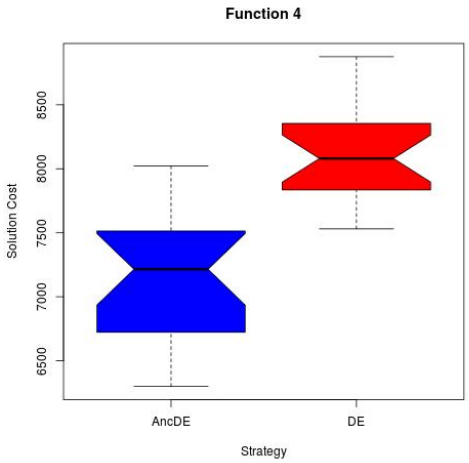
Adopting the method prescribed by IEEE CEC, results were collected for individual problems in the benchmark suite. We use *boxplots* to present comparative analysis between AncDE and DE. Boxplots, traditionally used in *descriptive statistics*, depicts groups of data through their quartile. Therefore, they are more suggestive and precise for our comparative analysis.

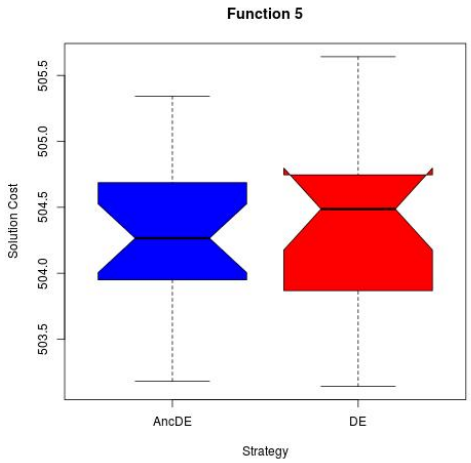
The box indicates the quartile range from 25% to 75% with the horizontal line in between indicating the median result. The "notch" indicates the confidence interval around the median – so if two boxes' notches do not overlap this often indicates the "strong evidence" that their medians differ. The upper and lower whiskers adding/subtracting 1.5 times the interquartile range and finally, possible outliers are indicated by unfilled circles.

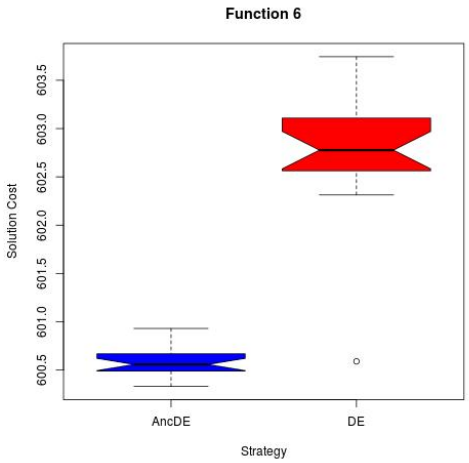

Function 1
Solution Cost
0.0e+00
5.0e+09
1.0e+10
1.5e+10
2.0e+10
2.5e+10
AncDE
DE
Strategy

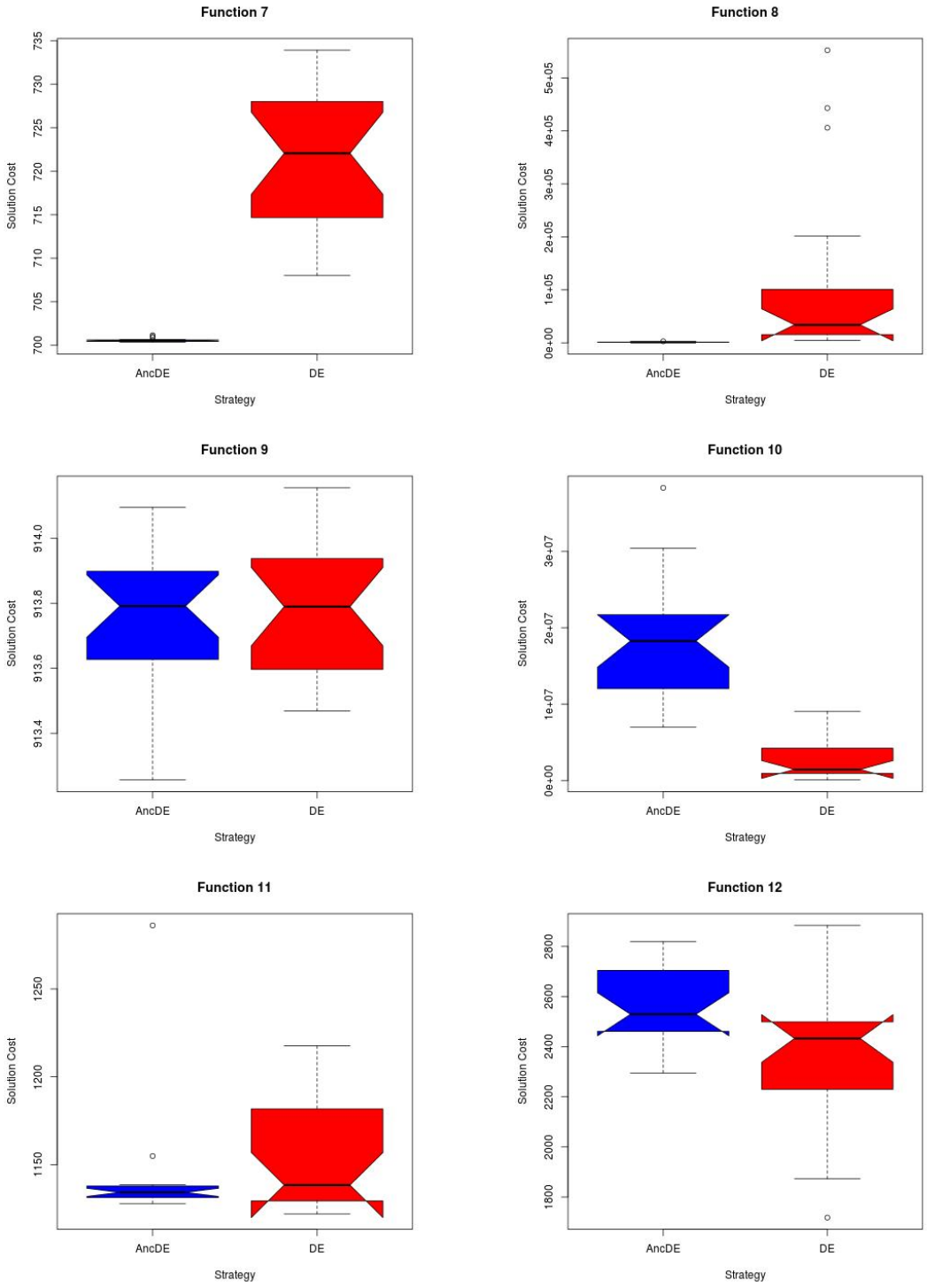

Function 2
Solution Cost
40000
60000
80000
120000
160000
AncDE
DE
Strategy


Function 3
Solution Cost
320
325
330
335
340
AncDE
DE
Strategy


Function 4
Solution Cost
6500
7000
7500
8000
8500
AncDE
DE
Strategy


Function 5
Solution Cost
503.5
504.0
504.5
505.0
505.5
AncDE
DE
Strategy


Function 6
Solution Cost
600.5
601.0
601.5
602.0
602.5
603.0
603.5
AncDE
DE
Strategy


Function 7
Solution Cost
700
705
710
715
720
725
730
735
AncDE
DE
Strategy
Function 8
Solution Cost
0e+00
1e+05
2e+05
3e+05
4e+05
5e+05
AncDE
DE
Strategy
Function 9
Solution Cost
913.4
913.6
913.8
914.0
AncDE
DE
Strategy
Function 10
Solution Cost
0e+00
1e+07
2e+07
3e+07
AncDE
DE
Strategy
Function 11
Solution Cost
1150
1200
1250
AncDE
DE
Strategy
Function 12
Solution Cost
1800
2000
2200
2400
2600
2800
AncDE
DE
Strategy

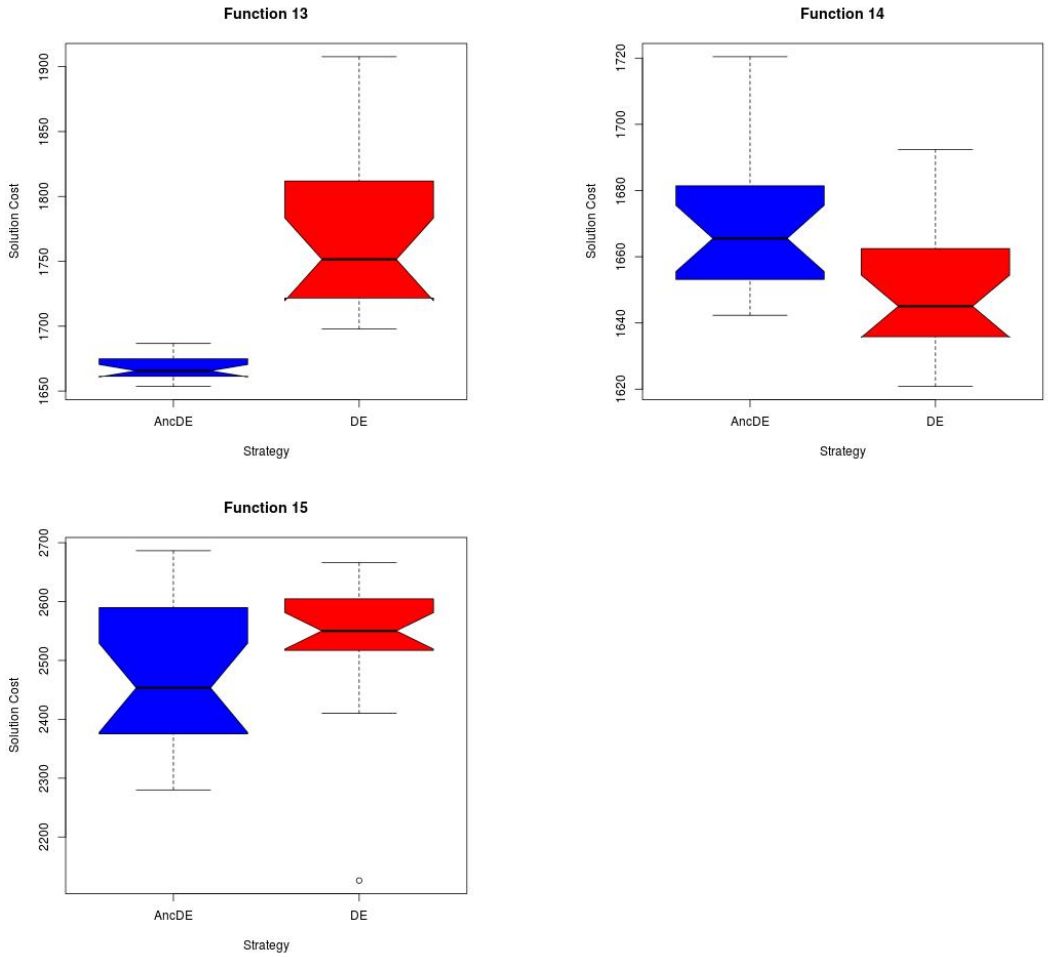


Figure 6.1: Box plots for AncDE and DE for 15 CEC benchmark problems

Figure 6.1 shows that AncDE is far better than DE for some of the problems, such as problem 1, 6, 7, 8, and 13. Similarly, AncDE is reasonably better than DE for other problems, such as problem 3, 4, 15. In the remaining problems (5,9,11,12, and 14) AncDE wins or looses only by a minuscule difference. Finally, problem 2 and 10 are the only instances where AncDE performs far worse than DE. These results strongly suggest that AncDE preforms better than DE. Note that, if we consider problems for which difference is minuscule as a "tie", then DE wins over AncDE only in two problems. It also suggests that when AncDE is not performing better than DE, it is as good as DE. Therefore these results also indicate that the controlled introduction of ancestors do not impede the convergence process.

The idea of ancestors helping in convergence seems counter intuitive at first, as ancestors produce values which are not better than values produced by current vectors. However, form the search space point of view, occasionally generated inter-generational difference vectors can probably help AncDE to *escape* local optima and *stagnation*.

Local optima can possibly be avoided by making big jumps (at the right time) towards global optimum and avoiding to be directed towards local optima. On the other hand, as

explained by Lampinen *et al.* [9], stagnation occurs when vectors in the current generation and starts producing identical offspring vectors during optimisation process. When such a situation occurs, no progress is made towards the global optimum even though the vector population is diverse and is not caught up in local optima. AncDE can escape stagnation situation with high probability by using ancestral vectors which are *guaranteed* to be different than the current generation by the virtue of the algorithm itself.

6.1.2 Comparative Analysis: Population Size

It is important to take into consideration the population size with which algorithms produce better results. After all, the more vector population is required, the more function evaluations are performed per generation. Algorithms requiring larger population size can face scalability problem when optimisation problems are costly. Therefore, it is apt to evaluate algorithm's capability on small population size (NP).

Wilcoxon Test.We compared the performance of the two algorithms using a *Wilcoxon Matched Pairs Signed Rank* test [32] on the mean result produced for each of the 15 problems, for these small population tests using NP=8. The small population AncDE significantly outperformed the small population DE algorithm (V=117; p<0.005; two tailed). This proves that under these conditions adding the ancestral cache significantly improves the quality of results produced on these challenging and diverse problems.

6.1.3 Comparative Analysis: CEC Method

IEEE CEC 2015 specified its own method to evaluate algorithm performance. It requires a *global matrix* representing all 15 benchmark problems. From the produced local matrix, as explained chapter 5, *best, worst* and *mean* cost of all twenty runs, along with median, and standard deviation, is recorded in the a global matrix. For example, the best cost recorded in the global matrix gives the best result achieved by the algorithm from all of the twenty best values recorded in local matrix at **MaxFES** (i.e. after 1500 or 500 evaluations based on 30D or 10D). There are total two global matrix generated this way; one is for 10D and on for 30D. As per the CEC method then mean values and median values obtained by each algorithm on all 15 problems for 10D and 30D is summed up as the final score as shown in equation 6.1.

$$\text{Total score} := \sum_{i=1}^{15} mean(f^*)\,|_{D=10} + \sum_{i=1}^{15} mean(f^*)\,|_{D=30} + \sum_{i=1}^{15} median(f^*)\,|_{D=10} + \sum_{i=1}^{15} median(f^*)\,|_{D=30} \tag{6.1}$$

CEC use this method to compare algorithms as shown in table 6.1. It shows results achieved for AncDE and DE using CEC method. With this method as well AncDE turns out to be better than DE, again due to the fact that for some optimisation problems, when AncDE is better in results with DE, its result is far more better than DE. Similarly, when DE won over AncDE, it did so by very small difference.

For 10D optimisation problems AncDE produces better results for 13 problems out of 15. That is, AncDE outperforms DE **86.87%** of the times for 10D problems. Similarly, for 30D optimisation problems, AncDE shows overall improvement by producing better results for 11 problems out of 15. That is, in case of 30D problems, AncDE is better than DE **73.34%** of the times.

Similar experiments were carried out on IEEE CEC 2013 benchmark[33] because they are defined for 50D and 100D as well. Comparative analysis on these larger problems showed that AncDE was better than DE 73.34% of the times. We do not present experiments for CEC 2013 benchmark here due to space restrictions, but the results are added in the appendix.

Functions	AncDE Mean	AncDE Median	DE Median	DE Mean
30D	5.21E+008	4.45E+008	1.11E+010	1.24E+010
	1.18E+005	1.15E+005	6.25E+004	6.37E+004
	2.67E+001	2.59E+001	2.74E+001	2.80E+001
	6.78E+003	6.81E+003	7.68E+003	7.77E+003
	4.24E+000	4.27E+000	4.27E+000	4.30E+000
	5.89E-001	5.58E-001	2.78E+000	2.76E+000
	5.77E-001	5.43E-001	2.21E+001	2.15E+001
	4.48E+002	2.43E+002	3.32E+004	1.07E+005
	1.38E+001	1.38E+001	1.38E+001	1.38E+001
	1.81E+007	1.83E+007	1.44E+006	2.45E+006
	4.24E+001	3.42E+001	3.84E+001	5.37E+001
	1.37E+003	1.33E+003	1.23E+003	1.14E+003
	3.68E+002	3.66E+002	4.51E+002	4.69E+002
	2.71E+002	2.66E+002	2.45E+002	2.51E+002
	9.71E+002	9.53E+002	1.05E+003	1.04E+003
10D	1.78E+007	8.84E+006	3.29E+008	4.33E+008
	3.53E+004	3.78E+004	1.93E+004	2.82E+004
	5.73E+000	5.84E+000	7.77E+000	7.64E+000
	1.63E+003	1.66E+003	1.95E+003	1.99E+003
	2.60E+000	2.66E+000	2.58E+000	2.75E+000
	5.50E-001	5.55E-001	1.02E+000	9.93E-001
	6.35E-001	5.34E-001	1.81E+000	2.27E+000
	6.26E+000	6.32E+000	1.23E+001	5.25E+001
	3.97E+000	3.93E+000	4.08E+000	4.05E+000
	2.62E+005	1.98E+005	8.02E+004	1.94E+005
	6.65E+000	6.39E+000	6.78E+000	7.76E+000
	2.24E+002	2.18E+002	2.80E+002	2.79E+002
	3.25E+002	3.24E+002	3.29E+002	3.32E+002
	2.04E+002	2.03E+002	2.06E+002	2.07E+002
	3.59E+002	4.06E+002	4.09E+002	3.68E+002
$\sum_{i=1}^{15}$	5.57E+008	4.73E+008	1.14E+010	1.28E+010

Table 6.1: Comparison with CEC method. First two columns show mean and median values obtained using AncDE on each of the 15 problems (10D first 15 and 30D remaining). Similarly last two columns are for DE. Score indicates application of formula stated in equation 6.1.

These results clearly shows that an ancestral extension to standard DE does have positive influence on the evolution process. For completeness, to match exactly with CEC evaluation method (as stated in 6.1), results obtained at the end of the table 6.1 for simply needs to be added (first two columns and last two columns) and compared; this does not change the result we see in the table however.

6.1.4 Algorithm Efficiency

As described in chapter 5, data to analyse computational complexity in the table 6.2 is generated by executing algorithm on each function and then dividing its runtime by the runtime of baseline function provided by CEC (presented in chapter 5).

Function	AncDE-T1/T0	DE-T2/T0
1	2.0763888889	2.0555555556
2	1.9652777778	1.9305555556
3	1.9652777778	1.9583333333
4	2.125	2.125
5	3.3263888889	3.3194444444
6	2.0347222222	2.0138888889
7	1.9513888889	1.9513888889
8	2.1597222222	2.1527777778
9	2.0208333333	1.9861111111
10	2.2638888889	2.2638888889
11	3.5	3.5
12	2.7777777778	2.7777777778
13	3.7777777778	3.75
14	3.4097222222	3.3888888889
15	9.9791666667	9.9583333333

Table 6.2: Computational complexity of both AncDE and DE. T0 denotes CEC baseline function, T1 denotes AncDE algorithm, and T2 denotes DE algorithm

Table 6.2 shows that computational complexity of AncDE is almost equivalent to the standard DE algorithm. The only overhead AncDE has to bear in computation is that of maintaining one extra cache and switching between strategies. As AncDE does add any extra computations in the process, such as local search or learning mechanisms, it stays lightweight as original DE, moreover, produces relatively better results.

Up to this point, our claim about convergence improvement due to occasional introduction of ancestors is valid. We stated that one of the reasons of this improvement is the faster exploration of search space by generating high magnitude difference vectors. Similar mechanism is utilised by ArpDE[1]. Therefore, next we compare ArpDE and AncDE, and observe that AncDE produces better results than ArpDE.

6.1.5 Comparative Analysis: ArpDE and AncDE

Using the same CEC method we can compare results achieved by AncDE and ArpDE. Similar global matrices as mentioned in previous section are generated for ArpDE as well. We present such comparison on 30D problems only.

As discussed in chapter 2 and 3, there are two prime differences between AncDE and ArpDE – 1. AncDE algorithm makes sure that ancestors are *at least* one generation old. No such guarantees can be made in case of ArpDE. Hence, AncDE is relatively better than ArpDE when it comes to stagnation characteristic (discussed later in the next subsection).

[1]Discussed in Related Works section of Chapter 2

2. AncDE has a completely different mutation strategy. It utilises target vector itself to produce mutant vector. Mutation scheme also switches between DE's *Best/1/bin* and *Target/1/bin*. On the other hand, ArpDE always utilises *Best/1/bin* strategy with one vector from ancestral generation.

Function	AncDE Mean	AncDE Median	ArpDE Mean	ArpDE Median
1	5.21E+008	4.45E+008	1.65E+010	1.55E+010
2	1.18E+005	1.15E+005	9.76E+004	9.34E+004
3	3.27E+002	3.26E+002	3.28E+002	3.28E+002
4	7.18E+003	7.21E+003	8.06E+003	8.17E+003
5	5.04E+002	5.04E+002	5.04E+002	5.05E+002
6	6.01E+002	6.01E+002	6.03E+002	6.03E+002
7	7.01E+002	7.01E+002	7.32E+002	7.30E+002
8	1.25E+003	1.04E+003	1.61E+005	1.07E+005
9	9.14E+002	9.14E+002	9.14E+002	9.14E+002
10	1.81E+007	1.83E+007	2.73E+006	2.28E+006
11	1.14E+003	1.13E+003	1.16E+003	1.15E+003
12	2.57E+003	2.53E+003	2.05E+003	1.98E+003
13	1.67E+003	1.67E+003	1.79E+003	1.80E+003
14	1.67E+003	1.67E+003	1.66E+003	1.66E+003
15	2.47E+003	2.45E+003	2.60E+003	2.60E+003
$\sum_{i=1}^{15}$	5.39E+008	4.64E+008	1.65E+010	1.55E+010

Table 6.3: Comparison with CEC method: AncDE vs ArpDE. Problems for which AncDE is better is marked with green.

Table 6.3 shows that overall performance of AncDE is better than ArpDE. Problems where AncDE performs better than ArpDE are marked in green color in the table. AncDE outperforms ArpDE 11 times out of the 15, that is **73.34%** of the times. This result suggests that AncDE is a notable improvement over ArpDE. This completes our formal evaluation AncDE for this thesis.

6.2 Sensitivity Analysis: ARP and AUP

AncDE's performance depends heavily on ARP (ancestor replacement probability) and AUP (ancestor usage probability). Right values for these two control parameters play important role in convergence rate of the algorithm. We have to evaluate algorithm's performance against various combinations of these two values to find a "good pair". For this thesis we have limited our exploration range to, ARP values 0.05, 0.15 and 0.3, and AUP values 0.1, 0.3, and 0.5. The matrix formed from combinations of ARP and AUP for each of the 15 optimisation problems is then used to analyse the sensitivity of the algorithm to these pairs.

For rigours evaluation of each ARP-AUP value combination, best cost produced for each cell in the matrix is collected from average of 20 individual runs of the algorithm with each combination.

We have to *freeze* other control parameter values while evaluating ARP and AUP. From our experiments, similar to the ones carried out for finding AUP and ARP, good values

discovered for other parameters are: N=25, F=0.6, CR=0.6 and Range=75.

We do not present experiments performed for these control parameters as they are not newly introduced for AncDE, and extensive literature is available on finding acceptable values for them[2].

In case of ARP-AUP pair, to provide a clearer idea (using less space) on what can be relatively good values, one matrix is presented for each set of problems having *similar* properties, rather than matrices for all problems.

		ARP		
		0.05	0.15	0.3
AUP	0.1	6.21E+008	7.30E+008	7.50E+008
	0.3	6.56E+008	6.26E+008	6.59E+008
	0.5	1.82E+009	1.67E+009	2.35E+009

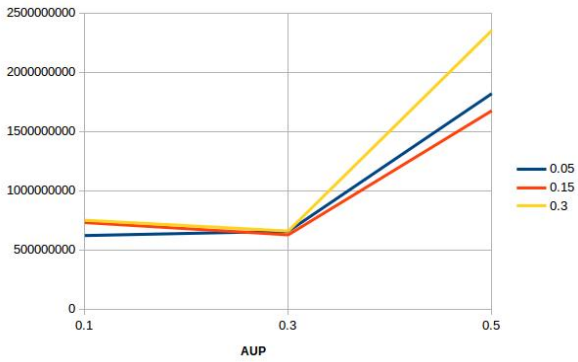


Figure 6.2: Sensitivity analysis for unimodal functions. Table shows best costs produced with each pair of ARP and AUP.

		ARP		
		0.05	0.15	0.3
AUP	0.1	1.60E+003	1.77E+003	2.02E+003
	0.3	1.46E+003	1.18E+003	1.35E+003
	0.5	1.73E+003	1.76E+003	2.12E+003

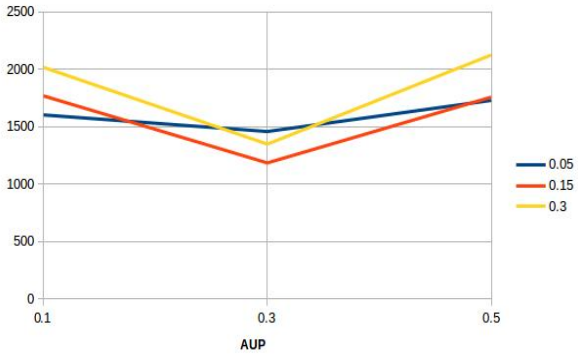


Figure 6.3: Sensitivity analysis for simple multimodal functions. Table shows best costs produced with each pair of ARP and AUP.

[2]Please refer to Chapter 2 for more details on this.

		ARP		
		0.05	0.15	0.3
AUP	0.1	9.14E+002	9.14E+002	9.14E+002
	0.3	9.14E+002	9.14E+002	9.14E+002
	0.5	9.14E+002	9.14E+002	9.14E+002

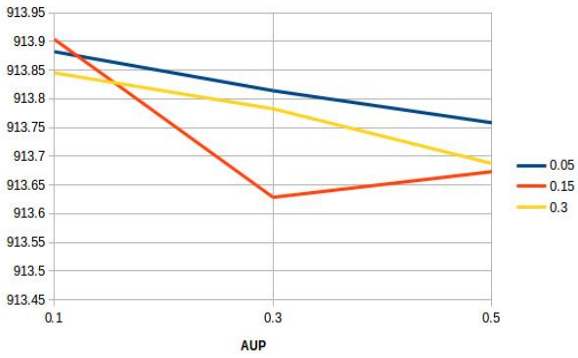


Figure 6.4: Sensitivity analysis for hybrid functions. Table shows best costs produced with each pair of ARP and AUP.

		ARP		
		0.05	0.15	0.3
AUP	0.1	2.64E+003	2.59E+003	2.58E+003
	0.3	2.47E+003	2.47E+003	2.52E+003
	0.5	2.39E+003	2.39E+003	2.43E+003

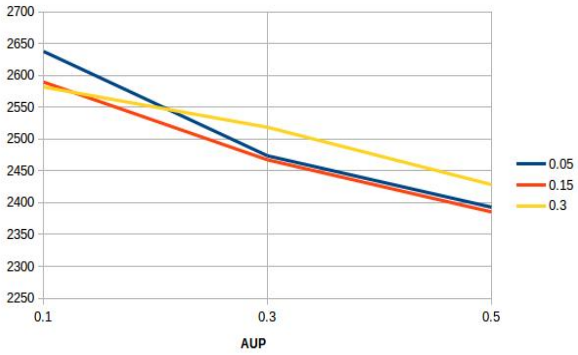


Figure 6.5: Sensitivity analysis for composition functions. Table shows best costs produced with each pair of ARP and AUP.

Figures 6.2, 6.3, 6.4, and 6.5 show that overall good values are: ARP=0.15 and AUP=0.3. That is, when ancestor replacement rate is 15% and usage rate is 30%, AncDE tends to have relatively better convergence rate. As mentioned earlier, we have crossover fixed to 0.6 and NP to 25.

An effect of higher crossover rate is that there is a high probability that AncDE's inter-generational mutation scheme will initiate. Also, with small population size, there are more evolution loops, and more generations are tried out with ancestors. As a result, we can see from convergence plots that AncDE progresses either better than DE or as good as DE, with *less* frequency to stagnate or get caught in local optima. However, we can say that this analysis is still limited by the matrix size we have selected. Due to time limitations, it is the part of the future work to analyse more exhaustive ranges for all control parameters.

6.3 Discussion

After presenting the evaluation of AncDE, it is apt to put together advantages and disadvantages of using AncDE. Question arises when one can use AncDE instead of using DE.

And how one can decide that the results obtained with AncDE are better than he/she could have achieved with DE. We discuss these questions in this section.
Advantages of AncDE:

- AncDE is applicable in all the instances where DE is.
- There is no performance hit in using AncDE instead of DE.
- AncDE requires relatively smaller vector population size to achieve competitive or better results than DE
- AncDE can stagnate relatively less frequently than DE
- Exploration capability of AncDE is relatively better than DE's

Limitations of AncDE. By introducing changes to the original algorithm we also create some drawbacks. In cases where these drawbacks cannot be ignored, using DE is the right choice.

- AncDE needs to store extra cache, along with current population.
- AncDE introduces to more new control parameters; AUP and ARP.
- Naively setting new control parameter values can result in AncDE producing worst results than DE.
- Similar to original DE, AncDE can only be applied to problems with continuous parameters. Extending AncDE to be applicable to discrete optimisation problems is considered as a probable future work.
- AncDE is not all the time better than DE. Results show that it is 73% of the times better, that is 11 out of 15. In earlier section we have also observed that DE performs far better than AncDE on problem 10 of CEC 2015 benchmark.

So when one can use AncDE? Since these are randomised algorithms, this question is of prime importance. With our experiments it has been observed that AncDE produces acceptable results with population of 25. This is true even for the higher dimension problems from IEEE CEC 2013[33]. Hence, appropriate approach to decide whether to use AncDE or not would be to:

1. perform at least 5 independent runs of AncDE with NP=25
2. similarly perform same number of independent runs of DE with its best know configuration for the given type of problem.
3. From the gathered data if it is evident that AncDE is producing competitive or better results, then use AncDE.

This is because one can always improve results produced by the algorithm by further tuning control parameters for that specific problem.

6.4 Conclusion

This chapter presented evaluation of AncDE algorithm using benchmark problems provided by IEEE CEC 2015. We showed AncDE performs better than DE with various comparative analyses and CEC method. We then presented convergence analysis to discuss how AncDE has less tendency to stagnate or get caught in local optima. We also presented ARP and AUP sensitivity analysis. This analysis showed that ARP=0.15 and AUP=0.3 along with small population and high crossover increases AncDE's performance. Finally, we concluded with summarising advantages and disadvantages of AncDE.

CHAPTER 7

Future Work

This chapter presents discussion on probable future work considering introduction of more than one ancestral vectors in optimisation process. We also discuss how currently limited analysis due to time constraints can be extended to cover larger data sets.

7.1 Multiple Ancestors

In this thesis we have shown that better results can be achieved with controlled introduction of ancestral vectors. It would be interesting to investigate whether these results can be further improved by introducing more than one ancestral vectors during mutation. Some experiments were carried out in this direction, but due to limited time further exploration was considered as part of the future work.

This involves two ancestral caches and finding the right strategy that can accommodate more than one ancestral vectors along with the same number of normal vectors. Same number of normal vectors constraint is essential to generate guaranteed inter-generational difference vectors. Of course, the more number of ancestral vectors used, the more number of difference vectors should be produced. The strategy then should utilise all such difference vectors, along with the base vector, to produce final mutant vector.

This task is non-trivial, since such a strategy can be discovered only with exhaustive experiments, like the one presented in this thesis. After finding the right strategy, its sensitivity to ARP and AUP needs to be analysed to find the best value combination for the new algorithm, again similar to the one presented in this thesis.

7.2 Extending Current Analysis

Due to time constraints, we have constrained our analysis in this thesis to limited ranges on control parameter values. Therefore, this analysis needs to be extended beyond this point.

On the other hand, benchmark provided by CEC is not defined for more than 30 dimensions, hence our analysis was also limited to 30 dimensions. Given enough time, same benchmark can be implemented to have more than 30 dimensional problems. Results obtained on higher dimension can provide more insights on how AncDE's ancestral mutation strategy performs.

Similarly, current analysis is restricted to limited combinations of control parameter values, including ARP and AUP. This constitutes one more direction to explore. Similar to DE, AncDE is also sensitive to optimisation problem properties and values of its control parameters. Experiments with extended ranges of parameter values can provide more information on how parameters can be tuned for specific class of optimisation problems.

7.3 AncDE for Discrete Optimisation

Recently DE has been extended to work on discrete optimisation problems[34]. These are the problems that have only integer parameter vectors as their input. Since AncDE is a direct extension of standard DE, it can also be extended to operate on discrete optimisation problems. It would be interesting to investigate how AncDE performs on such problems relative to discrete version of DE.

7.4 Conclusion

This chapter presented probable future work that can be carried out basing on the work presented in this thesis. We discussed to specific directions for the future work. First is to introduce multiple ancestral vectors in the optimisation process and measure their influence on convergence. Second one is to extend the analysis presented in this thesis to include large value ranges on control parameters and also to experiment with higher dimensional problems.

APPENDIX A

CEC 2013 Results

Comparison results for higher dimension problems from CEC 2013.

Function	AncDE Mean	AncDE Median	DE Mean	DE Median
1	6.14E+003	5.19E+003	4.71E+004	4.61E+004
2	4.95E+008	4.71E+008	3.71E+008	3.70E+008
3	1.51E+011	1.60E+011	1.56E+012	3.05E+011
4	1.85E+005	1.92E+005	1.32E+005	1.31E+005
5	4.79E+002	4.35E+002	8.74E+003	8.00E+003
6	-2.58E+002	-3.72E+002	2.47E+003	2.37E+003
7	-5.46E+002	-5.51E+002	4.86E+001	-3.24E+002
8	-6.79E+002	-6.79E+002	-6.79E+002	-6.79E+002
9	-5.21E+002	-5.20E+002	-5.36E+002	-5.38E+002
10	1.23E+003	1.28E+003	5.31E+003	5.25E+003
11	1.69E+002	1.67E+002	4.12E+002	3.91E+002
12	3.40E+002	3.43E+002	5.84E+002	5.41E+002
13	4.60E+002	4.43E+002	8.14E+002	7.72E+002
14	1.47E+004	1.46E+004	1.41E+004	1.42E+004
15	1.60E+004	1.61E+004	1.61E+004	1.61E+004
$\sum_{i=1}^{15}$	1.52E+011	1.60E+011	1.56E+012	3.05E+011

Table A.1: AncDE vs DE with CEC 2013 Benchmark Problems: 50D

Function	AncDE Mean	AncDE Median	DE Mean	DE Median
1	1.72E+004	1.19E+004	1.56E+005	1.54E+005
2	3.06E+008	2.82E+008	3.35E+009	3.21E+009
3	2.57E+012	6.14E+011	4.93E+019	3.91E+018
4	3.70E+005	3.68E+005	3.15E+005	3.11E+005
5	2.87E+003	2.79E+003	3.73E+004	3.47E+004
6	1.37E+003	1.29E+003	2.95E+004	2.89E+004
7	1.20E+002	-9.04E+001	8.80E+005	4.47E+005
8	-6.79E+002	-6.79E+002	-6.79E+002	-6.79E+002
9	-4.31E+002	-4.31E+002	-4.54E+002	-4.54E+002
10	2.24E+003	2.19E+003	2.16E+004	2.15E+004
11	5.24E+002	5.60E+002	2.29E+003	2.25E+003
12	1.17E+003	1.16E+003	2.40E+003	2.42E+003
13	1.27E+003	1.28E+003	2.56E+003	2.55E+003
14	3.09E+004	3.09E+004	2.57E+004	2.58E+004
15	3.30E+004	3.31E+004	3.25E+004	3.29E+004
SUM	2.57E+012	6.14E+011	4.93E+019	3.91E+018

Table A.2: Comparison: AncDE vs DE on CEC 2013 100D problems. Notice that AncDE is far better than DE.

References

[1] M. Lesnik, B. Boskovic, and J. Brest. Performance tuning of java ee application servers with multi-objective differential evolution. In *Differential Evolution (SDE), 2013 IEEE Symposium on*, pages 69–76, April 2013. doi: 10.1109/SDE.2013.6601444.

[2] Zyed Bouzarkouna, Didier Yu Ding, and Anne Auger. Well placement optimization under uncertainty with CMA-ES using the neighborhood. *CoRR*, abs/1209.0616, 2012. URL http://arxiv.org/abs/1209.0616.

[3] Jason D. Lohn, Derek S. Linden, Gregory Hornby, William F. Kraus, and Adaan Rodriguez-Arroyo. Evolutionary design of an x-band antenna for nasa's space technology 5 mission. In *Evolvable Hardware*, pages 155–163. IEEE Computer Society, 2003. ISBN 0-7695-1977-6.

[4] Garrison W Greenwood. Finding solutions to np problems: Philosophical difference between quantum and evolutionary search algorithms. *arXiv preprint quant-ph/0010021*, 2000.

[5] Anyong Qing. *Differential Evolution: Fundamentals and Applications in Electrical Engineering*. Wiley-IEEE Press, 2009. ISBN 0470823925, 9780470823927.

[6] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11(4):341–359, December 1997. ISSN 0925-5001. doi: 10.1023/A:1008202821328. URL http://dx.doi.org/10.1023/A:1008202821328.

[7] A.K. Qin, V.L. Huang, and P.N. Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *Evolutionary Computation, IEEE Transactions on*, 13(2):398–417, April 2009. ISSN 1089-778X. doi: 10.1109/TEVC.2008.927706.

[8] Roger Gämperle, Sibylle D. Müller, and Petros Koumoutsakos. A parameter study for differential evolution. In *WSEAS Int. Conf. on Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, pages 293–298. Press, 2002.

[9] Jouni Lampinen and Ivan Zelinka. ON STAGNATION OF THE DIFFERENTIAL EVOLUTION ALGORITHM. 2000.

[10] Magnus Erik Hvass Pedersen. Good parameters for differential evolution.

[11] B. Liu Q. Chen and P. N. Suganthan B. Y. Qu Q. Zhang, J. J. Liang. Problem definition and evaluation criteria for cec 2015 special session and competition on bound constrained single-objective computationally expensive numerical optimization, 2015. URL http://www.cec2015.org/. Accessed: 2015-5-20.

[12] O'Donoghue D.P Sawant R, Hatton D. An ancestor based extension to differential evolution (ancde) for single-objective computationally expensive numerical optimization. *The annual IEEE Congress on Evolutionary Computation (IEEE CEC)*, 2015 (In Press).

[13] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, 2006. ISBN 9780387400655.

[14] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, UK, 1996. ISBN 0-19-509971-0.

[15] Kenneth Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. ISBN 3540209506.

[16] S. Das and P.N. Suganthan. Differential evolution: A survey of the state-of-the-art. *Evolutionary Computation, IEEE Transactions on*, 15(1):4–31, Feb 2011. ISSN 1089-778X. doi: 10.1109/TEVC.2010.2059031.

[17] R. Mallipeddi, P.N. Suganthan, Q.K. Pan, and M.F. Tasgetiren. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied Soft Computing*, 11(2):1679 – 1696, 2011. ISSN 1568-4946. doi: http://dx.doi.org/10.1016/j.asoc.2010.04.024. URL http://www.sciencedirect.com/science/article/pii/S1568494610001043. The Impact of Soft Computing for the Progress of Artificial Intelligence.

[18] Yong Wang, Zixing Cai, and Qingfu Zhang. Differential evolution with composite trial vector generation strategies and control parameters. *Evolutionary Computation, IEEE Transactions on*, 15(1):55–66, Feb 2011. ISSN 1089-778X. doi: 10.1109/TEVC.2010.2087271.

[19] Haixiang Guo, Yanan Li, Jinling Li, Han Sun, Deyun Wang, and Xiaohong Chen. Differential evolution improved with self-adaptive control parameters based on simulated annealing. *Swarm and Evolutionary Computation*, 19(0):52 – 67, 2014.

ISSN 2210-6502. doi: http://dx.doi.org/10.1016/j.swevo.2014.07.001. URL http://www.sciencedirect.com/science/article/pii/S2210650214000522.

[20] Janez Brest and Mirjam Sepesy Maučec. Population size reduction for the differential evolution algorithm. *Applied Intelligence*, 29(3):228–247, 2008. ISSN 0924-669X. doi: 10.1007/s10489-007-0091-x. URL http://dx.doi.org/10.1007/s10489-007-0091-x.

[21] Daniela Zaharie. Control of population diversity and adaptation in differential evolution algorithms. In *Proc. of MENDEL*, volume 9, pages 41–46, 2003.

[22] S. Das, A. Abraham, U.K. Chakraborty, and A. Konar. Differential evolution using a neighborhood-based mutation operator. *Evolutionary Computation, IEEE Transactions on*, 13(3):526–553, June 2009. ISSN 1089-778X. doi: 10.1109/TEVC.2008.2009457.

[23] Jingqiao Zhang and A.C. Sanderson. Jade: Adaptive differential evolution with optional external archive. *Evolutionary Computation, IEEE Transactions on*, 13(5): 945–958, Oct 2009. ISSN 1089-778X. doi: 10.1109/TEVC.2009.2014613.

[24] Amy FitzGerald, DiarmuidP. O'Donoghue, and Xinyu Liu. Genetic repair strategies inspired by arabidopsis thaliana. In Lorcan Coyle and Jill Freyne, editors, *Artificial Intelligence and Cognitive Science*, volume 6206 of *Lecture Notes in Computer Science*, pages 61–71. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-17079-9. doi: 10.1007/978-3-642-17080-5_9. URL http://dx.doi.org/10.1007/978-3-642-17080-5_9.

[25] Wei-jie Yu and Jun Zhang. Adaptive differential evolution with optimization state estimation. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, GECCO '12, pages 1285–1292, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1177-9. doi: 10.1145/2330163.2330341. URL http://doi.acm.org/10.1145/2330163.2330341.

[26] Efrñn Mezura-Montes, Jesús Velázquez-Reyes, and Carlos A. Coello Coello. A comparative study of differential evolution variants for global optimization. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, pages 485–492, New York, NY, USA, 2006. ACM. ISBN 1-59593-186-4. doi: 10.1145/1143997.1144086. URL http://doi.acm.org/10.1145/1143997.1144086.

[27] Kenneth Price and Rainer Storn. Differential evolution (de) for continuous function optimization (an algorithm by kenneth price and rainer storn), 1999. URL http://www1.icsi.berkeley.edu/~storn/code.html. Accessed: 2014-12-20.

[28] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. ISBN 0-201-63361-2.

[29] Barbara Liskov. Keynote address - data abstraction and hierarchy. In *Addendum to the Proceedings on Object-oriented Programming Systems, Languages and Applications (Addendum)*, OOPSLA '87, pages 17–34, New York, NY, USA, 1987. ACM. ISBN 0-89791-266-7. doi: 10.1145/62138.62141. URL http://doi.acm.org/10.1145/62138.62141.

[30] Tom Lysaght Stephen Brown, Joe Timoney. *Software Testing: Principles and Practice*. China Machine Press, 2012.

[31] Vashek (Vaclav) Matyas. Matyas function, 2013. URL http://www.sfu.ca/~ssurjano/matya.html. Accessed: 2015-5-20.

[32] F. Wilcoxon. *Individual Comparisons by Ranking Methods*. Bobbs-Merrill Reprint Series in the Social Sciences, S541. Bobbs-Merrill, College Division. URL https://books.google.ie/books?id=BSdFHQAACAAJ.

[33] P. N. Suganthan Alfredo G. Hernández-Díaz J. J. Liang, B. Y. Qu. Problem definitions and evaluation criteria for the cec 2013 special session on real-parameter optimization, 2013. URL http://www.cec2013.org/. Accessed: 2015-5-20.

[34] Jingqiao Zhang, V. Avasarala, A.C. Sanderson, and T. Mullen. Differential evolution for discrete optimization: An experimental study on combinatorial auction problems. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 2794–2800, June 2008. doi: 10.1109/CEC.2008.4631173.