Design And Implementation Of Hybrid Evolutionary Algorithms: A case study with Differential Evolution variants

A thesis

Submitted in partial fulfillment of the requirements for the Degree of

Bachelor of Science in Computer Science and Engineering

Submitted by

Sadiq Mahbub Faisal	**140204087**
Ashikul Islam	**150104017**
Ahsanul Bari Romi	**150104019**
Kazi Hafsha Haque	**150104053**
Afrina Zahan Mithila	**150104115**

Supervised by

Dr. Mohammad Shafiul Alam

Associate Professor

Department of Computer Science and Engineering

Ahsanullah University of Science and Technology

Department of Computer Science and Engineering

Ahsanullah University of Science and Technology

Dhaka, Bangladesh

June 2019

CANDIDATES' DECLARATION

We, hereby, declare that the thesis presented in this report is the outcome of the investigation performed by us under the supervision of Dr. Mohammad Shafiul Alam, Associate Professor, Department of Computer Science and Engineering, Ahsanullah University of Science and Technology, Dhaka, Bangladesh. The work was spread over two final year courses, CSE4100: Project and Thesis-I and CSE4250: Project and Thesis-II, in accordance with the course curriculum of the Department for the Bachelor of Science in Computer Science and Engineering program.

It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Sadiq Mahbub Faisal
140204087

Ashikul Islam
150104017

Ahsanul Bari Romi
150104019

Kazi Hafsha Haque
150104053

Afrina Zahan Mithila
150104115

CERTIFICATION

This thesis titled, **"Design And Implementation Of Hybrid Evolutionary Algorithms: A case study with Differential Evolution variants"**, submitted by the group as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree B.Sc. in Computer Science and Engineering in June 2019.

Group Members:

Sadiq Mahbub Faisal	**140204087**
Ashikul Islam	**150104017**
Ahsanul Bari Romi	**150104019**
Kazi Hafsha Haque	**150104053**
Afrina Zahan Mithila	**150104115**

Dr. Mohammad Shafiul Alam
Associate Professor & Supervisor
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology

Prof. Dr. Kazi A. Kalpoma
Professor & Head
Department of Computer Science and Engineering
Ahsanullah University of Science and Technology

ACKNOWLEDGEMENT

We would like to express my deep gratitude to my supervisor, Dr. Mohammad Shafiul Alam, Associate Professor, Department of Computer Science and Engineering, Ahsanullah University Of Science and Technology, for his continuous support, advice and care. His patience, guidance, continuous encouragement, constant supervision, constructive criticism, reading many inferior drafts and correcting them at all stages have made it possible to complete this thesis. Without his continuous guidance and support, we would not be able to achieve what we have done in this thesis.

We also want to express special thanks to our head of the department Dr. Kazi A. Kalpoma Professor, Department of Computer Science and Technology, Ahsanullah University of Science and Technology.

We also owe special thanks to Mir Imtiaz Mostafiz Naved, Lecturer, Department of Computer Science and Engineering, Ahsanullah University Of Science and Technology, for his helpful advice, valuable suggestions and great questions and comments. His suggestions helped us not only to improve the quality of our thesis, but also to equip us with new knowledge and experience to do our research better along the course of this thesis.

Next we want to thank all the faculty members and staffs of Computer Science and Engineering department, Ahsanullah University of Science and Technology for their various support and cooperation.

At the very last we want to express our deepest gratitude to our beloved parents and family members for their support, sacrifice and unconditional love which helped us to complete this work.

Dhaka
June 2019

Sadiq Mahbub Faisal

Ashikul Islam

Ahsanul Bari Romi

Kazi Hafsha Haque

Afrina Zahan Mithila

ABSTRACT

Our thesis work is based on implementation and hybridization of evolutionary algorithms for bench mark function optimization. Optimization is the process of selecting the best element by following some rules and criteria from some set of available alternatives. Function optimization means finding the best available value of some given objective function in a defined domain. In this work we have taken a novel approach by hybridizing different variants of evolutionary algorithms and by varying the ratio of exploitation and exploration. There exists different types of evolutionary and Swarm Intelligence algorithms. In our thesis work we have implemented different variants of Differential Evolutionary(DE) algorithms on different families of benchmark problems, i.e; Unimodal and Multimodal, Low and High Dimensional functions. We have implemented different exploitative and explorative variants of DE, compared them side by side and made discussion on their results. From the comparative study, it is found that different variants work better on different families of benchmark functions. Also the experimental results change with the variation of generations. Finally we have hybridized a few variants of Differential Evolutionary Algorithms. A number of different combinations of migration strategies and migration size in island model are employed during hybridization. The experimental results of our proposed hybrid algorithms indicate that hybridization of exploitative and explorative evolutionary algorithms may often improve the optimization performance over the original component algorithms.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many recent problems in science, engineering, economics are expressed as computing global optimal solutions. Using classical non-linear techniques may fail to solve such problems because these problems usually contain multiple local optima. In recent years, there has been a great deal of interest in emerging some artificial intelligence tools in the area of optimization. These tools are normally called Metaheuristic, which is mainly proposed by simulating nature or by invoking intelligent learned procedures. Global optimization problems usually contain the following properties:

- Calculation of the objective function (or constraint functions if exist) is very expensive or time consuming
- The exact gradient of the objective function cannot be computed or its numerical approximation is time consuming or expensive
- The values of the objective function (or constraint function if exist) contain noises

Such problems exist in many real-world applications and achieving the exact global solution is neither possible nor desirable. Therefore, using derivative-free global search methods is highly needed in order to achieve acceptable solutions. Actually, metaheuristics fight courageously when applied to these problems and they could obtain highly accurate solutions in many cases. The power of metaheuristics comes from the fact that they are robust and can deal successfully with a wide range of problem areas. However, these methods, especially when they are applied to complex problems, suffer from the slow convergence that brings about the high computational cost. The main reason for this slow convergence is that these methods explore the global search space by creating random movements without using local information about promising search direction. In contrast, local search methods have faster convergence due to their using local information to determine the most promising search direction by creating logical movements. However, local search methods can easily

be entrapped in local minima. Actually, metaheuristics fight courageously when applied to these problems and they could obtain highly accurate solutions in many cases. The power of metaheuristics comes from the fact that they are robust and can deal successfully with a wide range of problem areas. One approach that recently has drawn much attention is to combine metaheuristics with local search methods to design more methods that are efficient. These hybrid methods are not easily entrapped in local minima because they still maintain the merits of the metaheuristics. Moreover, the field of global optimization has been very active, producing different kinds of deterministic and stochastic algorithms for optimization in the continuous domain. Among the stochastic approaches, evolutionary algorithms (EAs) offer a number of exclusive advantages: robust and reliable performance, global search capability, little or no information requirement, etc. These characteristics of EAs, as well as other supplementary benefits such as ease of implementation, parallelism, no requirement for a differentiable or continuous objective function etc. make it an attractive choice. Mainly our focus will be on how to solve continuous optimization problems using different Metaheuristics approaches. We will step by step explain the algorithm family, algorithms we have used and overall discussion on the work done and future ideas.

1.1 Optimization

Optimization is essentially everywhere from engineering design to economics and from holiday planning to Internet routing. As money, resources and time are always limited, the optimal utility of these available resources is crucially important. It is an act, process or methodology of making something as fully perfect or effective as possible. Almost everything can be improved, so optimizations' relevance spans to almost every business or process to make it operate more efficiently and effectively. In mathematics, computer science and operations research, mathematical optimization is the selection of a best element from some set of available alternatives.

Optimization consists in trying variations on an initial concept and using the information gained to improve on the idea. A computer is the perfect tool for optimization as long as the idea or variable influencing the idea can be input in electronic format. Optimization is a math tool. It is a mathematical discipline that concerns the finding of minima and maxima of functions, subject to so-called constraints.

1.1.1 Continuous Optimization Problem

Many real-world problems can be formulated as optimization problems of the parameters that assume values from the continuous domain i.e. the continuous optimization problems. Continuous optimization has been grounded in the well-developed mathematical theory of convex analysis and geometry. It has been inspired by numerous applications in operations research, statistics, control theory and most recently, machine learning. The substantial overlap with scientific computing has led continuous optimization to become a basic tool in many areas of science that adopt continuous models to describe and understand natural phenomena.
In computer science, an optimization problem is the problem of finding the best solution from all feasible solutions. Continuous optimization approach uses the continuous range of values of variables. The main concern is continuous optimization problem by using meta-heuristics approaches. This means maximizing or minimizing a real function or finding 'best available' values of some objective function given a defined domain. It also includes a variety of different types of objective functions and different types of domain. That assume values from the continuous domain. A continuous optimization problem can be formalized as follows.

$$\begin{aligned} &\underset{x}{\text{minimize } f(x)} \\ &\text{subject to: } x \in S \end{aligned}$$

Here, the goal is to find a vector $x_{min} \in S$ such that $f(x_{min})f(x)$ for all $x \in S$, where the search space $S \subseteq R^n$ is a bounded subset of R^n and the objective function, $f : S \rightarrow R$ is an n-dimensional real valued function that is to be optimized over its' parameter x. Each element x of the vector x is a real-valued variable : $x = [x_1, x_2, ..., x_n]^T \in R^n$.
The task of continuous optimization is generally referred with many different names, such as real parameter optimization, function optimization and numeric optimization. However, all of them actually refer to the general task of finding a solution across a real valued (usually) multi-dimensional search space such that the solution gives the best value such as minimum or maximum value of an objective function depending on whether it is a minimization or maximization task. This solution should have not only the best objective function value around its local neighborhood, but also the best objective value overall the feasible solutions across the entire search space.

1.1.2 Discrete optimization problems

Discrete optimization means searching for an optimal solution in a finite or countably infinite set of potential solutions. Optimality is defined with respect to some criterion function, which is to be minimized or maximized. Examples:

- minimization: cost, distance, length of a traversal, weight, processing time, material, energy consumption, number of objects
- maximization: profit, value, output, return, yield, utility, efficiency, capacity, number of objects.

A discrete optimization problem can be expressed as a tuple (S, f). The set S is a finite or countably infinite set of all solutions that satisfy specified constraints. The function f is the cost function that maps each element in set S onto the set of real numbers R. The objective of a DOP is to find a feasible solution x_{opt} such that $f(x_{opt}) \leq (x)$ for all $x \in S$. The study of discrete optimization has been inter twined with that of theoretical computer science: the foundations of computational complexity and algorithm design including NP-completeness, approximation algorithms and in approximability, all blossomed around the study of discrete optimization problems.

1.2 Local and Global Optima

In applied mathematics and computer science, a local optimum of an optimization problem is a solution that is optimal (whether maximal or minimal) within a neighboring set of candidate solutions. This is in contrast to a global optimum, which is the optimal solution among all possible solutions, not just those in a particular neighborhood of values.

1.3 Optimization Algorithm

Most real-world optimizations are highly nonlinear and multimodal, under various complex constraints. Different objectives are often conflicting. Even for a single objective, sometimes, optimal solutions may not exist at all. In general, finding an optimal solution or even sub-optimal solutions is not an easy task. To solve the optimization problem, efficient search or optimization algorithms are needed. There are many optimization algorithms which can be classified in many ways, depending on the focus and characteristics. Mainly optimization algorithms can be classified into two categories.

- Deterministic Algorithms and
- Probabilistic Algorithms.

In computer science, a deterministic algorithm is an algorithm which has given a particular input, will always produce the same output, with the underlying machine always passing through the same sequence of states. Deterministic algorithms are by far the most studied and familiar kind of algorithm, as well as one of the most practical, since they could be run on real machines efficiently.
If an algorithm works in a mechanical deterministic manner without any random nature, it is called deterministic. For such an algorithm, it will reach the same final solution if we start with the same initial point.

- State Space Search Algorithms
- Branch and Bound Algorithms
- Algebraic Geometry Algorithms

In computer science, a non deterministic algorithm is an algorithm that even for the same input, can exhibit different behaviors on different runs, as opposed to a deterministic algorithm. There are several ways an algorithm may behave differently from run to run. A concurrent algorithm can perform differently on different runs due to a race condition. A probabilistic algorithms' behaviors depend on a random number generator. An algorithm that solves a problem in non deterministic polynomial time can run in polynomial time or exponential time depending on the choices it makes during execution. The non deterministic algorithms are often used to find an approximation to a solution, when the exact solution would be too costly to obtain using a deterministic one. If there is some randomness in the algorithm, the algorithm will usually reach a different point every time the algorithm is executed, even though the same initial point is used. In this category, algorithms are mainly bio inspired or natural optimization algorithm. The following algorithms are in this category

- (Stochastic) Hill Climbing
- Evolutionary Computation
- Random Optimization
- Simulated Annealing
- Tabu Search
- Parallel Tempering
- Stochastic Tunneling

1.4 Heuristic and Single State Methods

Heuristic is an approach to problem solving, learning or discovery that employs a practical method not guaranteed to be optimal or perfect but sufficient for the immediate goals. Where finding an optimal solution is impossible or impractical, heuristic methods can be used to speed up the process of finding a satisfactory solution. It is a technique designed for solving a problem more quickly when classical methods are too slow.

- Employs Heuristics and Randomness During Search
- Examples: Simulated Annealing, Tabu Search, Iterated Local Search, Hill Climbing with Several Variants
- Limited Strength Against Local Optima

Heuristic is better choice when

- We have incomplete information
- We have complete information but the amount of time to find the optimal solution is too high

Example: Alpha-Beta pruning in Chess. Alpha-Beta pruning is an adversarial search algorithm commonly used for machine playing of two player games (Chess, Tic-tac-toe, go etc). It stops completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. By applying alpha beta pruning we search for an immediate goal but it cannot be guaranteed that it is the optimal solution. So, we can say that it is a heuristic search.

1.5 Population Based and Meta-Heuristic Methods

Metaheuristic is a higher-level procedure to find, generate or select a heuristic. It means heuristics about heuristics. It is not problem specific. Metaheuristic uses stochastic optimization (uses some degree of randomness) to solve many problems. It explores the search space for near-optimal solution. It does not guarantee a globally optimal solution. Candidate solutions are used to find a nearly optimal solution. Metaheuristic algorithms are used to find answers to problems when we have little amount of information such as we do not know how the optimal solution of the problem looks like, we do not know how to go about finding things in a principled way, we have very little heuristic information to go on,

and brute-force search is out of the questions because the space is too large. Examples of metaheuristic algorithm:

- Evolutionary Computation Family: Genetic Algorithm, Evolution Strategy, Evolutionary Programming.
- Swarm Intelligence Family: Ant Colony Optimization, Particle Swarm Optimization, Artificial Bee Colony Algorithms.

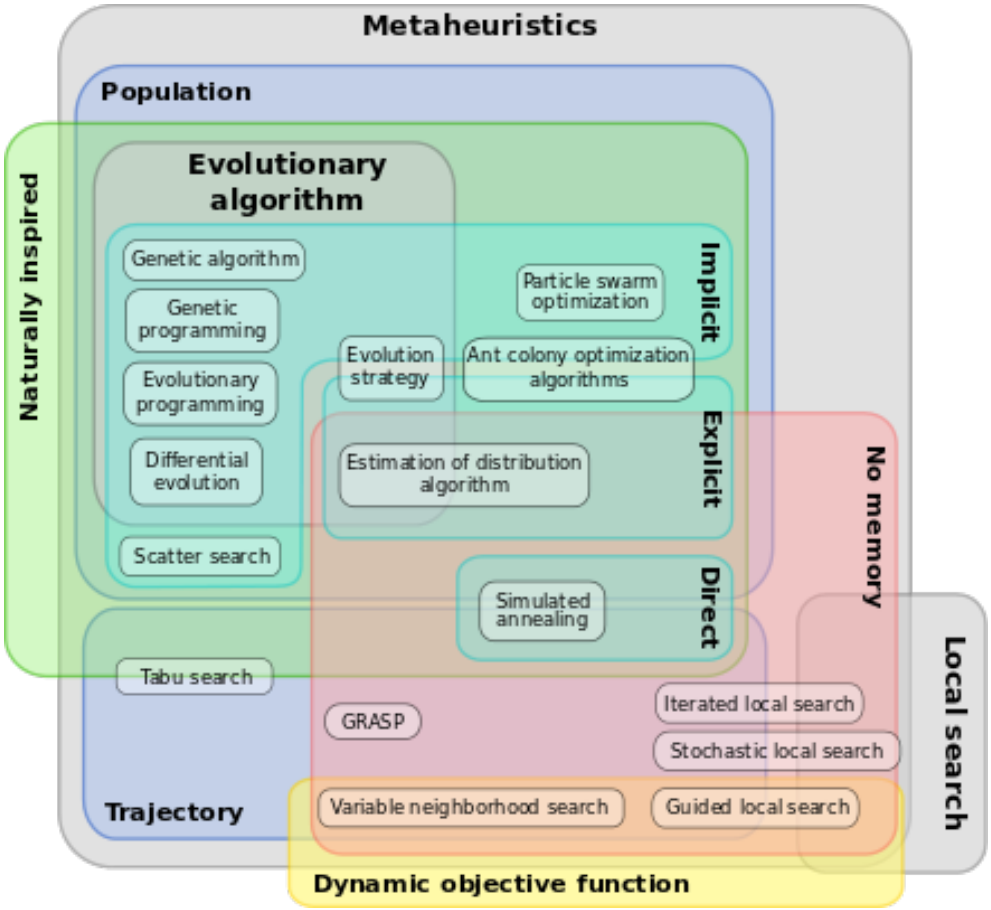


Figure 1.1: Classification of Metaheuristics

1.6 Hybrid Metaheuristic Algorithms

The term ***hybrid*** refers to 'something heterogeneous in composition or origin' or 'something that has two different types of components performing essentially the same function'. While the current entry in Wiktionary defines this term as,

- Offspring resulting from cross-breeding different entities e.g. different species
- Something of mixed origin or composition.

The motivation behind such hybridization of different algorithmic concepts is usually to obtain better performing systems that exploit and unite advantages of the individual pure strategies i.e. such hybrids are believed to benefit from synergy. In fact, today it seems that choosing an adequate hybrid approach is determinant for achieving top performance in solving most difficult problems.

1.7 Objective and Motivations

Evolutionary and swarm intelligence algorithms both maintain a population of candidate solutions that is expected to be improved gradually through successive perturbation and selection operations until they reach sufficiently close to the globally optimum point. But practical experiences, often show that the evolving population loses its diversity and explorative capacity too soon and the candidate solutions may prematurely get trapped around the locally optimal points of the fitness landscape. This general problem of premature convergence often originates from the lack of balance between global explorations and local exploitations during the optimization process. The prime motivation of this thesis is to concentrate on designing new and improved evolutionary algorithms, specially Differential evolution for continuous optimization that will address the conflicting goals of global explorations and local exploitations, which is often considered a key factor for the good performance of the algorithm, in terms of both convergence speed and the quality of the final solution. Exploration refers to the capability of the algorithm to examine new, non-visited search regions more extensively, which is necessary for robustness of the algorithm against the locally optimal points. In contrast, exploitation refers to concentrating the search on local neighborhoods of the already found solutions, which often leads to increased convergence speed, but possibly towards some locally optimal point.Therefore,balancing explorations with exploitations may lead to both increased convergence speed with improved solution quality, which is the prime concern of most EAs and SIAs. Therefore, the main objective of our study is to introduce novel evolutionary algorithms for the continuous optimization problem that try to balance between global explorations and local exploitations

and maintain sufficient population diversity in order to avoid the problems of premature convergence and fitness stagnations. The issues of explorations vs. exploitations and population diversity vs convergence are often closely related and addressing these issues is often considered essential for both improved final solution quality and higher convergence speed of an algorithm. Along the course of this thesis, we plan to develop a number of improved evolutionary and algorithms that would address these closely related issues.

1.8 Aims of this Thesis

Along the course of this thesis, we plan to develop a number of hybrid algorithms by using evolutionary and swarm intelligence algorithms that would be able to cover up each others' weakness and give better solutions of the problems. More specifically, the aim of this thesis is to accomplish the following objectives and to achieve the following possible outcomes.

- Study of the current state of art of Evolutionary Algorithms (EA) and try to understand how those algorithms maintain a population of candidate solutions that improves gradually through successive perturbation and selection operations which takes them very close to the globally optimum point. We will also try to seek the reason for which the evolving population loses its diversity and explorative capacity too and the candidate solutions may prematurely get trapped around the locally optimal points of the fitness landscape.

- Closely observe Evolutionary Programming (EP) and Classical Evolutionary Programming (CEP), Fast Evolutionary Programming (FEP) and Improved Fast Evolutionary Programming (IFEP). We will try to find out the performances of CEP, FEP and IFEP on some benchmark functions and compare those performances. We will also discuss about the problems of CEP, FEP and IFEP for which they struggle to reach the globally optimum solution.

- Take a close look at Differential Evolution (DE) and know how DE optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality and what kind of obstacles DE faces to reach the global optimum point.

- Analyze the variants of DE and learn how they work. We will also implement them and try to find out their limitations.

- We will discuss about hybridization in a brief manner and also try to hybridize among different variants of Differential Evolution Algorithms in order to find out whether we can get better results from the hybrid algorithms or not. We will also try to find out, if

the hybrid algorithms can solve the general problem of premature convergence. Examine whether the hybrid algorithms are capable of creating balance between global explorations and local exploitations during the optimization process.

- Lastly, if we can get promising results then we will try to find out the reasons behind the success in our future works and also make some guidelines for further research and future researchers in this field.

To achieve our goal, we will implement some proposal using C++ and try to obtain encouraging results.

1.9 Organization of the Chapters

This thesis has been organized in different chapters, with each chapter discussing different aspects of the study. The areas covered by different chapters are briefly as follows:

Chapter 2 briefly introduces the fundamentals of metaheuristics algorithms and their classification, the fundamentals of evolutionary algorithms (EAs), with their many different variants, models, operators and processes. The essential details such as flowchart, algorithms etc. which are related to EAs are explained with figures and examples. Chapter 2 also discusses about how EAs are employed to solve the continuous optimization problems. In this chapter we will discuss about evolutionary programming (EP) and Swarm Intelligence Algorithms in a very brief manner. This chapter also expresses the steps of EP and shows the flowchart and the algorithm. We will also discuss different EP and have comparative analysis different types of SIA.

Chapter 3 explains benchmark functions those are use in our algorithms.

Chapter 4 describes differential evolution (DE) elaborately. We will also talk about process and different strategies of DE. In chapter 6 we will also implement DE with different strategies and have comparative analysis.

Chapter 5 introduces a short description on hybrid metaheuristics, their classification and advantages and proposes some hybridized algorithms that we have implemented in our thesis and implementation of them.

Chapter 6 This chapter includes the conclusions by presenting a summary of the algorithms implemented so far and suggestion for future work.

Chapter 2

Metaheuristic Algorithms

A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms (Sorensen and Glover, 2013). Notable examples of metaheuristics include genetic/evolutionary algorithms, tabu search, simulated annealing, variable neighborhood search, (adaptive) large neighborhood search and ant colony optimization, although many more exist. A problem-specific implementation of a heuristic optimization algorithm according to the guidelines expressed in a metaheuristic framework is also referred to as a metaheuristic. The term was coined by Glover (1986) and combines the Greek prefix *meta* (meta, beyond in the sense of high-level) with heuristic (from the Greek *heuriskein* or *euriskein*, to search).

2.1 Overview of Metaheuristic Algoithms

Metaheuristic algorithms i.e. optimization methods designed according to the strategies laid out in a metaheuristic framework, are as the name suggests always heuristic in nature. This fact distinguishes them from exact methods that do come with a proof that the optimal solution will be found in a finite (although often prohibitively large) amount of time. Metaheuristics are therefore developed specifically to find a solution that is 'good enough' in a computing time that is 'small enough'. As a result, they are not subject to combinational explosion, the phenomenon where the computing time required to find the optimal solution of NP-hard problems increases as an exponential function of the problem size. Metaheuristic have been demonstrated by the scientific community to be a viable, and often superior, alternative to more traditional (exact) methods of mixed-integer optimization such as branch and bound and dynamic programming. Especially for complicated problems or large problem instances, metaheuristics are often able to offer a better trade-off between solution quality and computing time. Moreover, metaheuristics are more flexible

than exact methods in two important ways. First, because metaheuristic frameworks are defined in general terms, metaheuristic algorithms can be adapted to fit the needs of most real-life optimization problems in terms of expected solution quality and allowed computing time, which can vary greatly across different problems and different situations. Secondly, metaheuristics do not put any demands on the formulation of the optimization problem (like requiring constraints or objective functions to be expressed as linear functions of the decision variables). However, this flexibility comes at the cost of requiring considerable problem-specific adaptation to achieve good performance. The underlying foundations of different metaheuristics vary significantly. Some model the optimization process by using a metaphor seemingly unrelated to optimization, such as natural evolution (genetic/evolutionary algorithms), the cooling of a crystalline solid (simulated annealing) or the behavior of animal swarms (e.g. ant colony optimization). Others, like tabu search, do not use such an intermediary level of explanation but rather focus on exploiting the problem structure to improve the search for good solutions. In general, metaheuristics frameworks rely heavily on the use of randomness, although some completely deterministic strategies have also been proposed. Metaheuristic algorithms attempt to find the best (feasible) solution out of all possible solutions of an optimization problem. To this end, they evaluate potential solutions and perform a series of operations on them in order to find different, better solutions. Metaheuristics operate on a representation or encoding of a solution, an object that can be stored in computer memory and can be conveniently manipulated by the different operators employed by the metaheuristic. Three fundamental classes of metaheuristics can be distinguished, based on the way in which solutions are manipulated. Local search metaheuristics iteratively make small changes to a single solution. Constructive metaheuristics construct solutions from their constituting parts. Population-based metaheuristics iteratively combine solutions into new ones. However, these classes are not mutually exclusive and many metaheuristic algorithms combine ideas from different classes. Such methods are called hybrid metaheuristics.

2.2 Classification of metaheuristics

A metaheuristic will be successful on a given optimization problem if it can provide a balance between the exploitation of the accumulated search experience and the exploration of the search space to identify regions with high quality solutions in a problem specific, near optimal way. The main difference between the existing metaheuristics concerns the particular way in which they try to achieve this balance. The different metaheuristic approaches can be characterized by different aspects concerning the search path they follow or how memory is exploited. In this section, we discuss these aspects according to some general criteria which may be used to classify the presented algorithms.

Local search vs. global search One approach is to characterize the type of search strategy [3]. One type of search strategy is an improvement on simple local search algorithms. A well known local search algorithm is the hill climbing method which is used to find local optimums. However, hill climbing does not guarantee finding global optimum solutions.

Many metaheuristic ideas were proposed to improve local search heuristic in order to find better solutions. Such metaheuristics include simulated annealing, tabu search, iterated local search, variable neighborhood search, and GRASP [3]. These metaheuristics can both be classified as local search-based or global search metaheuristics.

Other global search metaheuristic that are not local search-based are usually population-based metaheuristics. Such metaheuristics include ant colony optimization, evolutionary computation, particle swarm optimization, and genetic algorithms [3].

Single-solution vs. population-based Another classification dimension is single solution vs population-based searches [3, 53]. Single solution approaches focus on modifying and improving a single candidate solution; single solution metaheuristics include simulated annealing, iterated local search, variable neighborhood search, and guided local search [53]. Population-based approaches maintain and improve multiple candidate solutions, often using population characteristics to guide the search; population based metaheuristics include evolutionary computation, genetic algorithms, and particle swarm optimization [53]. Another category of metaheuristics is Swarm intelligence which is a collective behavior of decentralized, self-organized agents in a population or swarm. Ant colony optimization, [11] particle swarm optimization [53], social cognitive optimization, Harris hawks optimization [21], penguins search optimization algorithm, and artificial bee colony [26] algorithms are examples of this category.

Hybridization and memetic algorithms A hybrid metaheuristic is one which combines a metaheuristic with other optimization approaches, such as algorithms from mathematical programming, constraint programming, and machine learning. Both components of a hybrid metaheuristic may run concurrently and exchange information to guide the search. On the other hand, Memetic algorithms [39] represent the synergy of evolutionary or any population-based approach with separate individual learning or local improvement procedures for problem search. An example of memetic algorithm is the use of a local search algorithm instead of a basic mutation operator in evolutionary algorithms.

Parallel metaheuristics A parallel metaheuristic is one which uses the techniques of parallel programming to run multiple metaheuristic searches in parallel; these may range from simple distributed schemes to concurrent search runs that interact to improve the overall solution.

Nature-inspired metaheuristics A very active area of research is the design of nature-inspired metaheuristics. Many recent metaheuristics, especially evolutionary computation-based algorithms, are inspired by natural systems. Nature acts as a source of concepts, mechanisms and principles for designing of artificial computing systems to deal with complex computational problems [20]. Such metaheuristics include simulated annealing, evolutionary algorithms, ant colony optimization and particle swarm optimization. A large number of more recent metaphor-inspired metaheuristics have started to attract criticism in the research community for hiding their lack of novelty behind an elaborate metaphor.

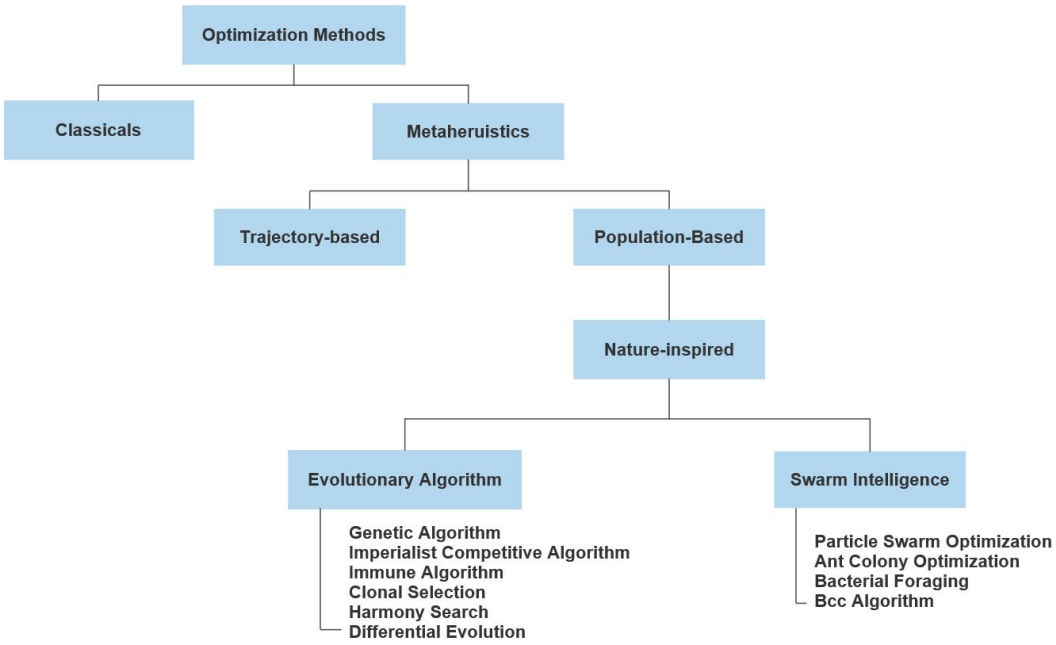


Figure 2.1: Classification of Metaheuristics Search Methods

2.3 Biological Basis of EA

In this section some biological terms, related with EA, are defined. This will be helpful to have some comprehension of the biological processes upon which EAs are based on. Every living organism consists of one or more cells. Inside each cell, there is a nucleus, which is

known as the central part of the cell. The nucleus of every cell of an organism contains the same set of chromosomes. Chromosomes are strings of DNA and serve as a model or blue print" of the whole organism. Blocks of DNA within a chromosome are known as genes. Each gene encodes a particular trait, for example color of eyes or hair. Each gene is located at a particular position in the chromosome. This location is the identity of the gene, and determines the trait to which it is related. The collection of all the genetic materials within all the chromosomes is called genome. A specific set of genes in genome is called genotype. The genotype is directly related with the organisms' phenotype i.e. its physical and mental characteristics, such as hair color, personality, complexion etc. When two organisms mate, their reproduction makes some shuffling of genetic materials of chromosomes from both the parents. A pair of chromosomes exchange genetic information and produce offspring that contain a combination of information from each parent. This is the recombination operation which is often referred to as crossover. Random effects are usually involved in the selection of parents and in the process of shuffling of genes among the chromosomes. Usually organisms with higher fitness get better chance of mating and surviving. EAs usually use some function of the fitness measure to select individuals probability to undergo the mating and reproduction procedure using the genetic operations such as crossover/recombination and mutation. Evolution requires some amount of diversity to work appropriately. In nature, an important source of diversity is *mutation*, which changes a randomly selected gene in the chromosome by a random amount. In an EA, a large amount of diversity is usually introduced at the start of the algorithm, by randomizing the genes across the population. However, this diversity may fall rapidly during the next generations because both recombination and selection operations are usually diversity decreasing operations. Thus, the importance of mutation which introduces further diversity while the algorithm is running cannot be overemphasized.
However, some researchers like crossover/recombination as the main search operator and prefer mutation as a background operator to reintroduce some of the original diversity that may have been lost, while others view mutation as playing the dominant role in the evolutionary process. For example, in the EP-based algorithms mutation is the sole evolutionary search operator, with no recombination or crossover operation.

2.4 Basic Terminology of EA

Since EA is inspired by the concepts of natural genetics and the Darwinian theory of evolution, they use lots of terms from biology and genetics. Since these terms are prevalent and ubiquitous across the literature on evolutionary computation, we briefly present them below

Individual is the carrier of the genetic information. It is characterized by its state in the search space, its fitness and where applicable, a set of strategy parameters. The individual is the unit of selection.

Population is the pool of individuals exhibiting equal or similar genome structures, which allows the application of genetic operators. Total number of individuals in a population is known as population size.

Genome is often used interchangeably with the term individual. Actually genome is the data structure of an individual, which is used during genetic operations, such as crossover and mutation.

Genotype is the representation on which the crossover and mutation operators are applied to.

Chromosome is a set of parameters which define a proposed solution to the problem that the evolutionary algorithm is trying to solve.

Parent (i.e., existing individual or candidate solution) is perturbed by genetic operations (i.e., mutation, crossover/recombination) to produce a new candidate solution (child or offspring).

Gene is known as a particular slot position or a sub unit of a chromosome which, as a rule, represents or codes an (object) parameter or block. In other words, each chromosome is a list (sequence) of gene values.
in biology, is the term given to the appropriate range of values for genes. In evolutionary algorithms, an allele is the value of the gene (or genes).

Phenotype is expression of the properties coded by the individual's genotype. The precise definition is mostly problem-dependent. For parameter optimization the phenotype is usually identical with the object parameters, whereas for structure optimization (e.g. of neural networks) the phenotype represents a specific structure.

Fitness is evaluation of an individual with respect to its reproduction capability. Selection in EA is based on the fitness. Generally, it is determined on the basis of the objective value(s) of the individual in comparison with all other individuals in the selection pool.

Fitness assessment is the task of computing the fitness of an individual.

Fitness Landscape is a metaphor that expresses the dependency of the fitness, resp. objective function, on the location of the individual in the search space.

Generation is a one complete cycle (iteration) of fitness assessment, breeding and reinsertion to form the population for the next cycle. The term generation may also refer to the population produced in each such cycle.

Selection is a necessary operator in EA, which, depending on the fitness or the objective function values, respectively, decides on the reproduction of the individuals. Selection gives a direction to evolution, it guides the search.

Breeding is the procedure of producing one or more children (new candidate solutions) from a pool of parents (i.e., existing candidate solutions) through a process of selection and genetic operations (crossover/recombination and/or mutation).

Crossover is a form of genetic operation related to sexual breeding. A typical recombination or crossover operation selects two individuals (parents), combines or swaps their information in some way to produce (usually) two new individuals (offspring).

Mutation is a form of genetic operation that selects an individual (candidate solution) and randomly perturbs it to form a new individual.

Re-insertion is constructing the population of the next generation from the union of the parent and offspring solutions.

2.5 Overview of EA

In the past few decades, many nature-inspired optimization algorithms have been developed successfully for solving a wide range of optimization problems [31]. Evolutionary

Algorithms (EAs) are one of those. Evolutionary Algorithms consist of several heuristics, which are able to solve optimization tasks by imitating some aspects of natural evolution. They may use different levels of abstraction but they are always working on whole populations of possible solutions for a given task. EAs are an approved set of heuristics, which are flexible to use and postulate only negligible requirements on the optimization task. Evolutionary algorithm is an iterative, stochastic search and optimization technique based on the concepts of the Darwinian theory of evolution. EA maintains a population of individuals or chromosomes (i.e. candidate solutions) that are selected using Darwinians' 'laws of natural selection' with 'survival of the fittest' and updated using operators borrowed from natural genetics like ***crossover, recombination*** and ***mutation***. EA is an iterative algorithm, it runs generation by generation. In each generation, a typical EA employs selection, crossover/recombination and/or mutation operations to produce a pool of new candidate solutions (i.e. offspring) from the existing solutions (i.e. parents). From the union of the parents and offspring, the selection and/or reinsertion policy tries to keep the better candidate solutions and wipe out the low quality individuals during constituting the next generation population. Crossover and recombination are like 'sexual breeding' operations that produce new candidate solutions by combining information from two (or more) different existing solutions, while mutation is like an 'asexual' genetic operation that randomly alters 'gene' value(s) of an individual. Evolutionary algorithms model natural processes, such as selection, recombination, mutation, migration, locality and neighborhood. Figure 2.2 shows the structure of a simple EA. Algorithm 1 shows the pseudo code of a typical EA. Figure 2.3 shows the gradual improvement of the objective value during the run of a typical EA.

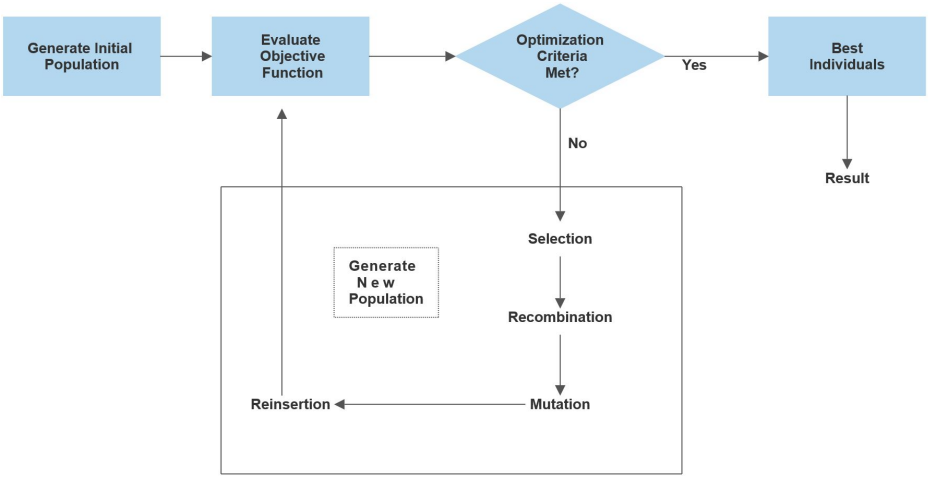


Figure 2.2: Block Diagram of typical EA

Algorithm 1: A typical EA

1 *Initialize*. Generate a random population of n chromosomes (individuals). Each chromosome is a candidate solution for the problem.
2 *Fitness evalution*. Evaluate the fitness $f(x)$ of each x chromosome of the population.
3 *Generate new population*. Create a new population by repeating the following steps until the new population is complete.
4 *Selection*. Select two parent chromosomes from the population according to their fitness values (the better the fitness, the bigger chance to be selected).
5 *Recombination*. Combine the gene values of the two parents to produce new offspring.
6 *Mutation*. With a mutation probability, mutate the new offspring at each gene/locus (position in its chromosome).
7 *Reinsertion*. Either accept or reject the new offspring in a new population.
8 Test for termination.
9 If the stopping criteria are satisfied, then stop and return the best solution in the current generation population.
10 Otherwise, continue.
11 *Loop back*. Go to the step 2 to continue the evolution with the new population of candidate solutions.

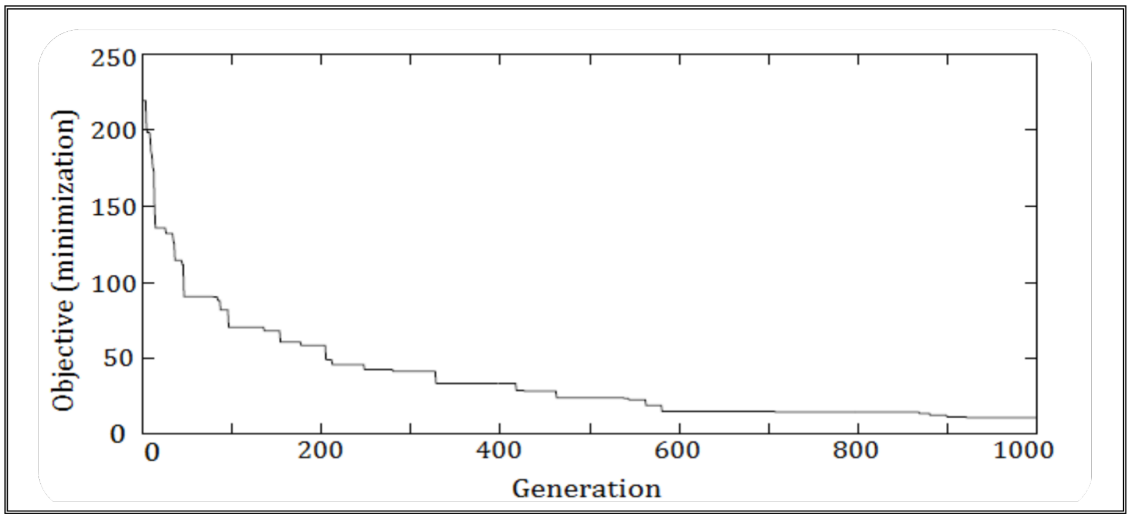


Figure 2.3: Gradual improvement of the objective value during the run of a typical EA.

2.6 Components of EA

EAs have a number of component procedures and operators that must be described to have a proper comprehension of the algorithm. When solving a particular problem using EA, each of these components is required to be specified precisely in order to describe the EA. The essential components of an EA are as follows.

- Representation (Encoding)
- Fitness (or, evaluation) Function
- Selection
- Recombination/Crossover
- Mutation
- Reinsertion

2.6.1 Representation

Representation means to represent or encode a candidate solution of the actual problem as a data object inside a computer program, suitable to be manipulated by the simulated evolutionary computation. A data object representing a potential solution of the actual problem is called a 'chromosome', 'genotype', 'genome' or 'individual' interchangeably while the actual physical solution in the problem domain is referred as 'phenotype'. A candidate solution may be encoded as a collection (e.g. vector, list) of attributes (parameters or search variables) within a chromosome or individual. Each attribute inside a chromosome is called a 'gene'. An appropriate encoding for the parameters in the genotype depends on the problem at hand. However, encoding with binary bit string is very common because almost all problem parameters can be encoded using binary representations. The genetic operations often depend on the encoding of the chromosome, so special encoding may necessitate designing specialized mutation and crossover/recombination operators. Although there exist many EA variants with many different encodings and operators, the two most commonly used encodings are for numeric search domain and permutation domain. Most of the problems handled by EAs appear from these two domains. Numeric domains cover all the problems where the goal is to find a numerical vector (i.e. real-valued vector) that optimizes an objective function value. Most of the applications of EA come from this domain, so there already exist a lot of works addressing the issues of encodings and operators for numeric domains. The two key encodings for numeric domain are the binary string (bit string) encoding and the value encoding. In binary string encoding, every chromosome is a string of bits, 0 or 1.

Chromosome A	white, gray, gray, black, brown
Chromosome B	b a c d c c a b g a c a d b d d c c b
Chromosome C	0.5498 1.5329 2.1092 9.2143 0.2241

For example, a chromosome **A** may be represented as: **1101010100100101 11**. However, in many problems of the numeric domain, value encoding is used, which directly encodes the chromosome as a sequence of its attribute values. For example, if each attribute is a real number, the chromosome is represented as a string (or vector) of real values. Values can be anything connected to the problem, from numbers or characters to some complicated structures or objects. Some examples of value encoding are shown below. Another form of representation is the permutation encoding, which is mostly used in ordering (or, permutation) problems, such as the traveling salesman problem or task ordering problem. In permutation encoding, every chromosome is a string of numbers, each number representing a position in a sequence. For example, a chromosome *D* may have the permutation encoding: 7 2 4 1 6 5 9 3 8, where the original *sequence* has 9 different members. For the traveling salesman problem, the above chromosome *D* may represent the order at which the cities are travelled, such as the city no.7 been travelled first, followed by the city no.2, then city no.4, and so on. Another special kind of representation, which is used with genetic programming [28], is the tree encoding. In tree encoding, each chromosome represents a computer program or expression that is evolved through the evolutionary process. In this scheme, every chromosome is a tree of some objects, such as instructions, or procedures of a programming language.

2.6.2 Fitness Function

The fitness function (or evaluation function) performs the role to define how 'well' each individual chromosome is carrying out as a possible solution of the actual problem. Actually, it is a particular type of objective function but is called 'fitness function' because it assigns a level of fitness to each individual. This fitness measure summarizes how close is an individual to achieve the objective of the given problem at hand. The selection and reinsertion phases usually depend significantly on the fitness values of the individuals assigned by the fitness function. As for example, suppose we want to find the value of x within the domain of 8-bit integers so that the value of the objective function $f(x) = x^2$ is maximized. Here, the phenotype search space contains all possible integers within the range. The evolution will start with a limited number of chromosomes, sampled over the range of 8-bit integers. If we use a binary encoding then a chromosome *00010100* will represent a phenotype integer 20. To measure the fitness of the chromosome, the fitness function may simply compute the square of the corresponding phenotype: $20^2 = 400$. The more the value of the square, the better the chromosome is. Thus, the evaluation function builds a bridge from the genotype

space of the simulated evolution towards the phenotype space of the actual problem domain. The direction of the evolution depends entirely on how the fitness function interprets the fitness and evaluates the chromosomes.

Designing a workable fitness function may not be straightforward. Even though it is no longer the human designer but the computer, that comes up with the final design, it is the human designer who has to design the fitness function. If the fitness function is not designed appropriately, the algorithm may converge to undesired solutions or even fail to converge at all. Besides, the fitness function should be designed such a way that it not only closely expresses the goal of the human designer, but also can be computed quickly with small computational expense. The execution speed of an EA or SIA is directly dependent on how efficiently the fitness function can be computed. In some instances, fitness approximation is allowed instead of exact fitness computation. This reason might be an extremely high computational need of fitness evaluation, or the lack of an appropriate and known model of the fitness function. In some cases, it may become completely impossible to even guess the definition of the fitness function (e.g. the aesthetics of a fashion product or the taste of a coffee). The interactive genetic algorithm is a family of EA that handle this difficulty by employing external agents (e.g. humans, polls, online forums) for fitness evaluations [6].

2.6.3 Selection

Inspired by the 'laws of natural selection' and 'survival of the fittest' in the Darwinian theory of evolution, a typical EA selects a number of individuals (parents) from the population to constitute a mating pool where they go through the genetic procedures of crossover/recombination with each other to reproduce a number of new individuals (offspring). Selection is usually based on fitness to provide the fitter individuals with better chance of mating, reproduction, and survival in order to simulate the Darwinian evolutionary principle of survival of the fittest.

There are a number of standard techniques to make the selection of individuals (parents). For example, the roulette wheel selection, rank based selection, tournament selection, steady state selection and scaling selection. In Roulette Wheel Selection, parents are selected in proportion to their fitness. The better an individual, the more likely it is to be selected. Consider a roulette wheel where all the individuals are placed. The slice of every individual is as large as (or proportional to) the fitness of the individual, assigned by the fitness function. The wheel is rotated at a random pace and a marble is thrown into it to select a chromosome. Chromosomes with higher fitness and occupying more area on the roulette wheel will have better chance to be selected. If we need N individuals, the marble is thrown Ntimes; each time it returns an individual.

When the best individual of the population is exceptionally better than the other individuals, the roulette wheel selection will cause problems by selecting the best individual several times, and filling the populations with its multiple copies. For example, if the best chromosomes' fitness covers 70 percentage of the area of the roulette wheel, then the other chromosomes will have few chances to be selected. Rank based selection eliminates such problem. Rank based selection first ranks all the individuals of the population based on their objective function value. Then every chromosome receives fitness from its ranking, not from their actual objective values. Selection is made from these rank based fitness values.

In tournament selection, a number (say, T) of individuals is chosen uniformly at random from the population and the individual with the best fitness of this group is selected as a parent. This process is repeated as many times as individuals are needed as parents. The parameter T values ranging from 2 to N where $N$is the number of individuals in the population. Large values of T(e.g., $T=10$) intensifies the degree of exploitations because it increases the likelihood of picking from the very fit individuals during each selection, while smaller values of T (e.g. $T=2$) makes the algorithm more explorative because it picks both high and low quality individuals with significant probability allowing some diversity to persist throughout the entire course of the evolution, which may be necessary to avoid premature convergence for some problems.

Another commonly used selection scheme is the Scaling Selection. As the average fitness of the population increases gradually, the scaling selection scheme ensures that the strength of the selective pressure also increases and the fitness function gradually becomes more and more discriminating with the ongoing generations. This method can be helpful in making the best selection later on when all individuals have relatively high fitness values and only small differences in their fitness values distinguish them one from another.

Another selection scheme- Steady State Selection ensures that most of the chromosomes of the current generation survive to the next generation. Such a selection scheme usually exhibits steady improvement of fitness values, and avoids wild oscillations in the average fitness values. In each generation, some good (or best) chromosomes are selected for mating. The offspring replace some bad (or, worst) chromosomes from the population. The rest of the population survives to the new generation. The opposite of Steady State Selection is the Generational selection-the offspring of the parent individuals selected from i^{th} generation become the entire next (i.e., $(i+1)^{th}$) generation. No individuals are retained between the generations.

A term closely connected with selection is Elitism. Elitism is a method which safeguards the best part of the population. During each generation, the best chromosome or a pre-defined number (say, k of best chromosomes are copied to the new population. Then any selection scheme completes the rest of the selection. The parameter $k$is called the *elite size*. Elitism

has shown good performance with a number of problems [36, 38] because it protects the best solutions. However, both steady state selection and too strong elitism (i.e. large value of the elite size k can turn an EA into too much exploitative by filling the population only by the best individual and its offspring solutions.

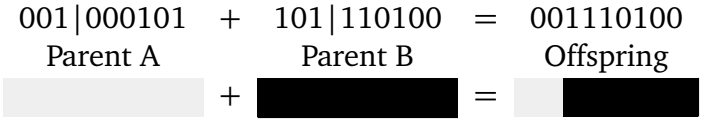
2.6.4 Recombination/Crossover

Recombination is the process of generating new individuals (i.e. offspring candidate solutions) by combining the information of two or more existing individuals (parent candidate solutions). Each individual contains a number of attributes (i.e. parameters or search variables). Recombination is done by combining the attribute values of the parents. There are several ways of recombination. The representation (encoding) of the parents play an important role in determining the method of recombination to be applied on the parents. For individuals with real valued encoding of the attributes, several variants of recombination is defined. For example, line recombination, extended line recombination, intermediate recombination. In intermediate recombination, the attribute variables of the offspring are randomly chosen somewhere around and between the 'parents' attribute variables. Offspring are produced according to the following rule.

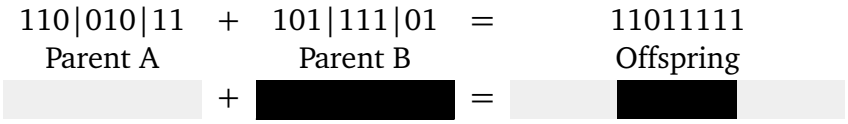
$$Var_i^0 = Var_i^{p1} * \alpha_i + Var_i^{p2} * (1 - \alpha_i); \alpha_i \in [-d, 1+d]; i \in 1, 2, ..., n \tag{2.1}$$

$\alpha_i \in [-d, 1+d]$ Here,n is the number of variables in each individual (which is usually same as the dimensionality of the problem),α_i is a scaling factor chosen uniformly at random from the interval $[-d, 1+d]$. If d is set to 0, offspring are always generated at the intermediate region of the parents. Thus, over the generations, the area spanned by the offspring gradually reduces than the area of the parents. From statistical studies, an appropriate value of $d = 0.25$ has been chosen, which ensures that the area spanned by the offspring does not shrink by successive recombination operations over the generations. The difference between intermediate recombination and the line (or extended line) recombination is that the latter always produces the offspring solutions on the line (or extended line) connecting both the parent solutions, while the former one (i.e. intermediate recombination) produces the offspring at any point inside the hypercube (or extended hypercube, based on the value of d in eq. (2.1)) surrounding the parents. When parents have binary encoding, their recombination constitutes a special case, known as crossover. During crossover, a random bit position, k is selected from the range $[1...n]$. Then, the pair of mating parent exchanges all their bits starting from the k^{th} position. The random bit position k is called the crossover point and selected anew for each crossover operation. Depending on the number of crossover points, there exist single-point, two-point and multi-point crossover. An example of single point

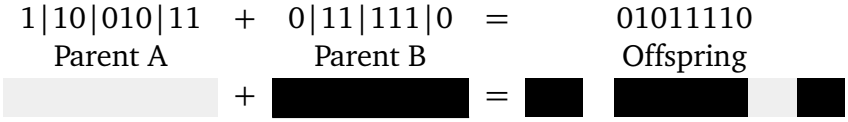
crossover is shown in the following Figure.



For two-point crossover, two crossover points are selected at random and binary string from the beginning of chromosome to the first crossover point is copied from one parent, the part from the first to the second crossover point is copied from the second parent and the rest is copied from the first parent. This is illustrated with an example in the following Figure.



Another form of crossover is uniform crossover in which bits are randomly copied from the first or from the second parent, based on a crossover probability p_c The following Figure shows this kind of crossover using an example.



2.6.5 Mutation

Mutation means randomly altering the chromosomes. In EP-based algorithms, mutation is considered the sole genetic operator, while other EA families employ both the recombination and mutation operations. Two parameters are associated with mutation-mutation step and mutation rate. Mutation step controls the amount of variation incurred by the mutation. Mutation rate controls the probability of an attribute (gene) within a chromosome being mutated. Usually mutations are applied with small mutation steps and low mutation rate. Offspring are probabilistically mutated after being created by the recombination step. Like recombination, there exist several standard ways of mutation. The encoding of the chromosome determines the possible means of mutation that may be applied on it. If the chromosome has value encoding, with real values for the attributes, then mutation means adding randomly created real values with the attributes. For binary encoding, mutation means the flipping of the bits, since every bit has to be either 0 or 1.

The mutation rate usually depends on the problem at hand. For many problems, the mutation rate is made inversely proportional to the number of variables. The more attributes (genes or search parameters) an individual has, the smaller is set the mutation probability. There exist a number of research works that try to find the optimal mutation rate and step

size for a problem. However, a mutation rate of $1/n$ (n: number of genes within a chromosome) has been reported to exhibit satisfactory results for a wide range of problems. With this mutation rate, only one variable per individual is altered. Thus, the mutation rate is become independent of the population size. Another strategy suggested in [2] is to set higher mutation rates at the beginning of the evolution, and declining this rate with the increasing generations, which has shown an acceleration of the search process for many problems.

The optimal size for the mutation step is usually difficult to realize. It always depends on the problem at hand and often varies during the ongoing optimization process. Small mutation steps are usually successful, while larger steps, when successful, produce good results much quicker. For this reason, a good mutation operator should produce both small and large step-sizes in suitable proportions. Such a mutation operator, as proposed and employed in [41, 42] is specified the following eq. (2.2).

$$Var_i^{Mut} = Var_i + s_i * r_i * a_i \tag{2.2}$$

$$\begin{gathered} i \in 1,2,...n \; uniform\; at\; random, s_i \in -1,+1 \; uniform\; at\; random \\ r_i = r.domain_i;\; r:\; mutation\; range(standard : 10\; percent) \\ a_i = 2^{-u.k}; u \in [0,1]\; uniform\; at\; random,\; k:\; mutation\; prec \end{gathered}$$

This mutation is able to generate most points in the hypercube defined by the domain of the attributes (search variables) of the individuals. Most mutated individuals tend to be near the parent individual. Only some mutated individuals will be far away from the parent. Thus, the probability of small step-sizes is greater than larger steps which is proper for most problems.

For binary valued individuals, mutation means the flipping of the attribute (i.e. gene) values because every gene can have either of only two states - 0 and 1. Thus, the size of the mutation step is always 1. For every individual, the gene value to be mutated is chosen mostly uniform at random. The following example in the figure below shows how a binary mutation alters a chromosome with 10 bits. Here, mutation randomly flips the bit at position 6 from 0 to 1.

Choromosome (before mutation)	0	0	1	0	0	0	1	0	1	1
Randomly selected gene position to be muted:6										
Choromosome (after mutation)	0	0	1	0	0	1	1	0	1	1

In order to mutate real variables, it is possible to adapt the direction and step-size to conduct a more effective search process. These methods are from evolutionary strategies [50] and also from evolutionary programming [13]. The extensions of these methods and new

developments include several new schemes, such as - adaptation of n (number of search variables) step-sizes but no direction [17,34] adaptation of n step-sizes and only one direction [17] and adaptation of n step-sizes and n directions [12]

2.6.6 Reinsertion

Reinsertion is the process of constructing the next generation population from the union of the parents and offspring. There exist several schemes, each with its own merits and demerits. In pure reinsertion [50], the number of offspring reproduced is equal to the number of parents and the offspring population replaces the parent population entirely. Thus, each individual, even the best one, lives for only a single generation. In uniform reinsertion [13], fewer offspring are produced than parents and offspring replace parents uniformly at random. As a result, better individuals may be replaced by weaker offspring. In elitist reinsertion [13], the worst individuals of the current population are replaced by the new offspring, which means the elitist reinsertion can become very exploitative. Another scheme, fitness based reinsertion [13], produces more offspring than needed, and they are inserted into the population based on their fitness. A number of reinsertion schemes may be combined to construct a new scheme. For example, the elitist scheme combined with fitness-based reinsertion prevents the best individuals from being lost and it is recommended for many problems. In this scheme, a given number of the least fit parents are replaced by the same number of the fittest offspring. Some EAs [52] consider not only the fitness value, but also the diversity of a candidate solution to allow it to the next generation. A more diverse individual promotes more search space explorations and helps the algorithm avoid premature convergence to any locally optimal point.

In addition to all the global reinsertion schemes described above, reinsertion may also be based on local policies. In local reinsertion, individuals are considered within its bounded neighborhood only. The reinsertion of an offspring usually takes place in exactly the same neighborhood from where it is selected. This preserves the locality of the genetic information. Examples of some local reinsertion policies are as follows.

- Insert every offspring and replace weakest individuals in the neighborhood.
- Insert offspring fitter than weakest individual in the neighborhood and replace the weakest individuals in the neighborhood.
- Insert offspring fitter than the parent and replace the parent.
- Insert every offspring and replace individuals in the neighborhood randomly.
- Insert offspring fitter than weakest individual in the neighborhood replacing parent.

- Insert offspring fitter than the weakest individual in the neighborhood and replace individuals in the neighborhood uniformly at random.

During the reinsertion step, a number of better parent solutions may be replaced by some worse offspring. However, this does not cause trouble for most problems, since if the inserted offspring are extremely bad, they are very likely to be replaced with new, better offspring solutions in the next generation.

2.7 Advantages and Strengths of EA

EAs belong to the population-based, stochastic, meta-heuristic family of algorithms and they share many strengths and advantages over other analytical, deterministic, direct-search and single-state algorithms. Their most prominent strengths and advantages are listed in the following paragraphs [40].

(i) EAs inherently parallel in nature, because they have to maintain a population of candidate solutions. The candidate solutions can produce multiple offspring to explore the solution space in multiple directions simultaneously. If a particular path reaches some dead end or a poor local optimum, it can be easily discarded and search can continue with more efforts along the more promising paths [40].

(ii)EAs are not just parallel hill-climbers; rather they are much better than that. Each candidate solution can share information with other solutions and affect how they do the hill-climbing in the fitness landscape. The better solutions can share their information with the weaker ones, through the recombination and perturbation operations, thus causing them to move towards the good quality search regions [40].

(iii)EAs can often produce good quality solutions by searching only a small fraction of the search space. This is why they are particularly effective on the problems with very large search space - too large for any exhaustive search. The source of this strength is their inherent parallelism and their ability to implicitly evaluate many schema (i.e. patterns within candidate solutions) simultaneously as demonstrated in with an example of the Schemata theorem [40].

(iv)EAs are especially suited for non-linear problems with extremely large search space. Most real world problems are non-linear, where the search variables are not independent of one another - changing a single variable may have ripple effects on the others and can affect the objective value in an non-predicted way. Non-linearity results in an exponential increase of the search space. Since EAs and SIAs can usually find good quality solutions by searching only a small fraction of the search space and within a reasonable amount of

time, they are often a good choice for the nonlinear problems [40].

(v) EAs are well suited for problems with complex fitness landscape where the fitness function may be discontinuous, noisy or multimodal (having many local optima). Many real-world problems have a huge search space with several local optima where the search algorithm may get stuck with premature convergence. Both EAs and SIAs have proven their effectiveness at escaping the locally optimal points and locating the neighborhood of the global optimum, even in a very complex fitness landscape [40].

(vi)EAs are exceptionally good for multi objective and multi parameter optimization. They can handle large number of parameters and work on many objectives simultaneously. Their use of parallelism enables them to produce multiple good solutions to the same problem, possibly with one candidate solution optimizing one objective and another solution optimizing a different one [40].

(vii)EAs often come up with novel and unconventional solutions to a problem. This is because they don't use any prior knowledge or domain specific information of the problem during their search. They are like 'blind watchmakers', because they simply make random changes to their candidate solutions and then use the fitness function to determine whether those changes produce any improvements. An interesting example of novel design from the EA is where the EA came up with a high performance jet engine turbine design that was three times better than a human designed configuration and 50 percentage better than a configuration designed by an expert system. To find the design, the EA successfully searched across a huge solution space with more than 10^{387} possible solutions, which clearly indicates the strength of EAs on large and complex optimization problems [40].

2.8 Disadvantages and Limitations of EA

Despite their many strengths and advantages, as presented in the previous section, EAs do have some limitations too. However, these limitations are mostly well addressed by the researchers and many of them have firm roots in their biological and natural inspirations. Some of their limitations are as follows: **Need for an appropriate representation:** During solving any problem with EA, the first step is to represent a candidate solution as an individual of the population. This requires an effective and appropriate representation of the problem. The representation must be robust and fault to learn as it has to ensure that random mutation or perturbations of the existing candidate solutions can not produce invalid or inconsistent candidate solutions. One simple way to ensure this is to employ a repair operator after each perturbation that ensures the validity of the newly perturbed solution, possibly by doing some extra tasks [40].

Need for a well designed fitness function: The fitness function should be thoughtfully designed such that higher fitness values are assigned only to those candidate solutions that are actually better solutions to the given problem at hand. If the fitness function is not designed properly, the EA may be unable to find a solution or even worse, may wind up solving a completely wrong problem, an example of which can be found in where the objective was to design a circuit that can produce an oscillating signal. When EA finishes its execution using the improper fitness function, a circuit is evolved that really outputs an oscillating signal but not producing by itself, rather by acting as a radio receiver to pick up and relay an oscillating signal from the nearby electronic devices, which was highly undesirable [40].

High computational complexity: For complex problems, the repeated fitness evaluations may become computationally prohibitive and may limit the applicability of EAs. In many real world problems, the fitness evaluation is computationally very expensive. For example in some structural optimization problems, a single function evaluation may require hours or even days of simulations. For such instances, EAs may not be a good choice. However in such cases, an EA may use approximate fitness evaluations which is less computation intensive than the exact fitness evaluations [40].

Limitation with dynamic data: Both EAs and SIAs have inherent limitations to deal with dynamic data sets. This is because the population or swarm usually converges to the best solutions of the already seen data, which may no longer be the best solution (or even a good solution) for the newly arrived dynamic data points [40].

Limitation with decision problems: For decision problems, the fitness measure of a candidate solution is just a right/wrong measure, so the fitness value is either 1 or 0. This makes EAs to be ineffective for such problems, because there is no fitness hill to climb, rather steep fitness cliffs with no gradient direction to follow [40].

Limitation with deceptive functions: Another type of problem where both EAs face difficulty is the problems with 'deceptive' fitness functions. For these problems, the gradient directions often take the algorithm farther from the global optimum. This can happen when a strong local optimum is located far from the narrow global optimum and the gradients mostly point towards that local optimum. For such problems, an EA is likely to find the strong local optimum instead of the global optimum [40].

Limitation with fine tuning: EA are good to produce decent near optimum solutions within bounded computation time and resource. Usually they can quickly reach close to the peak of a good local or the global optimal hill of the fitness landscape. But precisely pinpointing the peak is usually difficult for them and can be done only by random mutations or perturbations (i.e. by pure chance only). However, for most real world situations, a near optimum solution is good enough, given a bounded, reasonable amount of time and

efforts [40].

Premature convergence: One crucial issue for the success of both EAs is the premature convergence [36], which means that the entire population of candidate solutions has converged to one or a few locally optimal points, failing to locate the global optimum. As a result, the algorithm returns a sub-optimal solution which may be nowhere around the globally optimum solution. This usually happens when one or a few candidate solutions suddenly become exceptionally fitter than the rest of the population causing the selection operator to pick them several times for reproduction which may gradually fill the entire population only by their descendants. This drastically drives down the diversity of the population because the entire population now represents only a single (or a few) locally optimal point(s) around those exceptionally fit candidate solutions. Premature convergence is followed by fitness stagnation for many successive iterations. In the ideal cases, such fitness stagnation of the population coincides with the successful discovery of the global optimum, which is highly desirable. But stagnation may also occur with premature convergence, because all the individuals have become so similar that the crossover/recombination operation can't produce any new or better candidate solutions. At this point, mutation is the only way to break the premature convergence and explore other areas of the search space. But mutation often requires a number of random downhill steps to get away from the peak of the locally optimal hill where the entire population has converged, which may not be allowed by the selection operator that tries to prematurely dismiss those downhill steps produced by mutation operation [40].

The likelihood of premature convergence often depends on the characteristics of the fitness landscape - some functions may provide an easy ascent to the globally optimum peak of the fitness landscape, while some other may allow easier rides to the locally optimal points. This may happen more easily with the deceptive problem or with the problems that have wide, strong locally optimal points far from the global optimum. The problem of premature convergence is common to arise with EAs and SIAs and this is particularly common with small population/swarm size, because the possibility of large improvements by a mutation on a particular individual is high with small population size and it may cause the individual to become dominant over the rest of the small population. It is interesting to note that premature convergence does occur in nature, which is called 'genetic drift' by the biologists. This is nothing unusual with the real evolution, because evolution has no commitment to find the global best solution; rather it can find locally optimal solutions that are just good enough. However, premature convergence in nature is less common, because the most beneficial mutations on living things usually produce small, incremental fitness improvements, allowing very little chance to an individual to be dominant over the rest of the population [40].

2.9 Classification and Use of EA

Evolutionary algorithms (EA) can be classified into following algorithms,

- Genetic Algorithms (GA)
- Genetic Algorithms (GA)
- Evolutionary Programming (EP)
- Evolution Strategy (ES)
- Learning Class Classifier (LCS)

Genetic Algorithms (GAs): Genetic Algorithms are adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics. As such they represent an intelligent exploitation of a random search used to solve optimization problems. Although randomized, GAs are by no means random, instead they exploit historical information to direct the search into the region of better performance within the search space. The basic techniques of the GAs are designed to simulate processes in natural systems necessary for evolution, especially those follow the principles first laid down by Charles Darwin of “survival of the fittest". Since in nature, competition among individuals for scanty resources results in the fittest individuals dominating over the weaker ones.

Genetic Programming (GP): One of the central challenges of computer science is to get a computer to do what needs to be done, without telling it how to do it. Genetic programming addresses this challenge by providing a method for automatically creating a working computer program from a high-level problem statement of the problem. Genetic programming achieves this goal of automatic programming by genetically breeding a population of computer programs using the principles of Darwinian natural selection and biologically inspired operations. The operations include reproduction, crossover, mutation, and architecture-altering operations patterned after gene duplication and gene deletion in nature. Genetic programming is a domain-independent method that genetically breeds a population of computer programs to solve a problem. Specifically, genetic programming iteratively transforms a population of computer programs into a new generation of programs by applying analogs of naturally occurring genetic operations. The genetic operations include crossover mutation, reproduction, gene duplication and gene deletion.

Learning Classifier System (LCS): Learning Classifier Systems (LCS) is one of the major families of techniques that apply evolutionary computation to machine learning tasks. LCS are almost as ancient as GAs, Holland made one of the first proposals.

Evolutionary programming(EP): Evolutionary programming(EP) is one of the four major evolutionary algorithm paradigms. A stochastic optimization strategy originally conceived by Lawrence J Fogel in 1960. It is similar to genetic programming but the structure of the program to be optimized is fixed while its numerical parameters are allowed to evolve.

Procedure of EP For EP, like GAs, there is an underlying assumption that a fitness landscape can be characterized in terms of variables, and that there is an optimum solution (or multiple such optima) in terms of those variables. For example, if one were trying to find the shortest path in a Traveling Salesman Problem, each solution would be a path. The length of the path could be expressed as a number which would serve as the solution's fitness. The fitness landscape for this problem could be characterized as a hyper surface proportional to the path lengths in a space of possible paths. The goal would be to find the globally shortest path in that space or more practically, to find very short tours very quickly. The basic EP method involves 3 steps (Repeat until a threshold for iteration is exceeded or an adequate solution is obtained):

- Choose an initial population of trial solutions at random. The number of solutions in a population is highly relevant to the speed of optimization, but no definite answers are available as to how many solutions are appropriate (other than > 1) and how many solutions are just wasteful.
- Each solution is replicated into a new population. Each of these offspring solutions are mutated according to a distribution of mutation types, ranging from minor too extreme with a continuum of mutation types between. The severity of mutation is judged on the basis of the functional change imposed on the parents.
- Each offspring solution is assessed by computing its fitness. Typically, a stochastic tournament is held to determine N solutions to be retained for the population of solutions, although this is occasionally performed deterministically. There is no requirement that the population could be held constant, however nor that only a single offspring be generated from each parent.

It should be pointed out that EP typically does not use any crossover as a genetic operator [26].

Algorithm and Flowchart of EP Figure 2.4 shows the flowchart of a typical evolutionary programming (EP). Algorithm 2 shows the pseudocode of a typical evolutionary programming (EP).

Algorithm 2: A typical EP

1 Start with an initial time $t = 0$;
2 Initialize a usually random population of individuals $initpopulationP(t)$;
3 Evaluate fitness of all initial individuals of population $evaluateP(t)$;
4 Test for termination criterion (time, fitness etc.) ;
5 **while** *not done* **do**
6 Perturb the whole population stochastically $P(t)^{'} = mutateP(t)$;
7 Evaluate its new fitness $evaluateP(t)^{'}$;
8 Stochastically select the survivors from actual fitness $P(t+1) = surviveP(t)^{'}, P(t)$;
9 Increase the time counter $t = t + 1$;
10 **end**

Figure 2.4: Flowchart of evolutionary programming (EP)

2.9.1 Evolutionary Programming (CEP, FEP and IFEP)

Classical Evolutionary Programming It is one of the variations of evolutionary programming (EP). It uses population of 100 individuals with 30 dimensions. It calculates fitness score from benchmark functions. In this method, Guassian distribution is used. This method converges slowly for multimodal functions. It follows the long self-adaptation process. The offspring is generated by

$$X_i^{'}(j) = X_i(j) + \eta_i(j) * N_j(0,1) \tag{2.3}$$

$$\eta_i^{'}(j) = \eta_i * e^{\tau^i N(0,1) + \tau N_j(0,1)} \tag{2.4}$$

Where,
$X_i(j)$ denote the j-th component of the vector $X_i(j)$
$X_i^{'}(j)$ denote the j-th component of the vector $X_i^{'}(j)$
$\eta_i(j)$ denote the j-th component of the vector $\eta_i(j)$
$\eta_i^{'}(j)$ denote the j-th component of the vector $\eta_i^{'}(j)$

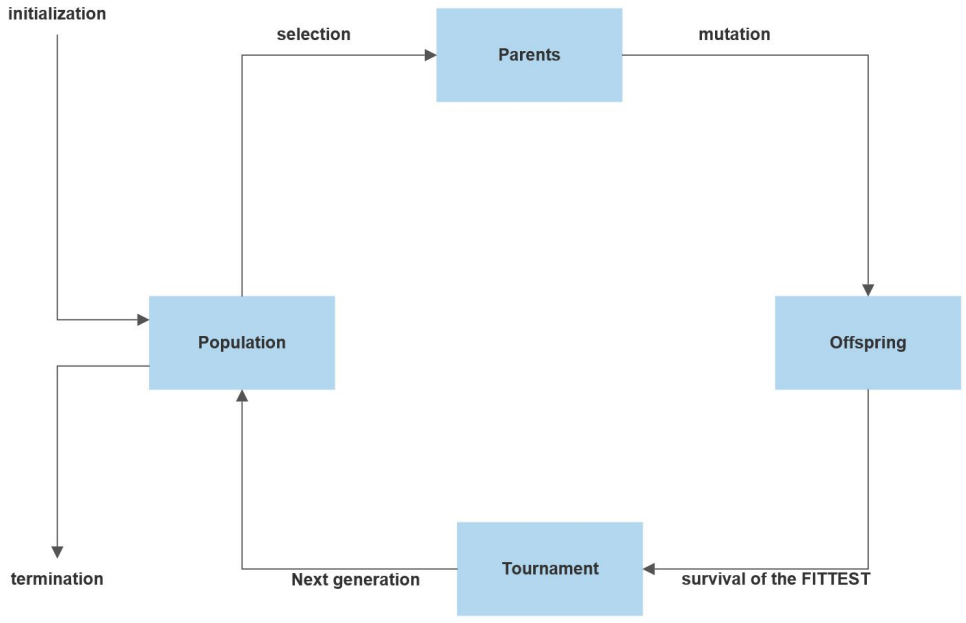


Figure 2.5: Flowchart of CEP

Algorithm 3: CEP

```
t=0, initialize strategy parameters ;
Create and initialize the population,C = 0, of n_s individuals ;
Evaluate the fitness, f(x_i(t)), of each individual ;
while stopping condition (s) not true do
    for each individual,x_i(t) ∈ C(t) do
        Create an offspring, x_i'(t) by applying the mutation operator ;
        Evaluate the fitness,f(x_i'(t)) ;
        Add x_i'(t) to the set of offspring, C'(t) ;
    end
end
```

Fast Evolutionary Programming (FEP) This is another variation of evolutionary programming. FEP uses Cauchy distribution for generating offspring. Mutational noise is sampled from the Cauchy distribution with v=1. Cauchy distribution provides larger step sizes therefore, faster convergence and better exploration is possible using this method. The equation for generating the offspring:

$$X_i^{'}(j) = X_i(j) + \eta_i(j)\delta_j \tag{2.5}$$

Cauchy mutation generates offspring far away from its parents than Gaussian mutation. That's why Cauchy distribution is far better for multimodal functions.

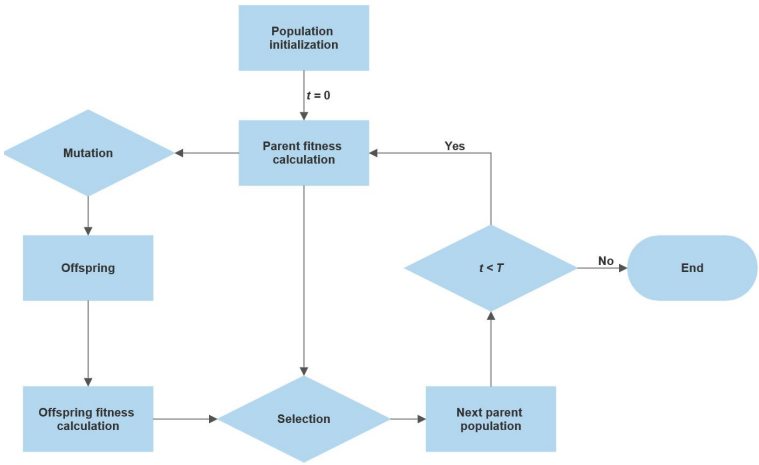


Figure 2.6: Flow chart of FEP

Improved Fast Evolutionary Programming (IFEP) For each parent IFEP generates two offspring. One offspring is generated using Gaussian mutation and another is generated using Cauchy mutation. The best offspring is chosen as the surviving offspring.

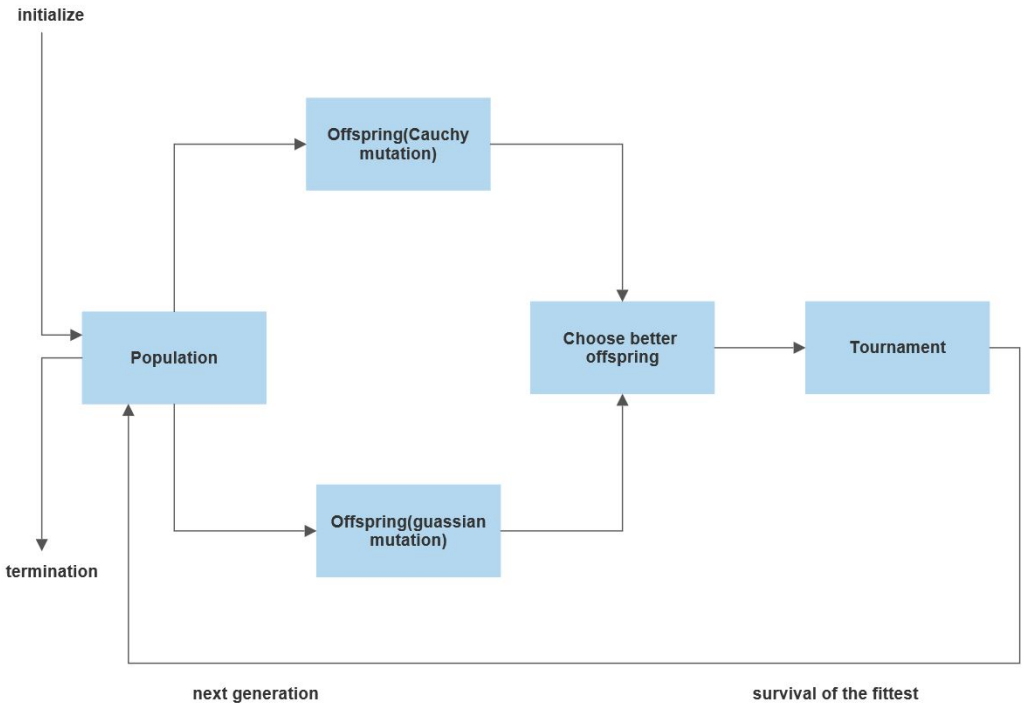


Figure 2.7: Flow chart of IFEP

Algorithm 4: IFEP

1 Initialize the population of μ individuals, $(x_i, \eta_i), \forall_i \in 1, ..., \mu$;
2 Evaluate the fitness of each individual, $(x_i, \eta_i), \forall_i \in 1, ..., \mu$;
3 **for** *each individual* $(x_i, \eta_i), \forall_i \in 1, ..., \mu$ **do**
4 create a single offspring $(x_i^i, \eta_i^{'})_1$ from (x_i, η_i) by Gaussian mutatin ;
5 create a single offspring $(x_i^i, \eta_i^{'})_2$ from (x_i, η_i) by Cauchy mutation ;
6 evaluate the fitness of $(x_i^i, \eta_i^{'})_1$ and $(x_i^i, \eta_i^{'})_2$ select the best offspring $(x_i^i, \eta_i^{'})$ of $(x_i^i, \eta_i^{'})_1$ and $(x_i^i, \eta_i^{'})_2$
7 **end**
8 Select the ν individuals out of (x_i, η_i) and $(x_i^{'}, \eta_i^{'}), \forall_i \in 1, ..., \mu$;
9 **if** *the halting criterion is satisfied* **then**
10 stop ;
11 **else**
12 go to step 2 ;
13 **end**

2.10 Swarm Intelligence Based Algorithms

2.10.1 SIA Overview

Swarm of insects that show social behavior, such as ants, bees, wasps and termites, have fascinated scientists and researchers as well as naturalists and poets for hundreds of years. Each member of a swarm, e.g., an ant or a bee, seems to perform very simple and basic operations, but together all their activities seamlessly integrate in an unsupervised and distributed way, without any central control, and yet lead to extraordinary results, such as the construction of an amazing ant colony, bee hive, termite nest or finding a set of optimal routes to good quality food sources over large distances and across lots of obstacles from their nests. Swarm intelligence is the branch of science that studies such emergent collective intelligence from the groups of decentralized, self-organized simple agents, in both natural and artificial systems. In particular, swarm intelligence based algorithms focus on the communal behavior and collective intelligence that may result from the local interactions and self-organizations of the individuals with each other and with their environment. The motivation of SI based algorithms lies in numerous natural systems, like the colonies of ants and termites, hives of bees, schools of fish, flocks of birds and herds of land animals. Examples of Swarm Intelligence Based algorithm

- Particle swarm optimization
- Ant colony optimization
- Artificial bee colony algorithm
- Differential evolution
- Bat Algorithm

2.10.2 Properties of Swarm Intelligence Algorithms

A typical swarm intelligence based system is a multi-agent system that shows self organized behavior based on the local interactions of the agents among themselves and with the surrounding environment. The self-organization behavior, usually combined with some significant degree random fluctuations, often leads to the emergence of globally intelligent behavior in a distributed style without the need of any central coordination. A typical swarm intelligence based system can be characterized by the following distinct properties.

- A swarm is composed of many agents (individuals).
- The individuals are usually homogeneous or nearly homogeneous in nature. They are either identical or belong to only a few closely related typologies.
- The individuals interact with each other and with the environment. The interactions are based on only a few simple behavioral rules that exploits some local information that they exchange, either directly or via some medium or environment (e.g. stigmergy).
- The overall behavior of the swarm system is a direct result of the interactions of the self-organized individuals with each other and with their environment. This group behavior shows the emergence of intelligent behavior from the collaborative, distributed, self-organized behavior of the individual swarm members.

The distinguished property of a swarm intelligence based algorithm is its ability to show collective, coordinated, intelligent group behavior without the presence of any central or external coordinator, which results from the interactions of spatially distributed individuals that exchange important local information among themselves to organize their behavior based on some simple, basic rules. Also, some randomness and stochastic behavior should also be present in each individual of the swarm, which may depend on the information it receives from the neighboring individuals and also from the surrounding environment. These properties make it possible to design swarm intelligence based algorithms that environment. These properties make it possible to design swarm intelligence based algorithms that are significantly scalable, parallel and fault-tolerant.

- Scalability is possible in a swarm intelligence based system because the interactions are only among the neighboring individuals, so the total number of interactions don't increase severely with the number of individuals in the swarm.
- Parallelism is possible in swarm intelligence based algorithms because the swarm individuals can simultaneously perform different actions in different regions of the search space without waiting for each-other or without the need for any central control.
- Since every individual swarm member is autonomous, decentralized and self-organized by using local information only, they can easily aggregate to form a fault tolerant system. The individuals are homogeneous or nearly homogeneous, so they are easily interchangeable. A failed individual can easily be replaced by another one, thus making a highly fault-tolerant system.

2.10.3 Principles of Swarm Intelligence Algorithms

In swarm based algorithms, the local interactions among simple self-organized agents lead to the development of collective intelligence at the global level. It has been showed [4] that such self-organization is the key feature of a swarm system, from which results a global (macroscopic) level intelligent response from the local (microscopic) level interaction In [4], the self-organization in a swarm system is characterized through four characteristics.

- Positive feedback from the agents-iIt is often a simple behavioral rule that helps to create a collective and convenient behavioral structure. For example, recruitment and reinforcement of the insects such as depositing pheromones along the trails by the ants, following nearby ants or their pheromone trails dancing of bees after returning to hive after they find some good quality food source (i.e. nectar) all these are the examples of positive feedback.

- Negative feedback from the failures of the agents-This counterbalances the positivee , which is necessary to stabilize the collective search pattern. Examples of negative feedback in natural swarms include the evaporation of pheromones from longer, unattractive ant trails, the abandonment of a food source by the bees when its nectar amount exhausts, or avoiding food sources that gets too much crowded from the competition of many other foragers. In such situations, a negative feedback helps to stabilize the system.

- Some random fluctuations that act as a source of creativity and innovation and helps avoid the local minima. Some examples of random fluctuations include random walks by ants, random task switching among the foraging ants, random explorations by scout bees, some random low-rate errors made by each swarm member and so on. Randomness is always important to break free from any locally optimal point and it also promotes the discovery of new, innovative solutions.

- Multiple interactions among the swarm individuals-This usually occurs automatically from the interaction among agents and their positive and negative feedbacks, because the agents in the swarm use the information from each-other to self-organize their own behavior. Gradually, the information, data and their impact spread throughout the swarm.

For the emergence of swarm intelligence, the following five requirements have been identified in [37] that are to be satisfied by the swarm behavior.

- The proximity principle-The swarm individuals should be able to do simple, basic space and time computations.

- The quality principle-The swarm should be able to respond to quality factors in the environment such as the quality of food sources or the safety of a location.
- The principle of diverse response The swarm should not allocate all of its resources along excessively narrow channels and it should distribute resources into many nodes.
- The principle of stability-The swarm should not changeits mode of behavior upon every fluctuation of the environment.
- The principle of adaptability-The swarm must be able to change behavior mode when the investment in energy is worth the computational price.

In some recent works (e.g., [1, 58]), it is emphasized that an intelligent division of labor and performing specialized tasks simultaneously by specialized agents are also important for the emergence of swarm intelligence. The division of labor is also seen in nature in many social insects, such as the different species of leaf-cutter ants performing different tasks or the different groups of bees (e.g., the employed, onlooker and scout bees in the ABC algorithm [25]) performing differentiated foraging tasks.

2.10.4 Examples of Swarm Intelligence Algorithms

Over the last few years, the field of swarm intelligence has been very active, producing several algorithms that mimic the intelligent swarm behaviors found in the natural world. From these algorithms, we briefly present only a few in the following paragraphs.

Ant colony optimization: Ant colony optimization (ACO) [33] is based onthe intelligent ant swarm behavior to find an (or, a set of) optimal route(s) from their nest to some distant food source(s).ACO-based algorithms are suitable for the optimization problems that need to find routes from a source (or, initial state) to a destination (or, goalstate). In ACO, artificially simulated ant agents search across a parameter space that representsthe search space ofall possible solutions to find some optimal solution. To do so, the artificial ants put pheromones, like natural ants, along their way from their nest to the resources. Their search behavior is also randomly affected by the pheromones laid down by other ants. Better trails have smaller lengths, so they require less time to travel and hence have higher density of pheromones, which in turn attracts more ants to deposit pheromones along them. Gradually an optimal or nearoptimal trail is found by such distributed, self-organized behavior and interactions among the simulated ants.

Artificial bee colony algorithm: The Artificial bee colony(ABC)algorithmis a recently introduced [25] population based meta-heuristic algorithm that mimics the intelligent food

foraging behavior of the honey bees found in nature. There exist three categories of honey bees-the employed, onlooker and scout bees. The ABC algorithm uses the same three groups of bee agents for differentiated degree of explorations and exploitations of the search space. The employed and onlooker bees exploit the already found food positions (i.e., candidate solution), while the scout bees randomly explore the search space for newer food sources. Based on the interaction among the onlooker and employed bees (recruitment), self-organization of each bee agent and an intelligent division of work by the different groups of bees, the ABC algorithm shows very good performance on a wide range of optimization problems, including both discrete (e.g., [64]) and continuous (e.g., [24]) problems.

Artificial immune systems algorithm: The Artificial immune systems(AIS) algorithms are a class of swarm intelligence algorithms that are based on the principles and processes found in the immune systems of the vertebrate and mammalian species. AIS mimics some of the characteristics of the immune system, such as learning and memory. Some examples of AIS based algorithms include clonal selection algorithms [10], Negative selection algorithm [15], Immune network algorithms [55] and dendritic cell algorithms [19].

Bacterial Foraging Optimization: m that is inspired by the food-seeking and reproductive behavior of common bacteria such as E. coli.A common bacteriumcan perform a number of operations, such as swim towards nutrient, tumble to change directions and reproduce into two identical bacteria. The action that a bacterium takes depends probabilistically on its surrounding environment and its neighboring bacteria. The BFO is a probabilistic and population based metaheuristic that model such bacteriabehavior across the parameter space to solve many search and optimization problems [46].

Bat algorithm: The Bat algorithm (BA) [61]has been motivated by the intelligent echolocation behavior of certain species of bats. BA employs a strategy that mimics the frequency-tuning based control of the loudness and the rate of emission of the sound wave pulses by the bats in order to balance between global explorations and local exploitations of the search space.

Cuckoo search: The Cuckoo Search (CS) [8] is a recent algorithm based on the brooding behavior of cuckoos. Some cuckoo species use other host birds for hatching their eggs and raising their offspring. In CS, such brooding behavior is hybridized with perturbation steps produced from Levy distribution.The more explorative Levy distribution makes the CS more robust than some other swarm intelligence based algorithms, such as PSO and ABC, as demonstrated in [8].

Differential search algorithm: The Differential Search algorithm (DSA) [7] is another recent SI-based algorithm, motivated by the self-organized migration behavior of some superorganisms. The performance of DSA is evaluated on the numerical optimization problems and compared with some other recent evolutionary and swarm intelligence based algorithms, such as ABC, JDE, JADE, SADE, EPSDE, GSA, PSO and CMA-ES [7].

Firefly algorithm: The Firefly algorithm (FA) [60] mimics the swarming and re-grouping behavior of fireflies. Each firefly has some flashing capability and the intensity of its light can make it more or less attractive to the other fireflies.Using its light intensity, a firefly can attract other fireflies to form a subgroup or sub-swarm where the other fireflies crowd around the locally best firefly (i.e., the firefly with maximum flash intensity within its neighborhood). This makes FA particularly suitable for the multimodal optimization problems.

Glowworm swarm optimization: The Glowworm swarm optimization (GSO) [29] is another swarm intelligence algorithm that models the self-organization and swarming behavior of the glowworms. Eachagent in GSO moves across a search space and carry a luminescence amount called luciferin. The fitness of the current search space location of each particular glowworm is represented by its luciferin amount. Each glowworm probabilistically selects a subset of its neighboring glowworms. Then it moves towards any of its neighbors thathasa higher lucifer in value than itself.Such selections and movements, based only on local information, enable the swarm of glowworms to partition into disjoint subgroups. Each subgroup gradually converges to a locally optimal point.The GSO algorithm is particularly suitable for simultaneous discovery of multiple optimal points of the multimodal functions [29]- [30].

Multi-swarm optimization: The Multi-swarm optimization (MSO) [22]is a novel variation of the particle swarm optimization (PSO) algorithm to more effectively deal with multimodal functions. MSO employs multiple sub-swarms to search through different regions of the search space. A specific diversification method makes decisions about where and when to initiate the sub-swarms. Employing multiple swarms ensure better search space explorations, which is especially suitable for multi-modal problems with many local optima.

Particle swarm optimization: The Particle swarm optimization(PSO) [45] algorithm is motivated by the social behavior of bird flocking and fish schooling. In PSO, each candidate solution is represented as a particle or point in the D-dimensional search space. During initialization, each particle is usually created at a random location of the D-dimensional space with a random velocity along a random direction. Each particle can also communicate

with its neighboring particles and possibly with the best-so-far found particle.Particles then fly through the solution space and often accelerated towards those particles within their neighborhood which have better fitness values. The velocity, acceleration and direction information of each particle may also be perturbed randomly and possibly using some key information, such as the best-so-far location seen by each particle or its neighbors or the global best particle found so far. There exist several variants of the PSO algorithm (e.g., [23, 32, 59, 63]) and they have been applied on wide and diverse range of optimization problems [45]. The above list is no way an exhaustive or comprehensive list of the existing swarm intelligen ce based algorithms. There exist many other SI based algorithms, such as the Gravitational search algorithm [49], Central force optimization [14], Intelligent water drops algorithm [51], Altruism algorithm [57], Magnetic optimization algorithm [54],Krill herd algorithm [16], River formation dynamics (RFD) [48], Self-propelled particles (SPP) [56],stochastic diffusion search (SDS) [43], Invasive weed optimization [35] and so on. Evolutionary algorithms are used in various fields like

- Robotics
- Automotive Designs
- Evolvable Hardware
- Engineering Design
- Optimized Telecommunication Routing
- Computer Gaming
- Encryption and Code breaking
- Computer Aided Molecular Design
- Finance and Investment Strategies And so on.

Chapter 3

Benchmark Functions

The test of reliability, efficiency and validation of optimization algorithms is frequently carried out by using a chosen set of common standard benchmarks or test functions from the literature. The number of test functions in most papers varied from a few to about two dozen. Ideally, the test functions used should be diverse and unbiased. However, there is no agreed set of test functions in the literature. Therefore, the major aim of this paper is to review and compile the most complete set of test functions that we can find from all the available literature so that they can be used for future validation and comparison of optimization algorithms.

3.1 Types of Benchmark Functions

For any new optimization, it is essential to validate its performance and compare with other existing algorithms over a good set of test functions. A common practice followed by many researches is to compare different algorithms on a large test set, especially when the test involves function optimization (Gordon 1993, Whitley 1996). However, it must be noted that effectiveness of one algorithm against others simply cannot be measured by the problems that it solves if the set of problems are too specialized and without diverse properties. Therefore, in order to evaluate an algorithm, one must identify the kind of problems where it performs better compared to others. This helps in characterizing the type of problems for which an algorithm is suitable. This is only possible if the test suite is large enough to include a wide variety of problems, such as unimodal, multimodal, regular, irregular, separable, non-separable and multi-dimensional problems. The benchmark functions can be classified into these four groups:

- Unimodal Function

- Basic Multimodal Function
- Expanded Function
- Composition function

3.1.1 Standard Benchmark Functions

Many test functions may be scattered in different textbooks, in individual research articles or at different web sites. Therefore, searching for a single source of test function with a wide variety of characteristics is a cumbersome and tedious task. The standard benchmark suite (Table 3.1) contains both unimodal(F1-F9) and multimodal(F10-F29), which has separable (e.g., F1, F3) and non-separable (e.g., F2, F4), high dimensional (F1-F18) and low dimensional (F19-F29) functions. A function is called multidimensional if it has multiple local optima. To optimize such a function, the search algorithm must process both exploitative and explorative characteristics so that it can explore the locally optimal points without being trapped anywhere and thus keep exploring further for better unvisited regions of the search space. Some of the multimodal functions can have hundreds of local optima, even when the dimensionality is just two or three. The number of local maxima is usually increases exponentially with the number of dimensions. This often makes the minimization of the high dimensional multimodal functions extremely difficult. Although the Rosenbrock function F7 is usually considered to be a unimodal function. As in [5, 44] there is some evidence (e.g., [62], [18]) that is contains several minima in high dimensional instances. The global minimum is situated inside a narrow and flat valley. Finding the valley is straightforward, but precisely pinpointing the global minimum in the almost flat valley is extremely difficult, because the flat region does not provide any useful gradient direction pointing towards the global minimum. They Ackley function F13 has an exponential term with cosine sum which issues numerous local minima surrounding one narrow basin for the global minimum. The Griewank function F14 has a product term implementing interdependence among all the variables, so any technique trying to optimize each variable separately is bound to failure for the function. The complexity of the Schwefel function F12 is due to having a strong second-best local minimum which is very far from the global minimum of the search space. The low dimensional functions F19-F29 have dimensionality D<= 10, and most of the contain only a small number of local minima. A brief summary on the standard benchmark suite is presented in Table 3.1 in the next page

Table 3.1: The standard benchmark suite functions for the evolution and comparison of each of our algorithms. Here, D: dimensionality of the function, S: search space, F_{min}: function value at global minimum

No.	Function name	S	F_{min}	Function
f_1	*Sphere*	$[-100,100]^D$	0	$\sum_{i=1}^{D} x_i^2$
f_2	*Schwefel*	$[-10,10]^D$	0	$\sum_{i=1}^{D}\|x\|+\prod_{i=1}^{D} x_i$
f_3	*Schwefel*	$[-10,100]^D$	0	$max_i\{\|x\|, 1 \le i \le D\}$
f_4	*Schwefel*	$[-100,100]^D$	0	$\sum_{i=1}^{D}(\sum_{j=1}^{i} x_j)^2$
f_5	*Powell*	$[-4,5]^D$	0	$\sum_{i=1}^{D/k}(x_{4i-3}+10x_{4i-2})^2+5(x_{4i-1}-x_{4i})^2+(x_{4i-2}-x_{4i-1})^4+10(x_{4i-3}-x_{4i})^4$
f_6	*Dixon price*	$[-10,10]^D$	0	$(x_1-1)^2+\sum_{i=2}^{D} i(2x_i^2-x_{i-1})^2$
f_7	*Rosenbrock*	$[-30,30]^D$	0	$\sum_{i=1}^{D-1}(100(x_{i+1}-x_i^2)^2+(x_i-1)^2)$
f_8	*Step*	$[-100,100]^D$	0	$\sum_{i=1}^{D}(\|x_i+0.5\|)^2$
f_9	*Quartic*	$[-1.28,1.28]^D$	0	$\sum_{i=1}^{D} ix_i^4+random(0,1)$
f_{10}	*Rastrigin*	$[-5.12,5.12]^D$	0	$\sum_{i=1}^{D}[x_i^2-10cos(2\Pi x_i)+10]$
f_{11}	*Non continuous Rastrigin*	$[-5.12,5.12]^D$	0	$\sum_{i=1}^{D}[y_1^2-10cos(2\pi y_i)+10]$ $where, y_1 = \begin{cases} x_i \|x_i\|<.5 \\ .5*round(2x_i) \|x_i\|\ge .5 \end{cases}$
f_{12}	*Schwefel*	$[-500,500]^D$	12569.5	$-\sum_{i=1}^{D} x_i sin(\sqrt{\|x_i\|})$
f_{13}	*Ackley*	$[-32,32]^D$	0	$-20exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2})$ $-exp(\frac{1}{D}\sum_{i=1}^{D} cos(2\pi x_i))$ $+20+e$
f_{14}	*Griewank*	$[-600,600]^D$	0	$\frac{1}{4000}\sum_{i=1}^{D} x_i^2-\prod_{i=1}^{D} cos(\frac{x_I}{\sqrt{i}})$
f_{15}	*Alpine*	$[-10,10]^D$	0	$\sum_{i=1}^{D}\|x_i sin(x_i)+0.1x_i\|$

Table 3.1 continued from previous page

f_{16}	*Weierstrass*	$[-0.5, 0.5]^D$	0	$\sum_{i=1}^{D}(\sum_{k=0}^{k_{max}}[a^k cos(2\pi b^k (x_i + 0.5))])D\sum_{k=0}^{k_{max}} [a^k cos(2\pi b^k 0.5)]$; $a = 0.5, b = 3, k_{max} = 20$
f_{17}	*Penalized*	$[-50, 50]^D$	0	$\frac{\pi}{D}[10sin^2(\pi y_1) + (y_D - 1)^2] + \sum_{i=1}^{D} u(x_i, 10, 100, 4) + \frac{\pi}{D} \sum_{i=1}^{D-1}(y_i - 1)^2(1 + 10sin^2 (\pi y_{i+1})), y_i = 1 + \frac{1}{4}(1 + x_i)$
f_{18}	*Penalized2*	$[-50, 50]^D$	0	$0.1[sin^2(\pi x_1)+ (x_D 1)^2(1 + sin^2(2\pi x_D))]+ \sum_{i=1}^{D} u(x_i, 5, 100, 4) + 0.1\sum_{i=1}^{D-1} (x_i - 1)^2(isin^2 3\pi x_{i+1})$
f_{19}	*Foxholes*	$[-65.5, 65.5]^D$	1	$(\frac{1}{500} + \sum_{j=1}^{25}(\frac{1.0}{j}+ \sum_{i=1}^{2}(x_i - a_{ij})^6))^{-1}$
f_{20}	*Kowalik*	$[-5, 5]^D$	0.000037	$\sum_{i=1}^{11}(a_i - x_1(b_i^2 + b_i x_2) /(b_i^2 + b_i x_3 + x_4))^2$
f_{21}	*Six Hump Camel Back*	$[-5, 5]^D$	-1.032	$4x_1^2 - 2.1x_1^4 + (\frac{1}{3})x_1^6 + x_1 x_2 - 4x_2^2 + 4x_2 4$
f_{22}	*Branin*	$[-5, 10]x[0, 15]$	0.398	$(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6)^2 + 10(1 - \frac{1}{8\pi})cos(x_1) + 10$
f_{23}	*Hartman3*	$[0, 1]^D$	-3.86	$-\sum_{i=1}^{4} c_i exp[-\sum_{j=1}^{D} a_{ij}(x_j - p_{ij})^2]$; $D = 3, 6$

Table 3.1 continued from previous page

f_{24}	*Hartman6*	$[0,1]^D$	-3.32	$-\sum_{i=1}^{4} c_i exp[-\sum_{j=1}^{D} a_{ij}(x_j - p_{ij})^2]; D = 3,6$
f_{25}	*Shekel5*	$[0,10]^D$	-10.6	$-\sum_{i=1}^{m}[(x-a_i)(x-a_i)^T + c_i]^{-1}; m = 5,7,10$
f_{26}	*Shekel7*	$[0,10]^D$	-10.4	$-\sum_{i=1}^{m}[(x-a_i)(x-a_i)^T + c_i]^{-1}; m = 5,7,10$
f_{27}	*Shekel10*	$[0,10]^D$	-10.5	$-\sum_{i=1}^{m}[(x-a_i)(x-a_i)^T + c_i]^{-1}; m = 5,7,10$
f_{28}	*Fetchwe powell*	$[-\Pi,\Pi]$	0	$-\sum_{i=1}^{D}(A_i - B_i)^2$ $A_i = \sum_{j=1}^{D}(a_{ij}sin\alpha_j + b_{ij}cos\alpha_j)$ $B_i = \sum_{j=1}^{D}(a_{ij}sinx_j + b_{ij}cos_j)$
f_{29}	*Michalewicz*	$[0,\Pi]^D$	-9.66	$-\sum_{i=1}^{D} sin(x_i)(sin(ix_i^2/\pi))^{2m}; m = 10$
f_{30}	*Langerman*	$[0,10]^D$	-1.4	$-\sum_{i=1}^{10} c_i exp(-\frac{1}{\pi}\sum_{j=1}^{D}(x_j - a_{ij})^2)cos(\pi\sum_{j=1}^{D}(x_j - a_{ij})^2)$

3.2 Details of Benchmark Functions

3.2.1 Unimodal Functions

F1: Sphere Function: Sphere Function is an unconstrained, unimodal, additively separable function. It is defined by the following equation-

$$f(x) = \sum_{i=1}^{D} x_i^2 \tag{3.1}$$

Where, $X = [X_1, X_2, ..., X_D], I = 1, ..., DandX \in [100, 100]^D$
Global Minimum $f(X_1, X_2, ..., X_D) = f(0, 0, ..., 0) = 0$

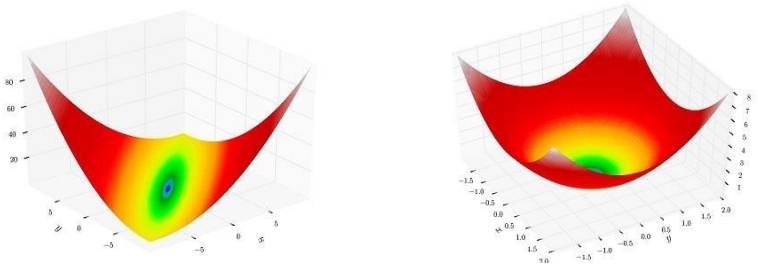


Figure 3.1: 3D plot of the Sphere Function

F2: Schwefel 2.22 Function: This is a unimodal, non-separable function. It is defined by the following equation:

$$f(x) = \sum_{i=1}^{D} |x| + \prod_{i=1}^{D} x_i \tag{3.2}$$

Where, $X = X \in [-10, 10]^D$

F3: Schwefel 2.21 Function: This is a unimodal and separable function. It is defined by the following equation:

$$f(x) = max_i\{|x|, 1 \leq i \leq D\} \tag{3.3}$$

Where, $X = X \in [-10, 10]^D$

F4: Schwefel 2.21 Function: This is a unimodal and separable function. It is defined by the following equation:

$$f(x) = \sum_{i=1}^{D} (\sum_{j=1}^{i} x_j)^2 \tag{3.4}$$

Where, $X = X \in [-100, 100]^D$

F5: Powell Function: This is a unimodal and separable function. This function need not be differentiable and no derivatives are taken. The function must be a real-valued function of a mixed number of real-valued inputs. It is defined by the following equation:

$$f(x) = \sum_{i=1}^{D/k} (x_{4i-3} + 10x_{4i-2})^2 + 5(x_{4i-1} - x_{4i})^2 + (x_{4i-2} - x_{4i-1})^4 + 10(x_{4i-3} - x_{4i})^4 \quad (3.5)$$

Where, $X = X \in [-4, 5]^D$

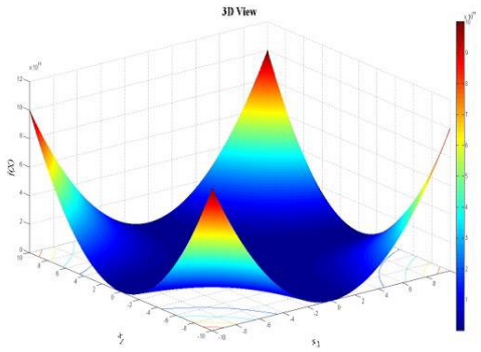


Figure 3.2: 3D plot of the Powell Function

F6: Dixon-Price Function: This is a unimodal and separable function. It is defined by the following equation:

$$f(x) = (x_1 - 1)^2 + \sum_{i=2}^{D} i(2x_i^2 - x_{i-1})^2 \quad (3.6)$$

Where, $X = X \in [-10, 10]^D$

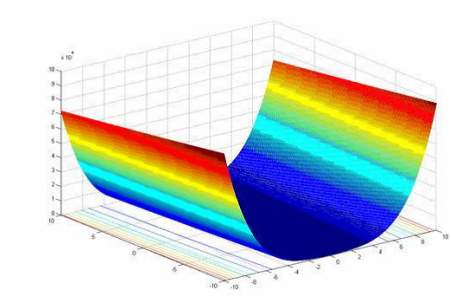


Figure 3.3: 3D plot of the Dixon-Price Function

F7: Rosenbrock Function: This is a unimodal and separable function. It is defined by the following equation

$$f(x) = \sum_{i=1}^{D-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2) \quad (3.7)$$

Where, $X = X \in [-30, 30]^D$

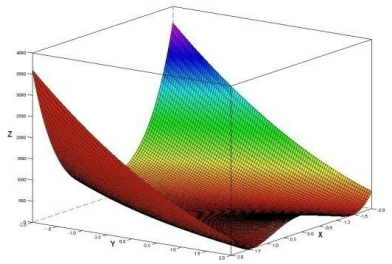


Figure 3.4: 3D plot of the Rosenbrock Function

F8: Step Function: This is a unimodal and separable function. It is defined by the following equation:

$$f(x) = \sum_{i=1}^{D} (|x_i + 0.5|)^2 \tag{3.8}$$

Where, $X = X \in [-100, 100]^D$

F9: Quartic Function: This is a unimodal and separable function. Sometimes it called a biquadratic function but the latter term occasionally also refers to a quadratic function of a square. It is defined by the following derivative equation:

$$f(x) = \sum_{i=1}^{D} i x_i^4 + random(0, 1) \tag{3.9}$$

Where, $X = X \in [-1.28, 1.28]^D$

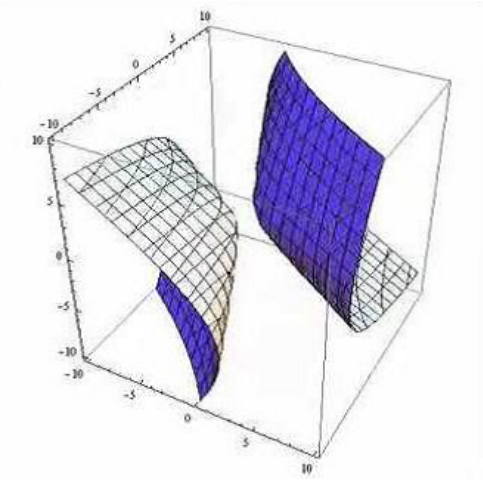


Figure 3.5: 3D plot of the Quartic Function

3.2.2 Multimodal Functions High Dimensional:

F10: Rastrigin Function: This is an multimodal and separable function. It is non-convex function used as a performance test problem for optimization algorithms. It is a typical example of non-linear multimodal function. This function is a fairly difficult problem due to its large search space and its large number of local minima. It is defined by the following equation:

$$f(x) = \sum_{i=1}^{D} [x_i^2 - 10cos(2\Pi x_i) + 10] \tag{3.10}$$

Where, $X = X \in [-5.12, 5.12]^D$

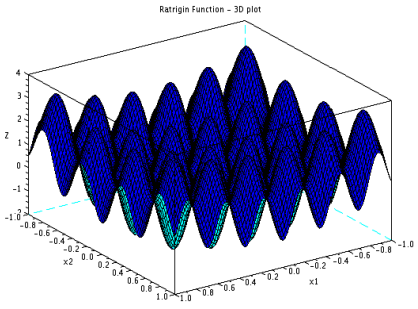


Figure 3.6: 3D plot of the Rastrigin Function

F11: Non-continuous Rastrigin Function: This is a multimodal and separable function. It is defined by the following equation:

$$f(x) = \sum_{i=1}^{D} [y_1^2 - 10cos(2\pi y_i) + 10] \tag{3.11}$$

Where,$X = X \in [-5.12, 5.12]^D$ and $y_1 = \begin{cases} x_i | x_i | < 0.5 \\ 0.5 * round(2x_i) | x_i | \geq 0.5 \end{cases}$

F12: Schwefel Function: This is a multimodal and separable function. It is defined by the following equation:

$$f(x) = -\sum_{i=1}^{D} x_i sin(\sqrt{|x_i|}) \tag{3.12}$$

Where, $X = X \in [-500, 500]^D$

F13: Ackley Function: This is a multimodal and non-separable function. It is defined by the following equation:

$$f(x) = -20exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}) - exp(\frac{1}{D}\sum_{i=1}^{D} cos(2\pi x_i)) + 20 + e \tag{3.13}$$

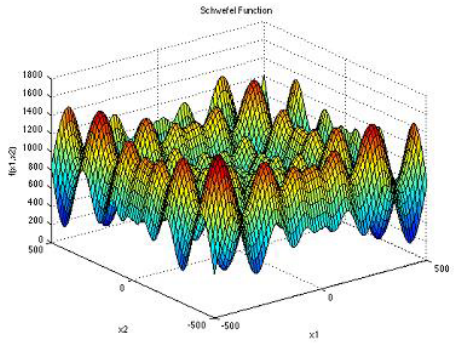


Figure 3.7: 3D plot of the Schwefel Function

Where, $X = X \in [-32, 32]^D$

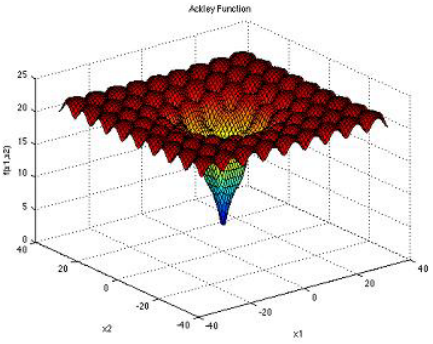


Figure 3.8: 3D plot of the Ackley Function

F14: Griewank Function: This is a multimodal and non-separable function. Several local minima in this function. The global minima: $F(x) = 0$. It is defined by the following equation:

$$f(x) = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} cos(\frac{x_I}{\sqrt{i}}) \tag{3.14}$$

Where, $X = X \in [-600, 600]^D$

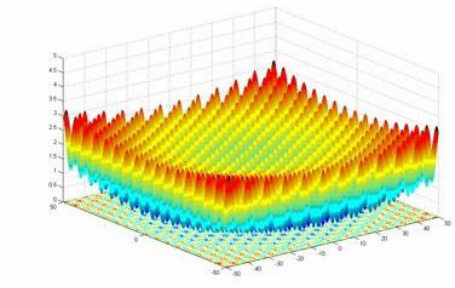


Figure 3.9: 3D plot of the Griewank Function

F15: Alpine Function: This is a multimodal and separable function. It is defined by the following equation:

$$f(x) = \sum_{i=1}^{D} |x_i sin(x_i) + 0.1x_i \tag{3.15}$$

Where, $X = X \in [-10, 10]^D$

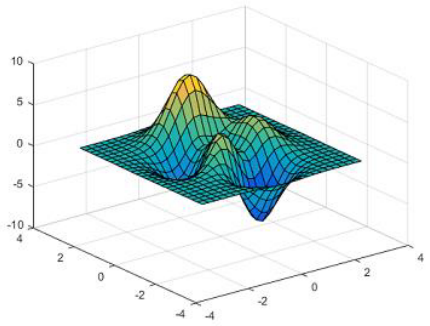


Figure 3.10: 3D plot of the Alpine Function

F16: Weierstrass Function: This is a multimodal and separable function. This function has the particularity of being continuous in all the domain but only differentiable in a particular set of points. It is defined by the following equation:

$$f(x) = \sum_{i=1}^{D} (\sum_{k=0}^{k_{max}} [a^k cos(2\pi b^k (x_i + 0.5))]) D \sum_{k=0}^{k_{max}} [a^k cos(2\pi b^k 0.5)]; \tag{3.16}$$

Where, $X = X \in [-0.5, 0.5]^D \; and \; a = 0.5, b = 3, k_{max} = 20$

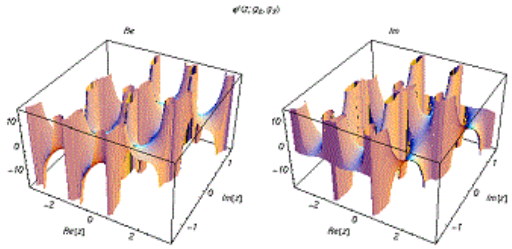


Figure 3.11: 3D plot of the Weierstrass Function

F17: Penalized Function: This is a multimodal and non-separable function. It is defined by the following equation:

$$\begin{aligned} f(x) = \frac{\pi}{D}[10sin^2(\pi y_1) + (y_D - 1)^2] + \sum_{i=1}^{D} u(x_i, 10, 100, 4) + \frac{\pi}{D} \\ \sum_{i=1}^{D-1} (y_i - 1)^2 (1 + 10sin^2(\pi y_{i+1})), \\ y_i = 1 + \frac{1}{4}(1 + x_i) \end{aligned} \tag{3.17}$$

Where, $X = X \in [-50, 50]^D$

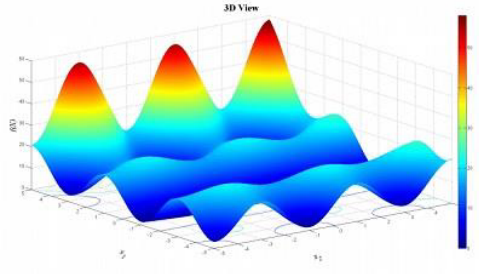


Figure 3.12: 3D plot of the Penalized Function

F18: Penalized2 Function: This is a multimodal and non-separable function. It is defined by the following equation:

$$f(x) = 0.1[sin^2(\pi x_1) + (x_D 1)^2(1 + sin^2(2\pi x_D))] + \sum_{i=1}^{D} u(x_i, 5, 100, 4) + 0.1 \sum_{i=1}^{D-1} (x_i - 1)^2 (isin^2 3\pi x_{i+1}) \tag{3.18}$$

Where, $X = X \in [-50, 50]^D$

3.2.3 Multimodal Functions Low Dimensional:

F19: Foxholes Function: This is a multimodal and separable function. It is defined by the following equation:

$$f(x) = [\frac{1}{500} + \sum_{j=1}^{25} (\frac{1.0}{j} + \sum_{i=1}^{2} (x_i - a_{ij})^6)]^{-1} \tag{3.19}$$

Where, $X = X \in [-65.536, 65.536]^D$

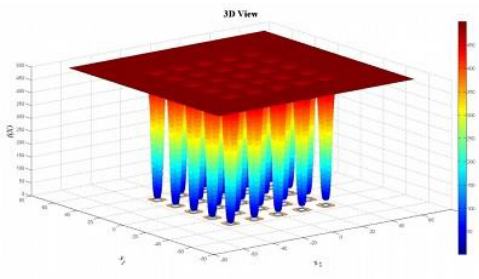


Figure 3.13: 3D plot of the Foxholes Function

F20: Kowalik's Function: This is a multimodal and separable function. It is defined by

Table 3.2: Coefficients for Kowalik's Function

I	a_i	b_i^{-1}
1	0.1957	0.25
2	0.1947	0.50
3	0.1735	1
4	0.1600	2
5	0.0844	4
6	0.0627	6
7	0.0456	8
8	0.0342	10
9	0.0323	12
10	0.0235	14
11	0.0246	16

the following equation:

$$f(x) = \sum_{i=1}^{11} (a_i - x_1(b_i^2 + b_i x_2)/(b_i^2 + b_i x_3 + x_4))^2 \tag{3.20}$$

Where, $X = X \in [-5,5]^D$

F21: Six Hump Camel Back Function: This is a multimodal and non-separable function and optimization test function. Within the bounded region it owns six local minima; two of them are global ones. The global minima: $X = (0.00898, -0.7126), (-0.0898, 0.7126)$ and $F(X) = 0$.It is defined by the following equation:

$$f(x) = 4x_1^2 - 2.1x_1^4 + (\frac{1}{3})x_1^6 + x_1x_2 - 4x_2^2 + 4x_24 \tag{3.21}$$

Where, $X = X \in [-5,5]^D$

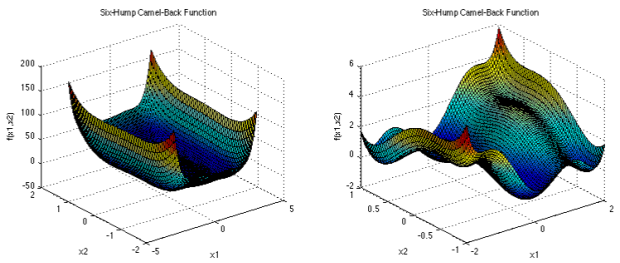


Figure 3.14: 3D plot of Six Hump Camel Back Function

F22: Branin Function: This is a multimodal and non-separable function. The global minima: $X = (-\pi, 12.275), (\pi, 12.275), (9.42478, 2.475)$, $F(X) = 0.0397887$. It is defined by the following equation:

$$f(x) = (x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6)^2 + 10(1 - \frac{1}{8\pi})cos(x_1) + 10 \tag{3.22}$$

Where, $X = X \in [-5, 10]x[0, 15]$

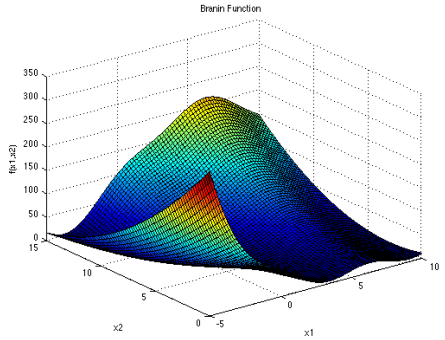


Figure 3.15: 3D plot of Branin Function

F23: Hartman3 Function: This is a multimodal and non-separable function. It is defined by following equation:

$$f(x) = -\sum_{i=1}^{4} c_i exp[-\sum_{j=1}^{D} a_{ij}(x_j - p_{ij})^2] \tag{3.23}$$

Where, $X = X \in [-5, 5]^D \; and \; D = 3$

Table 3.3: Coefficient for Hartman3 function

I	**j=1,2,3**				**j=1,2,3**		
1	3	10	30	1	0.3689	0.1170	0.2673
2	0.1	10	35	1.2	0.4699	0.4387	0.7470
3	3	10	30	3	0.1091	0.8732	0.5547
4	0.1	10	35	3.2	0.038150	0.5743	0.8828

F24: Hartman6 Function: This is a multimodal and separable function. It is defined by following equation:

$$f(x) = -\sum_{i=1}^{4} c_i exp[-\sum_{j=1}^{D} a_{ij}(x_j - p_{ij})^2] \tag{3.24}$$

Where, $X = X \in [-5, 5]^D \; and \; D = 6$

Table 3.4: Coefficient for Hartman6 function

I	j=1,2,3....6						j=1,2,3				
1	3	3	3.5	1.7	8	1	0.1312	0.1696	0.5569	0.8283	0.5886
2	0.05	10	0.1	8	14	1.2	0.2329	0.4135	0.8307	0.1004	0.9991
3	3	3.5	10	17	8	3	0.1415	0.1415	0.3522	0.3047	0.6650
4	17	8	0.05	0.1	14	3.2	0.4047	0.8828	0.8732	0.1091	0.0381

F25: Shekel5 Function: This is a multimodal and separable function. It is defined by following equation:

$$f(x) = -\sum_{i=1}^{m}[(x-a_i)(x-a_i)^T + c_i]^{-1} \tag{3.25}$$

Where, $X = X \in [0,10]^D \; and \; m = 5$

F26: Shekel7 Function: This is a multimodal and separable function. It is defined by following equation:

$$f(x) = -\sum_{i=1}^{m}[(x-a_i)(x-a_i)^T + c_i]^{-1} \tag{3.26}$$

Where, $X = X \in [0,10]^D \; and \; m = 7$

F27: Shekel10 Function: This is a multimodal and separable function. It is defined by following equation:

$$f(x) = -\sum_{i=1}^{m}[(x-a_i)(x-a_i)^T + c_i]^{-1} \tag{3.27}$$

Where, $X = X \in [0,10]^D \; and \; m = 10$

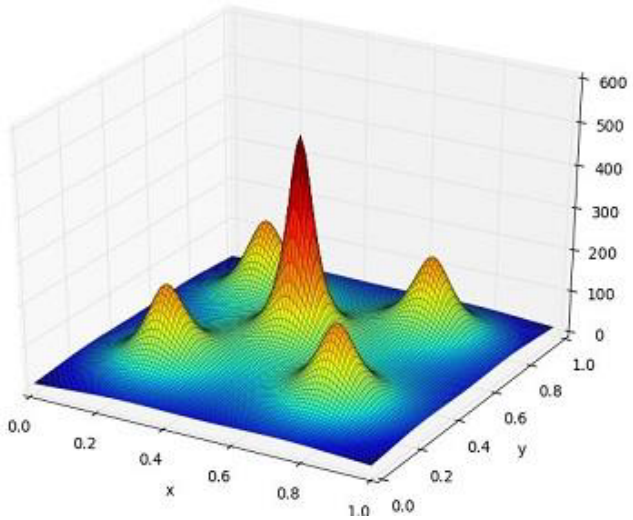


Figure 3.16: 3D plot of Shekel Function

F28: Fetchwe-powell Function: This is a multimodal and separable function. It is defined by following equation:

$$f(x) = -\sum_{i=1}^{D}(A_i - B_i)^2, A_i = \sum_{j=1}^{D}(a_{ij}sin\alpha_j + b_{ij}cos\alpha_j), B_i = \sum_{j=1}^{D}(a_{ij}sinx_j + b_{ij}cos_j) \quad (3.28)$$

Where, $X = X \in [\pi, \pi]$

F29: Michalewicz Function: This is multimodal function. It is defined by following equation:

$$f(x) = -\sum_{i=1}^{D} sin(x_i)(sin(ix_i^2/\pi))^{2m} \quad (3.29)$$

Where, $X = X \in [0, \Pi]^D \; and \; m = 10$

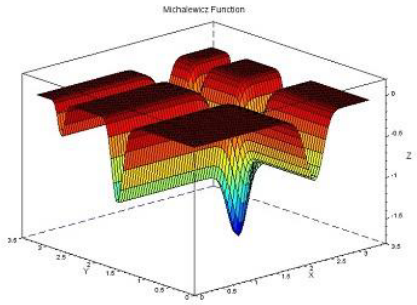


Figure 3.17: 3D plot of Michalewicz Function

F30: Langerman Function: This is multimodal function. It is defined by following equation:

$$f(x) = -\sum_{i=1}^{10} c_i exp(-\frac{1}{\pi}\sum_{j=1}^{D}(x_j - a_{ij})^2)cos(\pi)\sum_{j=1}^{D}(x_j - a_{ij})^2) \quad (3.30)$$

Where, $X = X \in [0, 10]^D$

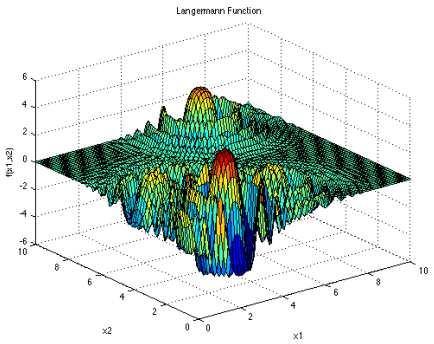


Figure 3.18: 3D plot of Langerman Function

Chapter 4

Differential Evolution

DE is a population-based algorithm, and a reliable and versatile function optimizer, which evolves a population of NP D-dimensional individuals towards the global optimum by enhancing the differences of the individuals. In brief, after initialization, DE repeats mutation, crossover, and selection operations to produce a trail vector and select one of those vectors with the best fitness value for each vector until satisfying some specific termination criteria. In this chapter, we will discuss about the essential strategies and concepts related to DE as well as the challenging research issues, such as avoiding premature convergence, preserving population diversity and balancing explorations with exploitations during the optimization process. In the following section, we will outline the classical DE and its different versions in sufficient details. Its remarkable performance as a global optimization algorithm on continuous numerical minimization problems has been extensively explored. We will implement some of the variances of DE such as DE with strategy 1 that is DE/best/1/exp , DE with strategy 2 that is DE/rand/1/exp , DE with strategy 6 that is DE/best/1/bin and strategy 7 DE/rand/1/bin. We will also try to represent a comparative analysis of DE (strategy 1), DE (strategy 2), DE (strategy 6) and DE (strategy 7).

4.1 Background of DE

During the last few decades, several stochastic, heuristic and meta-heuristic algorithms have emerged which showed significant success to deal with many complex scientific, mathematical and engineering problems, such as continuous optimization, combinatorial optimization, multi-objective optimization, industrial process control, engineering design, design of digital IIR filters, PID controllers, machine learning and so on. In the optimization process of a difficult task, the method of first choice will usually be a problem specific heuristics. These techniques using expert knowledge achieve a superior performance. If problem specific

technique is not applicable due to unknown system parameters, the multiple local minima, or non-differentiability, evolutionary algorithms (EAs) have the potential to overcome these limitations. EAs are a class of direct search algorithms. A conventional direct search method uses a strategy that generates variations of the design parameter vectors. Once a variation is generated, the new parameter vector is accepted or not. The new parameter vector is accepted in the case it reduces the objective function value. This method is usually named the greedy search. The greedy search converges fast but can be trapped by local minima. This disadvantage can be eliminated by running several vectors simultaneously. This is the main idea of differential evolution (DE) algorithm [27]. The Differential Evolution (DE) is a recently introduced meta-heuristic algorithm which belongs to the family of Evolutionary Algorithms (EAs) and is successfully employed to many complex search and optimization problems. Like most other population based meta-heuristic algorithms, the DE is usually resilient and robust against premature convergence and fitness stagnation. The reason is that the population of individuals (i.e., candidate solutions) can usually preserve sufficient amount of diversity and explorative search capability, which is necessary to continue the search space explorations around the locally optimal points without being trapped anywhere around them. The most popular EA is genetic algorithm. Although many genetic algorithm versions have been developed, they are still time consuming [62]. In 1995, in order to overcome this disadvantage, Price and Storn proposed a new floating point encoded evolutionary algorithm for global optimization and named it differential evolution (DE) owing to a special kind of differential operator, which they invoked to create new offspring from parent chromosomes instead of classical crossover or mutation [9].
It has been applied to several engineering problems in different areas. The main difference between the genetic algorithm and DE is the mutation scheme that makes DE self adaptive and the selection process. In DE, all solutions have the same chance of being selected as parents without dependence of their fitness value. DE employs a greedy selection process: The better one of new solution and its parent wins the competition providing significant advantage of converging performance over genetic algorithms. DE algorithm is a stochastic optimization method minimizing an objective function that can model the problem's objectives while incorporating constraints. The algorithm mainly has three advantages; finding the true global minimum regardless of the initial parameter values, fast convergence, and using a few control parameters. Being simple, fast, easy to use, very easily adaptable for integer and discrete optimization, quite effective in nonlinear constraint optimization including penalty functions and useful for optimizing multi-modal search spaces are the other important features of DE [27].

4.2 The DE Algorithm

We will represent subsequent generations in DE by discrete time steps like $t = 0, 1, 2, ..., t, t+1$ etc. Since the vectors are likely to be changed over different generations we may adopt the following notation for representing the i^{th} vector of the population at the current generation (i.e., at time $t = t$) as follows:

$$\vec{X}_i(t) = (x_{i,1}(t), x_{i,2}(t), x_{i,3}(t), ..., x_{i,D}(t)) \tag{4.1}$$

4.2.1 Procedure of DE

Initialization: First of all, uniformly randomize NP individuals within a D-dimensional real parameter search space. And the initial population should cover the entire search space constrained by the prescribed minimum and maximum bounds:

$$\vec{X}_{min} = (x_{min,1}, x_{min,2}, ..., (x_{min,D}) \tag{4.2}$$

$$\vec{X}_{max} = (x_{max,1}, x_{max,2}, ..., (x_{max,D}) \tag{4.3}$$

Hence, the initialization of the j^th element in the ith is taken as follows:

$$X_{i,j}(0) = x_{min,j} + rand(0,1) * (x_{max,j} - x_{min,j}), j = 1, 2, ..., D \tag{4.4}$$

Where $rand(0,1)$ represents a uniformly distributed random variable within the $range[0,1]$ and it is instantiated independently for each component of the i^{th} vector.

Mutation Operation: In the existing literature on DE, mutate vector $\vec{V}_i(t)$, called donor vector, is obtained through the differential mutation operation with respect to each individual $\vec{X}_i(t)$, is known as target vector, in the current population. For each target vector from the current population, the donor vector is created via certain mutation strategy. Several mutation strategies have been proposed. Here, we list one of the most popular and simplest forms of DE-mutation for the j^{th} component of each vector as follows:

$$v_{i,j}(t+1) = x_{r_1,j}(t) + F * (x_{r_2,j}(t) - x_{r_3,j}(t) \tag{4.5}$$

The indices r_1, r_2 and r_3 are mutually exclusive integers randomly generated within the $range[1, NP]$, which are different from the base vector index. These indices are randomly generated for each mutant vector. Now, the difference of any two of these three vectors is scaled by a mutation weighting factor F which typically lies in the interval $[0.4, 1]$ in the existing DE literature, and the scaled difference is added to the third one to obtain a donor

vector V.

Crossover Operation: After the mutation operation, according to the target vector $\vec{X}_i(t)$ and its corresponding donor vector $\vec{V}_i(t)$ a trail vector $\vec{U}_i(t)$ is produced by crossover operation. In traditional version, DE applies the binary defined crossover as follows:

$$u_{i,j}(t) = \begin{cases} v_{i.j}(t), & \text{if } rand_{i,j}[0,1] \leq C_r or j = j_{rand} \\ x_{i,j}(t), & \text{otherwise.} \end{cases} \tag{4.6}$$

Where C_r is a crossover rate within the $range[0,1]$, defined by user as a constant, which controls the probability of parameter values employed from the donor vector. j_{rand} is a randomly chosen integer within the $range[1,NP]$ which is introduced to ensure that the trial vector contains at least one parameter from donor vector.

Selection Operation: In classic DE algorithm, a greedy selection is adopted. The fitness of every trail vectors is evaluated and compared with that of its corresponding target vector in the current population. For minimization problem, if the fitness value of trial vector is not more than that of target vector, the target vector will be replaced by the trial vector in the population of the next generation. Otherwise, the target vector will be maintained in the population of the next generation. The selection operation is expressed as follows:

$$\vec{X}_i(t+1) = \begin{cases} \vec{U}_i(t), if f(\vec{U}_i(t)) \leq f(\vec{X}_i(t)), \\ \vec{X}_i(t), otherwise. \end{cases} \tag{4.7}$$

4.2.2 Strategies of DE

Actually, it is the process of mutation, which demarcates one DE strategy from another. In the former section, we have illustrated the basic steps of a simple DE. The mutation scheme in (5.3) uses a randomly selected vector $\vec{X}_{r_1}$and only one weighted difference vector $F * (\vec{X}_{r_2} - \vec{X}_{r_3})$ is used to perturb it. Hence, in literature the particular mutation scheme is referred to as DE/rand/1.

Differential Evolution has a specialized nomenclature that describes the adopted configuration. This takes the form of DE/x/y/z, where

- x represents the solution to be perturbed (such a random or best).
- The y signifies the number of difference vectors used in the perturbation of x, where a difference vectors is the difference between two randomly selected although distinct members of the population.

- Finally, z signifies the recombination operator performed such as bin for binomial and exp for exponential.

There is a nomenclature scheme developed to reference the different variants of Differential Evolution.

Table 4.1: DE variants used in experiments j_r is a random integer number generated between $[0.n]$,where n is the number of variables of the problem. $U_j(0,1)$ is a real number generated at random between 0 and 1. Both numbers are generated using a uniform distribution.

rand/p/bin	$u_{i,j} = \begin{cases} x_{r_3,j} + F * \sum_{k=1}^{p}(x_{r_1{}^p,j} - x_{r_2{}^p,j}), & \text{if } U_j(0,1) < C_r \text{ or } j \geq j_r \\ x_{i,j} & \text{otherwise} \end{cases}$
rand/p/exp	$u_{i,j} = \begin{cases} x_{r_3,j} + F * \sum_{k=1}^{p}(x_{r_1{}^p,j} - x_{r_2{}^p,j}), & \text{if } U_j(0,1) < C_r \text{ or } j \geq j_r \\ x_{i,j} & \text{otherwise} \end{cases}$
best/p/bin	$u_{i,j} = \begin{cases} x_{r_{best},j} + F * \sum_{k=1}^{p}(x_{r_1{}^p,j} - x_{r_2{}^p,j}), & \text{if } U_j(0,1) < C_r \text{ or } j \geq j_r \\ x_{i,j} & \text{otherwise} \end{cases}$
best/p/exp	$u_{i,j} = \begin{cases} x_{r_{best},j} + F * \sum_{k=1}^{p}(x_{r_1{}^p,j} - x_{r_2{}^p,j}), & \text{if } U_j(0,1) < C_r \text{ or } j \geq j_r \\ x_{i,j} & \text{otherwise} \end{cases}$
current-to-rand/p	$\vec{u_i} = \vec{x_i} + K * (\vec{x_3} - \vec{x_i}) + F * \sum_{k=1}^{p}(x_{r_1{}^p,j} - x_{r_2{}^p,j})$
current-to-best/p	$\vec{u_i} = \vec{x_i} + K * (\vec{x_{best}} - \vec{x_i}) + F * \sum_{k=1}^{p}(x_{r_1{}^p,j} - x_{r_2{}^p,j})$
current-to-rand/p/bin	$u_{i,} = \begin{cases} x_{r_3,j} + F * \sum_{k=1}^{p}(x_{r_1{}^p,j} - x_{r_2{}^p,j}) + K * (\vec{x_{r_3}} - \vec{x_i}) \text{ if } U_j(0,1) < C_r \text{ or } j \geq j_r \\ x_{i,j} \end{cases}$ otherwise
rand/2/dir	$\vec{v_i} = \vec{v_i} + (F/2) * (\vec{v_1} - \vec{v_2} + \vec{v_3} - \vec{v_4})$ *where* $f(\vec{v_1}) < f(\vec{v_2})$ *and* $f(\vec{v_3}) < f(\vec{v_4})$

In 2001 Storn and Price suggested total ten different working strategies of DE and some guidelines in applying these strategies to any given problem. These strategies were derived from the five different DE mutation strategies outlined above. Each mutation strategy was combined with either the “exponential" type crossover or the “binomial" type crossover. This yielded $5 * 2 = 10$ DE strategies, which are listed below.

1. DE/best/1/exp
2. DE/rand/1/exp
3. DE/rand-to-best/1/exp
4. DE/best/2/exp
5. DE/rand/2/exp
6. DE/best/1/bin
7. DE/rand/1/bin
8. DE/rand-to-best/1/bin
9. DE/best/2/bin
10. DE/rand/2/bin

4.2.3 Flowchart and Algorithm of DE

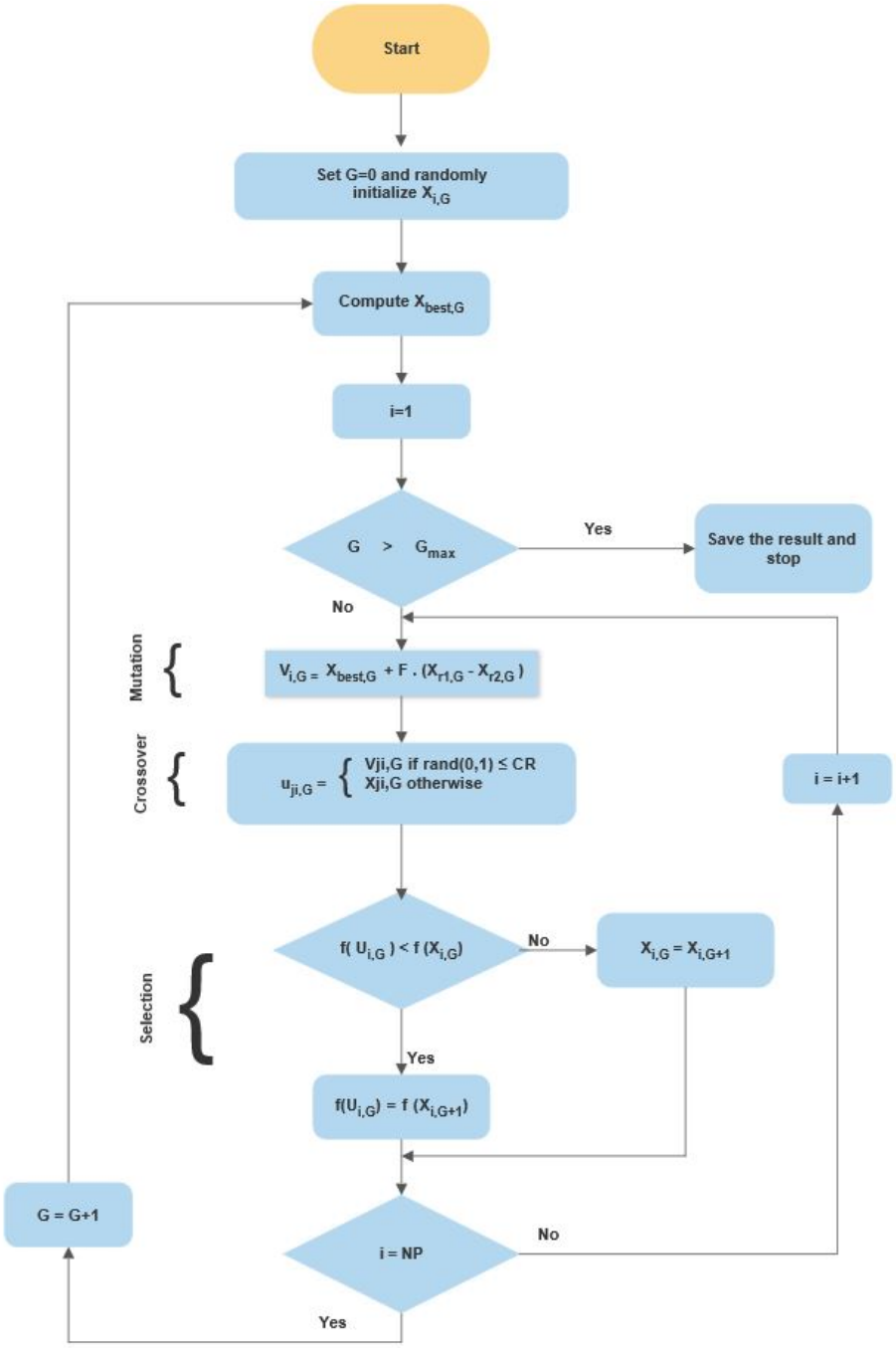


Figure 4.1: Block Diagram of typical DE

Algorithm 5: A typical DE

Input: Randomly initialized position and velocity of the particles:$\vec{x}_i(0)$
Output: Position of the approximation global optima $\vec{X}^*$

```
Begin:
Initialize population
Evaluate fitness
for i = 0 to max-iteration do
    Create Difference-Offspring
    Evaluate fitness
    if an offspring is better than its parent then
        replace the parent by offspring in the next generation
    end
end
End
```

4.3 Strategies for Experiment

Each of the five mutation strategy was combined either with the 'exponential' type crossover or the 'binomial' type crossover which gives 10 DE strategies. Among those we are going to use three of those for our experiment. They are as follows:

1. DE/best/1/exp
2. DE/rand/1/exp
3. DE/best/1/bin
4. DE/rand/1/bin

4.4 Functions Used for the Experiment

Among the 30 bench-mark functions, we have used 10 functions. 4 of them are unimodal and 6 of them are multimodal functions. Their name, domain and equations are given in the table below.

Table 4.2: Functions used for Experiment

No.	Function name	S	F_{min}	Function
f_1	*Sphere*	$[-100,100]^D$	0	$\sum_{i=1}^{D} x_i^2$
f_2	*Schwefel 2.2*	$[-10,10]^D$	0	$\sum_{i=1}^{D} \|x\| + \prod_{i=1}^{D} x_i$
f_3	*Schwefel 2.1*	$[-10,100]^D$	0	$max_i\{\|x\|, 1 \le i \le D\}$
f_7	*Rosenbrock*	$[-30,30]^D$	0	$\sum_{i=1}^{D-1}(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$
f_{10}	*Rastrigin*	$[-5.1,5.1]^D$	0	$\sum_{i=1}^{D}[x_i^2 - 10cos(2\Pi x_i) + 10]$
f_{11}	*Noncontinuous Rastrigin*	$[-5.1,5.1]^D$	0	$\sum_{i=1}^{D}[y_1^2 - 10cos(2\pi y_i) + 10]$ *where,* $y_1 = \begin{cases} x_i \|x_i\| < 0.5 \\ 0.5 * round(2x_i) \|x_i\| \ge 0.5 \end{cases}$
f_{14}	*Griewank*	$[-600,600]^D$	0	$\frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} cos(\frac{x_I}{\sqrt{i}})$
f_{15}	*Alpine*	$[-10,10]^D$	0	$\sum_{i=1}^{D} \|x_i sin(x_i) + 0.1x_i\|$
f_{20}	*Kowalik*	$[-5,5]D$		$\sum_{i=1}^{11}(a_i - x_1(b_i^2 + b_i x_2)/(b_i^2 + b_i x_3 + x_4))^2$
f_{29}	*Michalewicz*	$[0,10]^D$	-1.4	$-\sum_{i=1}^{D} sin(x_i)(sin(ix_i^2/\pi))^{2m}$

4.5 Setup for the Experiment

In all the experiments, the same self-adaptive method, the same population size, the same tournament size for selection, and same initial population is used for DE strategies.

4.6 Experimental Results

4.6.1 *Sphere function*(f_1)

Table 4.3: Optimization of *Sphere function*(f_1) by DE variants with different number of generations

Generation	DE/best/1/exp	DE/rand/1/exp	DE/best/1/bin	DE/rand/1/bin
1600	6.4059060e-009	0.021949	4.239e-008	0.05398
1100	9.7963	0.000901	0.000901	13.65
900	0.02453	115.578	0.0893	154.167
700	1.65269	1295.732	5.226	1651.456
500	81.254	14048.2793	235.526	16851.042
300	7260.458	104198.265	28410.96289	137911.5469
200	79974.96875	325852.1563	176992.0313	540198.75

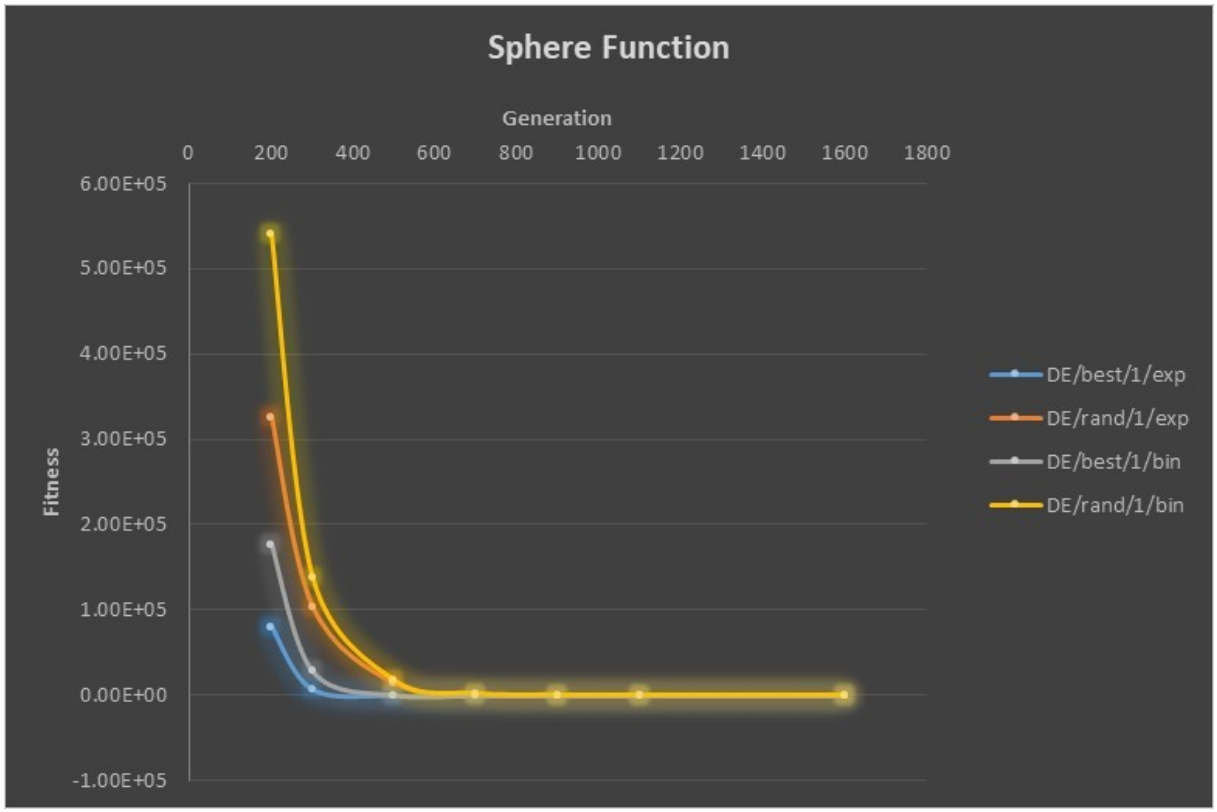


Figure 4.2: Comparison of DE/best/1/exp, DE/rand/1/exp, DE/best/1/bin and DE/rand/1/bin on *Sphere function* (f_1).

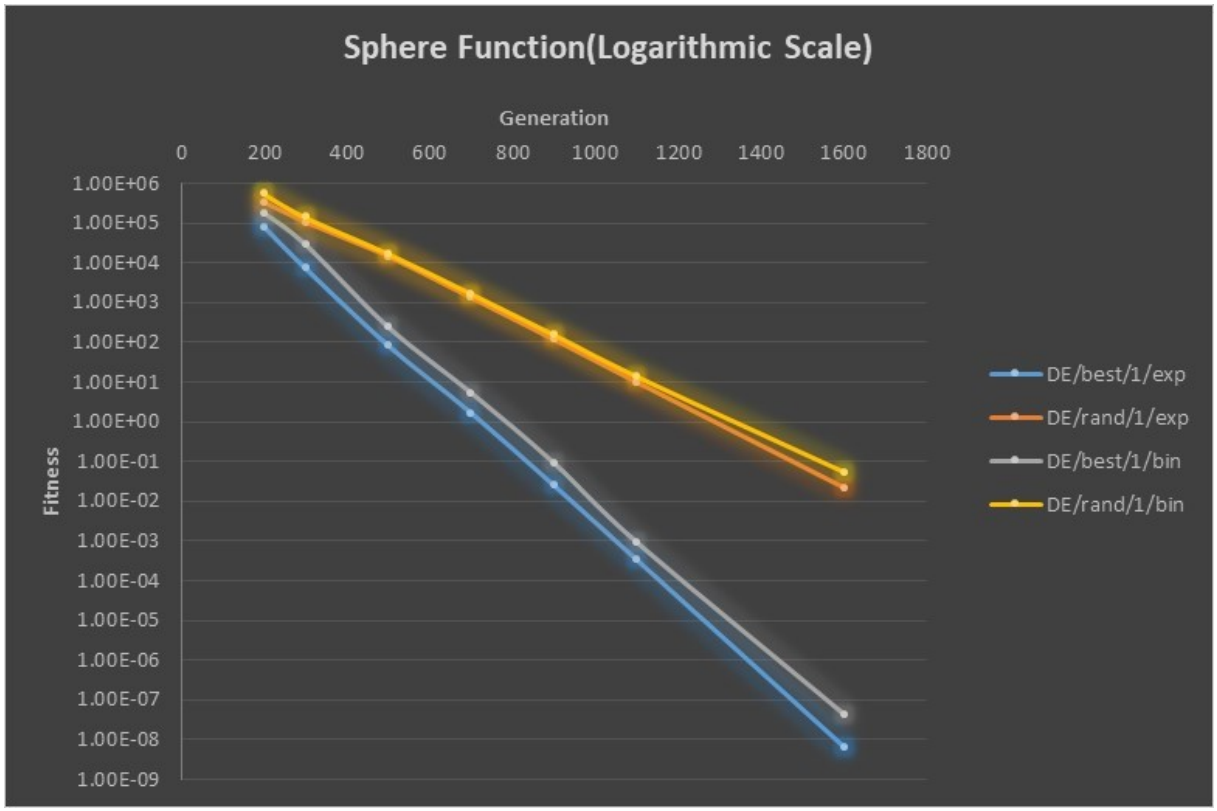


Figure 4.3: Comparison of DE/best/1/exp, DE/rand/1/exp, DE/best/1/bin and DE/rand/1/bin on *Sphere function* (f_1) in logarithmic scale.

4.6.2 *Schwefels - NonSep* (f_2)

Table 4.4: Optimization of *Schwefels - NonSep*(f_2) by DE variants with different number of generations

Generation	DE/best/1/exp	DE/rand/1/exp	DE/best/1/bin	DE/rand/1/bin
1600	3.02851574e-008	0.008325130679	1.89179286e-014	0.6266945004
1100	0.0001295143156	0.5688759685	7.27359150e-009	0.3713843226
900	0.004401491955	1.460704565	1.20718937e-006	0.001222378574
700	0.123017922	0.8084108829	0.0001013888032	0.02334097028
500	0.2383768857	1.382511258	0.01535930391	0.8691927791
300	3.795502901	22.37211418	2.890247107	53.3876152

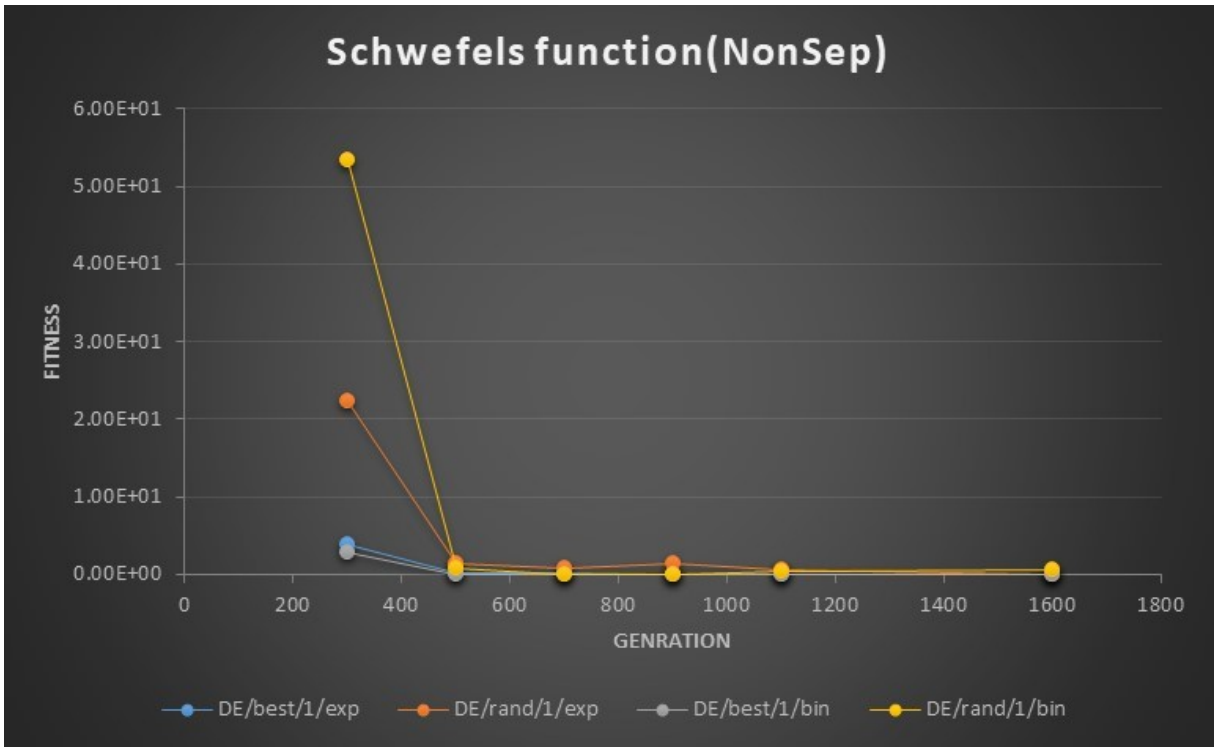


Figure 4.4: Comparison of DE/best/1/exp, DE/rand/1/exp, DE/best/1/bin and DE/rand/1/bin on *Schwefels - NonSep*(f_2)

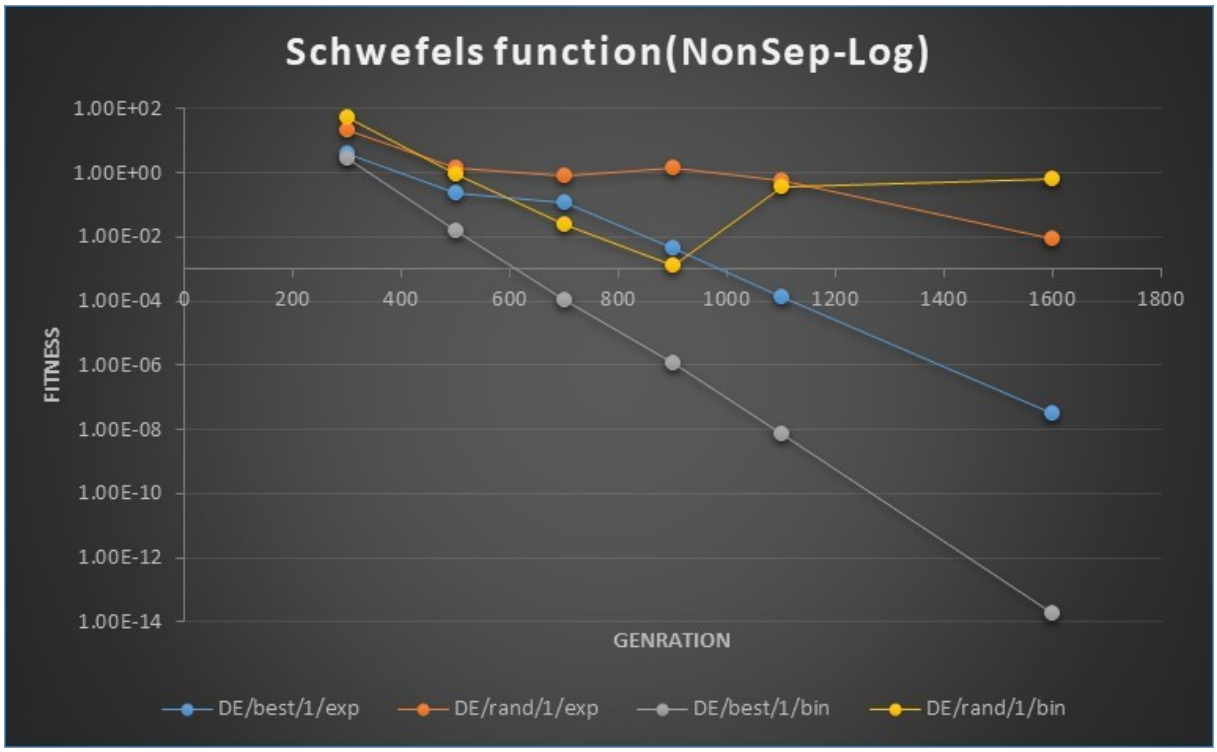


Figure 4.5: Comparison of DE/best/1/exp, DE/rand/1/exp, DE/best/1/bin and DE/rand/1/bin on *Schwefels - NonSep*(f_2) in logarithmic scale.

4.6.3 *Schwefels - Sep* (f_3)

Table 4.5: Optimization of *Schwefels - Sep* (f_3) by DE variants with different number of generations

Generation	DE/best/1/exp	DE/rand/1/exp	DE/best/1/bin	DE/rand/1/bin
1600	0.280887574	4.516009331	5.214056969	0.9924734235
1100	3.911840439	22.70979881	34.79998398	7.066696644
900	4.460326195	25.33384132	10.08383179	99.09114838
700	4.622749329	28.46648788	17.39812469	87.33509827
500	33.7862587	45.97307968	49.86231995	175.7211304
300	73.89970398	361.4726563	156.3807831	210.0697632

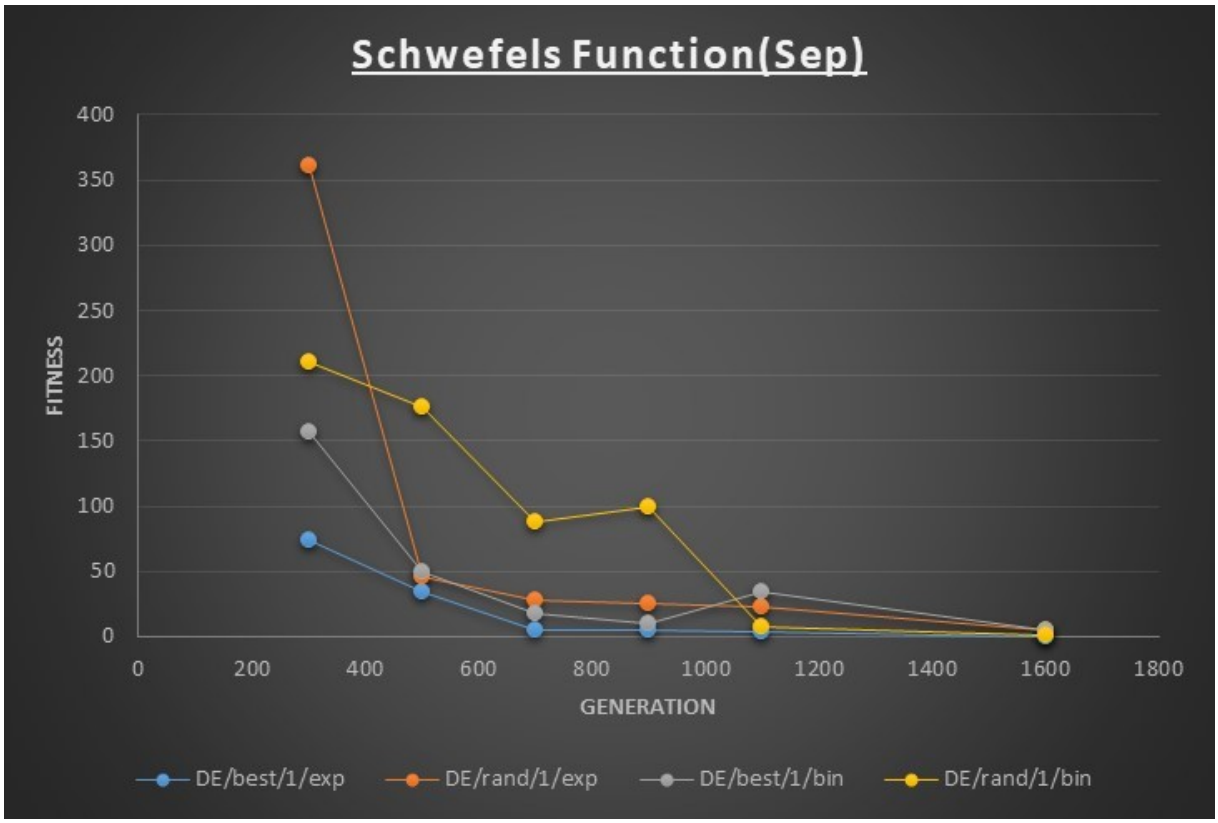


Figure 4.6: Comparison of DE/best/1/exp, DE/rand/1/exp, DE/best/1/bin and DE/rand/1/bin on *Schwefels - Sep*(f_3)

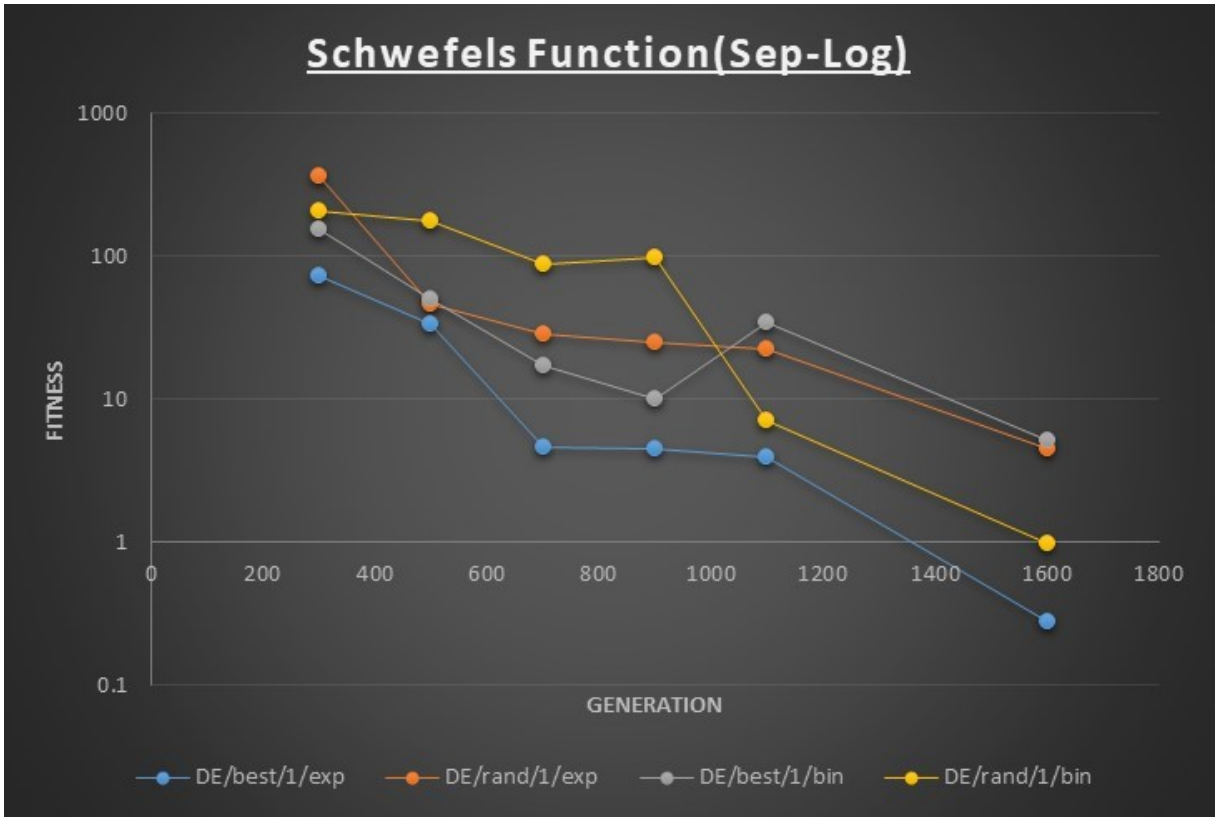


Figure 4.7: Comparison of DE/best/1/exp, DE/rand/1/exp, DE/best/1/bin and DE/rand/1/bin on *Schwefels - Sep*(f_3) in logarithmic scale

4.6.4 *Rosenbrock function*(f_7)

Table 4.6: Optimization of *Rosenbrock function*(f_7) by DE variants with different number of generations

Generation	DE/best/1/exp	DE/rand/1/exp	DE/best/1/bin	DE/rand/1/bin
1600	2.20702e+013	7.221627e+012	2.207021e+013	7.221627e+012
1100	2.547192e+013	7.244687e+012	2.547192e+013	7.244687e+012
900	2.681727e+013	7.315022e+012	2.681727e+013	7.315022e+012
700	2.773624e+013	7.725124e+012	2.773624e+013	7.725124e+012
500	2.931539e+013	1.09508e+013	2.931539e+013	1.09508e+013
300	3.025037e+013	5.81347e+013	3.025037e+013	5.81347e+013

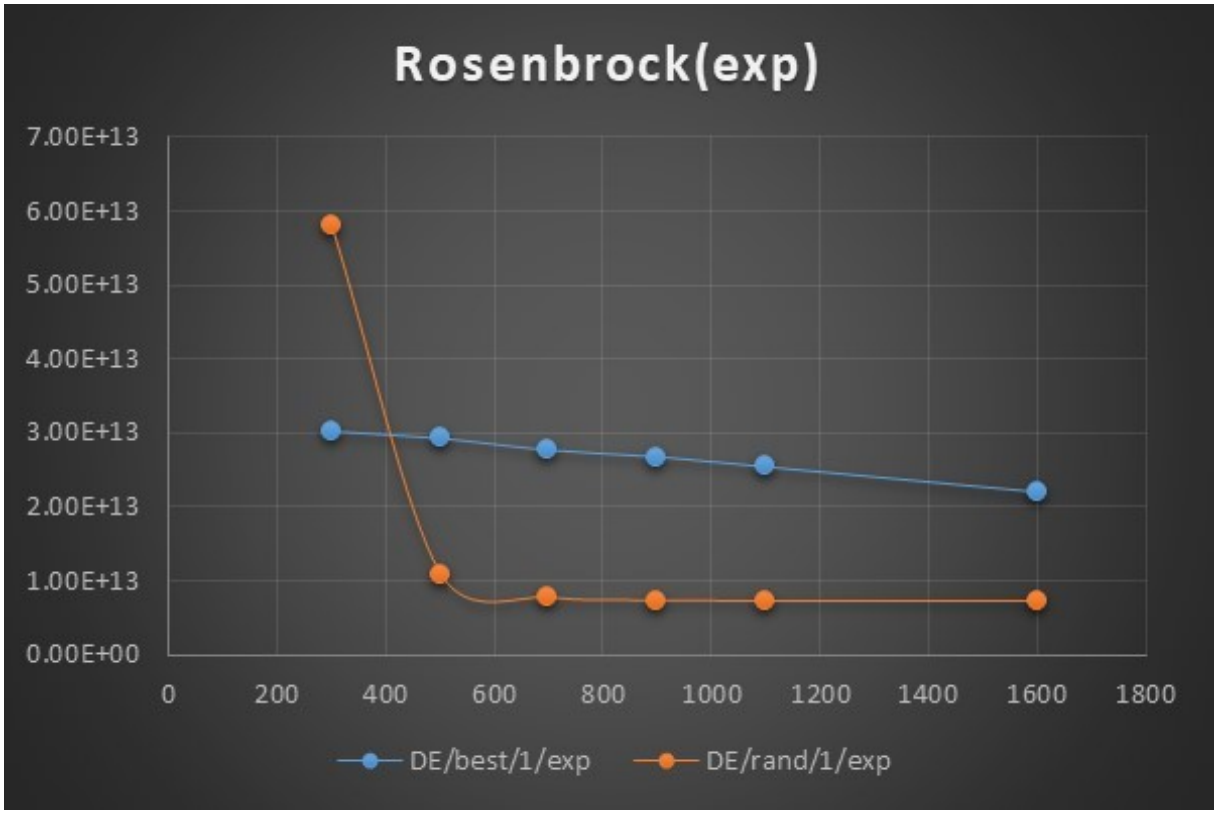


Figure 4.8: Comparison of DE/best/1/exp and DE/rand/1/exp on *Rosenbrock function*(f_7)

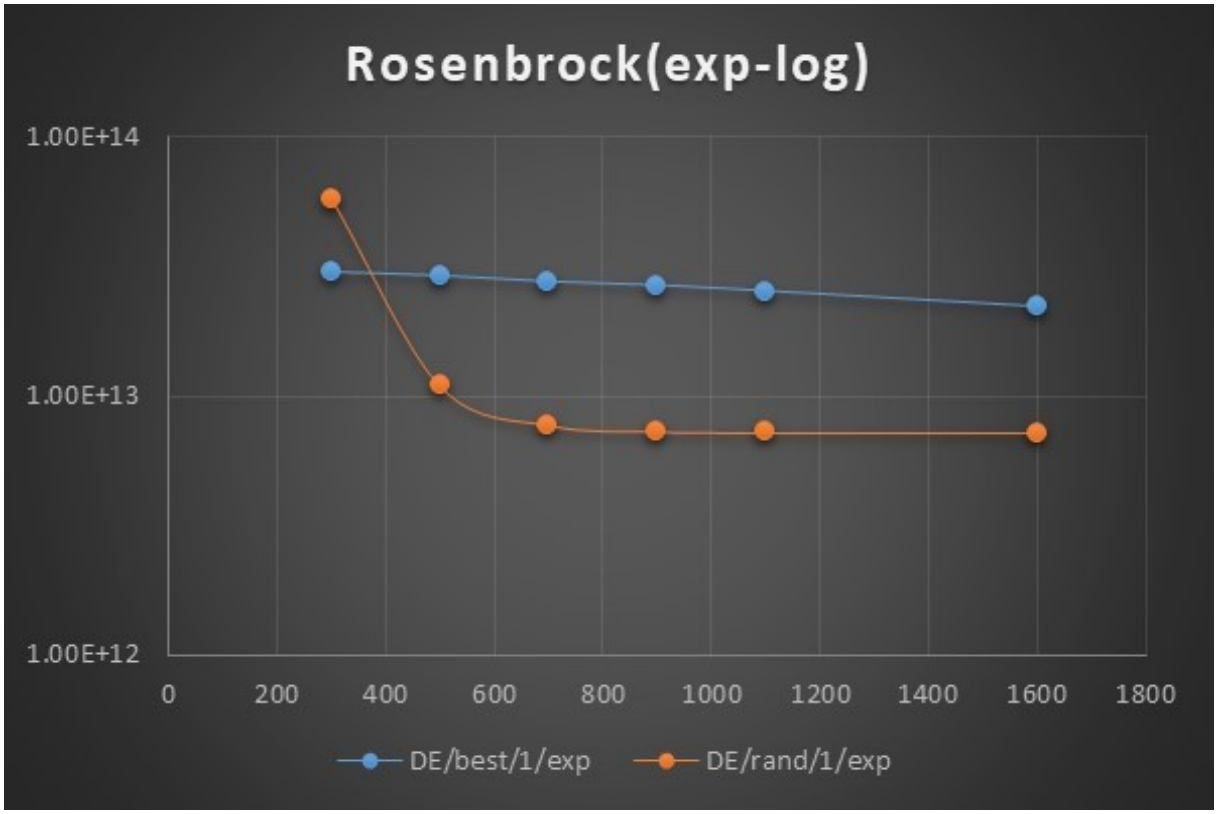


Figure 4.9: Comparison of DE/best/1/exp and DE/rand/1/exp on *Rosenbrock function*(f_7) in logarithmic scale

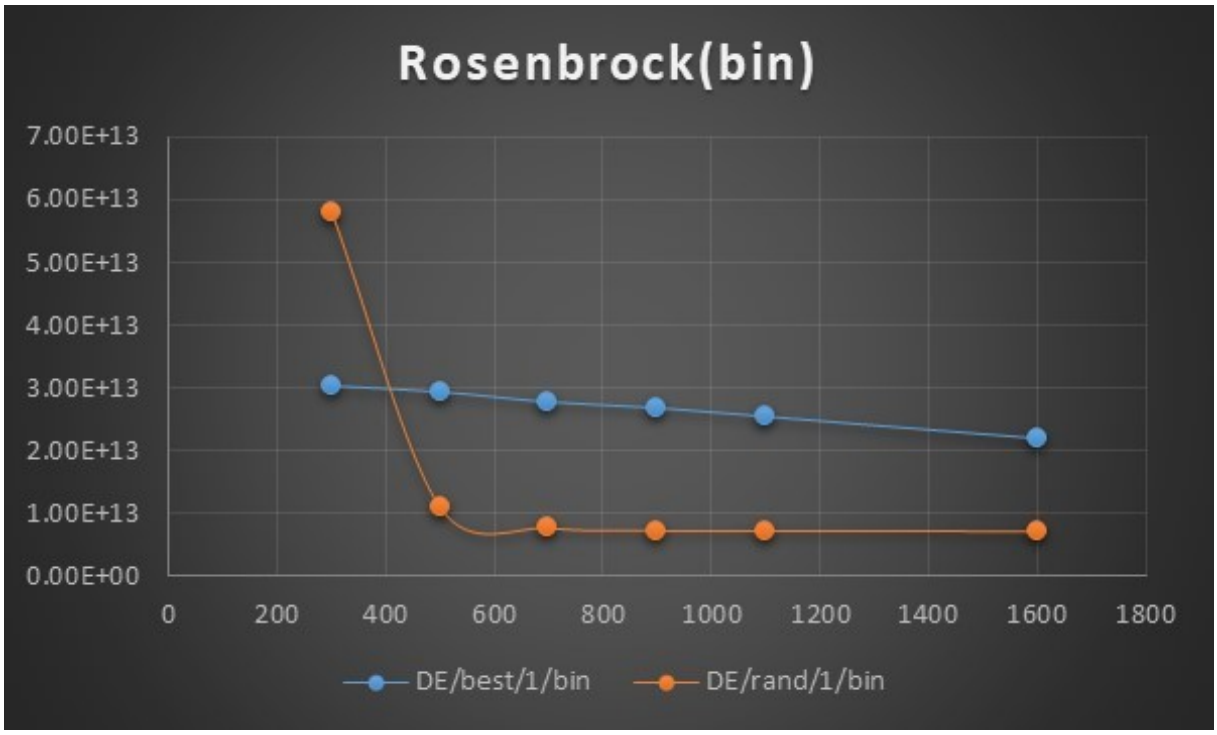


Figure 4.10: Comparison of DE/best/1/bin and DE/rand/1/bin on *Rosenbrock function*(f_7)

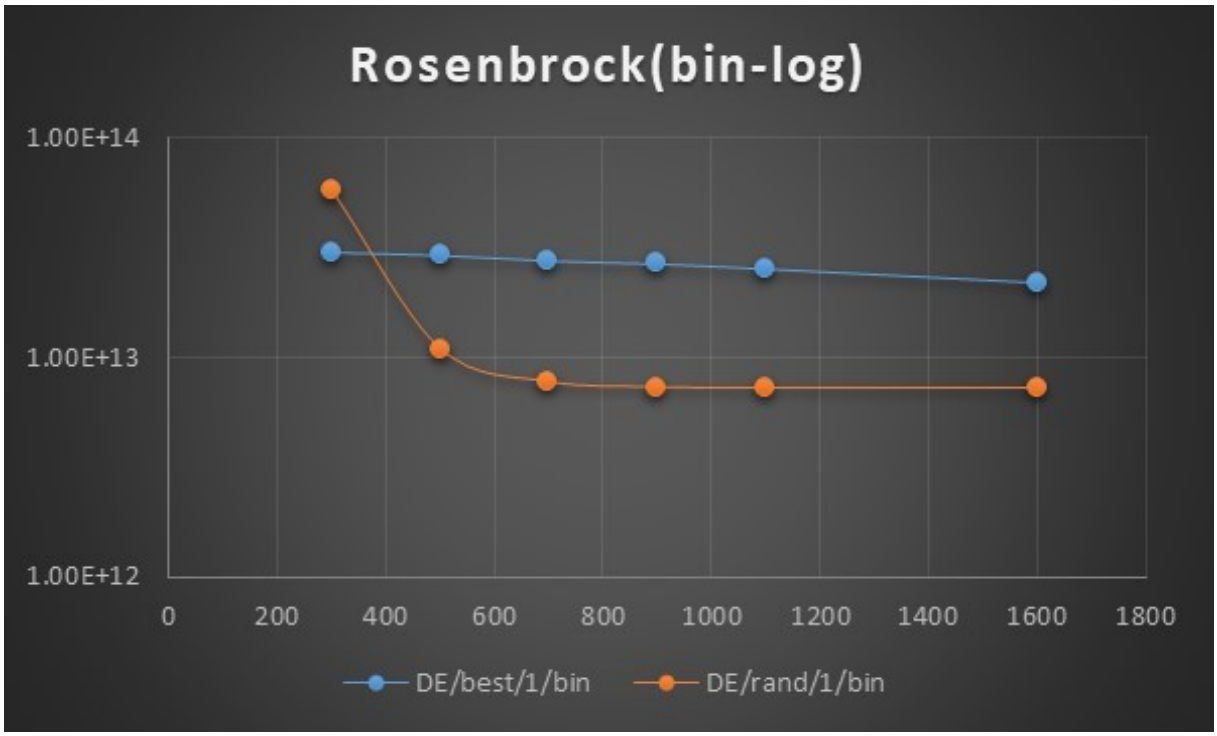


Figure 4.11: Comparison of DE/best/1/bin and DE/rand/1/bin on *Rosenbrock function*(f_7) in logarithmic scale

4.6.5 *Rastrigin function*(f_{10})

Table 4.7: Optimization of *Rastrigin function*(f_{10}) by DE variants with different number of generations

Generation	DE/best/1/exp	DE/rand/1/exp	DE/best/1/bin	DE/rand/1/bin
1600	0	2.44834446e-008	0	9862.541016
1100	1.817392103e-008	0.0009079681477	4.97379915e-014	62087.64844
900	4.848548542e-006	0.05491840094	5.566338501e-010	146964.625
700	0.001900542411	2.923788548	1.852597961e-006	199296.8281
500	0.8772598505	191.3851013	0.004891866352	575360.875
300	358.4184265	13030.75	23.80634117	1147348.75

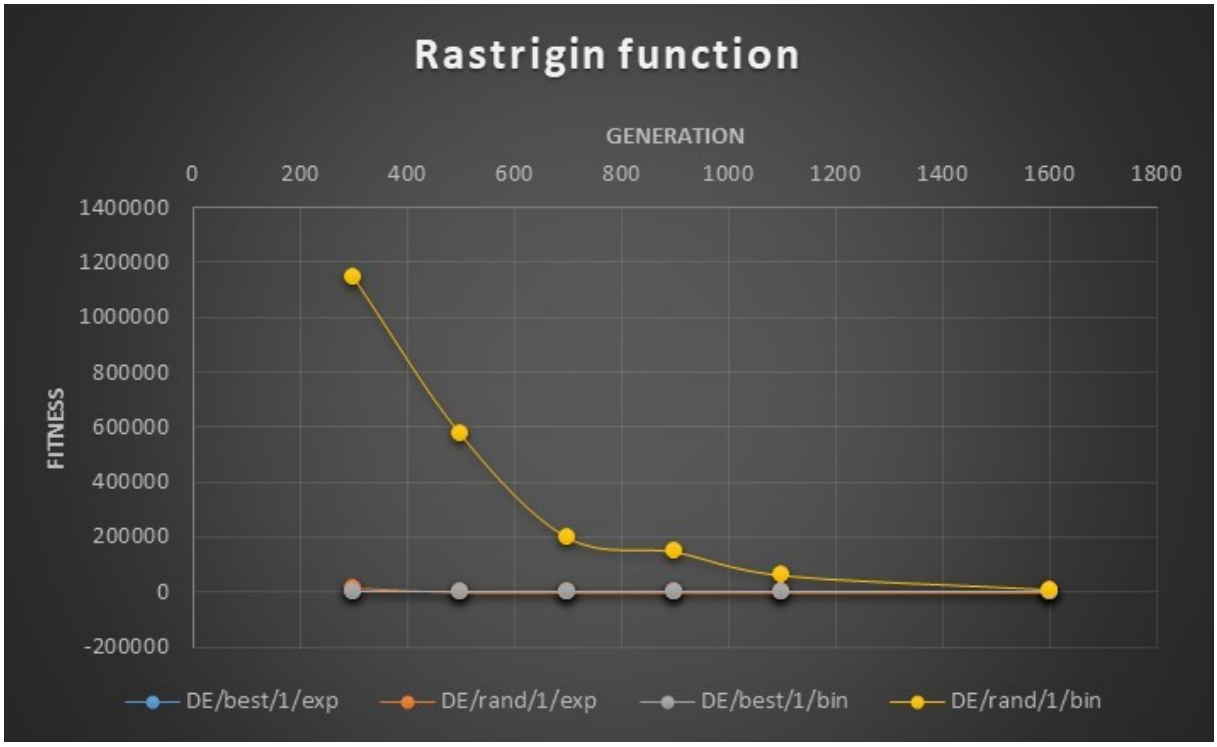


Figure 4.12: Comparison of DE/best/1/exp, DE/rand/1/exp, DE/best/1/bin and DE/rand/1/bin on *Rastrigin function*(f_{10})

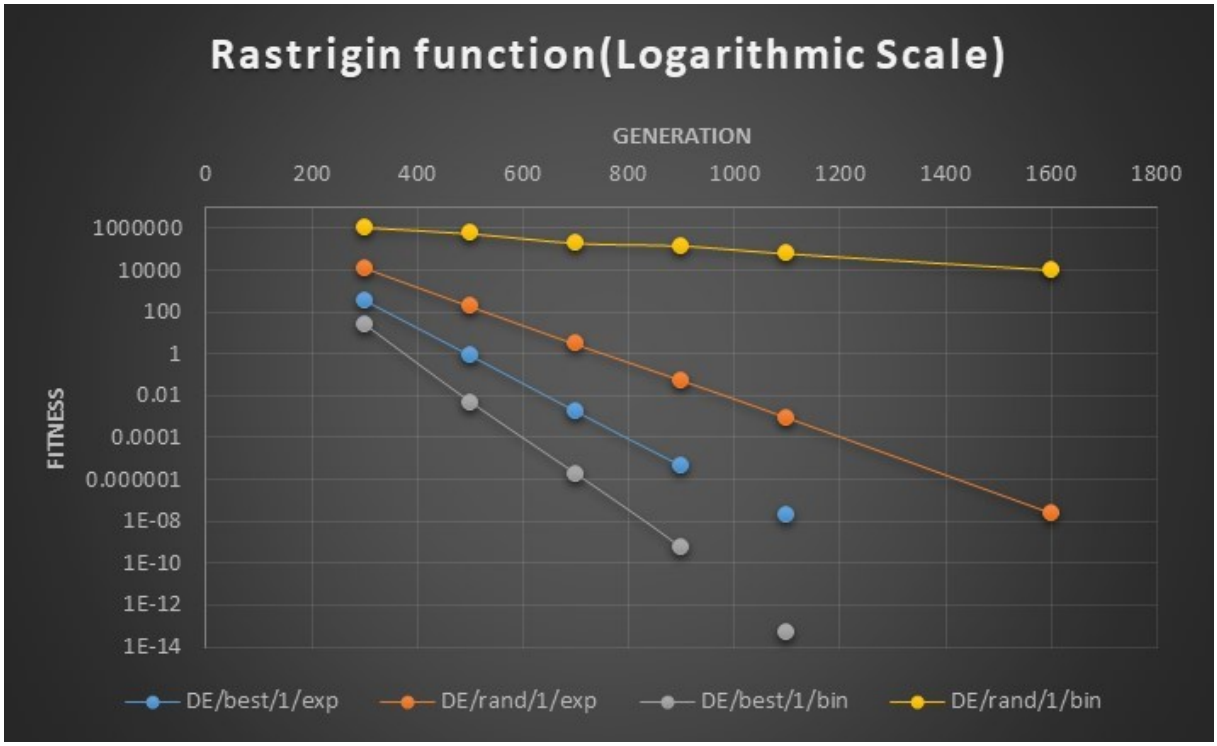


Figure 4.13: Comparison of DE/best/1/exp, DE/rand/1/exp, DE/best/1/bin and DE/rand/1/bin on *Rastrigin function*(f_{10}) in logarithmic scale

4.6.6 *Non-Continuous Rastrigin function*(f_{11})

Table 4.8: Optimization of *Non-Continuous Rastrigin function*(f_{11}) by DE variants with different number of generations

Generation	DE/best/1/exp	DE/rand/1/exp	DE/best/1/bin	DE/rand/1/bin
1600	0	0.0001333026157	0.2650287151	2.17291949e-006
1100	2.47161580e-009	0.2267789692	0.2650287151	3.24501843e-006
900	1.73597743e-006	5.169354439	0.2650287151	0.5704532862
700	0.001355597749	13.16416168	0.2650287151	19.85091209
500	0.5972465873	149.1278839	0.2653563917	24.23215485
300	1.173972607	198.3811646	3.191211462	265.7927856

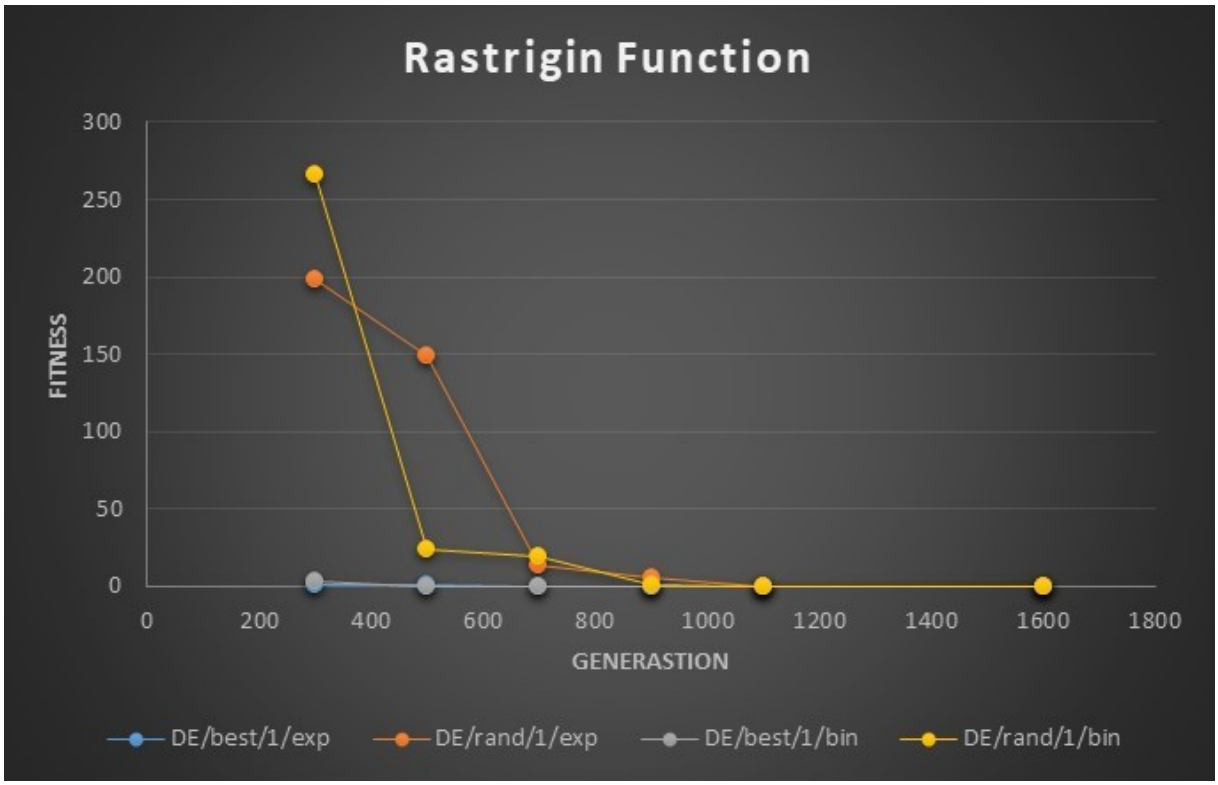


Figure 4.14: Comparison of DE/best/1/exp and DE/rand/1/exp on *Non-Continuous Rastrigin function*(f_{11})

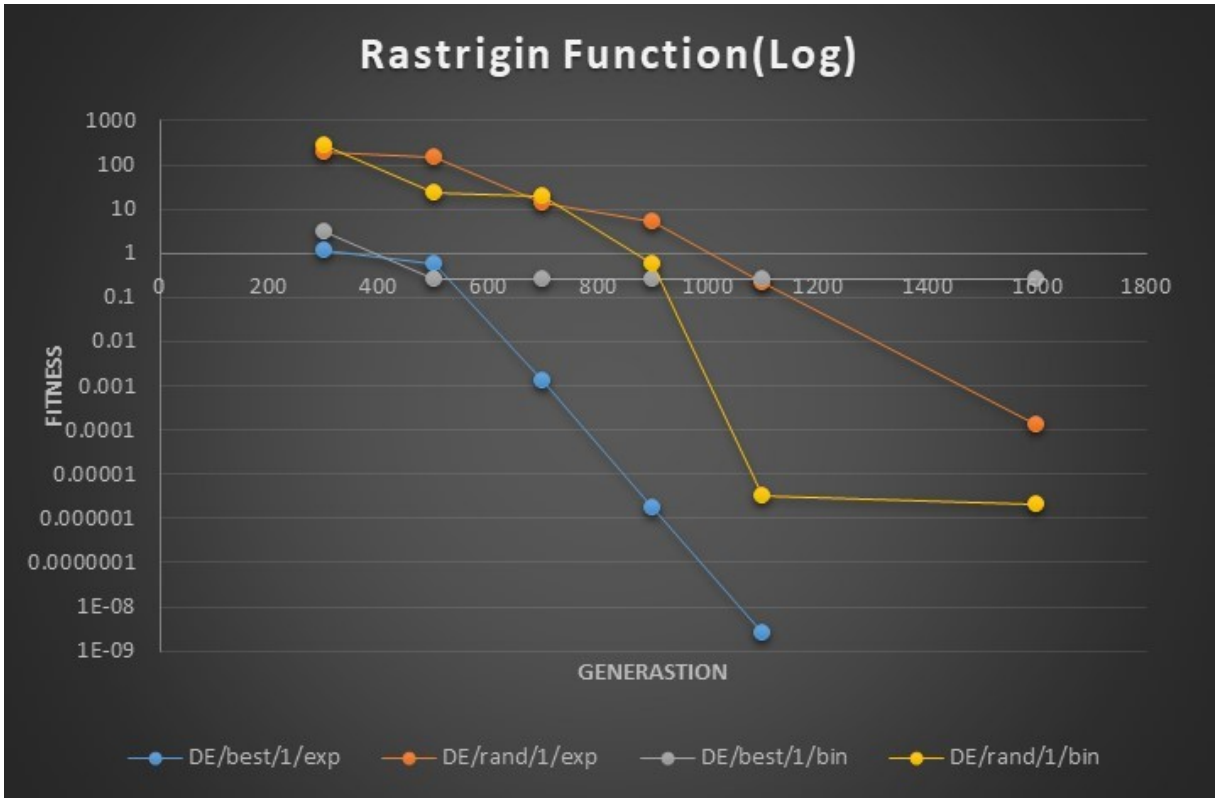


Figure 4.15: Comparison of DE/best/1/exp and DE/rand/1/exp on *Non-Continuous Rastrigin function*(f_{11}) in logarithmic scale

4.6.7 *Griewank function*(f_{14})

Table 4.9: Optimization of *Griewank function*(f_{14}) by DE variants with different number of generations

Generation	DE/best/1/exp	DE/rand/1/exp	DE/best/1/bin	DE/rand/1/bin
1600	0	0.0002096190146	0.01232099161	1.316485405
1100	1.02224118e-010	0.0507411994	0.01232099161	6.412651062
900	4.03430355e-008	0.2307763696	0.01232099161	16.39913368
700	3.23733220e-005	0.9333333373	0.01232099161	38.88303375
500	0.01709116623	1.432344198	0.01233339775	97.34729004
300	1.061983466	10.64492226	0.094392851	254.4597931

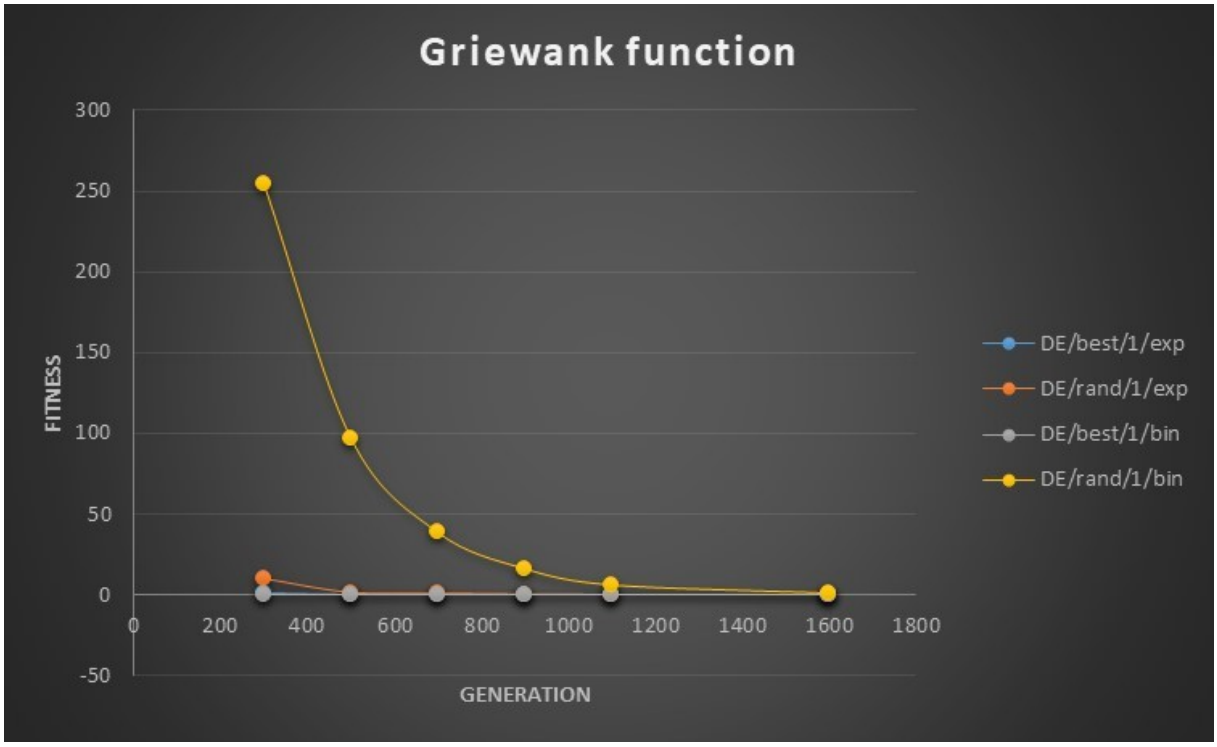


Figure 4.16: Comparison of DE/best/1/exp, DE/rand/1/exp, DE/best/1/bin and DE/rand/1/bin on *Griewank function*(f_{14})

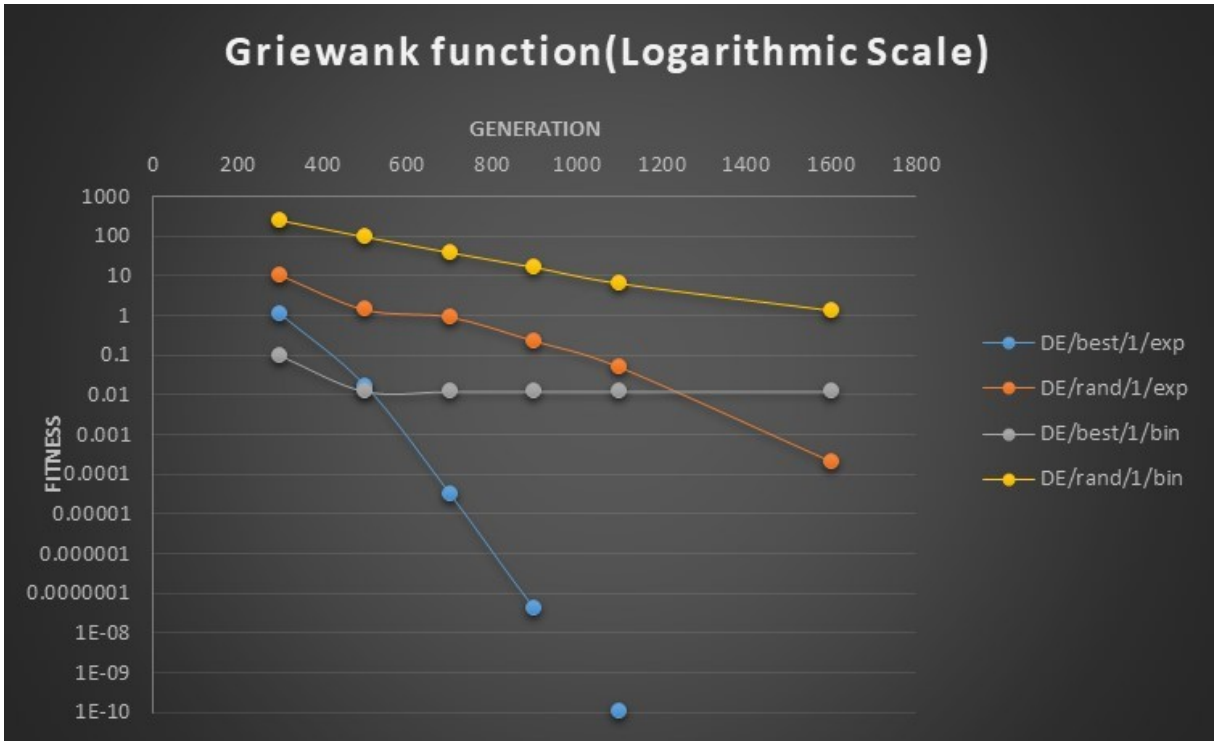


Figure 4.17: Comparison of DE/best/1/exp, DE/rand/1/exp, DE/best/1/bin and DE/rand/1/bin on *Griewank function*(f_{14}) in logarithmic scale

4.6.8 *Alpine function*(f_{15})

Table 4.10: Optimization of *Alpine function*(f_{15}) by DE variants with different number of generations

Generation	DE/best/1/exp	DE/rand/1/exp	DE/best/1/bin	DE/rand/1/bin
1600	0.01331150904	24.68152428	205.5137177	31.65343094
1100	11.2985611	81.38645172	284.5536499	105.3691559
900	26.1483593	101.0478058	205.539505	29.64492989
700	41.34587479	179.911026	273.5222778	110.540329
500	34.30560303	194.576889	370.39151	56.76514053
300	177.1293945	362.4794617	1304.365967	456.1977234

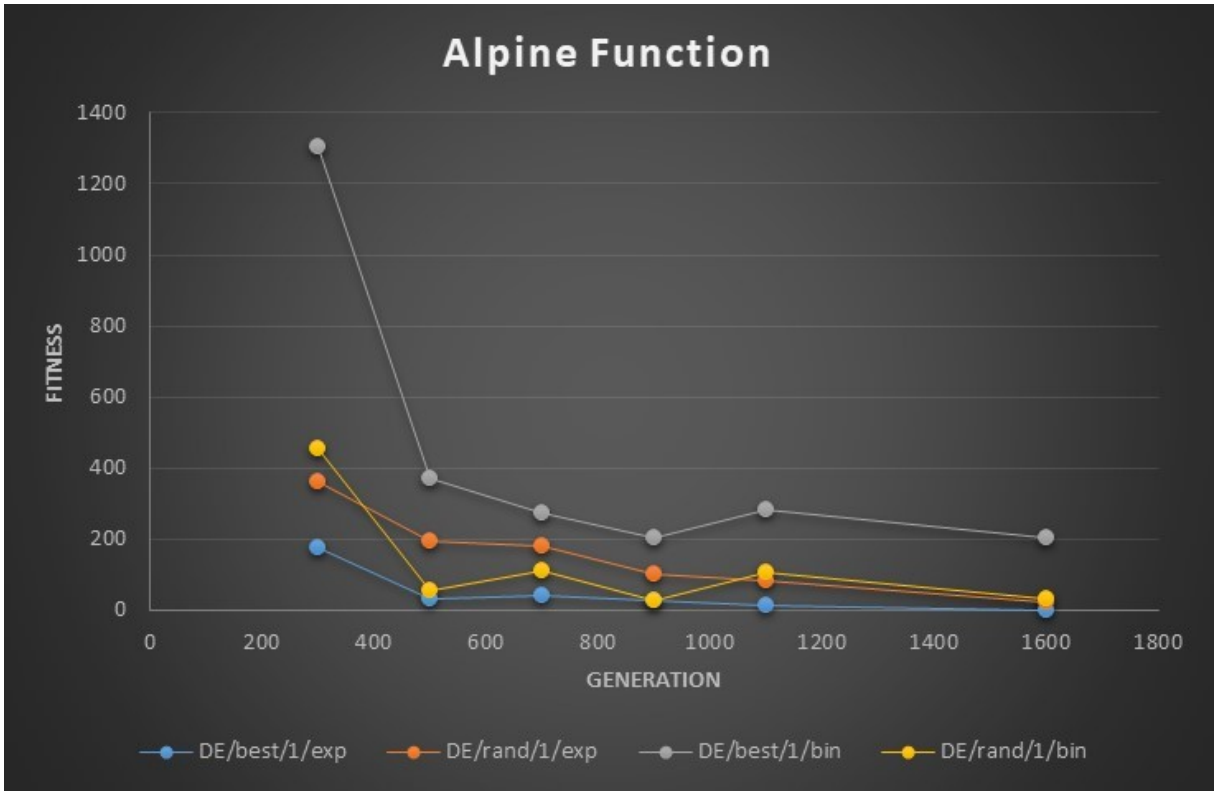


Figure 4.18: Comparison of DE/best/1/exp, DE/rand/1/exp, DE/best/1/bin and DE/rand/1/bin on *Alpine function*(f_{15})

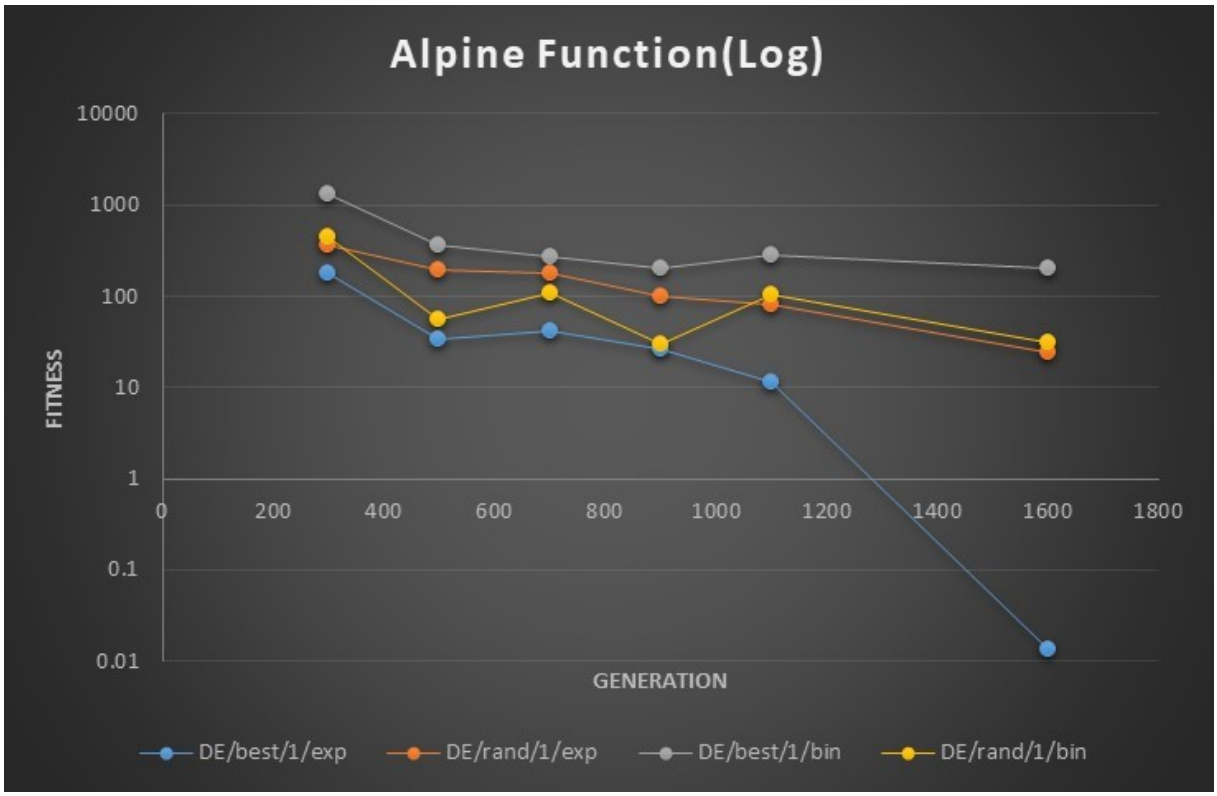


Figure 4.19: Comparison of DE/best/1/exp, DE/rand/1/exp, DE/best/1/bin and DE/rand/1/bin on *Alpine function*(f_{15}) in logarithmic scale

4.6.9 *Kowalik function*(f_{20})

Table 4.11: Optimization of *Kowalik function*(f_{20}) by DE variants with different number of generations

Generation	DE/best/1/exp	DE/rand/1/exp	DE/best/1/bin	DE/rand/1/bin
1600	0.0005940087722	0.005701813381	0.0005940087722	0.005701813381
1100	0.0005940087722	0.005701813381	0.0005940087722	0.005701813381
900	0.0005940087722	0.005701813381	0.0005940087722	0.005701813381
700	0.0005940087722	0.005701813381	0.0005940087722	0.005701813381
500	0.0005940087722	0.005701813381	0.0005940087722	0.005701813381
300	0.00141809124	0.006491493434	0.0005940087722	0.005701813381

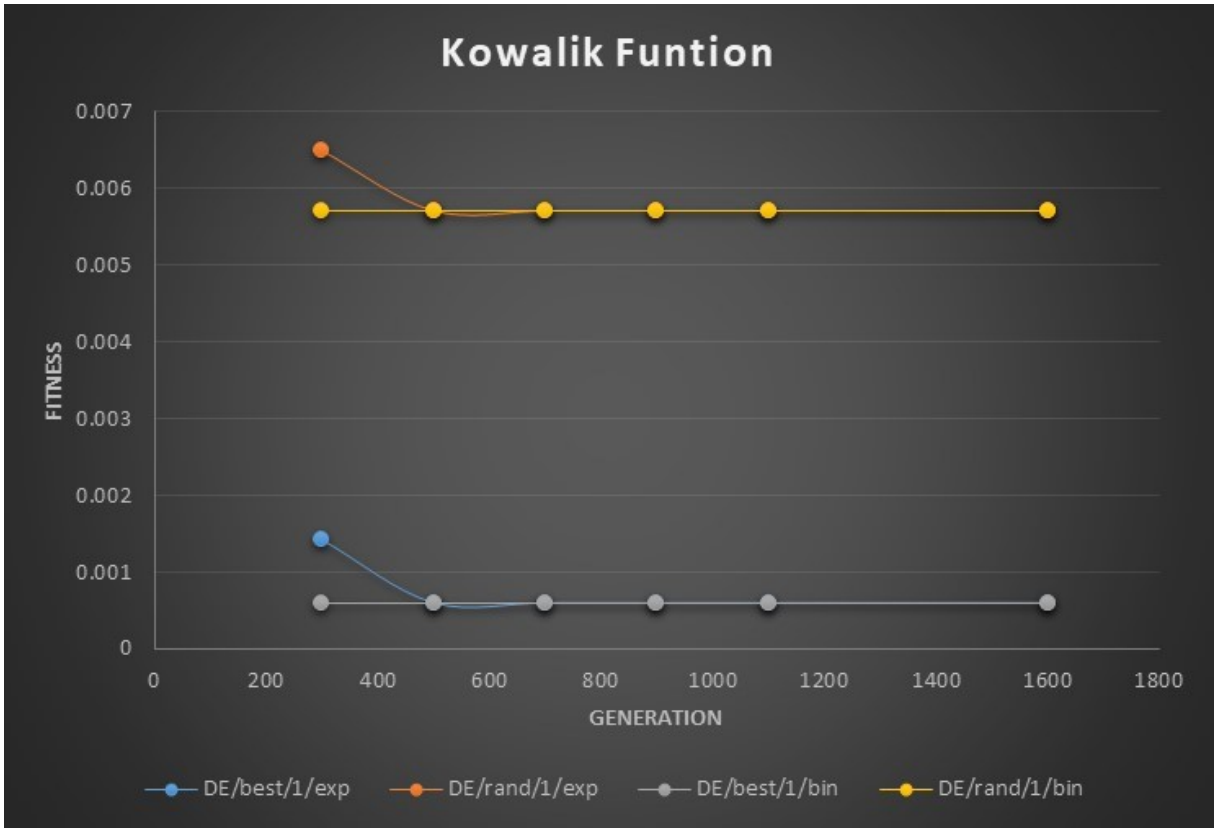


Figure 4.20: Comparison of DE/best/1/exp, DE/rand/1/exp, DE/best/1/bin and DE/rand/1/bin on *Kowalik function*(f_{20})

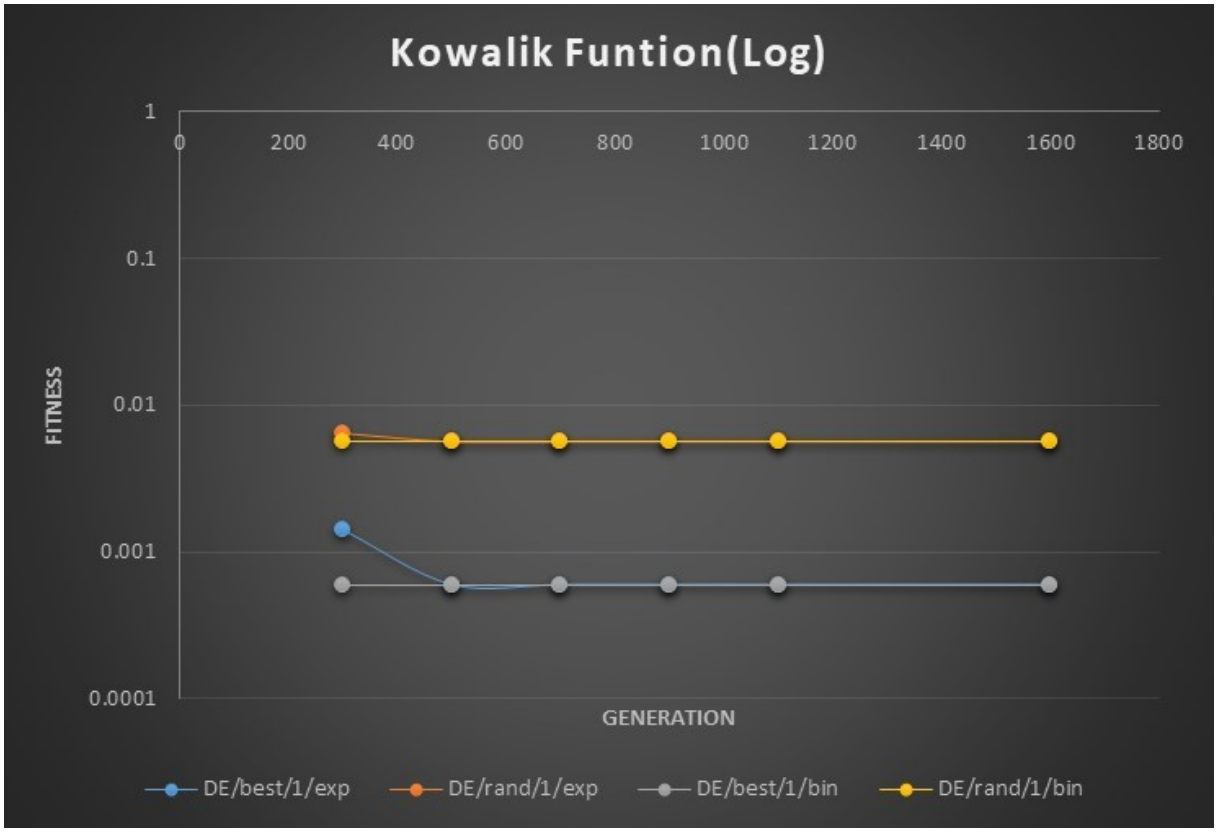


Figure 4.21: Comparison of DE/best/1/exp, DE/rand/1/exp, DE/best/1/bin and DE/rand/1/bin on *Kowalik function*(f_{20}) in logarithmic scale

4.6.10 *Michalewicz function*(f_{29})

Table 4.12: Optimization of *Michalewicz function*(f_{29}) by DE variants with different number of generations

Generation	DE/best/1/exp	DE/rand/1/exp	DE/best/1/bin	DE/rand/1/bin
1600	-18.81696701	-19.29681969	-23.45302963	-8.394515038
1100	-16.84629059	-17.7629528	-23.45302963	-6.424415112
900	-16.29207993	-17.44618416	-23.4141655	-6.408016205
700	-15.2630291	-16.54985237	-21.87807465	-6.408016205
500	-13.58875275	-15.26201439	-18.16374969	-6.408016205
300	-11.39054585	-13.64397812	-13.29537582	-6.408016205

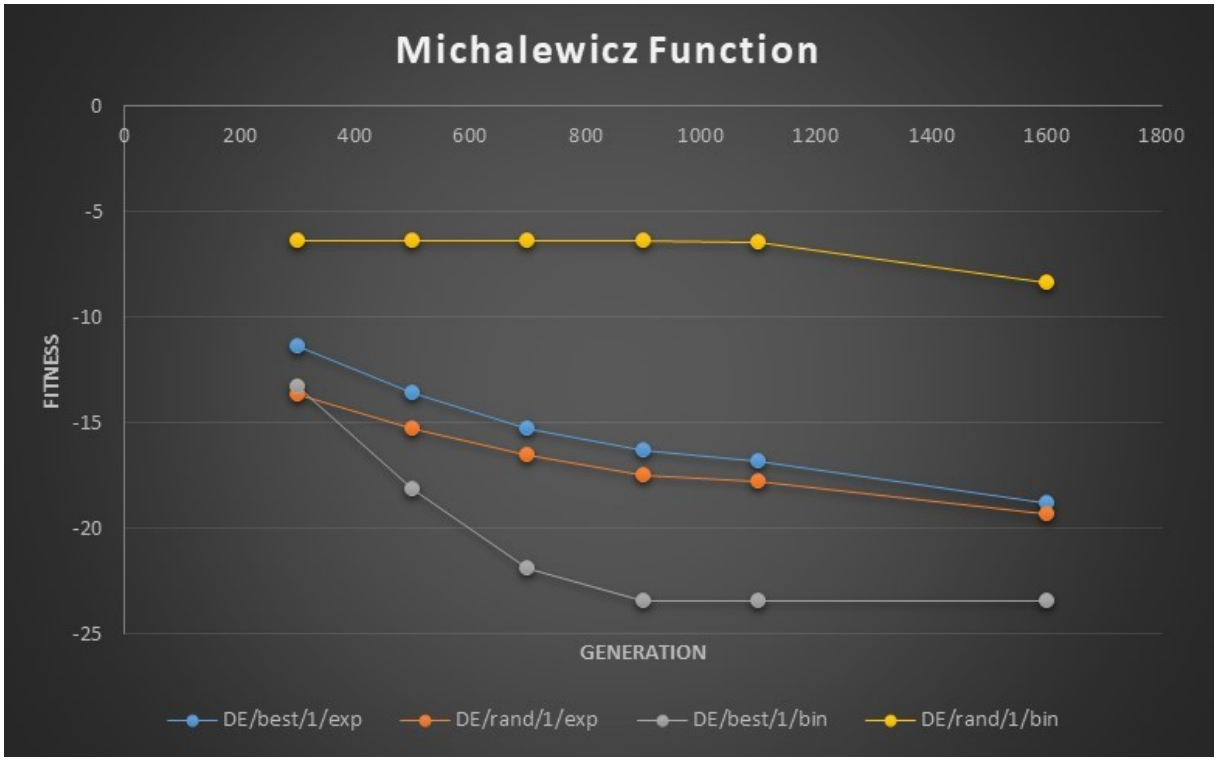


Figure 4.22: Comparison of DE/best/1/exp, DE/rand/1/exp, DE/best/1/bin and DE/rand/1/bin on *Michalewicz function*(f_{29})

4.6.11 *Comparison for DE variants : (Population 1600)*

Table 4.13: Data Table for *Comparison for DE variants: (Generation 1600)*

Functions	**DE/best/1/exp**	**DE/rand/1/exp**	**DE/best/1/bin**	**DE/rand/1/bin**
Sphere(f_1)	6.4059060e-009	0.02194936387	4.2399591e-008	0.05398572236
Schwefel(f_2)	3.02851574e-008	0.008325130679	1.89179286e-014	0.6266945004
Schwefel(f_3)	0.280887574	4.516009331	5.214056969	0.9924734235
Rosenbrock(f_7)	2.20702e+013	7.221627e+012	2.207021e+013	7.221627e+012
Rastrigin(f_{10})	0	2.44834446e-008	0	9862.541016
Non-Con-Rastrigin(f_{11})	0	0.0001333026157	0.2650287151	2.17291949e-006
Griewank(f_{14})	0	0.0002096190146	0.01232099161	1.316485405
Alpine(f_{15})	0.01331150904	24.68152428	205.5137177	31.65343094
Kawalik(f_{20})	0.0005940087722	0.005701813381	0.0005940087722	0.005701813381
Michalewickz(f_{29})	-18.81696701	-19.29681969	-23.45302963	-8.394515038

4.7 Performace Analysis of DE Variants

Table 4.14: Performance Factors for DE/best/1/exp

Functions	DE/best/1/exp	F_{min}	Absolute Error	Mean Error
Sphere(f_1)	6.4059060e-009	0	6.4059060e-009	2.21E+12
Schwefel(f_2)	3.02851574e-008	0	3.02851574e-008	
Schwefel(f_3)	0.280887574	0	0.280887574	
Rosenbrock(f_7)	2.20702e+013	0	2.20702e+013	
Rastrigin(f_{10})	0	0	0	
Non-Con-Rastrigin(f_{11})	0	0	0	
Griewank(f_{14})	0	0	0	
Alpine(f_{15})	0.01331150904	0	0.01331150904	
Kawalikf_{20}	0.0005940087722	3.07e-04	0.055635002	
Michalewickzf_{29}	-18.81696701	-9.66015	9.15681701	

Table 4.15: Performance Factors for DE/rand/1/exp

Functions	DE/rand/1/exp	F_{min}	Absolute Error	Mean Error
Sphere(f_1)	0.02194936387	0	0.02194936387	722166.6
Schwefel(f_2)	0.008325130679	0	0.008325130679	
Schwefel(f_3)	4.516009331	0	4.516009331	
Rosenbrock(f_7)	7.221627e+012	0	7.221627e+012	
Rastrigin(f_{10})	2.44834446e-008	0	2.44834446e-008	
Non-Con-Rastrigin(f_{11})	0.0001333026157	0	0.0001333026157	
Griewank(f_{14})	0.0002096190146	0	0.0002096190146	
Alpine(f_{15})	24.68152428	0	24.68152428	
Kawalikf_{20}	0.005701813381	3.07e-04	.050527198	
Michalewickzf_{29}	-19.29681969	-9.66015	9.63666969	

Table 4.16: Performance Factors for DE/best/1/bin

Functions	**DE/best/1/bin**	F_{min}	**Absolute Error**	**Mean Error**
Sphere(f_1)	4.2399591e-008	0	4.2399591e-008	2.21E+12
Schwefel(f_2)	1.89179286e-014	0	1.89179286e-014	
Schwefel(f_3)	5.214056969	0	5.214056969	
Rosenbrock(f_7)	2.207021e+013	0	2.207021e+013	
Rastrigin(f_{10})	0	0	0	
Non-Con-Rastrigin(f_{11})	0.2650287151	0	0.2650287151	
Griewank(f_{14})	0.01232099161	0	0.01232099161	
Alpine(f_{15})	205.5137177	0	205.5137177	
Kawalikf_{20}	0.0005940087722	3.07e-04	0.055635002	
Michalewickzf_{29}	-23.45302963	-9.66015	13.79287968	

Table 4.17: Performance Factors for DE/rand/1/bin

Functions	**DE/rand/1/bin**	F_{min}	**Absolute Error**	**Mean Error**
Sphere(f_1)	0.05398572236	0	0.05398572236	7.22163E+11
Schwefel(f_2)	0.6266945004	0	0.6266945004	
Schwefel(f_3)	0.9924734235	0	0.9924734235	
Rosenbrock(f_7)	7.221627e+012	0	7.221627e+012	
Rastrigin(f_{10})	09862.541016	0	9862.541016	
Non-Con-Rastrigin(f_{11})	2.17291949e-006	0	2.17291949e-006	
Griewank(f_{14})	1.316485405	0	1.316485405	
Alpine(f_{15})	31.65343094	0	31.65343094	
Kawalikf_{20}	0.005701813381	3.07e-04	0.050527198	
Michalewickzf_{29}	-8.394515038	-9.66015	1.265634962	

Chapter 5

Proposed algorithms: Hybrid Differential evolution

Introduction

Hybrid algorithms play a prominent role in improving the search capability of algorithms. Hybridization aims to combine the advantages of each algorithm to form a hybrid algorithm, while simultaneously trying to minimize any substantial is advantage. In general, the outcome of hybridization can usually make some interesting improvements in terms of either computational speed or accuracy. Hybrid algorithms are diverse and they form a long list of algorithms and variants. Here, we only brifly touch the tip of the algorithm iceberg. Hybrid algorithms are two or more algorithms that run together and complement each other to produce a profitable synergy from their integration. These algorithms are commonly known as Hybrid Metaheuristics (HMs). Basically, we are going to do hybridization among several DE variants on some benchmark functions.

5.1 Hybrid Metaheuristic Algorithms

Hybrid algorithms are two or more algorithms that run together and complement each other to produce a profitable synergy from their integration. These algorithms are commonly known as hybrid metaheuristics (HMs). There are in fact two prominent issues of EAs in solving global and highly non convex optimization problem. These are:

- **Premature convergence:** The problem of premature convergence results in the lack of accuracy of the ïňĄnal solution. The ïňĄnal solution is a feasible solution close to the global optimal, often regarded as satisfactory or close-to-optimal solution.

- **Slow convergence:** Slow convergence means the solution quality does not improve sufficiently quickly. It shows stagnation or almost ïňĆat on a convergence graph (either a single iteration or the average of multiple iterations).

These two issues above are often partly related to the solution diversity that an algorithm can produce in the searching process. In nature, the diversity is maintained by the variety (quality) and abundance (quantity) of organisms at a given place and time, and this principle is applicable to EAs. At the beginning of a search process, usually diversity is high, and it decreases as the population may move towards the global optimum. High diversity may provide better guarantee to find the optimal solution with better accuracy, but this will usually lead to slow convergence, and thus there are some trade offs between convergence and accuracy. On the other hand, low diversity may lead to fast convergence while sacrificing the guarantee to find global optimality and with poor solution accuracy. This scenario is illustrated in Fig. 3.1. Hereby, we call the intersection between the convergence rate and accuracy as the trade off point. Obviously, this is an ideally approach and the real-world is much grayer and things are not clearly cut. It is worth pointing out that the diversity is one factor which is related to the more general concept of exploration and exploitation. High diversity encourages exploration, and low diversity does not necessarily mean exploitation because exploitation requires the use of landscape information and the information extracted from the population during the search process. Even with sufiňĄcient diversity, there is no guarantee to solve the convergence problem because convergence is a much complicated issue. Simply balancing the exploration and exploitation may make an algorithm work to its best capability but this does not necessarily mean its convergence rate is high. Good convergence requires clever exploitation at the right time and at the right place, which is still an open problem. In addition, it is widely known that one prominent factor for premature convergence is the lack of diversity. Therefore, in order to escape and jump out of local optima, proper diversity should be maintained during the search process, even with the expense of slower convergence. To enable this, hybridization has been a widely acceptable strategy to promote diversity along the search for the global optimum.

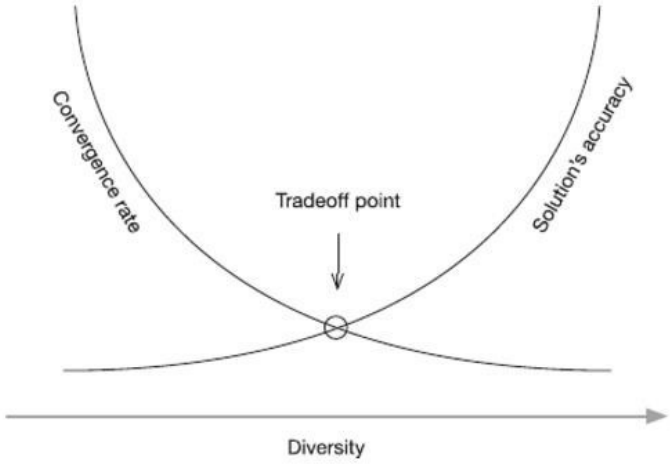


Figure 5.1: Compromising accuracy and convergence rate.

5.2 Taxonomy of Hybrid Algorithms

Generally speaking, hybrid algorithms can be grouped into two categories, which are described in the following subsections.

5.2.1 Collaborative Hybrids

- **Multi-stage:** There are two stages involved in this case. The ïňĄrst algorithm acts as the global optimizer whereas the second algorithm performs local search. A challenging issue in such an implementation is to know when to switch to the second algorithm.
- **Sequential:** In this structure, both algorithms are run alternatively until one of the convergence criteria is met. For simplicity, both algorithms will be run for similar number of iterations before proceeding to the next algorithm.
- **Parallel:** Two algorithms are run simultaneously, manipulating on the same population. One of the algorithms may be executed on a pre-speciïňĄed percentage of an algorithm.

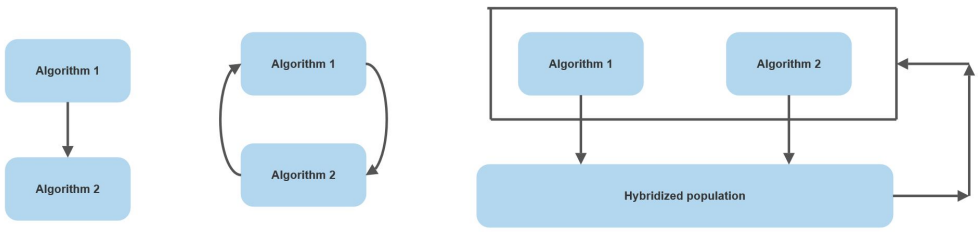


Figure 5.2: Collaborative framework of hybrid algorithm, depicting multi-stage, sequential and parallel structures.

5.2.2 Integrative Hybrids

In this aspect, one algorithm is regarded as a subordinate, embedded in a master metaheuristic. For this category, the contributing weight of the secondary algorithm is approximately 10-20 percentage. This involves incorporation of a manipulating operator from a secondary algorithm into a primary algorithm. Under this category, Fig. 5.3 illustrates the two possible approaches:

- **Full manipulation:** The entire population is manipulated at every iteration. Such operation can be integrated in line with the existing source code, usually as a subroutine/subfunction.
- **Partial manipulation:** In this manipulation, only a portion of the entire population is accelerated using local search methods such as gradient methods. Choosing the right portion and the right candidate to be accelerated pose a great challenge in assuring the success of this hybrid structure.

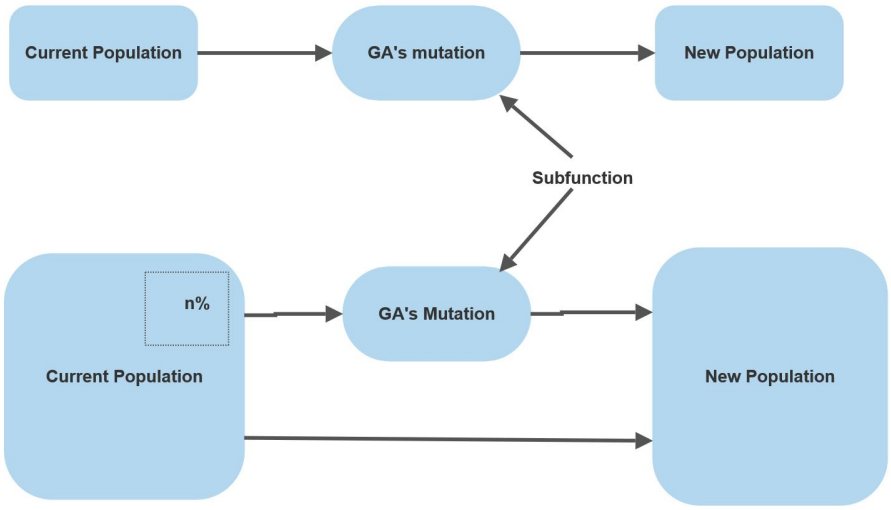


Figure 5.3: Integrative structure of a hybrid algorithm, with full and partial manipulation.

5.3 Advantages of Hybridization

In a hybrid algorithm, two or more algorithms are collectively and cooperatively solving a predefined problem. In some hybrids, one algorithm may be incorporated as a sub-algorithm to locating the optimal parameters for another algorithm, while in other cases, different components of algorithms such mutation and crossover are used to improve another algorithm in the hybrid structure. With regards to this nature, hybrid algorithms can loosely be divided into two categories:

- **Unified purpose hybrids:** Under this category, all sub-algorithms are utilized to solve the same problem directly and different sub-algorithms are used indifferent search stages. Hybrid metaheuristic algorithms with local search is atypical example. The global search explores the search space while the local search is utilized to refine the areas that may contain the global optimum.
- **Multiple purpose hybrids:** One primary algorithm is utilized to solve the problem while the sub-algorithm is applied to tune the parameters for the primary algorithm. For example, PSO can be applied to find the optimal value of mutation rate in GAs. Hereby, PSO is not solving the problem, but assisting in finding better solutions by searching for the optimal parameter for better performance.

Hybrid algorithms play a prominent role in improving the search capability of algorithms. Hybridization aims to combine the advantages of each algorithm to form a hybrid algorithm, while simultaneously trying to minimize any substantial disadvantage. The hybridization of different algorithms can exploit the complementary character of different optimization strategies, that is, hybrids are believed to be benefitted from synergy. In fact, choosing an adequate combination of algorithmic concepts can be the key for achieving top performance in solving many hard optimization problems. Unfortunately, developing an effective hybrid approach is in general a difficult task which requires expertise from different areas of optimization. Moreover, it is non-trivial to generalize, that is, a certain hybrid might work well for some specific problems, but it might work poorly for others. Nevertheless, there are hybridization types that have been shown to successful for many applications.

5.4 Disadvantages and Challenges of Hybrid Algorithms

Although hybrid algorithms offer great advantage of increasing the diversity in a population and hence enhancing the search capability of the developed hybrid algorithm, some drawbacks do exist which will be discussed in the following subsections.

5.4.1 Naming Convention

The inclusion of another algorithm usually leads to a naming issue. Some researchers adopt very different names to their hybrid algorithms. For instance, the GAAPI algorithm is an acronym for Hybrid Ant Colony-Genetic Algorithm, which is a bit confusing to other researchers. In comparison to both architectures mentioned in Section 5.2, the collaborative type of hybrid algorithm seems to create more sophisticated names.

5.4.2 Complexity of Hybrid Algorithm

In terms of algorithm architecture, the hybridization process usually creates extra components in the overall architecture of the hybrid algorithm. This increases the complexity of the hybrid algorithm. Due to a more complicated structure, hybrid algorithms have some resistance to be accepted by researchers.

5.4.3 Computational Speed

In many works, hybrid algorithms seem to improve results in terms of the overall convergence speed and accuracy. However, these convergence graphs are often plotted with respect to the number of iterations. This simply means that the faster convergence does not mean the true convergence rate because the hybrid usually uses a higher number of (internal or implicit) iterations. For example, for collaborative (sequential type) hybrid algorithm such as GA-PSO, a cycle, or one iteration comprises GA and PSO. For a fair comparison, this should be considered as two cycles instead of one in the convergence graph. To avoid this issue, the final run time should be utilized as a metric when comparing a hybrid algorithm with non-hybrid algorithms. Besides, due to a more complicated architecture in hybrid algorithms, the overhead arises alongside its complexity, often unavoidable. This affects the overall performance and thereby truncates its robustness. The time consumed by overheads should be taken into account for a fair comparison. Again, this is possible by recording the true number of iterations taken to reach a pre-specified target, though time complexity should be compared as well.

5.5 Examples of Hybrid Algorithms

In this section, we will give examples on some recent hybrid algorithms that have been performed on a wide range of numerical benchmark problems for global optimization. These hybrid algorithms are grouped either under collaborative (Table 5.1) or integrative (Table 5.2) categories. Collaborative algorithms (in Table 5.1) tend to have longer names compared to their integrative counterparts, as listed in Table 5.2. Also, from our analysis above, the n umber of journals published for the latter category is almost twice, compared to the first category. This means that the integrative hybrids are more widely accepted by researchers due to the simplicity in their development process.

Table 5.1: Some Recent collaborative hybrid algorithms, published after 2010

Abbreviation	**Full name**
GAAPI	Hybrid ant colony-genetic algorithm (GAAPI)
HPSO-BFGS	PSO-broyden-fletcher-goldfarb-shanno
PSO-ACO	Particle swarm optimization-ant colony optimization

Table 5.2: Some Recent integrative hybrid algorithms, published after 2010

Abbreviation	**Full name**
CPSO	Cellular particle swarm optimization
HABC	artificial bee colony
HMA	Hybrid memetic algorithm

5.6 Hybridization Models

5.6.1 Parallel Differential Evolution Proposals

By their nature, most metaheuristics are prone to parallelism, since most variation operations can be undertaken in parallel. But beyond the natural possibilities for parallelization, the difficulty lies in opting for parallel versions that,empowering the spirit of the metaheuristic, does not radically change the original method, nor be penalized by the portions of code that are not parallelizable. There are different approaches to parallelize population-based metaheuristics, and consequently DE, depending on the purpose to be achieved. On the one hand, it is desirable to improve the fitness value of the solution found with the sequential version. On the other hand, the execution time may be reduced, trying not to affect the quality of the solutions.The ideal case would be achieving both goals at the same time. In

recent years there have been developed different algorithmic versions to parallelize DE.Next subsections describe two parallel versions for DE. One follows what we call the subpopulationbased model, with ring interconnection topology and a semi-elitist replacement. The other,consist of an island model, having the entire population in each island,and a separate master process coordinating the system.

5.6.1.1 First algorithm: Subpopulation-based Model

The first model described is an “Standard Model with migration" version [11, 47]. It follows a master-worker scheme. The figure 5.4 describes this model. The master process initializes and divides the population (Pop. in the figure 5.4) into as many subpopulations (sp 1,....,sp n) as workers in the system, and sends one subpopulation per worker. Each worker receives its subpopulation, and starts with the evolutionary actions like a classic DE. The boxes in figure 5.4 named "SDE" represent worker processes, running a Subpopulation-based DE. The boxes with doted lines represent the processing node in which the worker is located. emigrant individuals is constituted by the best individual in the subpopulation (i.e.

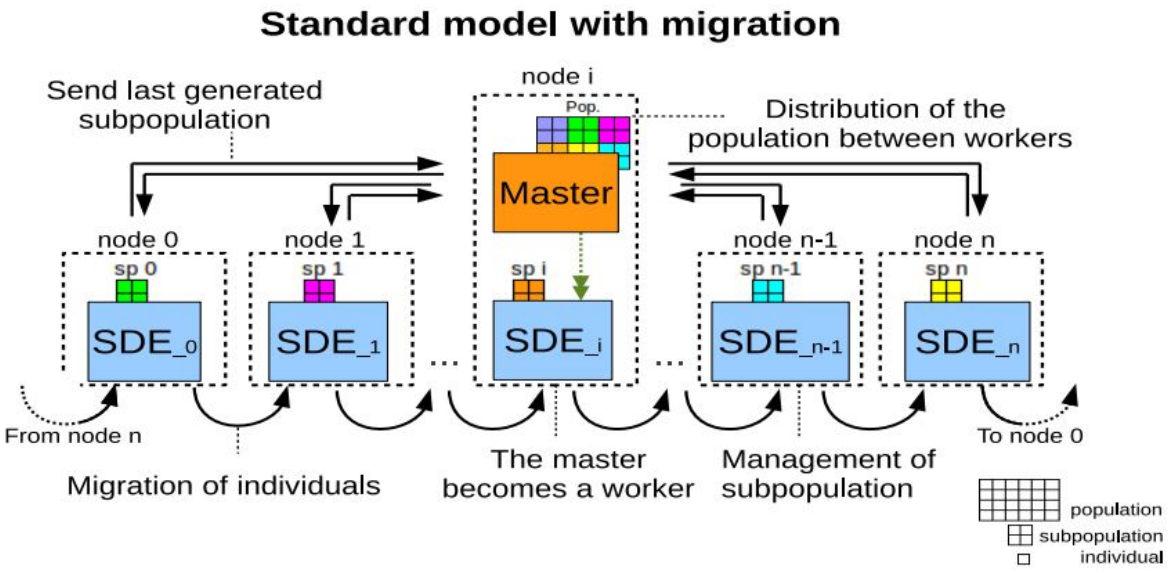


Figure 5.4: Subpopulation-based model: Population distribution between workers.

the one who has the best fitness value) and the rest is completed by means of a random selection. While selected individuals are migrating, the worker organizes its subpopulation to determine which will be the candidates to be replaced by the newly received ones. The replacement would be completely elitist, if the worst individuals were substituted by the bests of the source subpopulation. In our proposal, the set of emigrants is composed by some individuals randomly selected and the best member, thus balancing the diversity of the population. With this scheme, the master process remains idle while workers perform the evolutionary stages, and will have no further intervention until the merge of the results.

To avoid idle computing time, in our proposal the master process becomes another worker, and it applies DE with its corresponding portion of the initial population, like any of the other workers. As can be seen in figure 1, the master process shares the same processing node with a worker process, so as to maximize the use of the resources. Once the workers finish their generational evolutionary process, they communicate with the master in order to send the individuals of the last obtained generation. The master evaluates them, and stores the best individual found from all the subpopulations, then the process ends.

5.6.1.2 Second algorithm: Island Model

The second model described is framed within the classification “Algorithmic Level with cooperation" [53]. The figure 5.5 represents this model, whose processing scheme is described in the following. Multiple instances of DE are executed in parallel on different compute nodes, each one considering a different population of individuals (Pop. 1,..., Pop. n), and a different random initial seed. We call each computing node “an island". A master process is in charge of monitoring the system as a whole, and each worker process is dedicated to compute all the generations in that island. As can be seen, the master process is located in an exclusive computing node, so as to coordinate the system and to avoid delaying the response to the workers. Every certain migration rate, the worker communicates with a neighbor to send some individuals. The amount of individuals that migrate, in the same manner as was detailed before, is a certain percentage of the population. This percentage is a global value, calculated from the number of individuals in the island. Like the subpopulation model, the individuals to be migrated are the best member of the island plus other individuals randomly selected, and the received individuals will replace the worst individuals of the target population. After a migration phase and replacement process, the workers inform to the master which is the best individual found so far. The master receives this information and temporarily stores the best individual of all those who have been sent by the workers. Then, if the termination condition is met, the master sends a message to workers indicating the end of the process. Otherwise, the master informs to continue with their evolutionary process. This model has greater computational cost than the subpopulation based model, since the number of individuals of each island does not vary as the amount of workers increases. However, the island model can achieve an increment in the variability of the population when the number of individuals in each island is a considerable number. This can significantly promote the exploration of a larger search space, which may lead to better solutions quality.

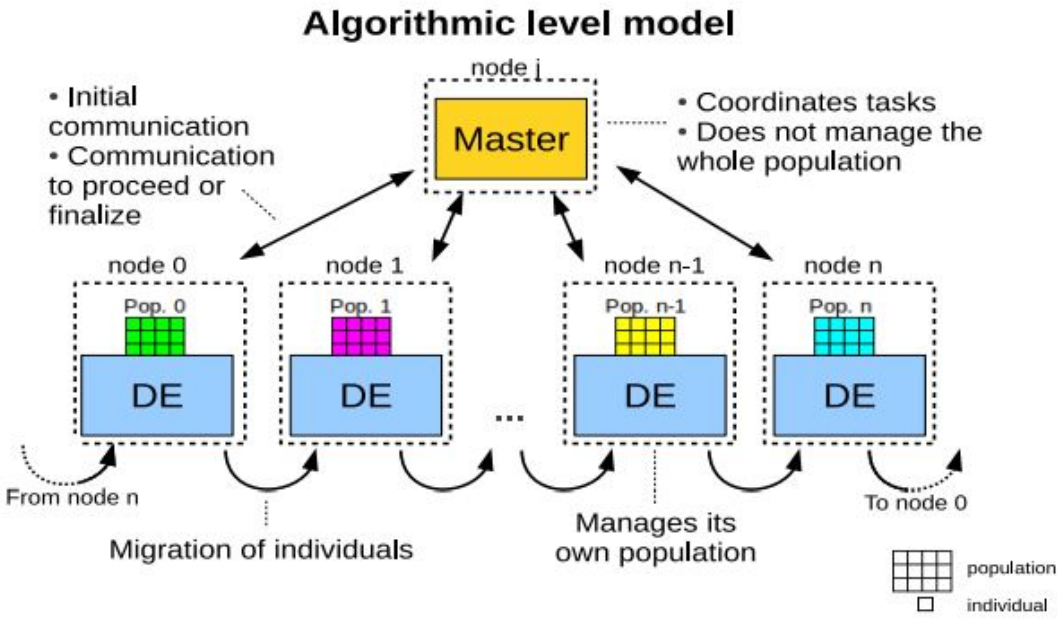


Figure 5.5: Island Model: independent or cooperating self-contained metaheuristics.

5.6.2 Comparison between Subpopulation based Model and Island Model

It was found that the subpopulation model reduces significantly the computing time, but the quality of the solutions is not the optimal that can be achieved. With the island model, the computing time is not reduced, because of the model characteristics, but the solutions quality is improved significantly with the increment of the number of workers involved in the process. This feature reflects the fact that the model explores a greater search space, since each island is configured with a different initial seed. The tendency indicates that better quality solutions would be obtained as the number of workers increases.

5.7 Proposed Hybrid 1: DE/best/1/exp and DE/rand/1/exp Hybrid using Island Model

1.INITIALIZATION:

The hybrid algorithm evolves a population of NP D-dimensional individual vectors i.e. solution candidate$X_i = (x_1, x_2, ..., x_D - 1, x_D) \in S$, i = 1,2,3,.....,NP.
from one generation to the next. The initial population should ideally cover the entire parameter of an individual vector with uniform distribution between the prescribed upper and lower parameter bounds $\max_j$ and $\min_j$using the following formula,

$$x_{i,j} = \min_j + rand[1,0].(x_{j,max} - x_{j,min}); \tag{5.1}$$

The fitness of the initial population is calculated. Then the whole population is equally distributed for Hybridization purpose.

2.MUTATION:

Mutation is done for each vectors means every member of the population.From the population NP we exclude the i_{th} member and then randomly select$r_1, r_2, ...r_n$,n = number of random number needed to do the operation.

The mutation formula varies with different variants of DE. Which has been shown in table 4.1 and as we are doing hybridization two of them will run in parallel with equal population that is NP/2.

$$u_{i,j} = \begin{cases} x_{r_{best},j} + F * \sum_{k=1}^{p}(x_{r_1{}^p,j} - x_{r_2{}^p,j}), & \text{if } U_j(0,1) < C_r \ or \ j \geq j_r \\ x_{i,j} & \text{otherwise} \end{cases} \tag{5.2}$$

$$u_{i,j} = \begin{cases} x_{r_3,j} + F * \sum_{k=1}^{p}(x_{r_1{}^p,j} - x_{r_2{}^p,j}), & \text{if } U_j(0,1) < C_r \ or \ j \geq j_r \\ x_{i,j} & \text{otherwise} \end{cases} \tag{5.3}$$

3.RECOMBINATION:

Exponential cross over can be used here. Which is actually a two point module. Choose an integer n(as starting point) and L(number of components donor actually contributes to the off spring) from the interval [1,D]

$$u_{j,i,G} = \begin{cases} V_{j,i,G}, \ for \ j = (n)_D.(n+1)_D.....(n+L-1)_D \\ X_{j,i,G}, \ for \ all \ other \ j \in [1,D] \end{cases} \tag{5.4}$$

Exploits linkages among the neighbouring decision variables.

4.SELECTION:

The competition is between the Trial offspring and parent vector. The one with the better fitness is admitted to the next generation population. Here fitness is basically the lower objective value.

5.MIGRATION:

The normal algorithm was supposed to jump to the Mutation step for the next generation. But In case of Hybrid algorithm As we are running two of the DE/best/1/exp and DE/rand/1/exp in parallel. After the end of the step 4 the migration operation will be done

for hybridization purpose. Three different Migration techniques are applied here. Any of them at a time.
This will be done in every iteration of the algorithm. After doing the migration it will jump to step 2 again.

5.7.1 Rand to Rand Migration 1

Only a single random value is exchanged. After doing all the steps of DE/best/1/exp and DE/rand/1/exp before starting the next generation a random index R1 from the new generation of variant 1 is selected. Another random index R2 is selected from the variant 2. The population of index R1 in variant 1 and the population of index R2 in variant 2 is exchanged.

5.7.2 Rand to Rand Migration 2

Two random values are exchanged. From the next generation of DE/best/1/exp two random variables are generated R1 and R2. Similarly two from DE/rand/1/exp , R3 and R4. The value of R1 index is exchanged with the value of R3 index and The value of R2 index is exchanged with the value of R4 index.

5.7.3 Best to Rand Migration 1

A random value is exchanged with the best. Variant 1 Generates a random value R1 after iteration 1. And the best member is found after the iteration, best1. Similarly R2 and best2 are generated from variant 2. Finally the value of R1 is exchanged with the best2 and the value of R2 is exchanged with the best1.

5.8 Proposed Hybrid 2: DE/best/1/bin and DE/rand/1/bin Hybrid using Island Model

1.INITIALIZATION:
This step is done similarly as proposed Hybrid 1.

2.MUTATION:
Mutation is done for each vectors means every member of the population.From the population NP we exclude the i_{th} member and then randomly select$r_1, r_2, ...r_n$,n = number of random number needed to do the operation.

The mutation formula varies with different variants of DE. Which has been shown in table 4.1 and as we are doing hybridization two of them will run in parallel with equal population that is NP/2. In this case DE/best/1/bin and DE/rand/bin have been used.

$$u_{i,j} = \begin{cases} x_{r_{best},j} + F * \sum_{k=1}^{p}(x_{r_1{}^p,j} - x_{r_2{}^p,j}), & \text{if } U_j(0,1) < C_r \ or \ j \geq j_r \\ x_{i,j} & \text{otherwise} \end{cases} \tag{5.5}$$

$$u_{i,j} = \begin{cases} x_{r_3,j} + F * \sum_{k=1}^{p}(x_{r_1{}^p,j} - x_{r_2{}^p,j}), & \text{if } U_j(0,1) < C_r \ or \ j \geq j_r \\ x_{i,j} & \text{otherwise} \end{cases} \tag{5.6}$$

3.RECOMBINATION:
Uniform/Binomial cross over is used for recombination. Components of the donor vector enters into the trial off spring vector in the following:
Let j_{rand} be a randomly chosen integer between 1,...,D.

$$u_{j,i,G} = \begin{cases} V_{j,i,G}, \ if(rand_{i,j}[0,1 \leq Cr or j = j_{rand}) \\ X_{j,i,G}, \ otherwise \end{cases} \tag{5.7}$$

4.SELECTION:
Survival of the fitter principle in selection is used. The trial offspring vector is compared with the target(parent) vector. And the one with the better fitness is admitted to the next generation population.

$$X_{i,G+1} = \begin{cases} U_{i,G}, \ if \ f(U_{i,G}) \leq f(X_{i,G}) \\ X_{i,G}, \ if \ f(U_{i,G}) > f(X_{i,G}) \end{cases} \tag{5.8}$$

5.MIGRATION:
In this case of Hybrid algorithm As we are running two of the DE/best/1/bin and DE/rand/1/bin in parallel. After the end of the step 4 the migration operation will be done for hybridization purpose. Three different Migration techniques are applied here. Any of them at a time. Similar to the previous one.

5.8.1 Rand to Rand Migration 1

Only a single random value is exchanged. After doing all the steps of DE/best/1/bin and DE/rand/1/bin before starting the next generation a random index R1 from the new generation of variant 1 is selected. Another random index R2 is selected from the variant 2. The

population of index R1 in variant 1 and the population of index R2 in variant 2 is exchanged.

5.8.2 Rand to Rand Migration 2

Two random values are exchanged. From the next generation of DE/best/1/bin two random variables are generated R1 and R2. Similarly two from DE/rand/1/bin , R3 and R4. The value of R1 index is exchanged with the value of R3 index and The value of R2 index is exchanged with the value of R4 index.

5.8.3 Best to Rand Migration 1

A random value is exchanged with the best. Variant 1 Generates a random value R1 after iteration 1. And the best member is found after the iteration, best1. Similarly R2 and best2 are generated from variant 2. Finally the value of R1 is exchanged with the best2 and the value of R2 is exchanged with the best1.

5.9 Functions Used for the Experiment

Among the 30 benchmark functions we have used 3 functions. 1 of those is unimodal and other 2 are multimodal functions. Their name, domain and equations are given in the table below.

Table 5.3: Functions used for Experiment

No.	Function name	S	F_{min}	Function
f_1	*Sphere*	$[-100, 100]^D$	0	$\sum_{i=1}^{D} x_i^2$
f_2	*Schwefel 2.2*	$[-10, 10]^D$	0	$\sum_{i=1}^{D} \|x\| + \prod_{i=1}^{D} x_i$
f_{10}	*Rastrigin*	$[-5.12, 5.12]^D$	0	$\sum_{i=1}^{D} [x_i^2$ -10cos$(2\Pi x_i) + 10]$
f_{14}	*Griewank*	$[-600, 600]^D$	0	$\frac{1}{4000} \sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} cos(\frac{x_I}{\sqrt{i}})$
f_{29}	*Michalewicz*	$[0, \Pi]^D$	-9.66	$-\sum_{i=1}^{D} sin(x_i)$ $(sin(ix_i^2/\pi))^{2m}; m = 10$

5.10 Setup for the Experiment

In all the experiments, the same self-adaptive method, the same population size, the same tournament size for selection, the same initialization and same initial population is used for Hybrids.

5.11 Results of the Experiment

5.11.1 *Sphere Function*(f_1)

Table 5.4: Optimization of *Sphere function*(f_1) by DE variants between DE/best/1/exp and DE/rand/1/exp Hybrid using island model on different number of generations

Generation	DE/best/1/exp	DE/rand/1/exp	Rand to Rand Migration 1	Rand to Rand Migration 2	Best to Rand Migration 1
1600	6.41e-009	2.20e-002	1.77e-005	5.33e-005	2.63e-012
1100	3.27e-004	9.80	6.23e-002	1.67e-001	1.46e-011
900	2.45e-002	1.16e+002	1.85	2.51	8.53e-009
700	1.65	1.30e+003	3.01e+001	5.81e+001	3.16e-005
500	8.13e+001	1.40e+004	8.31e+002	7.40e+002	4.11e-002
300	7.26e+003	1.04e+005	2.00e+004	2.78e+004	9.42e+001
200	7.99e+004	3.30e+005	2.06e+005	1.97e+005	3.23e+003

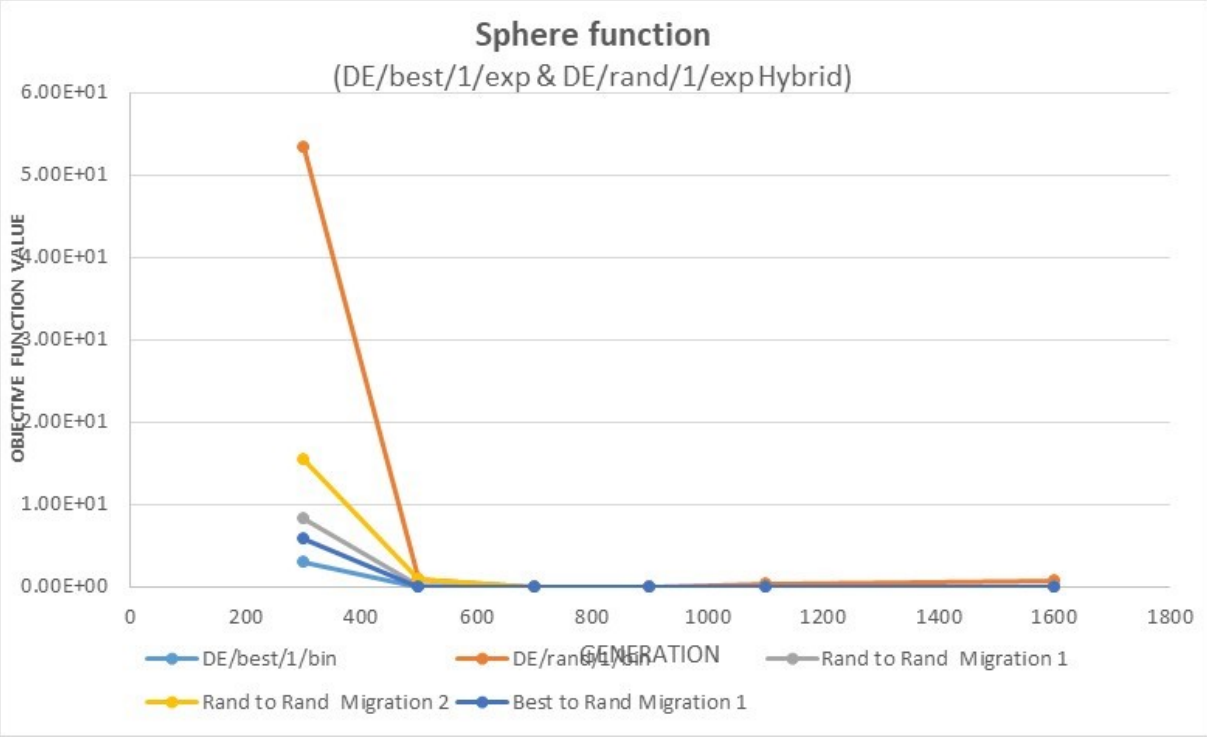


Figure 5.6: Comparison of DE/best/1/exp, DE/rand/1/exp, Rand to Rand Migration 1, Rand to Rand Migration 2, Best to Rand Migration 1 on *Sphere function* (f_1).

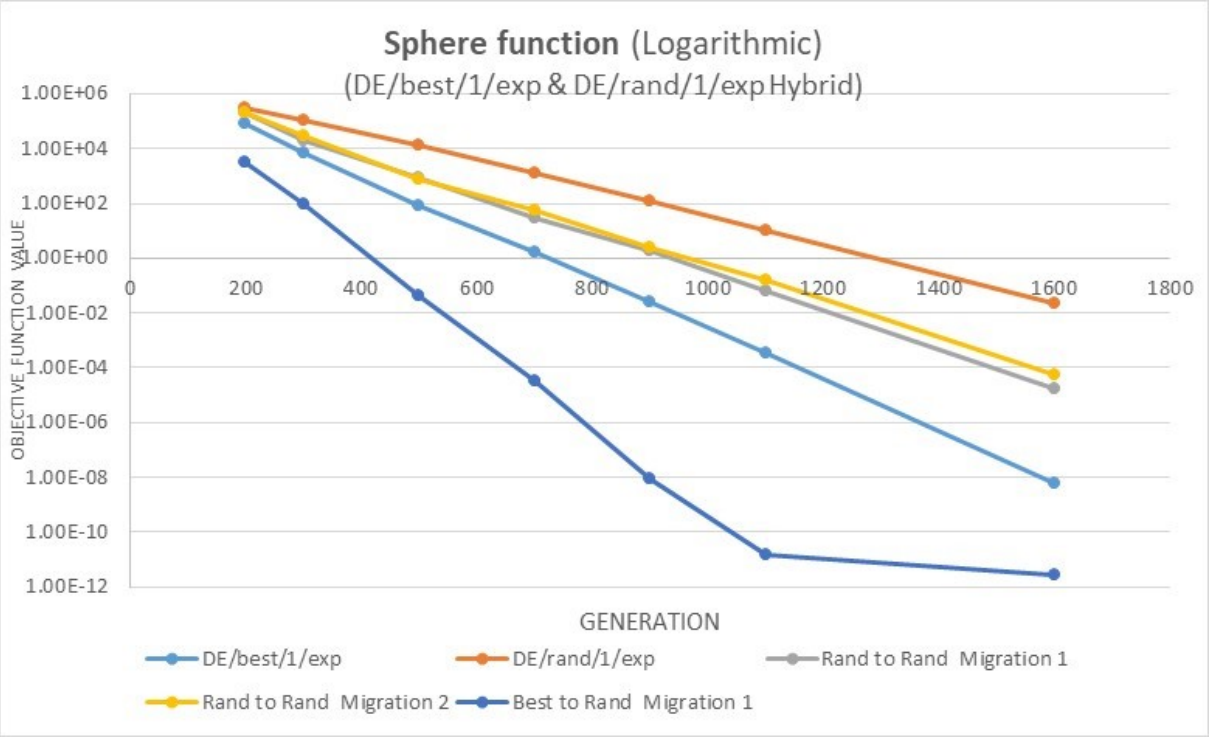


Figure 5.7: Comparison of DE/best/1/exp, DE/rand/1/exp, Rand to Rand Migration 1, Rand to Rand Migration 2, Best to Rand Migration 1 on *Sphere function*(f_1) in logarithmic scale.

Table 5.5: Optimization of *Sphere function*(f_1) by DE variants between DE/best/1/bin and DE/rand/1/bin Hybrid using island model on different number of generations

Generation	**DE/best/1/bin**	**DE/rand/1/bin**	**Rand to Rand Migration 1**	**Rand to Rand Migration 2**	**Best to Rand Migration 1**
1600	4.24e-008	5.40e-002	1.83e-006	1.00e-004	2.51e-014
1100	9.02e-004	1.37e+001	8.91e-003	1.23e-001	4.46e-008
900	8.93e-002	1.54e+002	4.53e-001	2.32	2.95e-006
700	5.23	1.65e+003	5.31	4.49e+001	1.72e-001
500	2.36e+002	1.69e+004	3.68e+002	5.16e+002	1.37e+001
300	2.84e+004	1.38e+005	1.52e+004	8.03e+003	2.15e+003
200	1.77e+005	5.40e+005	7.83e+004	5.47e+004	1.54e+004

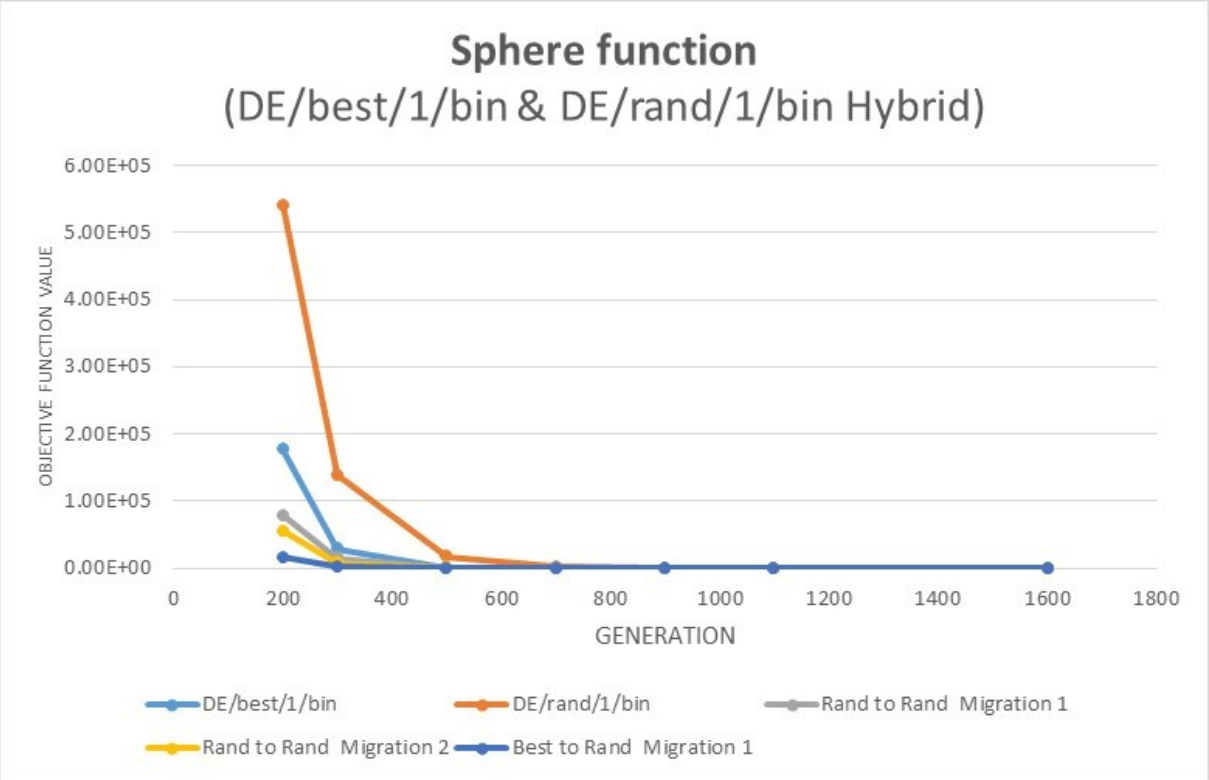


Figure 5.8: Comparison of DE/best/1/bin, DE/rand/1/bin, Rand to Rand Migration 1, Rand to Rand Migration 2, Best to Rand Migration 1 on *Sphere function*(f_1) .

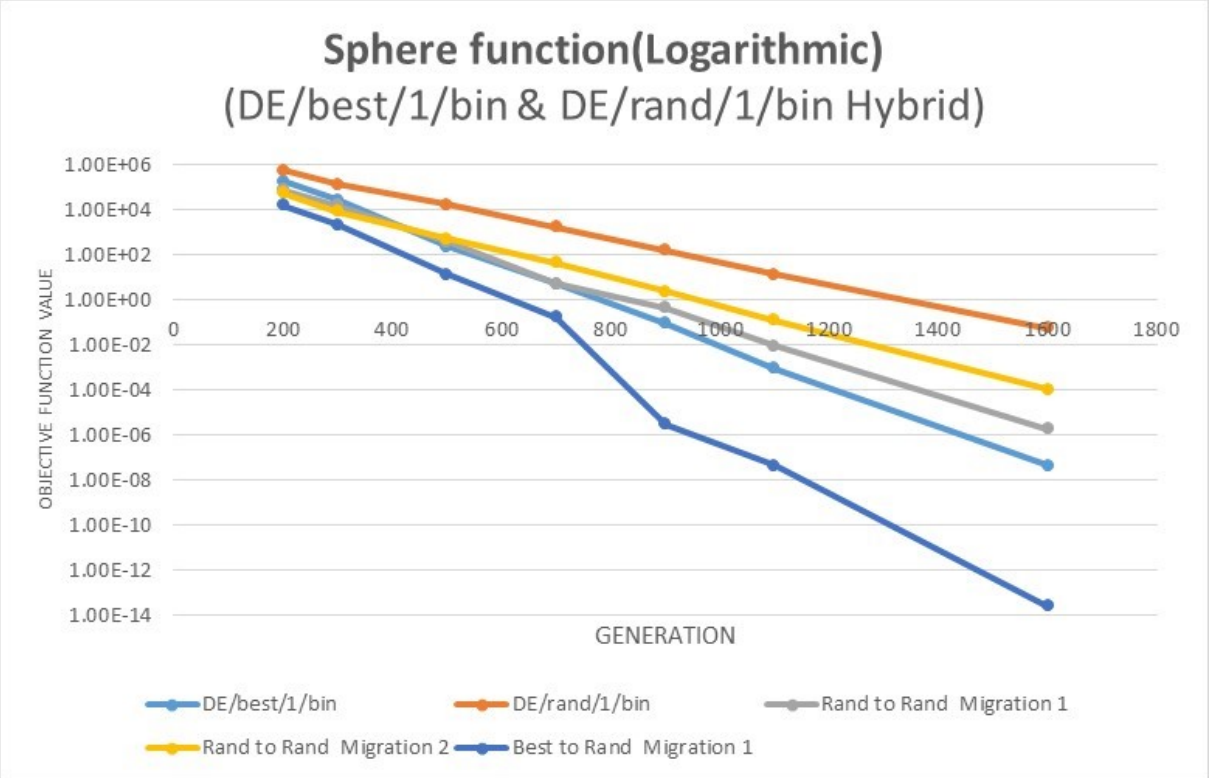


Figure 5.9: Comparison of DE/best/1/bin, DE/rand/1/bin, Rand to Rand Migration 1, Rand to Rand Migration 2, Best to Rand Migration 1 on *Sphere function*(f_1) in logarithmic scale.

5.11.2 *Schewefels' function*(f_2)

Table 5.6: Optimization of *Schewefels' function*(f_2) by DE variants between DE/best/1/exp and DE/rand/1/exp Hybrid using island model on different number of generations

Generation	DE/best/1/exp	DE/rand/1/exp	Rand to Rand Migration 1	Rand to Rand Migration 2	Best to Rand Migration 1
1600	3.02e-008	8.32e-003	3.11e-003	1.02e-005	9.33e-011
1100	1.30e-004	5.69e-001	3.40e-001	8.15e-003	9.21e-010
900	4.40e-003	1.46	1.69	9.47e-002	2.07e-007
700	1.23e+001	8.08e-001	8.12	1.00	4.35e-005
500	2.38e+001	1.38	3.28	9.36	1.21e-002
300	3.80	2.23e+001	4.40e+001	6.55e+001	3.46

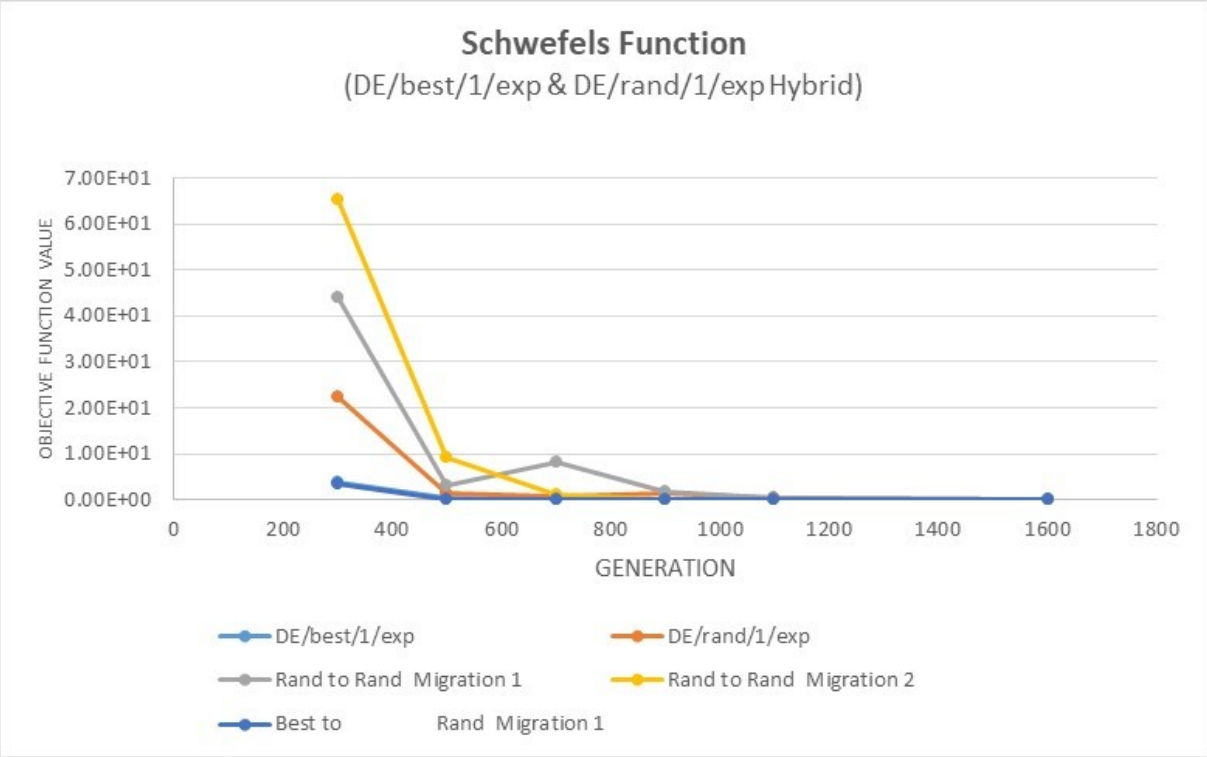


Figure 5.10: Comparison of DE/best/1/exp, DE/rand/1/exp, Rand to Rand Migration 1, Rand to Rand Migration 2, Best to Rand Migration 1 on *Schewefels' function*(f_2)

.

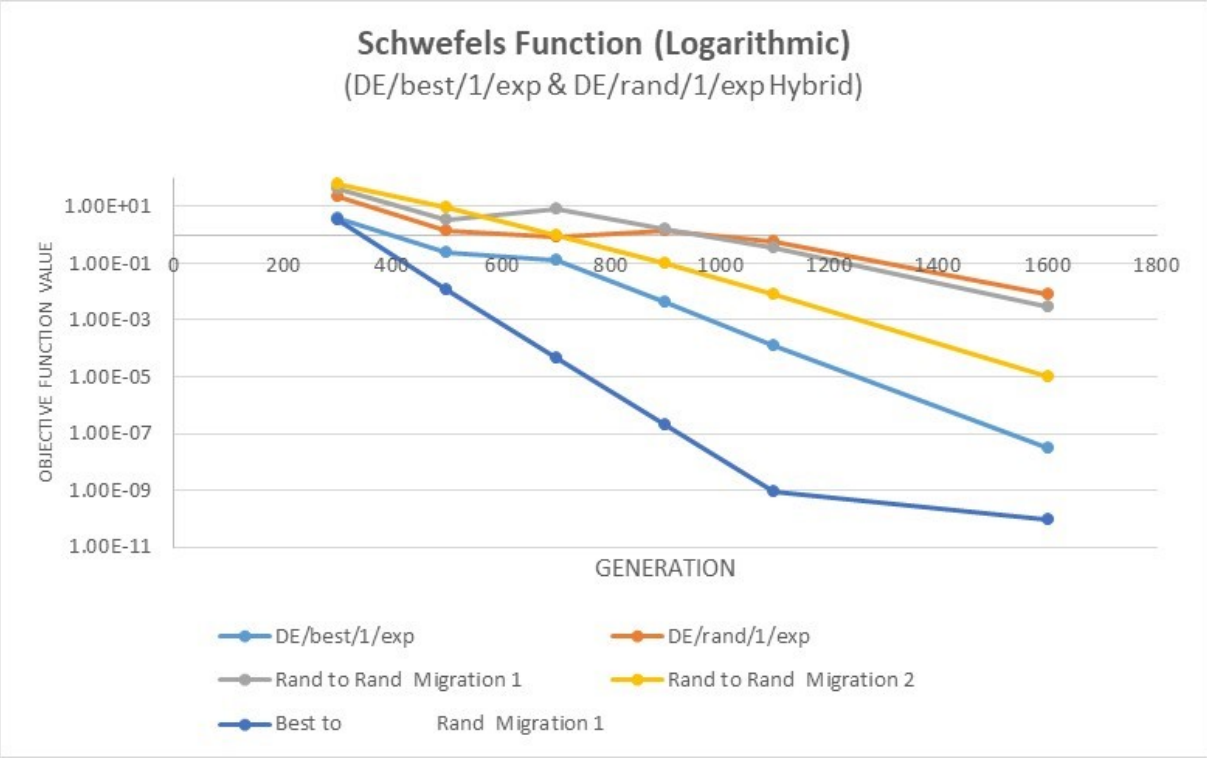


Figure 5.11: Comparison of DE/best/1/exp, DE/rand/1/exp, Rand to Rand Migration 1, Rand to Rand Migration 2,Best to Rand Migration 1 on *Schewefels' function*(f_2) in logarithmic scale.

Table 5.7: Optimization of *Schewefels' function*(f_2) by DE variants between DE/best/1/bin and DE/rand/1/bin Hybrid using island model on different number of generations

Generation	**DE/best/1/bin**	**DE/rand/1/bin**	**Rand to Rand Migration 1**	**Rand to Rand Migration 2**	**Best to Rand Migration 1**
1600	1.89e-014	6.27e-001	7.80e-010	6.08e-009	7.09e-016
1100	7.27e-009	3.71e-001	6.35e-007	1.60e-005	6.70e-010
900	1.20e-006	1.22e-003	4.50e-005	7.04e-004	1.70e-007
700	1.01e-004	2.33e-002	1.10e-003	1.73e-002	4.19e-005
500	1.54e-002	8.70e-001	5.52e-002	8.77e-001	9.12e-003
300	2.90	5.33e+001	8.25	1.53e+001	5.86

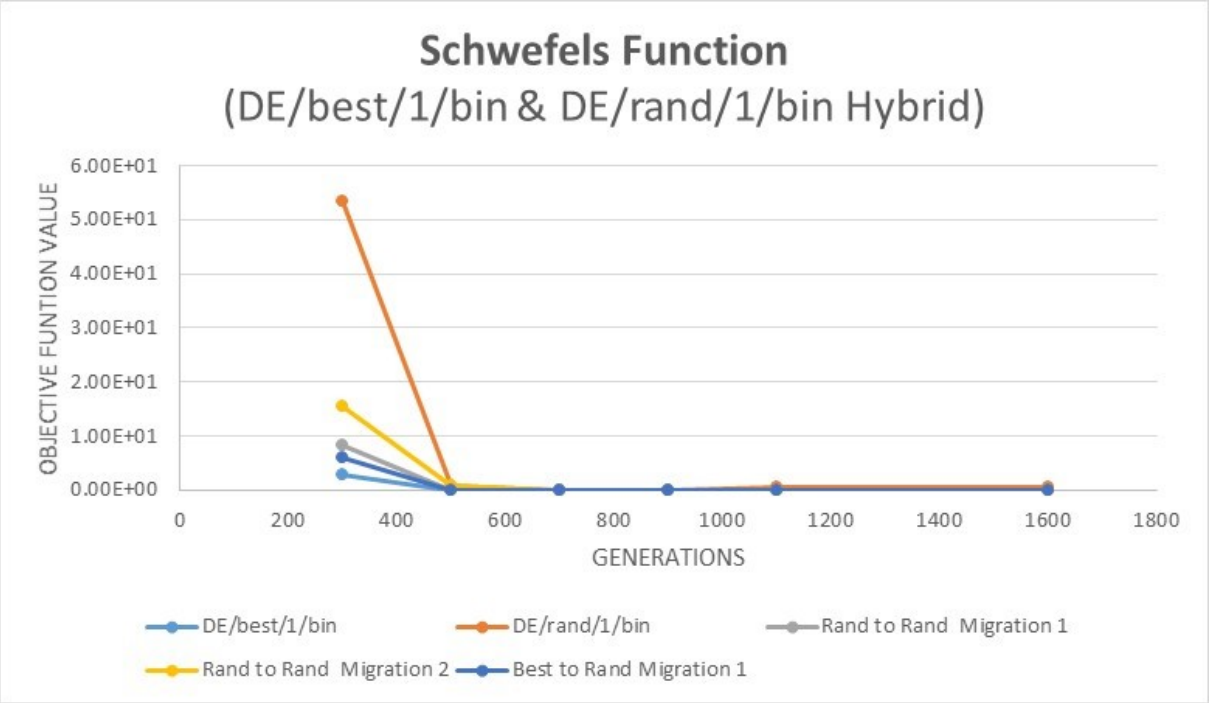


Figure 5.12: Comparison of DE/best/1/bin, DE/rand/1/bin, Rand to Rand Migration 1, Rand to Rand Migration 2, Best to Rand Migration 1 on *Schewefels' function*(f_2)

.

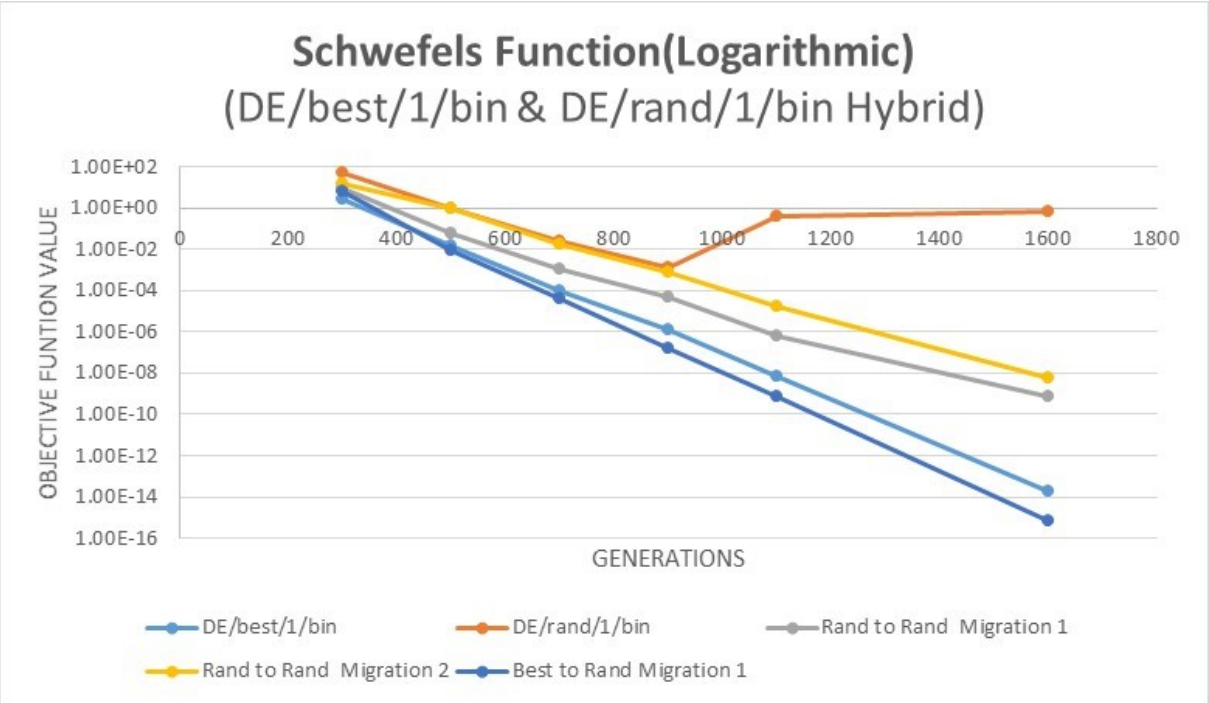


Figure 5.13: Comparison of DE/best/1/bin, DE/rand/1/bin, Rand to Rand Migration 1, Rand to Rand Migration 2, Best to Rand Migration 1 on *Schewefels' function*(f_2) in logarithmic scale.

5.11.3 *Rastrigin Function*(f_{10})

Table 5.8: Optimization of *Rastrigin function*(f_{10}) by DE variants between DE/best/1/exp and DE/rand/1/exp Hybrid using island model on different number of generations

Generation	DE/best/1/exp	DE/rand/1/exp	Rand to Rand Migration 1	Rand to Rand Migration 2	Best to Rand Migration 1
1600	0	2.45e-008	3.91e-005	6.56e-004	1.03e-001
1100	1.82e-008	9.07e-004	9.32e-002	1.25e-001	1.33e-001
900	4.85e-006	5.49e-002	2.58	2.43	1.32e-001
700	1.90e-003	2.92	4.83e+001	3.52e+001	1.32e-001
500	8.77e-001	1.91e+002	1.10e+003	1.26e+003	1.33e-001
300	3.58e+002	1.30e+004	2.47e+004	2.80e+004	8.15e+001

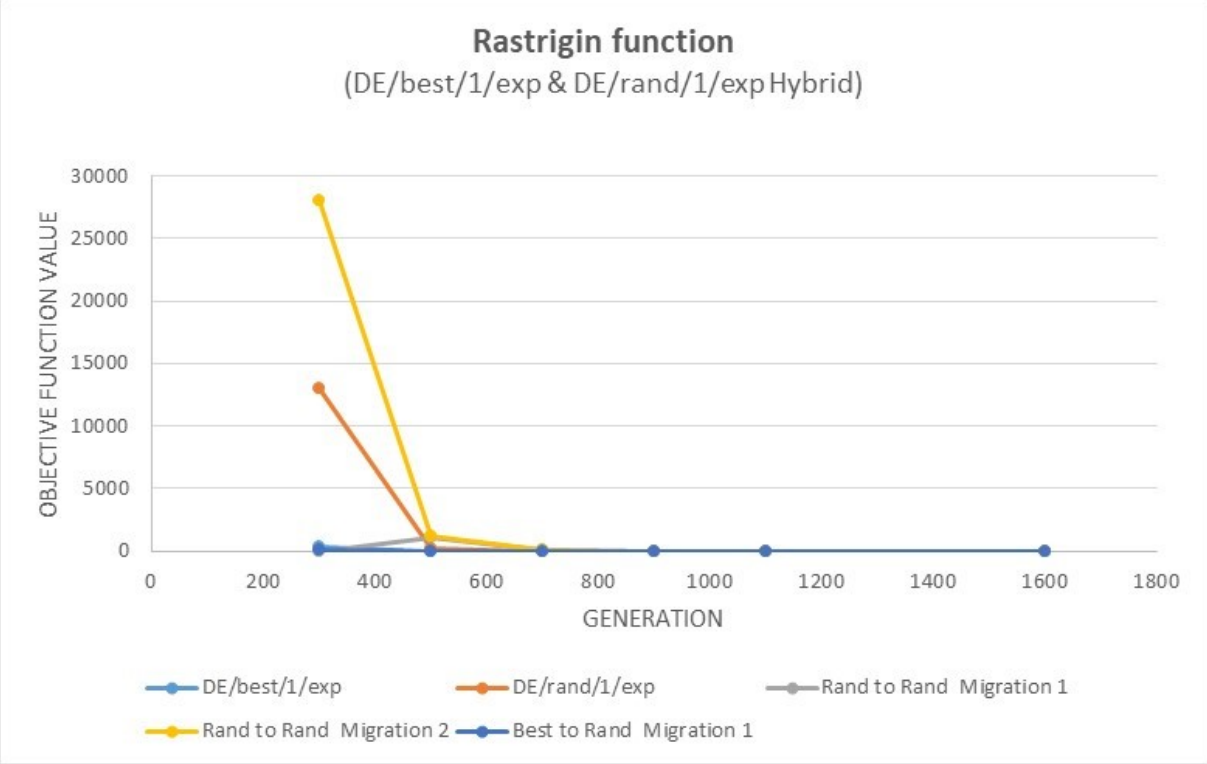


Figure 5.14: Comparison of DE/best/1/exp, DE/rand/1/exp, Rand to Rand Migration 1, Rand to Rand Migration 2, Best to Rand Migration 1 on *Rastrigin Function*(f_{10}).

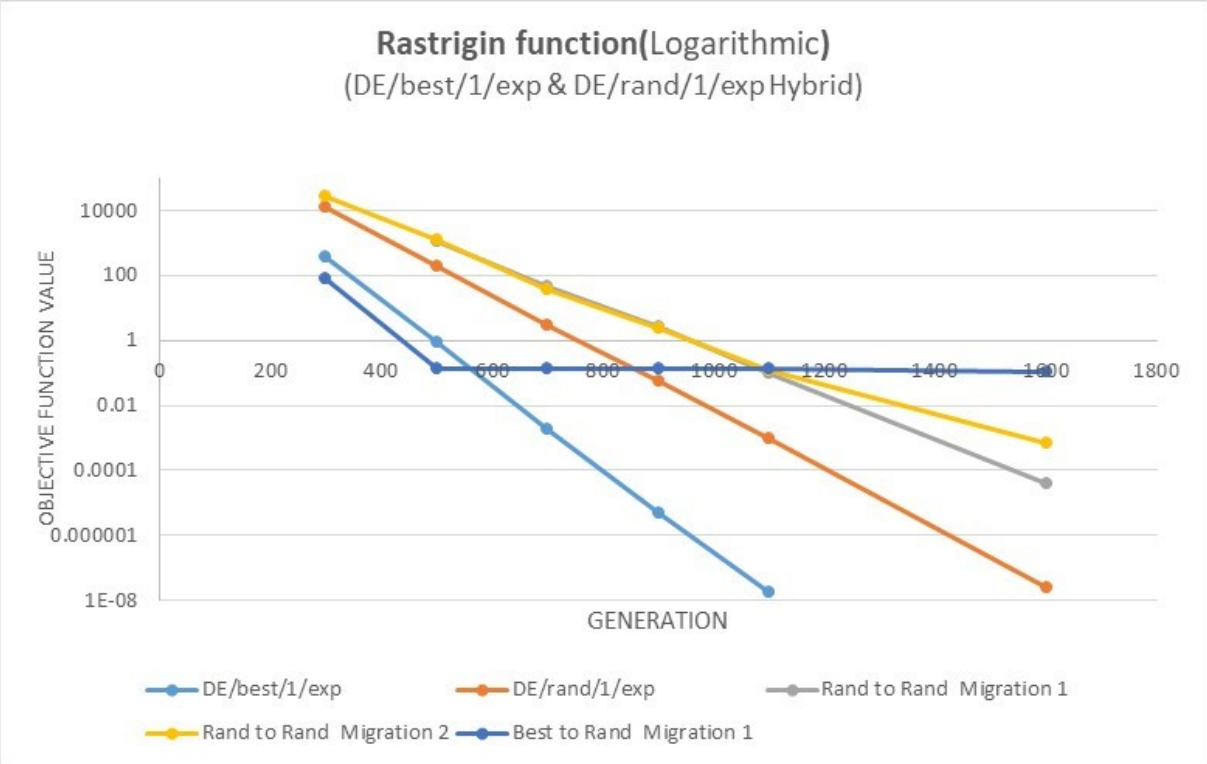


Figure 5.15: Comparison of DE/best/1/exp, DE/rand/1/exp, Rand to Rand Migration 1, Rand to Rand Migration 2, Best to Rand Migration 1 on *Rastrigin Function*(f_{10}) in logarithmic scale.

Table 5.9: Optimization of *Rastrigin function*(f_{10}) by DE variants between DE/best/1/bin and DE/rand/1/bin Hybrid using island model on different number of generations

Generation	**DE/best/1/bin**	**DE/rand/1/bin**	**Rand to Rand Migration 1**	**Rand to Rand Migration 2**	**Best to Rand Migration 1**
1600	0	9.86e+003	1.49e-006	1.39e-004	1.06e-014
1100	4.97e-014	6.20e+004	9.97e-003	1.84e-001	8.03e-008
900	5.57e-010	1.47e+005	1.88e-001	6.83	1.11e-006
700	1.85e-006	1.99e+005	5.50	1.42e+002	5.66e-004
500	4.89e-003	5.75e+005	2.77e+002	2.07e+003	9.79e-001
300	2.38e+001	1.15e+006	1.20e+004	7.19e+003	2.68e+002

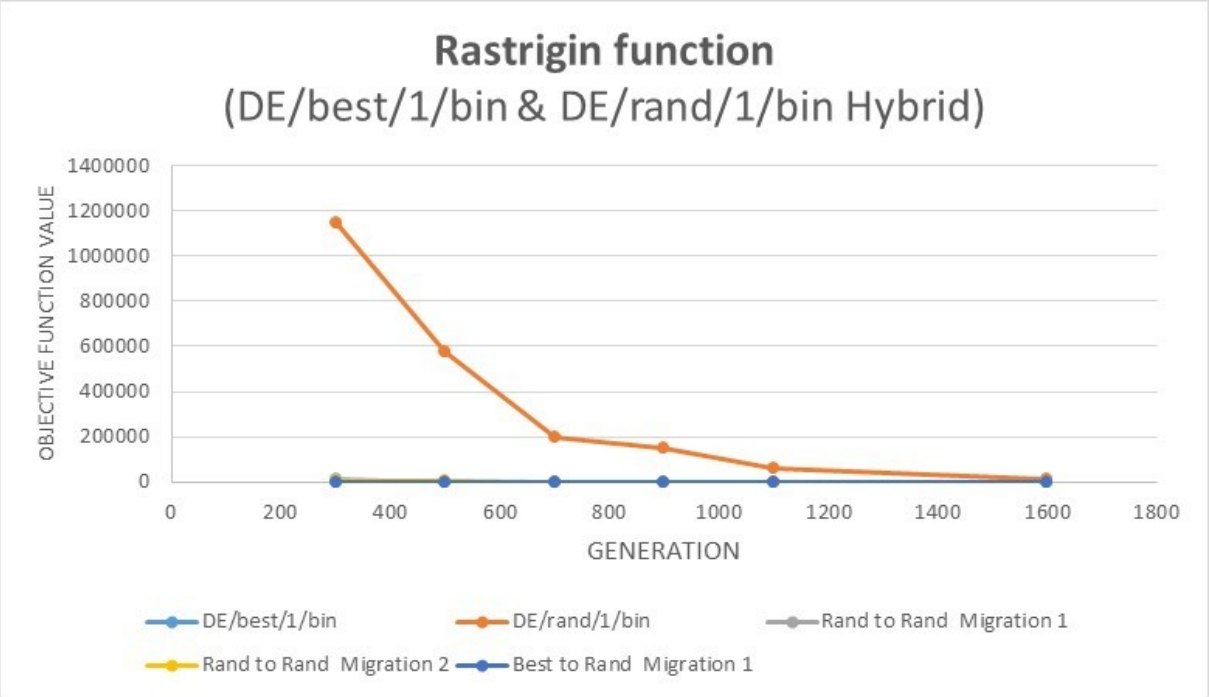


Figure 5.16: Comparison of DE/best/1/bin, DE/rand/1/bin, Rand to Rand Migration 1, Rand to Rand Migration 2, Best to Rand Migration 1 on *Rastrigin Function*(f_{10}).

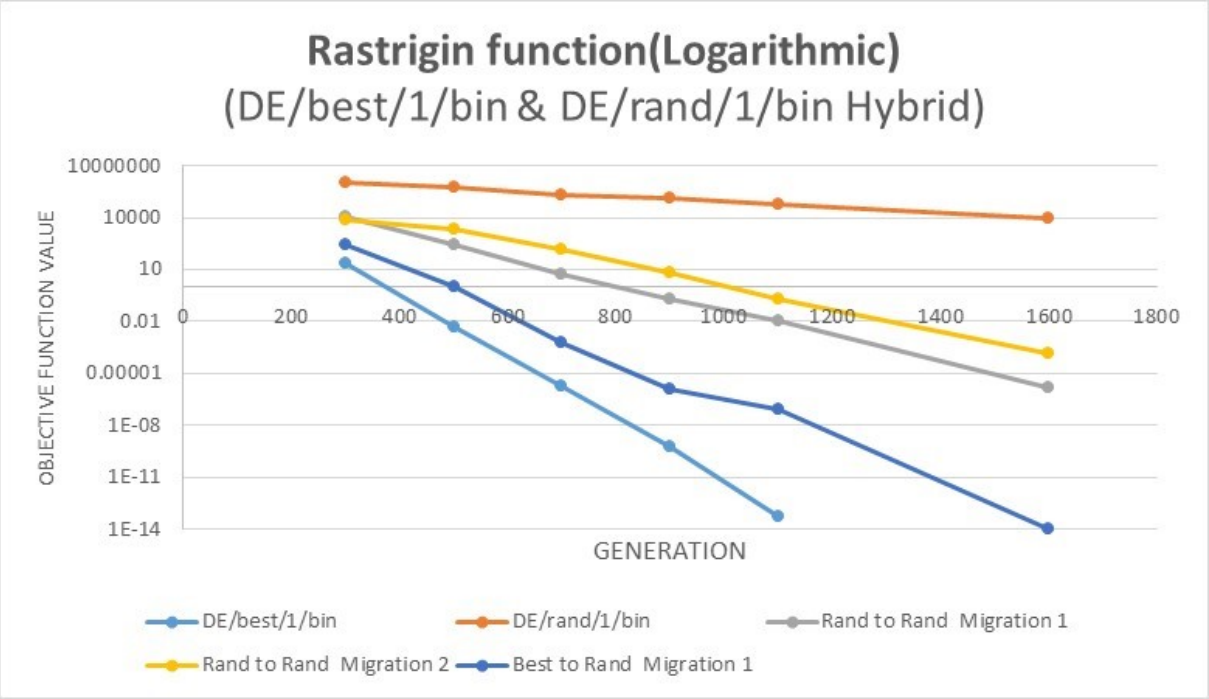


Figure 5.17: Comparison of DE/best/1/bin, DE/rand/1/bin, Rand to Rand Migration 1, Rand to Rand Migration 2, Best to Rand Migration 1 on *Rastrigin Function*(f_{10}) in logarithmic scale.

5.11.4 *Griewank Function*(f_{14})

Table 5.10: Optimization of *Griewank function*(f_{14}) by DE variants between DE/best/1/exp and DE/rand/1/exp Hybrid using island model on different number of generations

Generation	DE/best/1/exp	DE/rand/1/exp	Rand to Rand Migration 1	Rand to Rand Migration 2	Best to Rand Migration 1
1600	0	2.09e-004	3.44e-11	7.06e-012	0
1100	1.02e-010	5.07e-002	6.01e-006	5.06e-005	0
900	4.03e-008	2.31e-001	4.18e-004	1.36e-004	2.61e-016
700	3.24e-005	9.33e-001	1.18e-002	2.23e-002	2.91e-009
500	1.71e-002	1.43	2.78e-001	5.53e-001	1.36e-005
300	1.06	1.06e+001	1.17	1.36	2.05e-001

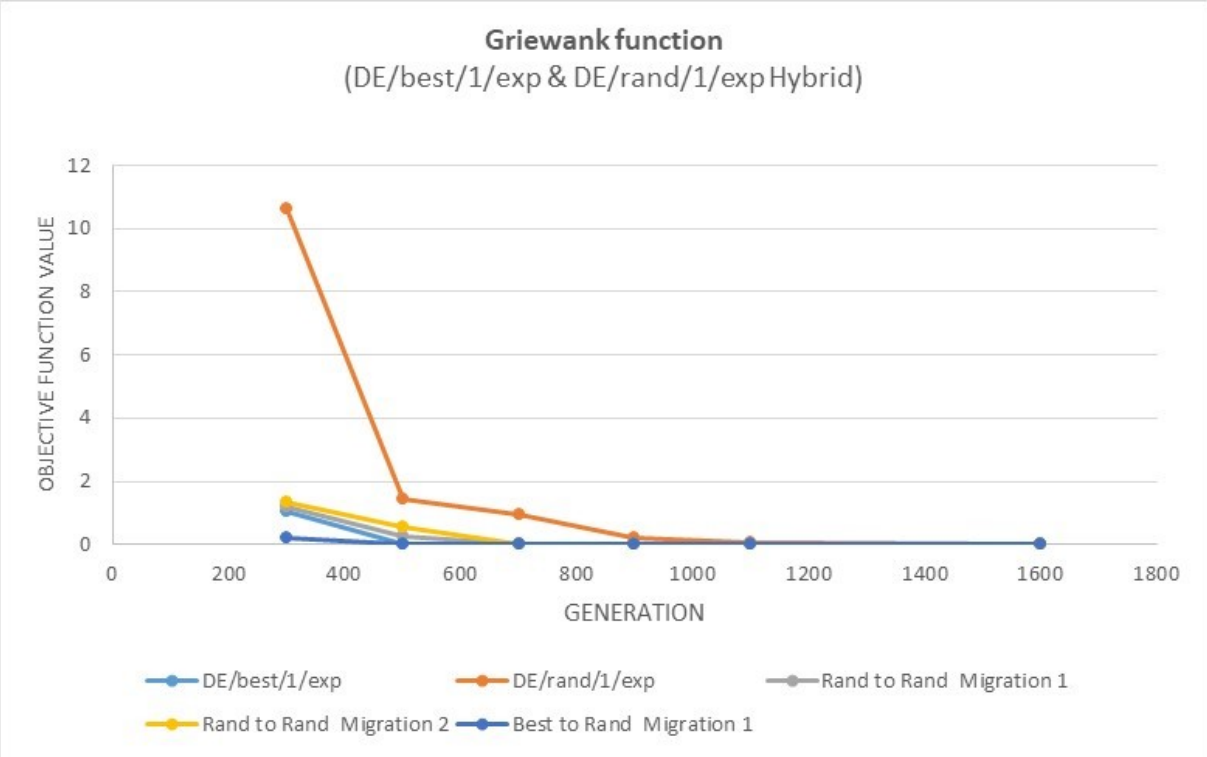


Figure 5.18: Comparison of DE/best/1/exp, DE/rand/1/exp, Rand to Rand Migration 1, Rand to Rand Migration 2, Best to Rand Migration 1 on *Griewank*(f_{14}).

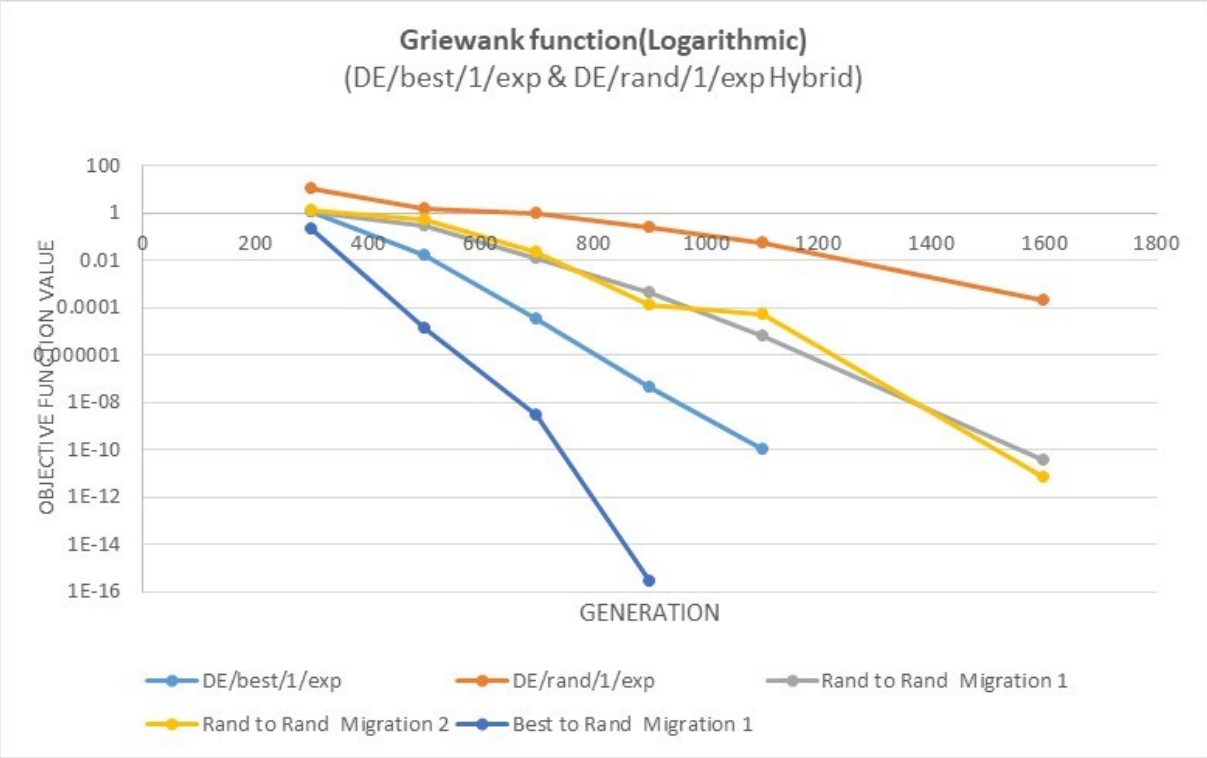


Figure 5.19: Comparison of DE/best/1/exp, DE/rand/1/exp, Rand to Rand Migration 1, Rand to Rand Migration 2, Best to Rand Migration 1 on *Griewank*(f_{14}) in logerithmic scale.

Table 5.11: Optimization of *Griewank function*(f_{14}) by DE variants between DE/best/1/bin and DE/rand/1/bin Hybrid using island model on different number of generations

Generation	**DE/best/1/bin**	**DE/rand/1/bin**	**Rand to Rand Migration 1**	**Rand to Rand Migration 2**	**Best to Rand Migration 1**
1600	1.23e-002	1.32	2.01e-012	1.52e-011	0
1100	1.23e-002	6.41	1.22e-006	1.42e-006	0
900	1.23e-002	1.63e+001	8.60e-005	9.38e-005	8.22e-15
700	1.23e-002	3.89e+001	1.48e-002	2.57e-002	9.79e-011
500	1.23e-002	9.73e+001	1.49e-002	4.09e-002	6.18e-006
300	9.44e-002	2.54e+002	4.42e-002	8.02e-001	5.97e-002

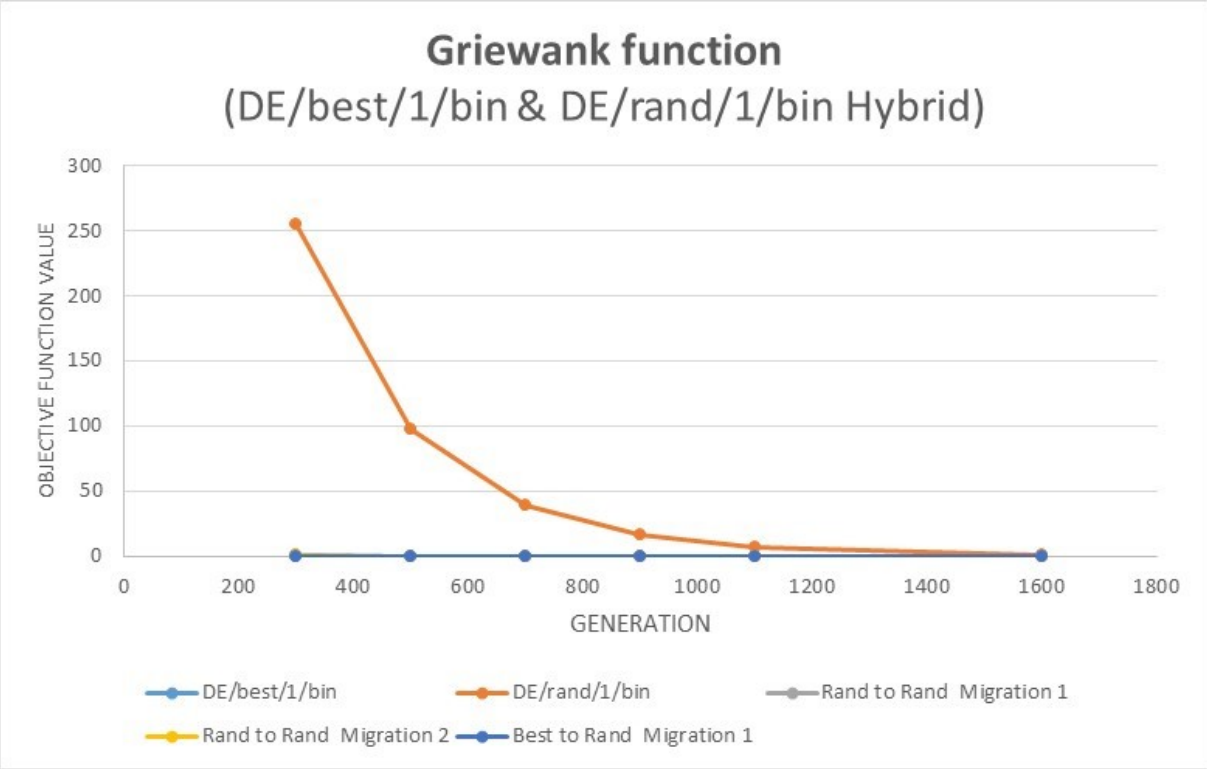


Figure 5.20: Comparison of DE/best/1/bin, DE/rand/1/bin, Rand to Rand Migration 1, Rand to Rand Migration 2, Best to Rand Migration 1 on *Griewank*(f_{14}).

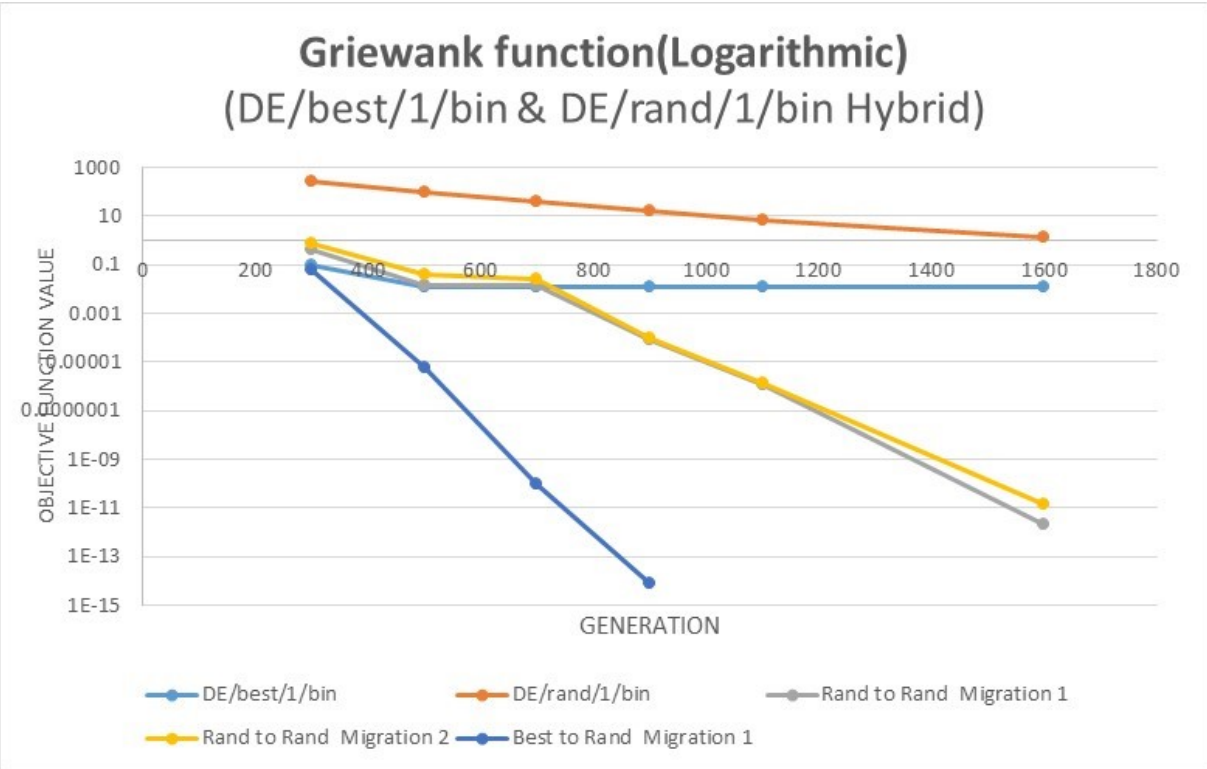


Figure 5.21: Comparison of DE/best/1/bin, DE/rand/1/bin, Rand to Rand Migration 1, Rand to Rand Migration 2, Best to Rand Migration 1 on *Griewank*(f_{14}) in logerithmic scale.

5.11.5 *Michalewicz Function* f_{29}

Table 5.12: Optimization of *Michalewicz*(f_{29}) by DE variants between DE/best/1/exp and DE/rand/1/exp Hybrid using island model on different number of generations

Generation	DE/best/1/exp	DE/rand/1/exp	Rand to Rand Migration 1	Rand to Rand Migration 2	Best to Rand Migration 1
1600	-1.88e+001	-1.93e+001	-1.85e+001	-1.89e+001	-2.38e+001
1100	-1.68e+001	-1.78e+001	-1.76e+001	-1.71e+001	-2.38e+001
900	-1.63e+001	-1.74e+001	-1.75e+001	-1.70e+001	-2.38e+001
700	-1.53e+001	-1.65e+001	-1.55e+001	-1.62e+001	-2.38e+001
500	-1.36e+001	-1.53e+001	-1.41e+001	-1.50e+001	-2.38e+001
300	-1.20e+001	-1.36e+001	-1.20e+001	-1.36e+001	-1.93e+001

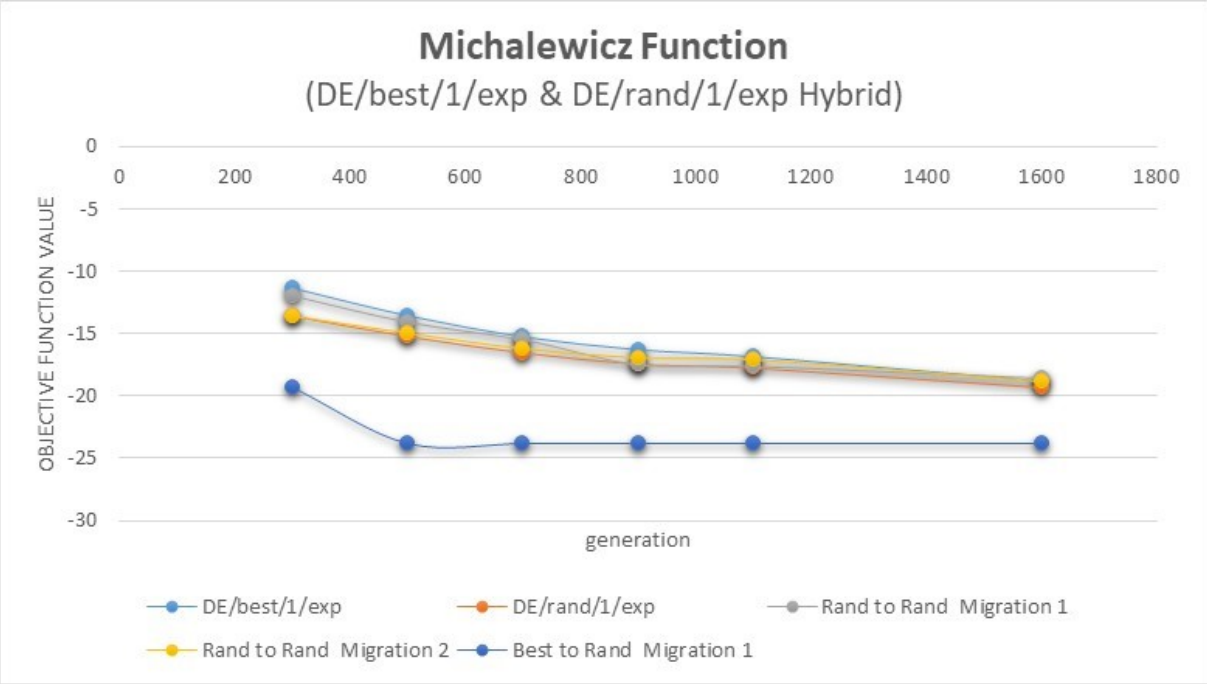


Figure 5.22: Comparison of DE/best/1/exp, DE/rand/1/exp, Rand to Rand Migration 1, Rand to Rand Migration 2, Best to Rand Migration 1 on *Michalewicz Function*(f_{29}).

Table 5.13: Optimization of *Michalwickz function*(f_{29}) by DE variants between DE/best/1/bin and DE/rand/1/bin Hybrid using island model on different number of generation

Generation	DE/best/1/bin	DE/rand/1/bin	Rand to Rand Migration 1	Rand to Rand Migration 2	Best to Rand Migration 1
1600	-2.35e+001	-8.39	-2.21e+001	-1.49e+001	-1.90e+001
1100	-2.35e+001	-6.42	-2.21e+001	-1.35e+001	-1.90e+001
900	-2.35e+001	-6.40	-2.21e+001	-1.30e+001	-1.90e+001
700	-2.19e+001	-6.40	-2.17e+001	-1.30e+001	-1.90e+001
500	-1.82e+001	-6.41	-1.77e+001	-1.30e+001	-1.90e+001
300	-1.33e+001	-6.41	-1.02e+001	-1.30e+001	-1.74e+001

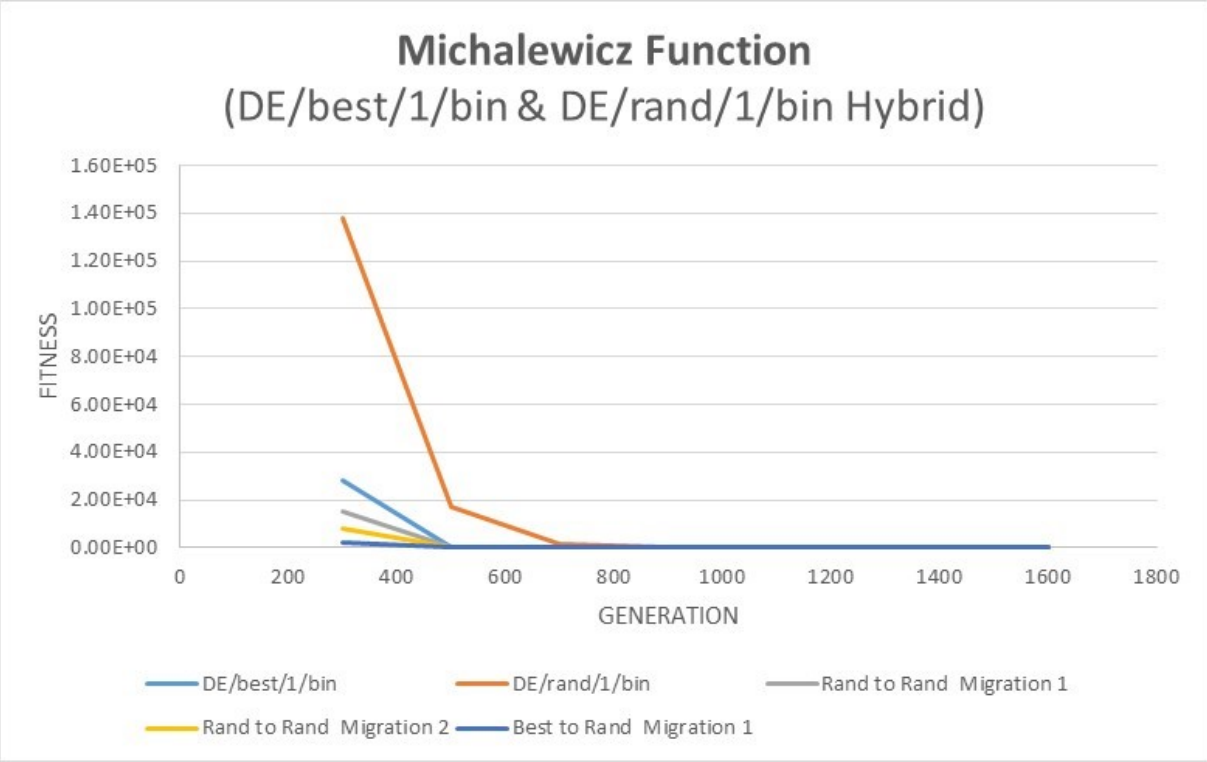


Figure 5.23: Comparison of DE/best/1/bin, DE/rand/1/bin, Rand to Rand Migration 1, Rand to Rand Migration 2, Best to Rand Migration 1 on *Michalewicz Function*(f_{29}).

5.12 Comparative Data Table

Table 5.14: Table for Mean Absolute Errors of algorithms

Function Name	Mean Absolute Error
DE/best/1/exp	1.83e+00
DE/rand/1/exp	1.93e+00
DE/best/1/bin	2.76e+00
DE/rand/1/bin	1.97e+03
Hybrid 1.1	1.78e+00
Hybrid 1.2	1.84e+00
Hybrid 1.3	2.85e+00
Hybrid 2.1	2.49e+00
Hybrid 2.2	1.05e+00
Hybrid 2.3	1.87e+00

5.13 Performance of implemented algorithms

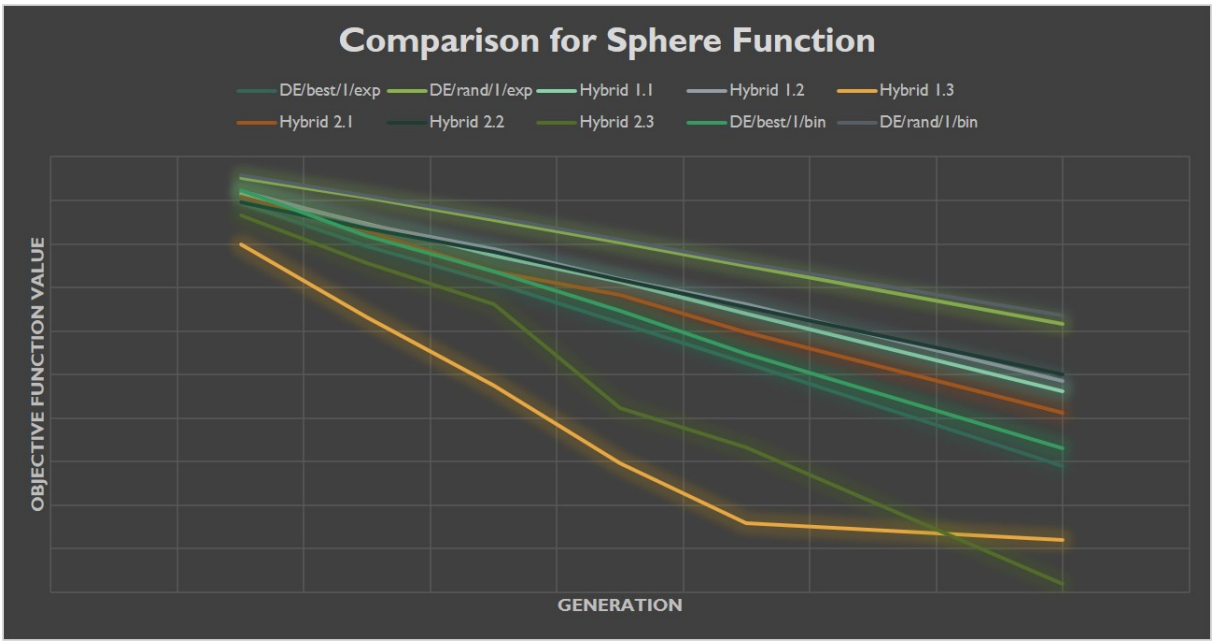


Figure 5.24: Performance of Different Algorithms on *Sphere function*.

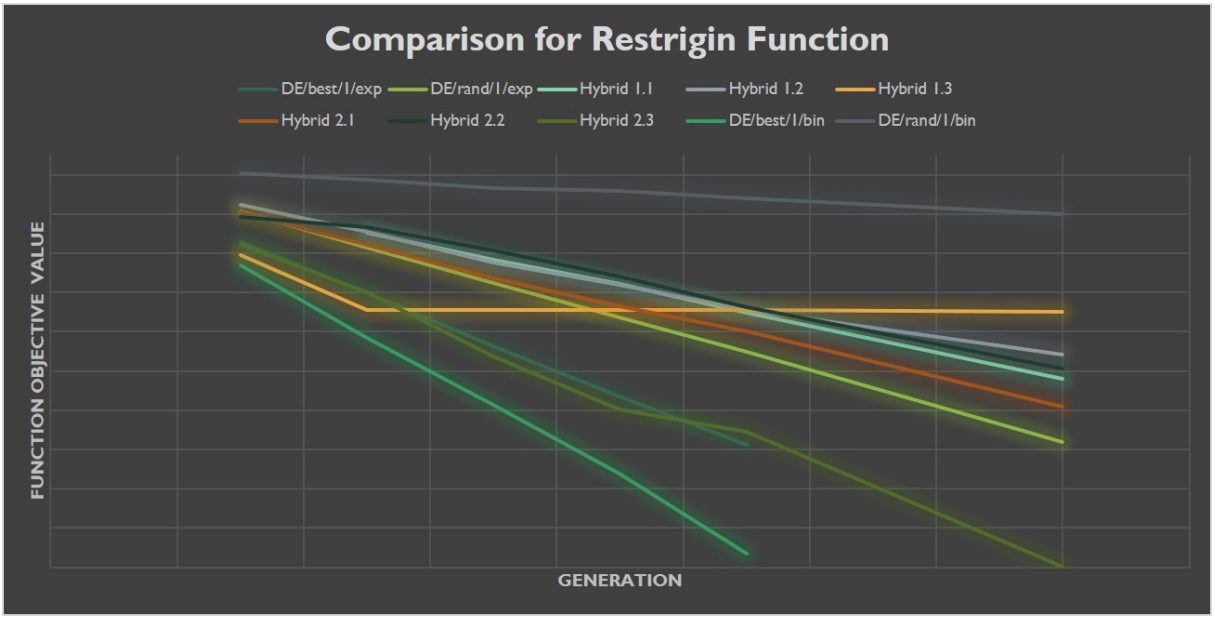


Figure 5.25: Performance of Different Algorithms on *Rastrigin function*.

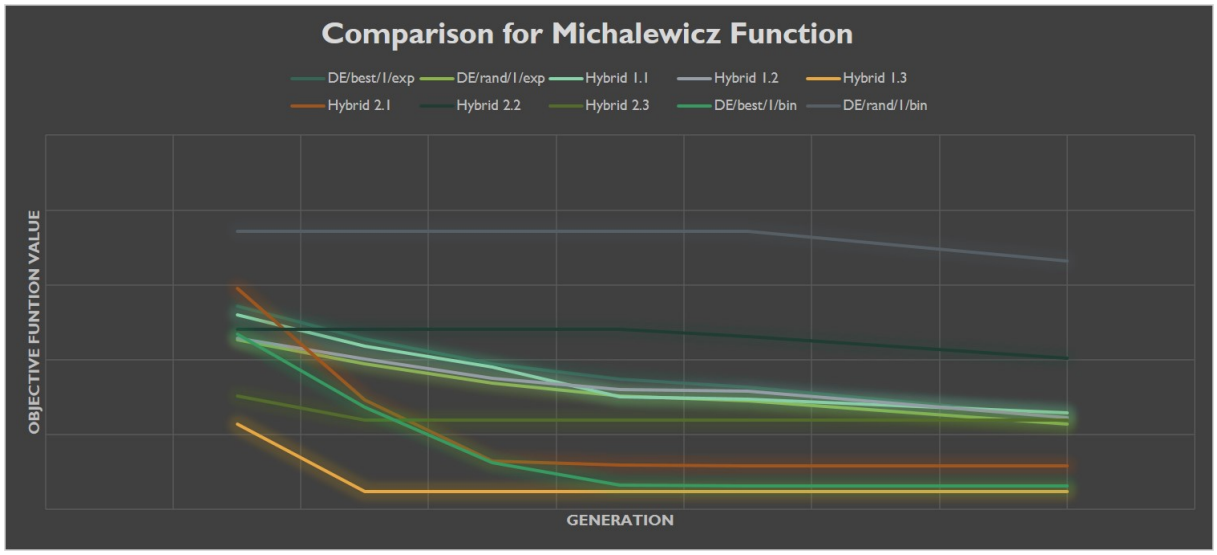


Figure 5.26: Performance of Different Algorithms on *Michalewickz function*.

5.14 Synergistic Effect of Hybrids

Table 5.15: Table for Function-Wise Synergistic Effects of Hybrids For High generations (900-1600)

	Sphere	***Schewefel***	***Rastrigin***	***Griewank***	***Michalewicz***
Hybrid 1.1	Not-Synergistic	Not-Synergistic	Not-Synergistic	Not-Synergistic	Not-Synergistic
Hybrid 1.2	Not-Synergistic	Not-Synergistic	Not-Synergistic	Not-Synergistic	Not-Synergistic
Hybrid 1.3	Synergistic	Synergistic	Not-Synergistic	Synergistic	Synergistic
Hybrid 2.1	Not-Synergistic	Not-Synergistic	Not-Synergistic	Synergistic	Not-Synergistic
Hybrid 2.2	Not-Synergistic	Not-Synergistic	Not-Synergistic	Synergistic	Not-Synergistic
Hybrid 2.3	Synergistic	Synergistic	Not-Synergistic	Synergistic	Not-Synergistic

Table 5.16: Table for Function-Wise Synergistic Effects of Hybrids For High generations (300-700)

	Sphere	***Schewefel***	***Rastrigin***	***Griewank***	***Michalewicz***
Hybrid 1.1	Not-Synergistic	Not-Synergistic	Not-Synergistic	Not-Synergistic	Not-Synergistic
Hybrid 1.2	Not-Synergistic	Not-Synergistic	Not-Synergistic	Not-Synergistic	Not-Synergistic
Hybrid 1.3	Synergistic	Synergistic	Synergistic	Synergistic	Synergistic
Hybrid 2.1	Not-Synergistic	Not-Synergistic	Not-Synergistic	Not-Synergistic	Not-Synergistic
Hybrid 2.2	Not-Synergistic	Not-Synergistic	Not-Synergistic	Not-Synergistic	Synergistic
Hybrid 2.3	Synergistic	Synergistic	Not-Synergistic	Synergistic	Synergistic

5.15 Result Analysis of Hybrid Algorithms

After completing the hybridization both better and worse outcomes are noticeable. When the hybridization of DE/best/1/exp and DE/rand/1/exp is applied on Uni-modal functions, in case of of rand to rand migration 1 the result was quite better comparing to the

DE/rand/1/exp. But wasn't satisfactory comparing to the DE/best/1/exp. rand to rand migration 2 provided a similar kind of output as the previous one. But when the best to rand is applied the result was noticeably satisfactory. The out put was much better than the both and the other hybrids as well. Even when the generation was pretty low the result was all right to be satisfied compared to the outputs of their single version.
The next hybridization technique was applied between DE/best/1/bin and DE/rand/1/bin maintaining the same migration techniques. The output gets better with the reduction of the generation while using rand to rand migration 1. But as we know in case of hybridization the generation basically considered double. So the the output is not that satisfactory considering the other facts to use that instead of DE/best/1/bin. Again here the best to rand migration provides much better result than all the variants.
When these Hybridization techniques were applied on High Dimensional Multi-Modal problems, in most of the cases of higher generations the outcome of DE/best/1/exp is better than the hybridized algorithms. Even DE/rand/1/exp gives a better outputs than the Hybrid algorithm. But when the generation goes down the result of the best to rand migration gets better in a rapid proportion compared to the DE/best/1/exp. At a lower generation the Hybridization of DE/best/1/exp and DE/rand/1/exp with best to rand migration 1 keeps giving better output.
In case of Hybridization of DE/best/1/bin and DE/rand/1/bin the outputs of rand to rand migration is better than DE/rand/1/bin when the generation is very high. Not that beneficial as the cost is much higher comparing to the slightly better output. For the high generations DE/best/1/bin provides the best result while for the mid range generations the best to rand migration 1 provides the better results comparatively.
When the hybridization of DE/best/1/exp and DE/rand/1/exp is applied on Low Dimensional Multi-Modal functions all the hybrid techniques give better output than these twos' separate outputs. Rand to rand migration 1 and rand to rand migration 2 these two are very much similar. The best to rand is better than all of them with a big margin in every possible generations.
Next the hybridization of DE/best/1/bin and DE/rand/1/bin on the same category. All the different migration techniques give better result than the DE/rand/1/bin. The rand to rand migration 1 provides the best output among the migration techniques almost as good as the best outcome. Though the best output came from the DE/best/1/bin rather than the hybridized algorithms.

Chapter 6

Conclusion and Future Work

Global optimization is necessary in fields such as engineering, statistics and finance. But many practical problems have objective functions that are no differentiable, non-continuous, non-linear, noisy, flat, multi-dimensional or have many local minima, constraints or stochasticity. Such problems are difficult if not impossible to solve analytically. DE can be used to find approximate solutions to such problems.
We have implemented just 4 of the variants of Deferential Evolutionary Algorithm on 4 bench mark problems and analyzed which DE provides better results in different generations. And how they change with that. Of course different variants are giving better results on different benchmark functions. We developed 4 hybrids between the evolutionary algorithms, conducted comparative study between the hybrids and some other relevant state-of-the-art meta-heuristic algorithms.In this ending final chapter we will try to summarize all the works we have done along the course of this thesis work, our findings and will discuss about our future plan.

6.1 Summary and Conclusion

In this thesis work we have implemented 4 variants of DE DE/rand/1/exp, DE/best/1/exp, DE/best/1/bin and DE/rand/1/bin on 10 benchmark functions.There are high dimensional uni-modal functions,high dimensional multi-modal functions and low dimensional multi-modal functions.And the results have been shown very precisely to make understand the differences between the algorithms. The graphs show their characteristics very accurately.In a nutshell, DE/best/1/exp and DE/best/1/bin algorithms give best result on uni-modal functions.In the cases of high dimensional multi-modal functions, DE/best/1/exp gives the best result and DE/rand/1/exp provides quite better result almost as good as the best. When these variants of DE are applied on the Low dimensional multi-modal functions

DE/best/1/bin provides the best result. DE/rand/1/exp is also satisfactory though.
After the implementation and evaluating the performance of the algorithms mentioned above, we have implemented a set of hybrid algorithms. There are 4 hybrid algorithms we have implemented and the results that we have found are pretty brilliant. The hybrid algorithms show synergistic effect which proves the effectiveness of the algorithms. From the analyzation part in chapter 5 we can conclude that in some cases there is no use of hybridizing the algorithm. And sometimes the output is slightly better which is not beneficial as the hybridization cost is pretty high comparing to the single variant. We must apply the hybridization where the results came much better and is beneficial even if the cost is higher.

6.2 Future Work

As our final aim was to hybridize the algorithms to get a more efficient output comparing to the existing ones, in future we will work with four more variants and implement all the 30 bench mark functions.We have applied these algorithms for just continuous optimization. In the future we will apply it on discreet optimization problems. We can alter the mutation process also to check the results. In the future, we will try to understand how and under which condition our hybrid algorithms give synergistic effect. We will also change different condition to check the results. More precisely after all these study and implementations we are pretty much sure the other variants of DE will give better output if they are hybridized in a proper way. Even we must check whether the efficiency increases or not if we apply different migration techniques such best to worst, best to best, rand to rand migration 3, rand to rand migration 5 etc. In near future, we will try to understand and implement other evolutionary algorithms like e BAT algorithm, PSO (Particle Swarm optimization) algorithms as we have studied these algorithms and understood quite well. We will apply our implemented hybrid in Data Mining and Artificial Neural Network. We will also try to apply these conventional and our designed hybrid algorithm to solve real-world problems and we will design more improved Differential Evolution [40] algorithms.

References

[1] ABRAHAM, A., GROSAN, C., AND RAMOS, V. *Stigmergic optimization*, vol. 31. Springer, 2006.

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

[3] BLUM, C., AND ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR) 35*, 3 (2003), 268–308.

[4] BONABEAU, E., MARCO, D. D. R. D. F., DORIGO, M., THERAULAZ, G., ET AL. *Swarm intelligence: from natural to artificial systems*. No. 1. Oxford university press, 1999.

[5] BROWNLEE, J. *Clever algorithms: nature-inspired programming recipes*. Jason Brownlee, 2011.

[6] CHAUDHURI, S., AND DEB, K. An interactive evolutionary multi-objective optimization and decision making procedure. *Applied Soft Computing 10*, 2 (2010), 496–511.

[7] CIVICIOGLU, P. Transforming geocentric cartesian coordinates to geodetic coordinates by using differential search algorithm. *Computers & Geosciences 46* (2012), 229–247.

[8] CIVICIOGLU, P., AND BESDOK, E. A conceptual comparison of the cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms. *Artificial intelligence review 39*, 4 (2013), 315–346.

[9] DAS, S., ABRAHAM, A., AND KONAR, A. Particle swarm optimization and differential evolution algorithms: technical analysis, applications and hybridization perspectives. In *Advances of computational intelligence in industrial systems*. Springer, 2008, pp. 1–38.

[10] DE CASTRO, L. N., AND VON ZUBEN, F. J. Learning and optimization using the clonal selection principle. *IEEE transactions on evolutionary computation 6*, 3 (2002), 239–251.

[11] DORIGO, M. Optimization, learning and natural algorithms. *PhD Thesis, Politecnico di Milano* (1992).

[12] ESHELMAN, L. J. Biases in the crossover landscape. *ICGA'89* (1989), 10–19.

[13] FOGEL, D. B. *Evolutionary computation: toward a new philosophy of machine intelligence*, vol. 1. John Wiley & Sons, 2006.

[14] FORMATO, R. Central force optimization: a new metaheuristic with applications in applied electromagnetics. prog electromagn res 77: 425–491, 2007.

[15] FORREST, S., PERELSON, A. S., ALLEN, L., AND CHERUKURI, R. Self-nonself discrimination in a computer. In *Proceedings of 1994 IEEE computer society symposium on research in security and privacy* (1994), Ieee, pp. 202–212.

[16] GANDOMI, A. H., AND ALAVI, A. H. Krill herd: a new bio-inspired optimization algorithm. *Communications in nonlinear science and numerical simulation 17*, 12 (2012), 4831–4845.

[17] GOLDBERG, D. E. Genetic algorithms in search, optimization and machine learning. addsion-wesley longman. *Reading* (1989).

[18] GOLDBERG, D. E., AND HOLLAND, J. H. Genetic algorithms and machine learning. *Machine learning 3*, 2 (1988), 95–99.

[19] GREENSMITH, J., AND AICKELIN, U. Artificial dendritic cells: multi-faceted perspectives. In *Human-Centric Information Processing Through Granular Modelling*. Springer, 2009, pp. 375–395.

[20] HARIFI, S., KHALILIAN, M., MOHAMMADZADEH, J., AND EBRAHIMNEJAD, S. Emperor penguins colony: a new metaheuristic algorithm for optimization. *Evolutionary Intelligence* (2019), 1–16.

[21] HEIDARI, A. A., MIRJALILI, S., FARIS, H., ALJARAH, I., MAFARJA, M., AND CHEN, H. Harris hawks optimization: Algorithm and applications. *Future Generation Computer Systems 97* (2019), 849–872.

[22] HENDTLASS, T. Wosp: a multi-optima particle swarm algorithm. In *2005 IEEE Congress on Evolutionary Computation* (2005), vol. 1, IEEE, pp. 727–734.

[23] HUANG, V. L., SUGANTHAN, P. N., AND LIANG, J. J. Comprehensive learning particle swarm optimizer for solving multiobjective optimization problems. *International Journal of Intelligent Systems 21*, 2 (2006), 209–226.

[24] IRANI, R., AND NASIMI, R. Application of artificial bee colony-based neural network in bottom hole pressure prediction in underbalanced drilling. *Journal of Petroleum Science and Engineering 78*, 1 (2011), 6–12.

[25] KARABOGA, D. An idea based on honey bee swarm for numerical optimization. Tech. rep., Technical report-tr06, Erciyes university, engineering faculty, computer âĂę, 2005.

[26] KARABOGA, D. Artificial bee colony algorithm. *scholarpedia 5*, 3 (2010), 6915.

[27] KARABOĞA, D., AND ÖKDEM, S. A simple and global optimization algorithm for engineering problems: differential evolution algorithm. *Turkish Journal of Electrical Engineering & Computer Sciences 12*, 1 (2004), 53–60.

[28] KOZA, J. R., AND KOZA, J. R. *Genetic programming: on the programming of computers by means of natural selection*, vol. 1. MIT press, 1992.

[29] KRISHNANAND, K., AND GHOSE, D. Detection of multiple source locations using a glowworm metaphor with applications to collective robotics. In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.* (2005), IEEE, pp. 84–91.

[30] KRISHNANAND, K., AND GHOSE, D. Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. *Swarm intelligence 3*, 2 (2009), 87–124.

[31] LI, X., TANG, K., SUGANTHAN, P. N., AND YANG, Z. Editorial for the special issue of information sciences journal (isj) on nature-inspired algorithms for large scale global optimization. *Information SciencesâĂŤInformatics and Computer Science, Intelligent Systems, Applications: An International Journal 316*, C (2015), 437–439.

[32] LIANG, J. J., QIN, A. K., SUGANTHAN, P. N., AND BASKAR, S. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE transactions on evolutionary computation 10*, 3 (2006), 281–295.

[33] LIAO, T., MONTES DE OCA, M. A., AYDIN, D., STÜTZLE, T., AND DORIGO, M. An incremental ant colony algorithm with local search for continuous optimization. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation* (2011), ACM, pp. 125–132.

[34] LIN, C. An adaptive genetic algorithm based on population diversity strategy. In *2009 Third International Conference on Genetic and Evolutionary Computing* (2009), IEEE, pp. 93–96.

[35] MEHRABIAN, A. R., AND LUCAS, C. A novel numerical optimization algorithm inspired from weed colonization. *Ecological informatics 1*, 4 (2006), 355–366.

[36] MEZURA-MONTES, E., AND VELEZ-KOEPPEL, R. E. Elitist artificial bee colony for constrained real-parameter optimization. In *IEEE congress on evolutionary computation* (2010), IEEE, pp. 1–8.

[37] MILLONAS, M. M. Swarms, phase transitions, and collective intelligence. *arXiv preprint adap-org/9306002* (1993).

[38] MOLINA, D., HERRERA, F., AND LOZANO, M. Adaptive local search parameters for real-coded memetic algorithms. In *2005 IEEE Congress on Evolutionary Computation* (2005), vol. 1, IEEE, pp. 888–895.

[39] MOSCATO, P., ET AL. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report 826* (1989), 1989.

[40] M.S.ALAM. *Continuous Optimization with evolutionary and swarm intelligence algorithms*. PhD thesis, Bangladesh University of Engineering and Technology, 2013.

[41] MUHLENBEIN, H. The breeder genetic algorithm-a provable optimal search algorithm and its application. In *IEE Colloquium on Applications of Genetic Algorithms* (1994), IET, pp. 5–1.

[42] MÜHLENBEIN, H., AND SCHLIERKAMP-VOOSEN, D. Predictive models for the breeder genetic algorithm i. continuous parameter optimization. *Evolutionary computation 1*, 1 (1993), 25–49.

[43] NASUTO, S. J., BISHOP, J. M., LAURIA, S., ET AL. Time complexity analysis of the stochastic diffusion search. In *NC* (1998), pp. 260–266.

[44] NOCEDAL, J., AND WRIGHT, S. *Numerical optimization*. Springer Science & Business Media, 2006.

[45] PARSOPOULOS, K. E., AND VRAHATIS, M. N. Recent approaches to global optimization problems through particle swarm optimization. *Natural computing 1*, 2-3 (2002), 235–306.

[46] PASSINO, K. M. Bacterial foraging optimization. *International Journal of Swarm Intelligence Research (IJSIR) 1*, 1 (2010), 1–16.

[47] PRICE, K., STORN, R. M., AND LAMPINEN, J. A. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[48] RABANAL, P., RODRÍGUEZ, I., AND RUBIO, F. Using river formation dynamics to design heuristic algorithms. In *International conference on unconventional computation* (2007), Springer, pp. 163–177.

[49] RASHEDI, E., NEZAMABADI-POUR, H., AND SARYAZDI, S. Gsa: a gravitational search algorithm. *Information sciences 179*, 13 (2009), 2232–2248.

[50] SCHWEFEL, H.-P. *Numerical optimization of computer models*. John Wiley & Sons, Inc., 1981.

[51] SHAH-HOSSEINI, H. The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm. *International Journal of Bio-Inspired Computation 1*, 1-2 (2009), 71–79.

[52] SYSWERDA, G. Uniform crossover in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms* (1989), Morgan Kaufmann Publishers, pp. 2–9.

[53] TALBI, E.-G. *Metaheuristics: from design to implementation*, vol. 74. John Wiley & Sons, 2009.

[54] TAYARANI-N, M.-H., AND AKBARZADEH-T, M. Magnetic optimization algorithms a new synthesis. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)* (2008), IEEE, pp. 2659–2664.

[55] TIMMIS, J., NEAL, M., AND HUNT, J. An artificial immune system for data analysis. *Biosystems 55*, 1-3 (2000), 143–150.

[56] VICSEK, T., CZIRÓK, A., BEN-JACOB, E., COHEN, I., AND SHOCHET, O. Novel type of phase transition in a system of self-driven particles. *Physical review letters 75*, 6 (1995), 1226.

[57] WAIBEL, M., FLOREANO, D., AND KELLER, L. A quantitative test of hamilton's rule for the evolution of altruism. *PLoS biology 9*, 5 (2011), e1000615.

[58] WAIBEL, M., FLOREANO, D., MAGNENAT, S., AND KELLER, L. Division of labour and colony efficiency in social insects: effects of interactions between genetic architecture, colony kin structure and rate of perturbations. *Proceedings of the Royal Society B: Biological Sciences 273*, 1595 (2006), 1815–1823.

[59] XIN, B., CHEN, J., ZHANG, J., FANG, H., AND PENG, Z.-H. Hybridizing differential evolution and particle swarm optimization to design powerful optimizers: a review and taxonomy. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42*, 5 (2011), 744–767.

[60] YANG, X.-S. *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

[61] YANG, X.-S. A new metaheuristic bat-inspired algorithm. In *Nature inspired cooperative strategies for optimization (NICSO 2010)*. Springer, 2010, pp. 65–74.

[62] YAO, X., AND LIU, Y. Fast evolutionary programming. *Evolutionary programming 3* (1996), 451–460.

[63] YEN, G. G., AND LEONG, W. F. Dynamic multiple swarms in multiobjective particle swarm optimization. *IEEE Transactions on systems, man, and cybernetics-part A: systems and humans 39*, 4 (2009), 890–911.

[64] ZHANG, X., BAI, Q., AND YUN, X. A new hybrid artificial bee colony algorithm for the traveling salesman problem. In *2011 IEEE 3rd International Conference on Communication Software and Networks* (2011), IEEE, pp. 155–159.

Generated using Undegraduate Thesis LaTeX Template, Version 1.4. Department of Computer Science and Engineering, Ahsanullah University of Science and Technology, Dhaka, Bangladesh.

This thesis was generated on Tuesday 2nd July, 2019 at 12:35am.