M.Sc. Engg. Thesis

A Recurring Multistage Evolutionary System for Balancing Exploitation and Exploration

Submitted By
Mohammad Shafiul Alam
M. Sc. Engineering Student
Department of Computer Science and Engineering
Student Id: 040405007P

A thesis submitted to the Department of Computer Science and Engineering in partial fulfillment of the requirements for the degree of
Master of Science in Engineering in
Computer Science and Engineering

Supervised by
Dr. Md. Monirul Islam
Associate Professor, Department of CSE, BUET

The thesis titled "**A Recurring Multistage Evolutionary System for Balancing Exploitation and Exploration**" submitted by Mohammad Shafiul Alam, Roll No. 040405007P, Session April 2004, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Engineering in Computer Science and Engineering (M.Sc. Engg. in CSE). Examination held on January 31, 2007.

DECLARATION

I, hereby, declare that the work presented in this thesis is the outcome of the investigation performed by me under the supervision of Dr. Md. Monirul Islam, Associate Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka. I also declare that no part of this thesis and thereof has been or is being submitted elsewhere for the award of any degree or diploma.

Signature

[illegible] 24/2/07

(Mohammad Shafiul Alam)

To

My Beloved Parents

&

Dear Sister

Without whom any achievement in life would be meaningless

Acknowledgements

Firstly, I would like to express my profound gratefulness and honor to Dr. Md. Monirul Islam, Associate Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, for his continuous support, advice and care. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this thesis.

Secondly, I would like to thank Dr. Muhammad Masroor Ali, Professor and Head, Department of CSE, BUET for his fruitful advice, care and suggestions.

Thirdly, I would like to express my gratitude to all the members of the graduate committee for their valuable suggestions.

I would also like to thank all the faculty members and staff of the CSE Department of BUET, for their support and cooperation.

Finally, I would like to express my deep respect and gratitude to my father A. F. M. Badrul Alam, my mother Mrs. Latifa Khanam and my sister Umme Nusrat Ara, without whose sacrifice, support and encouragement, this work would not been possible.

Abstract

Maintaining proper balance between exploitative and explorative operations of an evolutionary algorithm is essential for preventing premature convergence to local optima and for sustaining sufficient convergence speed throughout the evolution. This thesis introduces Recurring Multistage Evolutionary Algorithm (RMEA), a completely new framework to balance the exploitative and explorative features of the conventional evolutionary algorithm. The basis of RMEA is repeatedly alternating three different stages of evolution, each with its own explorative or exploitative objective and genetic operators. As the stages of RMEA repeat, the conflicting goals of exploitation and exploration are distributed gracefully across the generations of the different stages. The key concept of RMEA is to combine dissimilar information across the population for search space exploration and to combine similar information within population neighborhood for local exploitation. Performance of RMEA has been evaluated on a number of benchmark numerical optimization problems and results are compared with several existing algorithms. Experimental results show that RMEA performs better optimization with a higher rate of convergence for most of the problems. Also, an in-depth experimental study is carried out about the roles of the different stages and operators of RMEA, as well as the sensitivity of its parameters. An adaptive variant of RMEA is also proposed, which adjusts its parameters in an adaptive manner during the evolution, and does not require any problem specific knowledge from the user.

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Symbols

CEA	Cellular Evolutionary Algorithm
CEP	Classical Evolutionary Programming
DGEA	Diversity Guided Evolutionary Algorithm
EA	Evolutionary Algorithm
EP	Evolutionary Programming
FEP	Fast Evolutionary Programming
GA	Genetic Algorithm
IFEP	Improved Fast Evolutionary Programming
RMEA	Recurring Multistage Evolutionary Algorithm
SEA	Standard Evolutionary Algorithm
SOCEA	Self-organized Criticality Evolutionary Algorithm

Chapter 1

Introduction

1.1 Introduction

During the last three decades evolutionary algorithms (EAs) have been applied successfully in numerous and diverse application areas, such as different kinds of optimization and search problems [1], engineering design [2], automatic programming [3], combinatorial problems [4], data mining [5], game playing [6], evolvable hardware [7] and robotics [8]. EAs maintain a population of structures which represent approximate solutions to a particular problem. In each iteration, a new set of approximate solutions is generated by the process of selecting a number of individuals based on their fitness and applying some operators, borrowed from natural genetics, like recombination, crossover and mutation. To mimic the Darwinian evolutionary principle of *survival of the fittest*, better individuals are selectively given higher chance to reproduce and survive. In brief, the classical evolutionary approach for solving problems can be summarized into two major steps:

i) Change the individuals of the current population by using crossover and mutation.

ii) Create a new population by selecting individuals from the union of the changed and the current individuals for the next generation.

A major problem with classical EAs is that they have a tendency to converge to the local optima [9]. The primary reason for the premature convergence is the rapid loss of population diversity, starting from the very early phase of evolution. Diversity means the amount of 'variety' existing among the individuals across the population [10], where variety is based on the structural or behavioral differences across the individuals. Maintaining the population diversity has been cited as a key issue in a number of research works, e.g. [10-12], in order to prevent premature convergence and stagnation in local optima.

The loss of population diversity is closely related to the exploitation-exploration trade-off of the search process. If an EA is completely focused on local optimizations, i.e. exploitations, it becomes quite similar to greedy algorithms and soon the entire population is trapped around the local optimum points after loosing its diversity severely. On the other hand, if the EA puts emphasis only on diversity by more global explorations, it may not be able to completely analyze and exploit the already explored solutions. This is because every search algorithm is restrained by space and time constraints, and putting more efforts in one mode (say, exploitation) necessarily compels the algorithm to put less efforts in the other mode (i.e., exploration). Exploitations immediately improve a solution quality at the cost of reducing the population diversity and possible fitness stagnation by being trapped into local optima. In contrast, explorations enhance the diversity at the risk of deteriorating the solution quality and reducing the convergence rate. Thus an appropriate balancing between the explorative and exploitative features of the EA throughout the evolutionary process is crucial for maintaining both adequate convergence rate and sufficient population, and thus avoiding local optima in order to ensure better solution quality.

In order to achieve proper balance between the exploitative and explorative features of the conventional EA, this thesis work introduces a multistage framework of EA, which repeatedly executes and alternates different objective-driven stages, with their own operators, to fulfill the exploitative/explorative objectives of the current stage. This is the first time, to our best knowledge, that utilizes different stages, each with its separate specific objective and operators, in a recurring manner for balancing the conflicting goals of exploitation and exploration of the evolutionary approach. The proposed system will also maintain the population diversity in a recurring way to avoid entrapment into local optima. The proposed algorithm will be compared with several existing evolutionary systems on a number of benchmark numerical optimization problems.

1.2 Literature Review

The main problem of standard EA is its tendency of premature convergence around the local optimum points, without finding the global optimum. Several algorithmic features, especially high selection pressure and high gene flow among individuals, are considered responsible [12] for premature convergence. First, a high selection pressure will select an extraordinarily better but suboptimal solution multiple times for reproduction and fill the population with several copies of that individual. As a result, diversity drops quickly and the population no longer represents the entire search space, which leads the algorithm to entrapment into local optima. But lowering the selection pressure often slows down the convergence rate severely. Secondly, the standard EA allows any individual to mate with any other individual across the population. Therefore, better genes spread rapidly throughout the entire population and population diversity declines.

There is a number of existing works [9], [12-22] which try to achieve the conflicting goals of maintaining a diverse population and converging to the global optima with an acceptable rate. Most existing works fall in one of the following three categories.

- Complex population structures to lower gene flow, e.g., cellular EA [9], multinational EA [13], religion-based EA [14].
- Specialized operators to control and assist the selection procedure, e.g., crowding [15], fitness sharing [16].
- Reintroduction of genetic material, e.g., random immigrations [17], mass extinction models [18].

The cellular EA [9], also known as 'diffusion model', models the entire population as a two-dimensional grid of cells, with the edges and corners usually wrapped around. Every individual is located within a cell, and mating (recombination/crossover) is allowed only between an individual and its immediate neighbors. In such a framework, it becomes difficult for a solution to quickly take over the entire population, because it takes several generations to diffuse the genes throughout the cells. Thus limiting the interaction range lowers gene flow among the individuals. Cellular EA has been proved to be more diversity-preserving, and has displayed better performance than the standard EA for a number of problems.

Another scheme, multinational EA (MEA) [13] divides the population into a number of nations. Each nation searches for a potential peak in the fitness landscape. A nation consists of a population, a 'government', and a 'policy'. The government is a subset of the nation's individuals representing the peak in the fitness landscape to which the nation is approaching. The policy is a single point representing the peak and calculated from the individuals of the government. During the evolution, nations may merge together or split apart, and migration may occur from one nation to another. All these processes are controlled by a *hill-valley detection* procedure, which determines whether there exists a valley in the fitness landscape between two given individuals. During each generation, standard genetic operations alter the nations, and then the *hill-valley detection* procedure compares every member of every nation with the nation's policy. If a valley is detected between an individual, ***I*** and the policy of its nation, then ***I*** is assumed to no longer belong to its current nation. If another nation is found which is approaching the same peak as ***I*** does, then ***I*** migrates to that nation. Otherwise, ***I*** forms a completely new nation approaching the newly discovered peak. Besides, if two nations are found to approach the same peak, they are merged to constitute a single nation. MEA has been applied to a number of static problems and it has demonstrated to successfully locate a number of peaks for each problem. Besides, it has also exhibited satisfactory results for artificial dynamic problems.

The religion-based EA (RBEA) [14] is another model involving the concepts of religions, believers, and conversion to religions. RBEA uses a grid of cells, each cell being empty or occupied by a single individual. Individuals can move from its cell to the neighboring empty cells. During the evolution, each individual tries to occupy an empty cell, and tries to convert its surrounding individuals to its own religion. Such conversion is probabilistically based on the fitness of the individual. Besides, mating (i.e., recombination or crossover) is allowed within the same religion. RBEA is compared with standard EA and the diffusion model on six benchmark problems. It performed better than the standard EA in all the problems, and outperformed the diffusion model on five (out of six) problems.

Crowding [15] was one of the earliest attempts to deal with diversity. Crowding inserts newly created offspring into the population in such a way that it avoids crowding. When inserting an individual **I**, a number of individuals are selected at random and searched to find **I′**, which is most similar to **I**. Then, **I** replaces the **I′** if **I** has better fitness than **I′**. Crowding has been tested for a number of simple functions and displayed to maintain good diversity. A problem with crowding is the risk of replacement errors, i.e., a new individual may mistakenly replace a good individual from another peak.

The loss of diversity can also be prevented by changing the raw fitness values of the individuals in such a way that diverse individuals are favored by the selection and mating process. Goldberg and Richardson employed this idea by 'fitness sharing' [16] among the similar individuals. The individuals which are quite similar (i.e., close in the search space) are penalized by reducing their fitness values, while diverse individuals receive their raw fitness values unaltered. This algorithm was tested with two multi modal functions, each with five peaks. It was successful in locating all the five peaks simultaneously.

The random immigrations model [17] uses a very simple, but effective scheme to deal with dynamic problems. In each generation, a proportion of the population is replaced by randomly created individuals. The replacement rate is usually set from 5% to 10% of the population size. This model was proposed for handling time varying problems. The algorithm was tested on three dynamic benchmark problems and a static problem. It outperformed the standard EA on the dynamic problems, but not on the static one.

In mass extinction model [18], the diversity is ensured by forcefully exterminating a portion of the population. Two types of extinction events are employed: mass extinction event and background extinction event. During each generation, a stress factor (i.e., a random number within some range) is generated according to some distribution. After some necessary scaling of fitness values, all the individuals with fitness lower than the stress factor are eliminated. Then the vacant slots of the population are filled by new

```
1. Initialization. Generate a random population of $n$ chromosomes (individuals). Each
   chromosome is a potential solution for the problem.
2. Fitness Evaluation. Evaluate the fitness $f(x)$ of each chromosome, $x$ of the
   population.
3. Generate new population. Create a new population by repeating following steps
   until the new population is complete
    3.1 Selection. Select two parent chromosomes from a population according to
        their fitness values (the better the fitness, the bigger chance to be selected)
    3.2 Recombination. Combine the attributes of the two parents to form new
        offspring.
    3.3 Mutation. With a mutation probability, mutate new offspring at each locus
        (position in chromosome).
    3.4 Reinsertion. Either accept or reject the new offspring in a new population.
4. Test for termination. If the end condition is satisfied, stop, and return the best
   solution in current population. Otherwise, continue.
6. Loop back. Go to step 2 to continue the evolution with the population of potential
   solutions.
```

Algorithm 2.1: Standard evolutionary algorithm

2.3 Biological Basis

In this section some biological terms, related with EA, are defined. This will be helpful to have some appreciation 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 organism's *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 probabilistically to undergo genetic operations such as crossover or 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. 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 are diversity decreasing operations. So, the importance of mutation, which introduces further diversity while the algorithm is running, cannot be overemphasized. However, some researchers like mutation as a background operator, to reintroduce some of the original diversity that may have been lost, while others view it as playing the dominant role in the evolutionary process.

2.4 Components of Evolutionary Algorithm

EAs have a number of component procedures and operators that must be described to have a proper appreciation of the algorithm. Each of these components needs to be specified in order to define a particular EA. The most important components of an EA are as follows.

- Representation (Encoding)
- Evaluation Function
- Selection

- Recombination
- Mutation
- Reinsertion

2.4.1 Representation (Encoding):

Representation means to represent possible solutions of the actual problem as data objects 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', or 'individual' interchangeably while the actual physical solution in the problem domain is referred as 'phenotype'. The phenotype is encoded as a collection of attributes within a chromosome. Each attribute inside a chromosome is called a gene.

A chromosome, with its genes, may be encoded in several ways. Among them, *binary encoding* is the most common. In *binary encoding* every chromosome is a string of bits, 0 or 1. For example, a chromosome **A** may be represented as: **1101010100100010111.** However, in many problems, *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 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 provided here.

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 a a b
Chromosome C	0.5498 1.5329 2.1092 9.2143 0.2241

Another form of representation is *permutation encoding*, which is mostly used in ordering problems, such as traveling salesman problem or task ordering problem. In permutation encoding, every chromosome is a string of numbers, which represents positions in a *sequence*. For example, a chromosome **A** may have the permutation encoding: 7 2 4 1 6 5 9 3 8, where the original *sequence* has 9 different members. Another special kind of representation, which is used with genetic programming, 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.

2.4.2 Evaluation Function

The evaluation function performs the role to define how 'well' each individual chromosome is carrying out as a possible solution of the actual problem. It is also called 'fitness function' because it assigns a level of fitness to each individual. The selection and reinsertion phases depend significantly on the fitness values of the individuals assigned by the evaluation function. As an example, suppose, we want to find the value of x within the domain of 8-bit integers so that x^2 is maximized. Here, the phenotype 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 the phenotype integer, 20. To measure the fitness of the chromosome, the evaluation 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.

2.4.3 Selection

Inspired by the role of natural selection in evolution, EA selects a number of individuals (parents) from the population to constitute a mating pool, where they recombine with each other to reproduce a number of offspring individuals. 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 is a number of ways to make the selection of individuals. For example, roulette wheel selection, rank based selection, tournament selection, and steady state 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 each 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. So, 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 *N* times; each time it returns an individual.

When the best individual of the population is extremely better than the others, 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 chromosome's fitness covers 80% 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 the population based on its fitness. Then every chromosome receives fitness from its ranking, not from their actual fitness 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* takes values ranging from 2 to *N*, where *N* is the number of individuals in the population.

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.

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 few best chromosomes are copied to the new population. Then any selection scheme completes the rest of the selection. Elitism has shown good performance with a number of problems, because it protects the best solutions.

2.4.4 Recombination

Recombination is the process of generating new individuals (offspring) by combining the information of two or more existing individuals (parents). Each individual contains a number of attributes. 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:

$$Var_i^{o} = Var_i^{P_1} . \alpha_i + Var_i^{P_2} . (1 - \alpha_i); \quad i \in (1,2, \dots ,n)$$

$\alpha_i \in [-d, 1+d]$ uniform at random, $d = 0.25$,

α_i is generated anew for each i.

Here, n is the number of variables in each individual (which is 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. So, 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 recombination over generations.

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 $[1\dots n]$. 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 exists single point, two point and multi-point crossover. An example of single point crossover is shown here. The bits interchanged during crossover are shown by dotted outlining.

Parent 1: 0 0 1 0 0 0 1 0 1

Parent 2: 1 0 1 1 1 0 1 0 0

Crossover point: 4

Offspring 1: 0 0 1 1 1 0 1 0 0

Offspring2: 1 0 1 0 0 0 1 0 1

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 Figure 2.2.

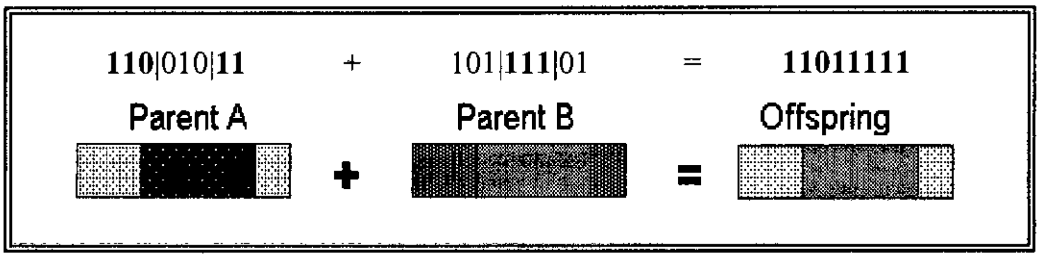

Figure 2.2: Two-point crossover

Another form of crossover is *uniform crossover* in which bits are randomly copied from the first or from the second parent. Figure 2.3 illustrates this kind of crossover.

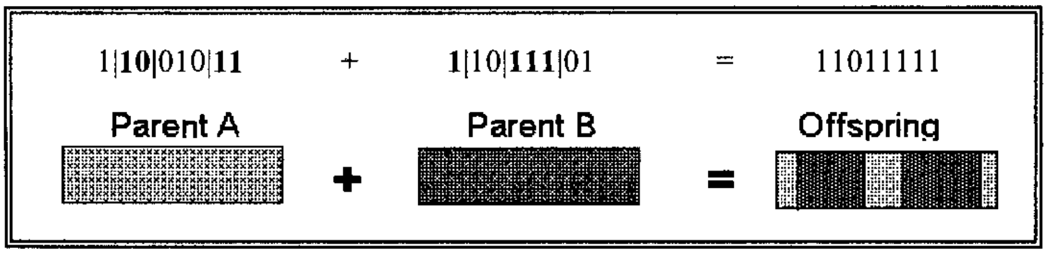

Figure 2.3: Uniform crossover

2.4.5 Mutation

Mutation means randomly altering the chromosomes. 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 a chromosome being mutated. Usually mutations are applied with small mutation step and low mutation rate. Offspring are probabilistically mutated after being created by the recombination step. Like recombination, there exist several methods 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 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 is usually inversely proportional to the number of variables. The more attributes (dimensions) one individual has, the smaller is the mutation probability. Several papers [28–30] researched for the optimal mutation rate. A mutation rate of $1/n$ (n: number of variables of an individual) 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 independent of the size of the population. Higher mutation rates at the beginning, and declining this rate with increasing generations has shown an acceleration of the search for many problems.

The optimal size for the mutation step is usually difficult to realize. It always depends on the problem at hand and may even vary during the search process. Small mutation steps are often successful, while larger steps, when successful, produce good results much quicker. So, a good mutation operator should produce both small and large step-sizes in suitable proportions. Such a mutation operator, first proposed and employed in [28], [31] is specified below.

$$Var_i^{Mut} = Var_i + s_i \,.\, r_i \,.\, a_i$$

$i \in \{1, 2, \ldots, n\}$ uniform at random,

$s_i \in \{-1, +1\}$ uniform at random

$r_i = r.\ domain_i$, r: mutation range (standard: 10%)

$a_i = 2^{-u.k}$, $u \in [0, 1]$ uniform at random, k: mutation precision.

This mutation is able to generate most points in the hyper-cube defined by the domain of the 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 that of larger steps, which is appropriate for most problems.

For binary valued individuals mutation means the flipping of variable values, because every variable has only two states. Thus, the size of the mutation step is always 1. For every individual the variable value to mutate is chosen mostly uniform at random. Following example shows how a binary mutation alters a chromosome with 10 bits. Here, mutation flips the bit at position 6 from 0 to 1.

Chromosome (before mutation): 0 0 1 0 0 $\boxed{\mathbf{0}}$ 1 0 1 1

Chromosome (after mutation): 0 0 1 0 0 $\boxed{\mathbf{1}}$ 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, [32, 33] and evolutionary programming [34]. The extensions of these methods and new developments include several new schemes, which includes

- Adaptation of *n* (number of variables) step-sizes [35, 36]
- Adaptation of *n* step-sizes and one direction [36]
- Adaptation of *n* step-sizes and *n* directions [37]

2.4.6 Reinsertion

Reinsertion is the process of constructing the next generation population from the union of parents and offspring. There exist several schemes, each with its own merits and demerits. In *pure reinsertion*, the number of offspring reproduced is equal to the number of parents and offspring replace the parents entirely. So each individual, even the best one, lives for only a single generation. In *uniform reinsertion*, fewer offspring are produced than parents and offspring replace parents uniformly at random. So, better individuals may be replaced by weaker offspring. In *elitist reinsertion*, the worst individuals of the current population are replaced by the new offspring. Another scheme, *fitness based reinsertion* 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* combined with *fitness-based reinsertion* prevents the best individuals from being lost, and it is the 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.

As well as these global reinsertion schemes, reinsertion may also be based on local policies. In local reinsertion, individuals are considered within its bounded neighborhood. The reinsertion of offspring 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 neighborhood.
- Insert offspring fitter than weakest individual in neighborhood and replace weakest individuals in neighborhood.
- Insert offspring fitter than parent and replace parent.
- Insert every offspring and replace individuals in neighborhood randomly.
- Insert offspring fitter than weakest individual in neighborhood and replace parent.
- Insert offspring fitter than weakest individual in neighborhood and replace individuals in neighborhood uniformly at random.

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

2.5 Applications of Evolutionary Algorithm

EAs have been widely used for complex optimization problems during the last three decades. The main advantage of EA is its ability to escape local minima and reach the global optima with relative ease. The basis of this strength lies in the parallelism of EA. EAs search throughout the search space with a population of individuals in parallel, so they are less likely to get stuck in local optima. Besides, they are quite easy to implement and to extend for other problems. Once an EA is designed, it only requires encoding new chromosome and designing new fitness function to solve another problem. EAs are robust. They perform surprisingly well with wide range of parameter values.

The main disadvantage of EAs is their high computational complexity. Besides, evaluation of fitness of the chromosomes may be quite difficult, and may incur additional computational cost, particularly for real world problems. However, EAs have been widely applied in diverse application areas for their general extensibility and robustness. Some applications of EA are mentioned here:

- Bioinformatics
- Cellular programming
- Game playing
- Automatic programming
- Non linear filtering
- Evolvable hardware
- Combinatorial problems
- Strategy planning
- Robot trajectory
- Time tabling
- Designing neural networks
- Evolving images and music

Chapter 3
Recurring Multistage Evolutionary Algorithm

3.1 Introduction

The standard evolutionary algorithm makes no explicit attempts to maintain balance between the exploitative and explorative operations. Exploitation is conducted by fitness based selection pressure and exploration is carried out by mutation, both in an implicit manner, without any explicit control of the algorithm. Such an imprecise management of exploitation and exploration often leads to quick loss of population diversity and stagnation around the local optima of the search space. In contrast, the proposed system, RMEA makes explicit attempts to ensure a proper balance between the exploitation and exploration. RMEA repeatedly applies three objective-driven evolutionary stages, each with its own operators to fulfill the specific exploitative/explorative objective of the stage. It has been tested against a number of benchmark numerical optimization problems, where the objective is to locate the global minima across the high dimensional multimodal search space. The experimental results establish that RMEA performs significantly better that several existing evolutionary systems, in terms of both convergence rate and solution quality. The following sections describe the framework of RMEA in details.

3.2 The Proposed System: RMEA

In order to ensure a proper balance between the exploitation and exploration, RMEA repeatedly alternates its conventional, exploration and exploitation stages during the course of evolution. In the conventional stage, fitness based tournament selection and gaussian mutation are employed for exploitation and exploration respectively. The exploration stage employs a number of explorative operators that always use dissimilar set of individuals from the population to reach the unexplored regions of the search space. Application of any explorative operator is followed by a number of uphill moves to evaluate the newly explored regions. The exploitation stage employs exploitative operators that always use sets of individuals belonging to some predefined vicinity (neighborhood) in the search space. In fact, all these operations are motivated by observing some important facts, such as:

- Exploration is a non-local operation. Therefore, genetic operations involving distant i.e., dissimilar individuals may lead the search process to unexplored regions of the search space.
- After exploring to a new region, some hill-climbing operations are necessary to realize the potentials of the newly explored regions and to avoid early rejection of better solutions.
- Exploitation is a local operation. Therefore, genetic operations among the neighbors and allowing only uphill moves are relevant to reach the local optimum points of the search space.

Like every evolutionary process, RMEA starts with some initial population of individuals, which represent approximate solutions to a particular problem. The solutions may be values of variables that optimize a function, process, plans, design, strategies, or any entity that need to be optimized. During each generation, RMEA employs the genetic operators of that stage on the individuals to produce offspring. The offspring are inserted into the population depending on the reinsertion scheme of the current stage. The process of RMEA is described in the following steps:

Step 1) Generate an initial population of M individuals. Each individual, **I** is represented as a pair of real valued vectors, $(\boldsymbol{x}_i, \boldsymbol{\eta}_i)$, for i=1,...,$M$; $\boldsymbol{x}_i$'s are objective variables and $\boldsymbol{\eta}_i$'s are standard deviations for Gaussian mutations. Each $\boldsymbol{x}_i$ (and $\boldsymbol{\eta}_i$) has n components, where n is the dimensionality of the problem. Each component of $\boldsymbol{x}_i$, for i=1,...,M, is generated uniformly at random within its domain. All the components of $\boldsymbol{\eta}_i$, for i=1,...,M, are initialized to some moderate value (e.g. 3), as is done in [22].

Step 2) Initialize parameters k_1, k_2 and k_3, which control the behavior of RMEA by defining the lengths, i.e., number of generations constituting the conventional, exploration and exploitation stages.

Step 3) Conventional stage. Repeat the following step 4 for k_1 times. This constitutes a single pass of the conventional stage.

Step 4) A single iteration of the classical evolutionary programming (CEP) throughout the population. Details of CEP are specified in subsection 3.2.1. In brief, an iteration of CEP consists of mutating every individual of the current generation to form M offspring, calculating the fitness of all the parents and offspring and a tournament among all the parents and offspring to select M individuals for the next generation.

Step 5) Repeat the following steps 6-7 for k_2 times. This constitutes a single pass of the exploration stage. Usually, population diversity increases in this stage due to the explorative nature of the operators.

Step 6) Calculate the distance of each individual **I** from every other individual in the population to find a set of N 'Strangers' (farthest individuals) for **I**. Strangers are utilized by the explorative operators.

Step 7) A single generation within the exploration stage. For every individual, **I** choose one of the six explorative operators (subsection 3.2.2) at random with equal probability. Apply the selected operator on **I** to obtain a new individual **I′**. Try to make a number (= k_4) of hill climbing moves from **I′** to obtain a new individual **I″**. These hill-climbing steps (subsection 3.2.3) are to estimate the potentials of the search space around **I′**. If **I″** has better fitness than the original individual **I**, then replace **I** by **I′** (not **I″**). Otherwise discard **I′** and **I″**. Repeat this entire process with all the individuals of the population.

Step 9) Repeat the following steps 10-11 for k_3 times. This constitutes a single pass of the exploitation stage. Population diversity decreases in this stage due to the exploitative nature of the operators.

Step 10) Calculate the distance of each individual **I** from every other individual in the population to find a set of N 'neighbours' (nearest individuals) for **I**. Neighbors are utilized by the exploitative operators.

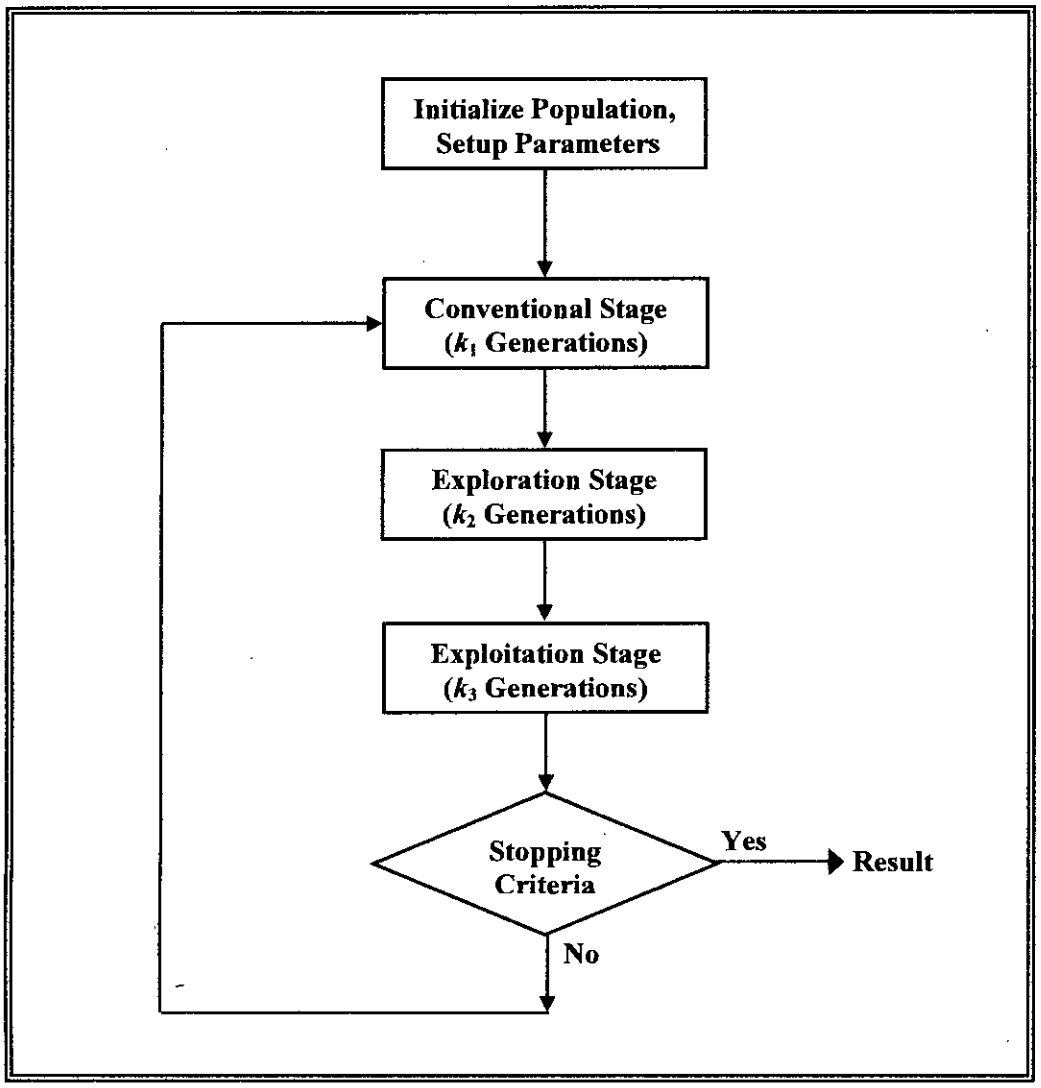
Initialize Population, Setup Parameters
Conventional Stage (k_1 Generations)
Exploration Stage (k_2 Generations)
Exploitation Stage (k_3 Generations)
Stopping Criteria
Yes
Result
No

Figure 3.1: Recurring multistage evolutionary algorithm (RMEA)

Step 11) A single generation within the exploitation stage. For every individual **I**, choose one of the six exploitative operators (subsection 3.2.4) at random with equal probability. Apply the selected operator on **I** to obtain a new individual **I′**. If **I′** has better fitness than **I**, then replace **I** by **I′**. Otherwise discard **I′**. Repeat this process with all other remaining individuals.

Step 12) Check for termination. If the best solution found is acceptable or the maximum number of generations has been elapsed, stop the evolutionary process and go to Step 13. Otherwise, go back to Step 3 to start another execution of conventional, exploration and exploitation stages.

Step 13) The evolutionary process is over. Obtain the best individual of the last generation which is the result of the entire evolutionary process.

```
1. Initialize Parameters. Generate Initial Population.

2. for $k_1$ generations          [Conventional Stage]
      Mutate each individual
      Execute tournament among parents and offspring to get the next generation

3. for $k_2$ generations          [Exploration Stage]
    for each individual, I
        Update the set of strangers of I
        Randomly choose one of the six explorative operators
        Apply the operator on I to obtain a new individual I′
        Perform some uphill steps from I′ to decide whether to accept or reject it.

4. for $k_3$ generations          [Exploitation Stage]
    for each individual, I
        Update the set of neighbors of I
        Randomly choose one of the six exploitative operators
        Apply the operator on I to obtain a new individual I′
        Accept I′ if it has better fitness than I. Otherwise Reject.

5. If the best solution found is acceptable or the maximum number of generations has
been elapsed, conclude RMEA and go to step 6. Otherwise, return to step 3 and start
another cycle of conventional, exploration and exploitation stages.

6. Obtain the best individual of the last generation, which is the output from the
evolutionary process.
```

Algorithm 3.1: Recurring multistage evolutionary algorithm

Algorithm 3.1 presents the pseudo-code of RMEA, which is apparently parallel to the Figure 3.1. The necessary details of the different stages and operators are provided in the following subsections.

3.2.1 Conventional Stage

The classical evolutionary programming (CEP) is among the most basic evolutionary systems and has been in use for many years for various optimization tasks. CEP does not explicitly consider the issues of exploration and exploitation. It mutates individuals of the current population for producing offspring, and selects the better individuals from the union of the parents and offspring. Thus, explorations are conducted by mutations, while exploitations are carried out by fitness based selection pressure. The following operations are performed in every generation of CEP.

- Evaluate the fitness score for each individual of the population.
- Mutate each individual ($\boldsymbol{x}_i$, $\boldsymbol{\eta}_i$), for i = 1...M, to create an offspring ($\boldsymbol{x}_i'$, $\boldsymbol{\eta}_i'$). That is, for j = 1,...,n,

  $$x_i'(j) = x_i(j) + \eta_i(j)N_j(0,1)$$

  $$\eta_i'(j) = \eta_i(j)\exp(\tau' N_j(0,1) + \tau N_j(0,1))$$

  $x_i(j)$, $x_i'(j)$, $\eta_i(j)$, and $\eta_i'(j)$ denote the j-th component of the vectors $\boldsymbol{x}_i$, $\boldsymbol{x}_i'$, $\boldsymbol{\eta}_i$ and $\boldsymbol{\eta}_i'$, respectively. $N_j(0,1)$ denotes a normally distributed one-dimensional random number with mean = 0 and standard deviation = 1. The subscript, j in $N_j(0,1)$ indicates that the random number is generated anew for each value of j. The factor τ and τ' are set to $(\sqrt{(2\sqrt{n})})^{-1}$ and $(\sqrt{(2n)})^{-1}$.
- Calculate the fitness of each offspring generated by the mutation.
- Conduct a pair wise tournament based competition over the union of the parents and offspring. For each individual, q opponents are chosen uniformly at random from all the parents and offspring. If the individual's fitness is not less than an opponent, then it receives a 'win'.
- Select the M individuals from the parents and offspring which have received the most number of 'win's. They become the parents of the next generation.

3.2.2 Exploration Stage

The objective of this stage is to explore new regions of the search space. To achieve this goal, genetic operators are applied on distant sets of individuals. The farthest (in terms of edit distance) N individuals from a particular individual are referred as its set of 'strangers'. Recombination and crossover are applied by using strangers and mutation is applied with a sufficiently large step size to facilitate exploration. As the attributes of distant individuals are combined and the large step size is used in the genetic operations, the population diversity tends to rise in this stage. The following scenario describes how operations involving distant individuals, in combination with limited hill climbing, will lead to an unexplored and better region of the search space.

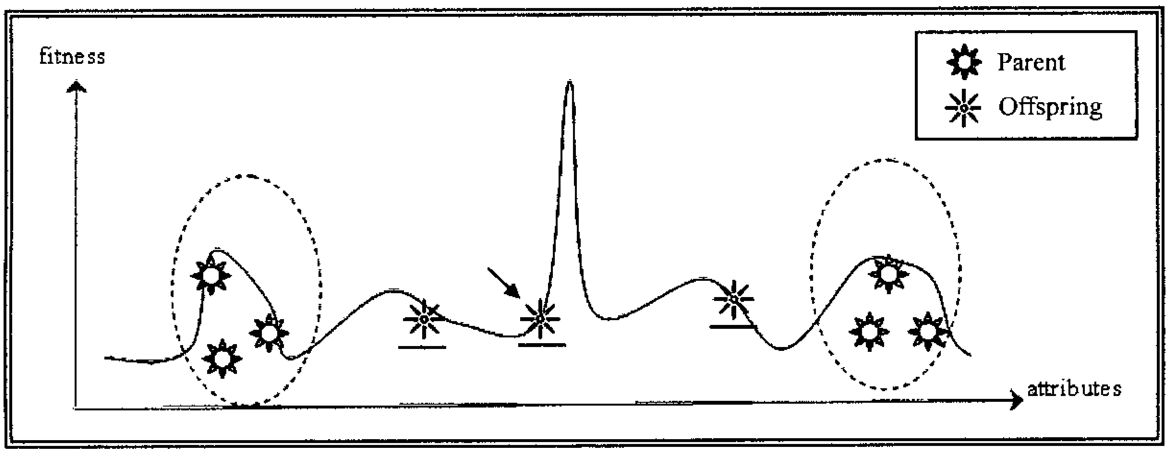

Figure 3.2: Exploration through the fitness landscape

Figure 3.2 shows two groups of parent individuals, which are marked by oval boundaries. When explorative genetic operators combine or mutate them, some intermediate offspring may be produced, which are marked by underlining in the figure. Initially, they seem to be quite similar in terms of fitness, but a small number of hill climbing steps prove one of them (i.e., the one marked by an arrow) to be much better. So it is allowed to enter the population and contribute to the optimization process. The fitness landscape shows that this new individual will guide the evolutionary process to better fitness values in comparison to its parents. Thus combination of information of distant individuals facilitates explorations and when it is combined with limited hill climbing, it may lead the search process to diverse, yet better regions of the fitness landscape.

Two variants, named partial and complete, of recombination, crossover and mutation are used as explorative operators. This means a total of six operators are used for exploration. The complete specification of all the six explorative operators and their pseudo-code is presented here. Suppose, **I** is an individual of the current population and $\{\mathbf{I}_S\}$ is its set of 'strangers'.

- Explorative Partial Crossover – Crossover is performed between **I** and a randomly selected stranger from $\{\mathbf{I}_S\}$ on a random number of attributes.

- ❖ Explorative Partial Recombination – Recombination is performed between **I** and a randomly selected stranger from {$\mathbf{I_S}$} on a random number of attributes.
- ❖ Explorative Partial Mutation – Mutation is performed on a random number of attributes of **I.** During mutation of a particular attribute, say x_i, the standard deviation of the Gaussian mutation is set to the average distance of **I** from all of its strangers along the attribute, x_i.
- ❖ Explorative Complete Crossover – For every attribute x_i of **I**, a stranger is selected uniformly at random from {$\mathbf{I_S}$} to perform crossover between **I** and the stranger on x_i.
- ❖ Explorative Complete Recombination - For every attribute x_i of **I**, a stranger is selected uniformly at random from {$\mathbf{I_S}$} to perform recombination between **I** and the stranger on x_i.
- ❖ Explorative Complete Mutation - Mutate every attribute x_i of **I.** The standard deviation of the Gaussian mutation is set uniformly at random from 0% to as large as 50% of the attribute value being mutated in order to ensure large jumps.

3.2.3 Hill Climbing Steps during Exploration

Every time an explorative operator is employed, RMEA makes a number (say, k_4) of uphill moves in the fitness landscape from the newly explored solution to decide whether to accept or reject the new solution. Suppose, an individual **I** is altered by an explorative operator to form a new individual **I′**. To make an uphill move from **I′**, one of its attributes is selected at random and is mutated by using the Gaussian distribution with a small standard deviation (e.g., 5% of the value being mutated). If the mutation improves the fitness of **I**, the mutation is accepted. Otherwise, the individual is rolled back to the state prior to this mutation. This process is repeated for k_4 times. If the final resultant individual, **I″** has a better fitness than the individual **I**, then **I′** (not the final individual **I″**) is accepted and it will replace **I**. Otherwise, **I′** and **I″** are rejected. It is necessary to determine an appropriate value of k_4, which will be generic enough to prove effective for a wide range of problems. It is intuitive that the value of k_4 is related to the properties of the fitness landscape. Since the size of the search space is exponentially related to the dimensionality of the problem, k_4 should be dependant on the dimensionality of the problem. In addition, the value of k_4 should also be dependant on the degree of exploration provoked by the explorative operator which alters **I** to **I′**. RMEA picks the

```
Explorative Partial Crossover
    Input:   I = { ($x_1$, $\eta_1$), ($x_2$, $\eta_2$), ... ... ..., ($x_n$, $\eta_n$) }
    Result:  I'
begin
    I' ← I   [start with an I' identical to I, then alter random no. of attributes of I']
    $I_S$ ← A Stranger of I, selected uniformly at random from the set of strangers {$I_S$}
    r ← a random integer from the range 1...(n-1)
    for i ← 1 to r do begin
        $x_i$ ← an attribute chosen at random from I'
        I'.$x_i$ ← $I_S$. $x_i$   [crossover between I' and $I_S$ on $x_i$]
    end do
    return I'
end

Explorative Partial Recombination
    Input:   I = { ($x_1$, $\eta_1$), ($x_2$, $\eta_2$), ... ... ..., ($x_n$, $\eta_n$) }
    Result:  I'
begin
    I' ← I   [start with an I' identical to I, then alter random no. of attributes of I']
    $I_S$ ← A Stranger of I, selected uniformly at random from the set of strangers {$I_S$}
    r ← a random integer from the range 1...(n-1)
    for i ← 1 to r do begin
        $x_i$ ← an attribute chosen at random from I'
        u ← uniformRandom(0, 1)
        I'.$x_i$ ← $I_S$. $x_i$ * u + I'.$x_i$ *(1−u)   [recombination between I' and $I_S$ on $x_i$]
    end do
    return I'
end

Explorative Partial Mutation
    Input:   I = { ($x_1$, $\eta_1$), ($x_2$, $\eta_2$), ... ... ..., ($x_n$, $\eta_n$) }
    Output:  I'
begin
    I' ← I   [start with an I' identical to I, then alter a single attribute of I']
    r ← a random integer from the range 1... (n-1)
    for i ← 1 to r do
        $x_i$ ← an attribute chosen at random from I'
        sd ← average distance of $x_i$ from the same attributes of all the strangers of I
        g ← GaussianRandom(0, sd)
        I'.$x_i$ ← I'.$x_i$ + I'.$x_i$ * g   [The actual mutation step]
    end do
    return I'
end
```

Algorithm 3.2: Explorative partial operators

```
Explorative Complete Crossover
        Input:    $I = \{ (x_1, \eta_1), (x_2, \eta_2), \ldots\ldots\ldots, (x_n, \eta_n) \}$
        Result:   $I'$
begin
    for i ← 1 to n do begin     [n = number of attributes in an individual]
        $I_r$ ← A Stranger of I chosen at random from its set of Strangers $\{I_S\}$
        $I'.x_i \leftarrow I_r.x_i$    [crossover for attribute $x_i$]
    end do
    return I'
end

Explorative Complete Recombination
        Input:    $I = \{ (x_1, \eta 1), (x_2, \eta 2), \ldots\ldots\ldots, (x_n, \eta_n) \}$
        Result:   $I'$
begin
    for i ← 1 to n do begin  [n = number of attributes in an individual]
        $I_r$ ← A Stranger of I chosen at random from its set of Strangers $\{I_S\}$
        u  ← uniformRandom(0, 1)
        $I'.x_i \leftarrow I_r.x_i * u + I'.x_i * (1-u)$    [recombination of attribute $x_i$]
    end do
    return I'
end

Explorative Complete Mutation
        Input:    $I = \{ (x_1, \eta 1), (x_2, \eta 2), \ldots\ldots\ldots, (x_n, \eta_n) \}$
        Result:   $I'$
begin
    for i ← 1 to n do begin          [n = number of attributes in an individual]
        r ← uniformRandom[0, 0.50]  [r is random, uniformly distributed in 0…0.50]
        sd ← $r * I.x_i$   [the s.d. of gaussian mutation is set to as large as 50% of the
                              attribute value being mutated, i.e. $x_i$]
        g  ← GaussianRandom(0, sd)
        $I'.x_i \leftarrow I'.x_i + I'.x_i * g$    [mutation of the attribute $x_i$]
    end do
    return I'
end
```

Algorithm 3.3: Explorative complete operators

value of k_4 uniformly at random from the range [1…*m*/2], where *m* is the number of attributes altered by the immediate explorative operator. If the immediate operator is a complete explorative operator, i.e. it alters all the *n* attributes of **I,** k_4 is selected from [1…*n*/2], where *n* is the dimensionality of the problem.

3.2.4 Exploitation Stage

The objective of the exploitation stage is to reach the optimum points represented by the local neighbourhoods of the population. Unlike exploration, genetic operators are applied on the sets of nearest 'neighbours' to achieve this goal. Individuals that are close to each other in the search space are referred as 'neighbours'. For each individual **I**, RMEA keeps track of the nearest *N* neighbours across the population. As exploitation is focused to reach the local optima within the neighbourhood, a small step size is appropriate for mutation. Six genetic operators, similar to the exploration stage, are used for exploitations. The only difference between the explorative and exploitative operators is that, recombination and crossover are performed by using neighbours in stead of strangers and mutations are performed with small step size. In this stage, only improvements are accepted, i.e., fitter offspring are allowed to replace the weaker parents to enter the neighbourhood. As a result, population diversity tends to drop in this stage.

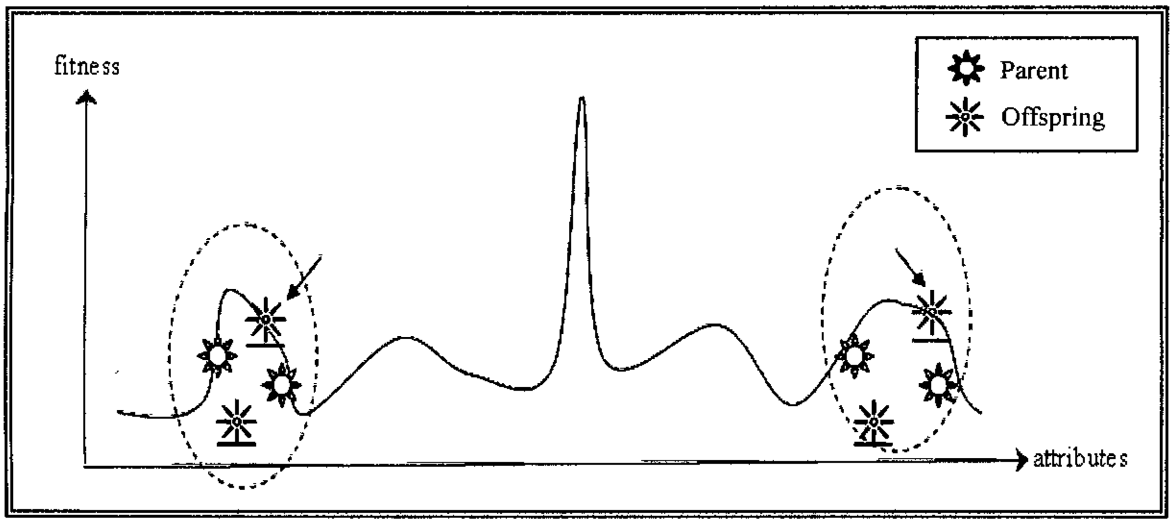

Figure 3.3: Exploitation through the fitness landscape

Figure 3.3 exemplifies how local operations help to reach the local optima. Two groups (neighbourhoods) of individuals are marked by oval boundaries. Recombination and crossover involving individuals within the same neighbourhood and mutation with small step size produce offspring at the same neighbourhood (marked by underlining). Since only better individuals are accepted, the two offspring, closer to the local peaks (pointed by arrows) are allowed to enter the population. This moves the two neighbourhoods closer to their local optimum points, which is the aim of the exploitation stage.

The six explorative operators are the 'partial' and 'complete' variants of crossover, recombination and mutation. Exploitative crossover and recombination differ from the explorative ones in the way that they involve the neighbours, in stead of the strangers. The standard deviation of the exploitative partial mutation is picked from the average distance of the neighbors, rather than strangers, along that attribute being mutated. The s.d. of exploitative complete mutation is picked uniformly at random from [0%...5%] of the attribute value being mutated, which ensures much smaller jumps than the explorative mutation that uses as large as 50% of the attribute value as s.d. The pseudo-code of each of the exploitative operators is presented in algorithm 3.4, 3.5.

3.3 Differences with Existing Works

RMEA differs from most existing approaches on a number of aspects. First, it realizes the possibility of automatically distributing the conflicting goals of exploitation and exploration across its recurring stages. Unlike some algorithms [12] that have to manually make decisions whether to switch from one stage to another, RMEA recurrently switches its stages at regular intervals. Since different stages repeat again and again, there is no need to make a 'perfect switching' and the conflicting goals of evolution are automatically distributed across the generations of the recurring stages.

Second, a number of control parameters are introduced in RMEA to provide a precise control over its degree of conventional, explorative and exploitative nature. Since optimization problems vary widely from each other, and no simple algorithm can perform well for all problems [21], an EA should be properly parameterized to enable the user to control several features of the algorithm. However, unlike some algorithms that require adequate problem specific knowledge from the user (e.g., threshold values of diversity [12], or relative weights between fitness and diversity [19, 20]), RMEA requires no deep knowledge from the user. Besides, an adaptive variant of RMEA, entitled as Ada-RMEA, is also proposed that requires no parameter from the user.

Third, RMEA defines a number of different exploitative and explorative operations, which select the participating individuals from a population based on their distance. Different operators are applied in different stages depending on the explorative or exploitative objective of that stage. This is quite different from most algorithms [12-22] that use the same set of operators throughout the evolution.

```
Exploitative Partial Crossover
      Input:   I = { (x_1, η_1), (x_2, η_2), ... ... ..., (x_n, η_n) }
      Result:  I'
begin
   I' ← I   [start with an I' identical to I, then alter random no. of attributes of I']
   I_N ← A neighbor of I, selected uniformly at random from the set of neighbors {I_N}
   r  ← a random integer from the range 1...(n-1)
   for i ← 1 to r do begin
        x_i  ← an attribute chosen at random from I'
        I'.x_i  ← I_N. x_i   [crossover between I' and I_N on x_i]
   end do
   return I'
end

Exploitative Partial Recombination
      Input:   I = { (x_1, η_1), (x_2, η_2), ... ... ..., (x_n, η_n) }
      Result:  I'
begin
   I' ← I   [start with an I' identical to I, then alter random no. of attributes of I']
   I_N ← A neighbor of I, selected uniformly at random from the set of neighbors {I_N}
   r  ← a random integer from the range 1...(n-1)
   for i ← 1 to r do begin
        x_i  ← an attribute chosen at random from I'
        u   ← uniformRandom(0, 1)
        I'.x_i  ← I_N. x_i * u + I'.x_i * (1−u)  [recombination between I' and I_N on x_i]
   end do
   return I'
end

Exploitative Partial Mutation
      Input:   I = { (x_1, η_1), (x_2, η_2), ... ... ..., (x_n, η_n) }
      Result:  I'
begin
   I' ← I   [start with an I' identical to I, then alter random no. of attributes of I']
   r  ← a random integer from the range 1... (n-1)
   for i ← 1 to r do begin
        x_i  ← an attribute chosen at random from I'
        sd  ← average distance of x_i from the same attributes of all the neighbors of I
        g   ← GaussianRandom(0, sd)
        I'.x_i  ←  I'.x_i + I'.x_i * g   [The actual mutation step]
   end do
   return I'
end
```

Algorithm 3.4: Exploitative partial operators

```
Exploitative Complete Crossover

        Input:    I = { (x_1, η_1), (x_2, η_2), ... ... ..., (x_n, η_n) }
        Result:   I'
begin
    for i ← 1 to n do begin    [n = number of attributes in an individual]
        I_r ← A Neighbor of I chosen at random from its set of Neighbors {I_N}
        I'.x_i ← I_r. x_i    [crossover of attribute x_i]
    end do
    return I'
end

Exploitative Complete Recombination

        Input:    I = { (x_1, η_1), (x_2, η_2), ... ... ..., (x_n, η_n) }
        Result:   I'
begin
    for i ← 1 to n do begin  [n = number of attributes in an individual]
        I_r ← A Neighbor of I chosen at random from its set of Neighbors {I_N}
        u ← uniformRandom(0, 1)
        I'.x_i ← I_r. x_i * u + I'.x_i * (1−u)  [recombination of attribute x_i]
    end do
    return I'
end

Exploitative Complete Mutation

        Input:    I = { (x_1, η_1), (x_2, η_2), ... ... ..., (x_n, η_n) }
        Result:   I'
begin
    for i ← 1 to n do begin        [n = number of attributes in an individual]
        r ← uniformRandom[0, 0.05] [r is random, uniformly distributed in 0...0.05]
        sd ← r * I'.x_i   [the s.d. of gaussian mutation is set to only ≤ 5% of the
                           attribute value being mutated, i.e. x_i]
        g ← GaussianRandom(0, sd)
        I'.x_i ← I.x_i + I.x_i * g   [mutation of the attribute x_i]
    end do
    return I'
end
```

Algorithm 3.5: Exploitative complete operators

Fourth, extensive empirical studies have been carried out to analyze and evaluate the performance of RMEA. A suite of 23 benchmark numerical optimization problems is used for empirical studies. Few evolutionary systems have been tested on a similar range of problems. The experimental results show that RMEA performs better than other algorithms for most of the problems.

3.5 Review of RMEA

A little reflection about the mechanism of RMEA makes it readily apparent that RMEA automatically and gracefully distributes the local and global search objectives through its recurring stages across the generations. The diversity of the population will usually increase and decrease as a result of the use of strangers and neighbors, respectively, by the genetic operators. The exploitative stage usually leads to the discovery of local optimum points and ends with quick decrease in population diversity, while the explorative stage, in combination with limited hill-climbing, enhances population diversity, and helps discover new optimum points of the fitness landscape which have better possibility of being the global optima. No existing evolutionary algorithm exhibit such an automatic dispense of exploitative, explorative and diversity preserving objectives.

Although it seems that RMEA is a bit more complex than CEP, its essence is the use of three different stages with specific object-oriented genetic operators for proper balancing between exploitations and explorations in the evolution. A drawback of RMEA is the increased computational cost. For each individual, the algorithm has to keep track of neighbors and strangers. Besides, hill-climbing steps in the fitness landscape during exploration require k_4 extra fitness evaluations.

RMEA represents a new framework for evolutionary processes to balance the exploitation and exploration. The applicability and usefulness of this framework can further be determined through extensive research and experiments. In an effort to discover the potentials of RMEA, this thesis work is oriented towards solving a number of benchmark numerical optimization problems by employing RMEA, which is presented in the following chapter.

Chapter 4

Experimental Studies

4.1 Introduction

This chapter presents an in-depth experimental study with RMEA. The performance of RMEA is evaluated with a suite of 23 benchmark functions and compared with standard evolutionary algorithm, cellular evolutionary algorithm, diversity guided evolutionary algorithm, self-organized criticality evolutionary algorithm, classical evolutionary programming, fast evolutionary programming and improved fast evolutionary programming. In addition, experiments have been carried out to analyze the sensitivity of the parameters with results, significance and effectiveness of the exploitative and explorative operators, roles and effects of the different stages, gradual variation of the population diversity by the evolutionary process and so on. An adaptive variant of RMEA, named as Ada-RMEA, is also proposed which adjusts the lengths of the conventional, exploitative and explorative stages depending on their relative effectiveness in fitness improvement. Ada-RMEA is also evaluated with the benchmark functions and compared with RMEA. The effect of adaptation on solutions and stage lengths is also examined.

4.2 Benchmark Functions:

RMEA is evaluated by using a suite of 23 benchmark numerical optimization problems. The objective of each problem is to find the global minimum of a multi-dimensional function. This becomes particularly difficult when the functions are high dimensional and multimodal, because the volume of the search space and the number of local minima increases exponentially with the number of dimensions. The suite of functions include both unimodal and multimodal, low and high dimensional functions. A brief summary of the functions is presented in Table 4.1. Among them, functions f_1 to f_7 are unimodal functions. Function f_6 is a step function, which is discontinuous, discreet valued and has a single global minima. Function f_7 is noisy quartic function, where $random[0,1]$ simulates random noise. Functions f_8 to f_{13} are high dimensional (with dimensionality = 30) multimodal functions, for which the number of local minima increases exponentially with the number of dimensions. Functions f_{14} to f_{23} are low dimensional multimodal functions with a smaller number of local minima.

Table 4.1
*The 23 benchmark functions used in the experimental study. Here n is the dimensionality of the function, **S** is the domain of the variables and f_{min} is the minimum value of the function. A detailed description of each function is provided in Appendix A.*

Test function	n	S	$f_{\min}$
$f_1(x)=\sum_{i=1}^{n} x_i^2$	30	$[-100, 100]^n$	0
$f_2(x)=\sum_{i=1}^{n} \lvert x_i\rvert + \prod_{i=1}^{n} \lvert x_i\rvert$	30	$[-10, 10]^n$	0
$f_3(x)=\sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)^2$	30	$[-100, 100]^n$	0
$f_4(x)=\max_i\left\{\lvert x_i\rvert,\ 1 \le i \le n\right\}$	30	$[-100, 100]^n$	0
$f_5(x)=\sum_{i=1}^{n-1}\left[100\left(x_{i+1}-x_i^2\right)^2+\left(x_i-1\right)^2\right]$	30	$[-30, 30]^n$	0
$f_6(x)=\sum_{i=1}^{n}\left(\lfloor x_i+0.5\rfloor\right)^2$	30	$[-100, 100]^n$	0
$f_7(x)=\sum_{i=1}^{n} ix_i^4 + random\left[0, 1\right)$	30	$[-1.28, 1.28]^n$	0
$f_8(x)=\sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i\rvert}\right)$	30	$[-500, 500]^n$	-12569
$f_9(x)=\sum_{i=1}^{n}\left[x_i^2-10\cos\left(2\pi x_i\right)+10\right]$	30	$[-5.12, 5.12]^n$	0

Table 4.1(continued): The 23 benchmark functions used in the experimental study

Test function	n	S	$f_{\min}$
$f_{10}(x)=-20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right)-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right)+20+e$	30	$[-32, 32]^n$	0
$f_{11}(x)=\frac{1}{4000}\sum_{i=1}^{n}\left(x_i^2\right)-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	30	$[-600, 600]^n$	0
$f_{12}(x)=\frac{\pi}{n}\left\{10\sin^2(\pi y_1)+(y_n-1)^2+\sum_{i=1}^{n-1}(y_i-1)^2\left[1+10\sin^2(\pi y_{i+1})\right]\right\}+\sum_{i=1}^{n}u(x_i, 10, 100, 4)$	30	$[-50, 50]^n$	0
$f_{13}(x)=0.1\left\{\sin^2(3\pi x_1)+(x_n-1)^2\left[1+\sin^2(2\pi x_n)\right]+\sum_{i=1}^{n-1}(x_i-1)^2\left[1+\sin^2(3\pi x_{i+1})\right]\right\}+\sum_{i=1}^{n}u(x_i, 5, 100, 4)$	30	$[-50, 50]^n$	0
$f_{14}(x)=\left[\frac{1}{500}+\sum_{j=1}^{25}\frac{1}{j+\sum_{i=1}^{2}\left(x_i-a_{ij}\right)^6}\right]^{-1}$	2	$[-65.536, 65.536]^n$	1
$f_{15}(x)=\sum_{i=1}^{11}\left[a_i-\frac{x_1\left(b_i^2+b_i x_2\right)}{b_i^2+b_i x_3+x_4}\right]^2$	4	$[-5, 5]^n$	0.000307
$f_{16}(x)=4x_1^2-2.1x_1^4+\frac{1}{3}x_1^6+x_1x_2-4x_2^2+4x_2^4$	2	$[-5, 5]^n$	-1.03162
$f_{17}(x)=\left(x_2-\frac{5.1}{4\pi^2}x_1^2+\frac{5}{\pi}x_1-6\right)^2+10\left(1-\frac{1}{8\pi}\right)\cos(x_1)+10$	2	$[-5, 10] \times [0, 15]$	0.398
$f_{18}(x)=\left[1+(x_1+x_2+1)^2\left(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2\right)\right]\times\left[30+(2x_1-3x_2)^2\left(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2\right)\right]$	2	$[-2, 2]^n$	3
$f_{19,20}(x)=-\sum_{i=1}^{4}c_i\exp\left[-\sum_{j=1}^{m}a_{ij}\left(x_j-p_{ij}\right)^2\right]\quad m=4, 6$	4, 6	$[0, 1]^n$	-3.86, -3.32
$f_{21,22,23}(x)=-\sum_{i=1}^{m}\left[(x-a_i)(x-a_i)^T+c_i\right]^{-1}\quad m=5, 7, 10$	4	$[0, 10]^n$	-10

Minimizing high dimensional multimodal functions is always considered challenging for any search algorithm. In the experiments with the multimodal functions f_8–f_{23}, escaping local optima is the focus of our interest, while for unimodal functions f_1–f_7, convergence rate is also important. Details of each function are available in the Appendix A. Few algorithms have been tested on a similar range of problems, as RMEA. Results, presented in the following sections, demonstrate that RMEA performs extremely well on most of these functions.

4.3 Experimental Setup

During the conventional stage, the same set of parameters, as in [22], is used: population size, $M = 100$, tournament size $q = 10$, initial standard deviations = 3.0 and self-adaptive standard deviations are used for all the test functions. The iteration counters k_1, k_2 and k_3 for the conventional, exploration and exploitation stages are set to 30 for the unimodal and multimodal functions f_1 - f_{13}, and f_{15}. However, for the rest of the functions, which require smaller number of generations (i.e., 100 or 200), k_1, k_2, and k_3 are equally set to 10% of their execution length (i.e., 10 or 20). For the high dimensional multimodal functions f_1 - f_{13}, the value of k_4, i.e., the number of hill climbing steps after exploration, is picked uniformly at random from $[1\ldots m/2]$, where m is the number of attributes affected by the preceding explorative operator. In each generation, the neighbourhood size is set randomly to 1% to 5% of the population size. Random values are chosen, because they always provide better immunity than fixed values against local minima. All these values are selected after some preliminary experiments and are not meant to be optimal. The encoding of a chromosome (individual) is pretty straightforward. Each chromosome possess n pairs of (x, η) values, where the i-th pair (x_i, η_i) represents the value of i-th objective variable, x_i and the corresponding standard deviation, η_i which is used as standard deviation to mutate x_i by CEP. RMEA is executed 50 times on each function. The mean and standard deviation of the best solutions found by RMEA are summarized in tables of the following sections.

4.4 Experimental Results

4.4.1 Unimodal Functions

The first set of experiments was aimed to compare the convergence rate of CEP and RMEA for the unimodal functions f_1–f_7. The average results and the progress of the mean best solutions with generations found by CEP and RMEA over 50 independent runs are summarized in Table 4.2 and Figure 4.1, respectively. It is apparent that RMEA performs much better both in terms of the convergence rate and final solution quality than CEP for all these functions, except f_5. For example, for function f_1, RMEA displays a faster convergence rate than CEP from the very beginning. It quickly minimizes the function down to 0.001 within only 700 generations, while CEP requires the entire evolutionary run of 1500 generations to reach the value of 0.001. RMEA continues to approach the global optima with the nearly same constant rate, and after 1500 generations, it reaches in the neighborhood of 10^{-12}, which proves the outstanding convergence rate and fine-tuning ability of RMEA.

CEP reaches the minimum value of 0.93 and 2.08, respectively, for functions f_3 and f_4, while RMEA minimizes these two functions in the range of 10^{-45}, and 10^{-10}. In fact, the optimization achieved by CEP for both these functions in 5000 generations is achieved by RMEA within only 800 generations, as indicated in Figure 4.1. Similar result is found for f_7, for which RMEA exhibits its excellent optimization ability by minimizing in the range of 10^{-51}. From the graphs in Figure 4.1, it is apparent that RMEA shows much better convergence rate throughout the entire evolution and the rate of minimization does not seem to stagnate during the late generations. This proves the effectiveness of RMEA with its periodically alternating stages in fulfilling both exploitative and explorative objectives during the evolution. The only exception in this group is function f_5, for which RMEA performs worse than CEP.

The step function f_6 is quite different from the others, which is distinguished by plateaus, and abrupt, discontinuous edges between plateaus. CEP fails miserably for this function because it employs $N(0,1)$ distribution for mutation, which usually produces small steps, but all the points around a point within a plateau have similar fitness value, except when the points cross they plateau boundaries. Hence it is quite difficult for CEP to generate a large jump and move from one plateau to another one. On the other hand, RMEA exhibits a much higher probability of generating longer jumps than CEP since its explorative operators are focused to do so by combining distant individuals and large mutation steps. This enables RMEA to move from one plateau to a lower one with relative ease. The rapid convergence of RMEA within only 350 generations in most of the runs, as we have observed, supports this claim.

However, this significantly superior performance is not absolutely 'free of cost'. In each generation, RMEA has to perform some extra computations during the exploration and exploitation stages. RMEA has to keep track of the neighbors and strangers for each individual and perform some additional fitness evaluations for hill climbing during exploration. On the other side, the application of the 'partial' genetic operations by RMEA requires less cost than the mutation of CEP that 'completely' alters an individual. Since the results are considerably better, as illustrated by the following tables, it certainly justifies the overhead of some extra computations.

Table 4.2
Comparison between CEP and RMEA on unimodal functions, f_1–f_7.
"Mean Best" is the mean of the best function values averaged over 50 Runs.

Function	Generations	Algorithm	Mean Best	Standard Deviation	Best Performance by
f_1	1500	CEP	1.09×10^{-3}	2.8×10^{-3}	RMEA
		RMEA	1.06×10^{-12}	1.13×10^{-12}	
f_2	2000	CEP	5.62×10^{-2}	7.2×10^{-3}	RMEA
		RMEA	2.66×10^{-11}	9.37×10^{-12}	
f_3	5000	CEP	0.93	1.15	RMEA
		RMEA	1.06×10^{-45}	7.19×10^{-46}	
f_4	5000	CEP	2.08	1.29	RMEA
		RMEA	6.81×10^{-10}	3.90×10^{-10}	
f_5	20000	CEP	12.77	6.28	CEP
		RMEA	20.89	0.748	
f_6	1500	CEP	82.17	151.43	RMEA
		RMEA	0	0	
f_7	3000	CEP	5.4×10^{-2}	1.7×10^{-2}	RMEA
		RMEA	1.24×10^{-51}	1.07×10^{-51}	

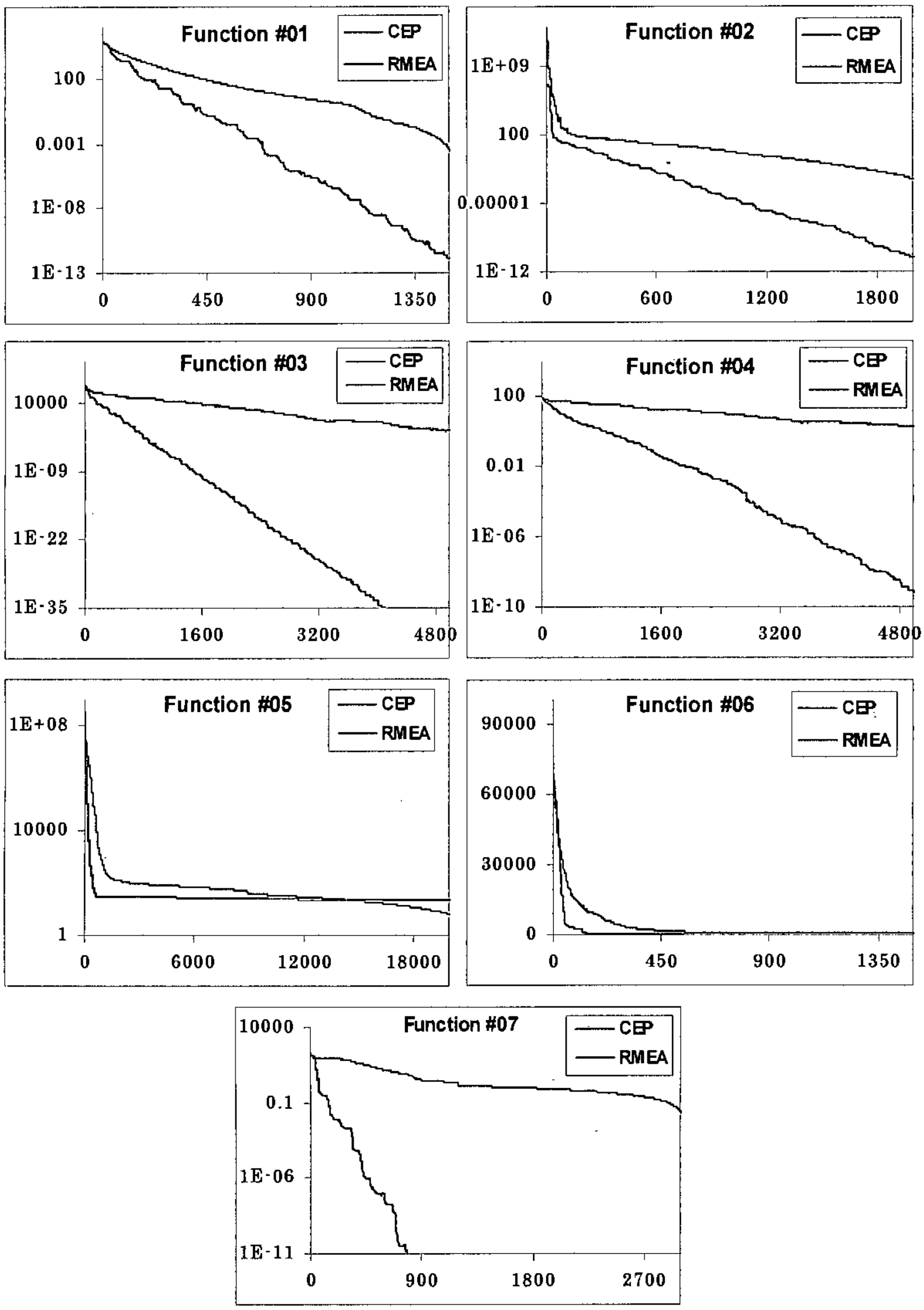

Figure 4.1: Comparison between CEP and RMEA on unimodal functions f_1–f_7. Vertical axis shows the function values, while the horizontal axis shows no. of generations. Results are averaged over 50 runs.

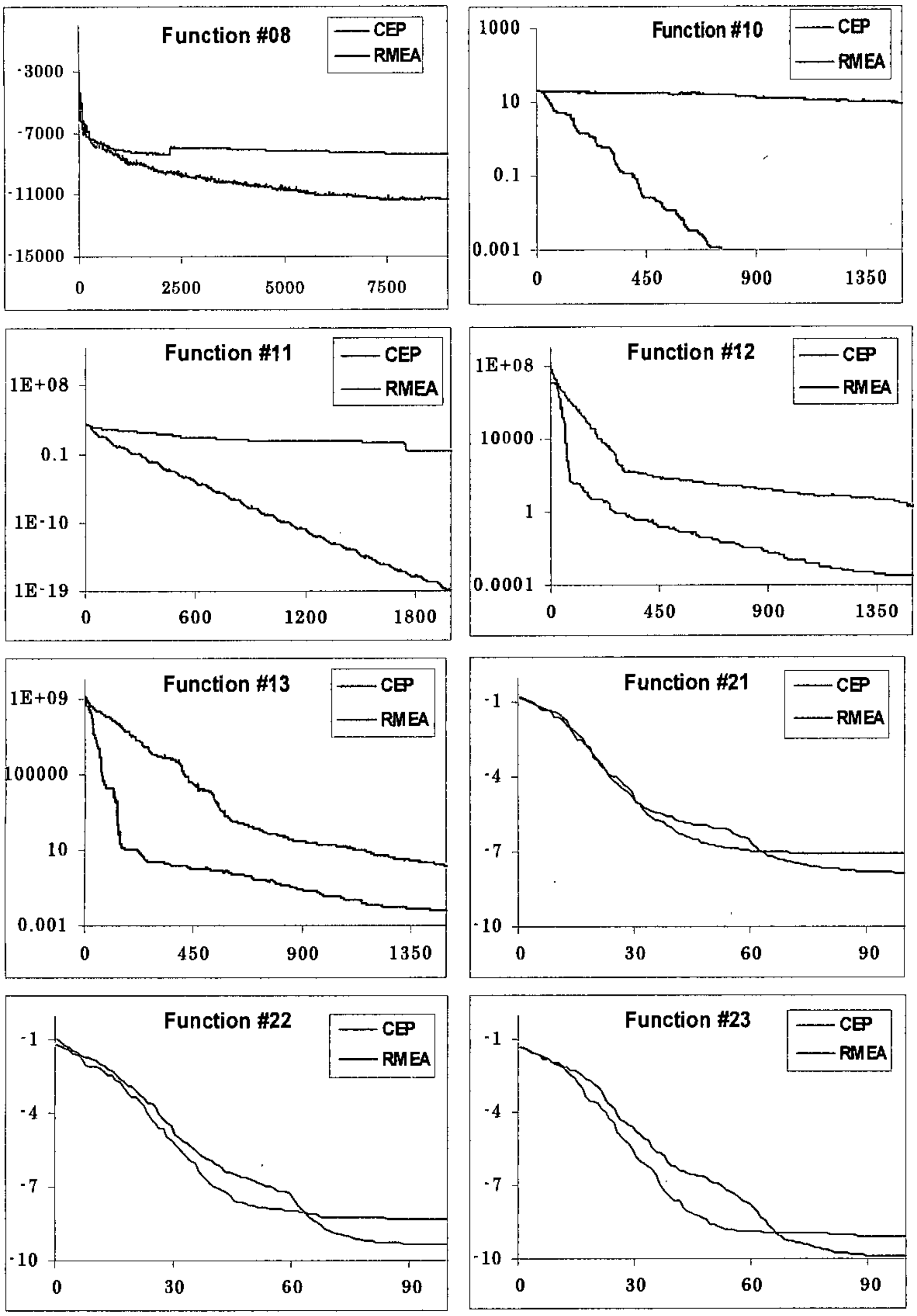

Figure 4.2: Comparison between CEP and RMEA on multimodal functions. Vertical axis shows the function values, while the horizontal axis shows no. of generations. Results are averaged over 50 runs.

4.4.2 Multimodal Functions with Many Local Minima

Multimodal functions have been proved to be quite troublesome for many optimization algorithms. Functions f_8–f_{13} are high-dimensional multimodal functions, with dimensionality set to 30. Their number of local minima increases exponentially with the number of dimensions. Table 4.3 summarizes the comparison results for these functions. It is apparent from the results that RMEA consistently performs significantly better than CEP for all these functions. CEP appears to be trapped in some poor local optima, while RMEA successfully gets rid of local minima and finds the global minima. RMEA discovers the global minima and reaches very close to it for all the functions in this group, except f_8. By examining the graphs, some of which are presented in Figure 4.2, a number of interesting results can be found. The minimization performed by CEP for f_8 and f_{10} in 9000 and 1500 generations, respectively, is achieved very rapidly i.e. within only 500 and 100 generations by RMEA. RMEA takes less than 400, 350 and 450 generations to minimize f_{11}–f_{13} to a level for which CEP requires 2000, 1500 and 1500 generations, respectively. It is observed from Figure 4.2 that RMEA shows better convergence rate from the very beginning and continues with similar convergence rate throughout the entire evolution. The only exception is function f_8, which has the global minima of -12569.5. RMEA starts with excellent convergence rate for f_8 and soon minimizes close to -10000. It then loses its convergence rate and almost stagnates at that value. This behavior is quite similar to CEP for function f_8 (and f_5). However, RMEA is somewhat better, since CEP is trapped to a worse local minimum with function value close to -8000. Such unexpected result with a few functions is nothing discouraging, since optimization problems have wide-ranging dissimilarity, and a particular approach never works well with every problem [21].

4.4.3 Multimodal Functions with Few Local Minima

RMEA was applied on the low dimensional multimodal functions f_{14}–f_{23}. Each function in this family has dimensionality ≤ 6, and has only a few local minima. So they are simpler than the high dimensional multimodal ones. Table 4.4 summarizes the results, averaged over 50 runs. RMEA performs similar as CEP for the functions f_{16}–f_{19}, and outperforms CEP for the rest of the functions, i.e., f_{14}–f_{15} and f_{20}–f_{23}. CEP is trapped in a poor local minimum for the functions f_{21}–f_{23}. Although, for the functions f_{16}–f_{19}, both CEP and RMEA reaches the global minima and show similar behavior, this is due to the fact that these functions have very low dimensionality (with dimensions = 2, 2, 2 and 4 respectively) and thus quite easy to minimize. However, functions f_{20}–f_{23} have relatively higher dimensionality (with dimensions = 6, 4, 4, 4) and the better performance of RMEA clearly speaks of the superiority of RMEA.

Table 4.3
Comparison between CEP and RMEA on high dimensional multimodal functions f_8–f_{13}. "Mean Best" is the mean of the best function values averaged over 50 Runs.

Function	Generations	Algorithm	Mean Best	Standard Deviation	Best Performance by
f_8	9000	CEP	-8174.8	572.2	RMEA
		RMEA	-11332.6	61.2	
f_9	5000	CEP	1.87×10^{-7}	9.18×10^{-8}	RMEA
		RMEA	5.34×10^{-51}	1.56×10^{-51}	
f_{10}	1500	CEP	9.35	4.72	RMEA
		RMEA	1.18×10^{-7}	6.27×10^{-8}	
f_{11}	2000	CEP	1.59×10^{-1}	0.27	RMEA
		RMEA	1.01×10^{-20}	4.64×10^{-21}	
f_{12}	1500	CEP	1.89	2.64	RMEA
		RMEA	2.77×10^{-4}	1.09×10^{-4}	
f_{13}	1500	CEP	1.95	3.79	RMEA
		RMEA	5.99×10^{-3}	4.13×10^{-3}	

Table 4.4
Comparison between CEP and RMEA on the low dimensional multimodal functions, f_{14}–f_{23}. "Mean Best" is the mean of the best function values averaged over 50 Runs.

Algorithm	Function Value (Mean Best)									
	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}	f_{21}	f_{22}	f_{23}
Generations	100	4000	100	100	100	100	200	100	100	100
CEP	1.41	$5.9\text{x}10^{-4}$	-1.03	0.398	3.0	-3.86	-3.31	-6.98	-8.32	-9.18
RMEA	1.19	$3.25\text{x}10^{-4}$	-1.03	0.398	3.00	-3.86	-3.38	-7.58	-9.41	-9.96
Best Performance by	RMEA	RMEA	Similar	Similar	Similar	Similar	RMEA	RMEA	RMEA	RMEA

4.5 Comparison with Existing Works

This section compares RMEA with standard evolutionary algorithm (SEA), self-organized criticality evolutionary algorithm (SOCEA), cellular evolutionary algorithm (CEA), diversity guided evolutionary algorithm (DGEA), fast evolutionary programming (FEP), and improved fast evolutionary programming (IFEP). The SEA uses Gaussian distribution for mutation, with mean = 0 and variance = 1.0/[√(t+1)]. The SOCEA shows much better performance than SEA by adjusting the mutation variance following a scheme [38] which ensures variable step sizes for mutation. The CEA employs a population with 400 individuals, organized as a 20 X 20 grid, with boundaries and corners wrapped around. An individual can mate with any of its four neighbours, selected at random, and better offspring replaces the central parent. The DGEA uses the diversity thresholds of d_{low} = $5\text{x}10^{-6}$ and d_{high} = 0.25. Further clarifications on each of the algorithms are available in [12], [38], [39].

Four standard benchmark optimization (i.e., minimization) problems are used to compare the algorithms. Based on the dimensionality, three variants of each problem is employed, with dimensions = 20, 50, and 100. For each problem, the total number of generations is set to 50 times the dimensionality of the problem, i.e., for 20, 50, and 100 dimensions, the algorithms are executed for 1000, 2500, and 5000 generations respectively. The four problems are: Rosenbrock's function: f_5, Rastrigin's function: f_9, Ackley's function: f_{10} and Griewank function: f_{11}. The details and analytical forms of each of the functions are provided in [12], and also in Appendix A. Table 4.5 compares the performance of RMEA with the others.

Table 4.5
Comparison among SEA, SOCEA, CEA and RMEA on a number of benchmark functions. Results represent the mean of the best function values, averaged over 20 runs.

Function	Dimensions	Algorithm				Best Performance By
		SEA	SOCEA	CEA	RMEA	
Rosenbrock f_5	20	8292.3	406.4	149.0	16.79	RMEA
	50	41425.6	4783.2	1160.0	48.2	
	100	91250.3	30427.6	6053.8	97.8	
Rastrigin f_9	20	11.12	2.87	1.25	4.53×10^{-12}	RMEA
	50	44.67	22.46	14.22	1.18×10^{-19}	
	100	106.21	86.36	58.38	4.93×10^{-25}	
Ackley f_{10}	20	2.49	0.63	0.23	1.76×10^{-6}	RMEA
	50	2.87	1.52	0.65	3.26×10^{-10}	
	100	2.89	2.22	1.14	2.23×10^{-12}	
Griewank f_{11}	20	1.17	0.93	0.64	25.99×10^{-13}	RMEA
	50	1.61	1.14	1.03	1.26×10^{-19}	
	100	2.25	1.62	1.17	1.21×10^{-23}	

The excellent performance of RMEA is apparent from the results of Table 4.5. RMEA outperforms the others by several orders of magnitude for all the four functions. For example, for the 100 dimensional variant of the Rastrigin, Ackley and Griewank functions, CEA performs minimization down to 106.21, 2.89, and 2.25, while RMEA minimizes them to a level as low as 10^{-8}, 10^{-4}, and 10^{-8} respectively. For the Rosenbrock function, all the SEA, SOCEA, and CEA are far away from the global minima of zero, while RMEA has rapidly converged much closer to the global minima, which is evident from the results.

Table 4.6 compares RMEA with DGEA on the same set of functions. DGEA controls the diversity of the population in an explicit manner [12], and shows significantly better performance than the SEA, SOCEA and CEA. Both DGEA and RMEA are executed for the same number of generations for each function, while DGEA* is executed until the best solution value does not improve for 500 generations.

Table 4.6 exhibits the superiority of RMEA over DGEA and DGEA*. Results show that RMEA outperforms both DGEA and DGEA* on almost all the functions. DGEA* shows better performance only with the low dimensional variant of the Rosenbrock function f_5, while RMEA surpasses DGEA* by orders of magnitude for all the rest of the functions. This performance from RMEA is quite encouraging, since DGEA* always gets the extra favour of executing more generations than the others, until its fitness values stagnate. Summarizing all these results, it becomes apparent that RMEA exhibits superior performance to all the other approaches.

Fast evolutionary programming (FEP), first introduced in [22], employs *cauchy* distribution, in stead of the widely used *gaussian* $N(0,1)$ distribution, to mutate the individuals. FEP is evaluated with the same suite of 23 benchmark functions, and it has demonstrated good performance, especially with the high dimensional multimodal functions, which are considered as the most difficult family of functions for optimization. Another improvement over FEP, Improved Fast Evolutionary Programming (IFEP) combines the features of CEP and FEP by using both *gaussian* $N(0,1)$ and *cauchy*(1) distributions in order to produce two offspring from each parent, and accepting only the better one. IFEP further improves the behaviour of FEP and exhibits better performance than FEP and CEP.

Table 4.6
Comparison among DGEA, DGEA and RMEA on a number of benchmark functions. Results represent the mean of the best function values, averaged over 20 runs.*

Function	Dimensions	Algorithm			Best Performance By
		DGEA	DGEA*	RMEA	
Rosenbrock f_5	20	96.00	8.12	16.79	DGEA*
	50	315.39	59.78	48.2	RMEA
	100	1161.55	880.32	97.8	RMEA
Rastrigin f_9	20	2.21×10^{-5}	3.37×10^{-8}	4.53×10^{-12}	RMEA
	·50	0.0166	1.97×10^{-6}	1.18×10^{-19}	RMEA
	100	0.1566	6.56×10^{-5}	4.93×10^{-25}	RMEA
Ackley f_{10}	20	8.05×10^{-4}	3.36×10^{-5}	1.76×10^{-6}	RMEA
	50	4.61×10^{-3}	2.52×10^{-4}	3.26×10^{-10}	RMEA
	100	0.01329	9.80×10^{-4}	2.23×10^{-12}	RMEA
Griewank f_{11}	20	7.02×10^{-4}	7.88×10^{-8}	25.99×10^{-13}	RMEA
	50	4.40×10^{-3}	1.19×10^{-3}	1.26×10^{-19}	RMEA
	100	0.01238	3.24×10^{-3}	1.21×10^{-23}	RMEA

In the following section, RMEA is compared with both FEP and IFEP. Table 4.7 shows results on unimodal functions. Results show that RMEA outperforms the others on five functions, shows similar result on one (f_6) and worse performance with only one (f_5). The superiority of RMEA is even by order of magnitude for the functions f_1-f_4, and f_7. For example, RMEA minimizes f_2, f_3 and f_7 down to 10^{-11}, 10^{-45}, 10^{-51}, while IFEP (or FEP) minimizes them only to 10^{-4}, 10^{-2}, and 10^{-3} respectively. For the step function f_6, RMEA finds the global minima quite easily, within the first 350 generations in most of the runs, as we have observed. The only exception in this family is the function f_5, for which FEP performs better than RMEA.

Table 4.8 compares RMEA on high dimensional multimodal functions f_8–f_{13}. FEP (and IFEP) is better effective for such difficult class of functions [22]. Out of the six functions, RMEA outperforms both FEP and IFEP in three, even by an order of magnitude. For f_9, f_{10}, and f_{11} RMEA performs minimization to 10^{-51}, 10^{-7}, 10^{-20} while IFEP or FEP exhibits minimization only to 10^{-2}, 10^{-3}, and 10^{-2} respectively. However, for the remaining three functions: f_8, f_{12} and f_{13}, FEP performs better than RMEA. An interesting optimization pattern is observed for Schwefel's problem, f_8. Here, the exploration stage is found to operate in the reverse direction of the exploitation and conventional stage to deteriorate solution quality. RMEA employs limited number of hill climbing steps to prevent such situation, but hill climbing seems to fail with exploration, only for this function. Some more sophisticated hill climbing considering the properties of the fitness landscape may improve this situation. However, for the low dimensional multimodal functions f_{14}-f_{23}, RMEA always performs better than (or at least as well as) FEP and IFEP, as exhibited by Table 4.9. Comparison of the results for the functions f_{14}, f_{18}, f_{20}, f_{21}, f_{22} and f_{23} speaks for the superiority of RMEA. Summarizing all the results, it may be concluded that, RMEA demonstrates very promising results and some more investigation with its workings may further improve its performance.

Table 4.7
Comparison among FEP, IFEP and RMEA on unimodal functions, f_1–f_7. "Mean Best" is the mean of the best function values averaged over 50 Runs. "N.A." means result is not available in [22].

Function	Generations	Algorithm	Mean Best	Standard Deviation	Best Performance by
f_1	1500	FEP	5.7×10^{-4}	1.3×10^{-4}	RMEA
		IFEP	4.6×10^{-5}	N.A.	
		RMEA	1.06×10^{-12}	1.13×10^{-12}	
f_2	2000	FEP	8.1×10^{-3}	7.7×10^{-4}	RMEA
		IFEP	2.44×10^{-4}	N.A.	
		RMEA	2.66×10^{-11}	9.37×10^{-12}	
f_3	5000	FEP	1.6×10^{-2}	1.4×10^{-2}	RMEA
		IFEP	N.A.	N.A.	
		RMEA	1.05×10^{-45}	7.19×10^{-46}	
f_4	5000	FEP	0.3	0.5	RMEA
		IFEP	N.A.	N.A.	
		RMEA	6.81×10^{-10}	3.90×10^{-10}	
f_5	20000	FEP	5.06	5.87	FEP
		IFEP	N.A.	N.A.	
		RMEA	20.89	0.748	
f_6	1500	FEP	0	0	RMEA, FEP
		IFEP	N.A.	N.A.	
		RMEA	0	0	
f_7	3000	FEP	7.6×10^{-3}	2.6×10^{-3}	RMEA
		IFEP	N.A.	N.A.	
		RMEA	1.24×10^{-51}	1.079×10^{-51}	

Table 4.8
Comparison among FEP, IFEP and RMEA on high dimensional multimodal functions, f_8–f_{13}. "Mean Best" is the mean of the best function values averaged over 50 Runs. "N.A." means result is not available in [22].

Function	Generations	Algorithm	Mean Best	Standard Deviation	Best Performance by
f_8	9000	FEP	-12554.5	52.6	FEP
		IFEP	N.A.	N.A.	
		RMEA	-11332.6	61.2	
f_9	5000	FEP	4.6×10^{-2}	1.2×10^{-2}	RMEA
		IFEP	N.A.	N.A.	
		RMEA	5.34×10^{-51}	1.56×10^{-51}	
f_{10}	1500	FEP	1.8×10^{-2}	2.1×10^{-3}	RMEA
		IFEP	4.83×10^{-3}	N.A.	
		RMEA	1.18×10^{-7}	6.27×10^{-8}	
f_{11}	2000	FEP	1.6×10^{-2}	2.2×10^{-2}	RMEA
		IFEP	4.54×10^{-2}	N.A.	
		RMEA	1.01×10^{-20}	4.64×10^{-21}	
f_{12}	1500	FEP	9.2×10^{-6}	3.6×10^{-6}	FEP
		IFEP	N.A.	N.A.	
		RMEA	2.77×10^{-4}	1.09×10^{-4}	
f_{13}	1500	FEP	1.6×10^{-4}	7.3×10^{-5}	FEP
		IFEP	N.A.	N.A.	
		RMEA	5.99×10^{-3}	4.13×10^{-3}	

Table 4.9
Comparison among FEP, IFEP and RMEA on low dimensional multimodal functions, f_{14}–f_{23}. "Mean Best" is the mean of the best function values averaged over 50 Runs. "N.A." means result is not available in [22].

Algorithm	Function Value (Mean Best)									
	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}	f_{21}	f_{22}	f_{23}
Generations	100	4000	100	100	100	100	200	100	100	100
FEP	1.22	5.0×10^{-4}	-1.03	0.398	3.02	-3.86	-3.27	-5.52	-5.52	-6.57
IFEP	N.A	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	-6.46	-7.10	-7.80
RMEA	1.19	3.25×10^{-4}	-1.03	0.398	3.00	-3.86	-3.38	-7.58	-9.41	-9.96
Best Performance By	RMEA	RMEA	Similar	Similar	RMEA	Similar	RMEA	RMEA	RMEA	RMEA

4.6 Experiments with Components of RMEA

This section presents a number of experiments with results in order to closely observe and achieve insights on different aspects of RMEA. The sensitivity of the parameters with solution quality, significance of the different operators, and roles of the different stages in RMEA are examined in the following sections.

4.6.1 Parameters

k_1, k_2 and k_3 are used in RMEA to define the length of its conventional, explorative and exploitative stages. The sensitivity of these parameters with the optimization performance is examined in this section. Table 4.10 shows the results of RMEA, which are averaged over 10 independent runs, with different values of k_1, k_2 and k_3 for 9 functions randomly selected from the functions f_1–f_{13}. Functions f_{14}–f_{23} are ignored because they are quite simple in comparison to the others due to their low dimensionality. The results show that RMEA is not significantly sensitive to k_1, k_2 and k_3. When the lengths of the stages are too short (e.g., 10) or too long (e.g., 100), the results are slightly worse. Somewhat better results are obtained with moderate stage lengths (e.g., 30 or 60). It is observed from Table 4.10 that the best results are always found with moderate stage lengths (i.e., 30 or 60). However, results are quite satisfactory for every choice of parameter values shown in Table 4.10. It, therefore, indicates that RMEA is not critically sensitive to k_1, k_2 and k_3, and any moderate choice of values can produce nearly similar results.

Table 4.10
Effect of varying lengths of the conventional, exploration and exploitation stages. k_1, k_2, k_3 are the lengths of the stages respectively.

Function	Short Stages $k_1= k_2= k_3= 10$	Moderate[1] Stages $k_1= k_2= k_3= 30$	Moderate[2] Stages $k_1= k_2= k_3= 60$	Prolonged Stages $k_1= k_2= k_3=100$	Best
f_1	3.53×10^{-11}	1.06×10^{-12}	4.72×10^{-11}	1.23×10^{-7}	Moderate[1]
f_3	8.52×10^{-38}	1.06×10^{-45}	2.49×10^{-45}	7.22×10^{-40}	Moderate[1]
f_4	3.86×10^{-9}	6.81×10^{-10}	1.27×10^{-10}	1.95×10^{-10}	Moderate[2]
f_7	8.30×10^{-45}	1.42×10^{-51}	7.31×10^{-50}	1.03×10^{-45}	Moderate[1]
f_8	-9843.93	-11332.6	-10614.86	-8328.66	Moderate[1]
f_{10}	1.17×10^{-6}	1.18×10^{-7}	5.87×10^{-5}	3.64×10^{-5}	Moderate[1]
f_{11}	6.63×10^{-19}	1.01×10^{-20}	6.89×10^{-18}	2.75×10^{-15}	Moderate[1]
f_{12}	3.12×10^{-4}	2.77×10^{-4}	3.28×10^{-4}	6.88×10^{-3}	Moderate[1]
f_{13}	2.83×10^{-3}	5.99×10^{-3}	1.42×10^{-3}	5.71×10^{-3}	Moderate[2]

In fact, what is crucial in RMEA is not the length of the stages, but their recurring nature. This is established by the results summarized in Table 4.11, where conventional, exploration and exploitation stages are run sequentially, not in a recurring manner. These schemes are named as sequential multi-stage evolutionary algorithm (*SMEA*). Depending on the sequencing of the stages, three different variants of *SMEA* are specified here. The first variant, $SMEA_{C,\,ER,\,ET}$ starts with conventional stage and executes it for the first one-third of the total generations needed for minimizing a function. Then explorative stage continues for the next one-third and finally exploitative stage executes for the last one-third of the total generations. For example, to minimize f_1 for 1500 generations, $SMEA_{C,\,ER,\,ET}$ executes the conventional stage for 500 generations, followed by explorative stage for 500 generations, followed by exploitative stage for the last 500 generations. Another variant, $SMEA_{ER,\,C,\,ET}$ executes the stages in this order: exploration stage, followed by conventional stage, followed by exploitative stage. The third variant, $SMEA_{ER,\,ET,\,C}$ executes the explorative stage first, then exploitative stage and finally concludes with the conventional stage. These three non- recurring variants of *SMEA* are applied on seven functions, chosen randomly from f_1-f_{23}. The results of the experiments are averaged over 50 independent runs and compared with RMEA in Table 4.11.

Table 4.11
Comparison between SMEA and RMEA on a number of unimodal and multimodal functions. RMEA outperforms the non-recurring SMEA variants in every case.

Function	$SMEA_{C,\,ER,\,ET}$	$SMEA_{ER,\,C,\,ET}$	$SMEA_{ER,\,ET,\,C}$	RMEA	Best
f_1	8.77×10^{-8}	2.60×10^{-4}	1.48×10^{-5}	1.06×10^{-12}	RMEA
f_4	4.87×10^{-9}	1.14×10^{-6}	1.76×10^{-6}	6.81×10^{-10}	RMEA
f_7	2.73×10^{-29}	8.08×10^{-33}	2.97×10^{-33}	1.24×10^{-51}	RMEA
f_{10}	1.31×10^{-4}	3.02×10^{-3}	5.30×10^{-4}	1.18×10^{-7}	RMEA
f_{11}	8.39×10^{-16}	1.12×10^{-12}	8.98×10^{-12}	1.01×10^{-20}	RMEA
f_{13}	6.26×10^{-3}	1.80×10^{-1}	9.86×10^{-2}	5.99×10^{-3}	RMEA
f_{23}	-9.90	-8.37	-9.21	-9.96	RMEA

To make fair comparisons, all three variants of *SMEA* employ the same number of conventional, exploration and exploitation stages in total as RMEA, but they do not mix and interleave different stage. The results in Table 4.11 show that, RMEA performs significantly better than the variants of non-recurring *SMEA*. This proves the necessity of interleaving and mixing different stages regularly instead of trying to make a perfect switch from one stage to another. Generally, an evolutionary approach needs to escape from several local minima when exploring through the fitness landscape for optimization. Such a situation makes a recurring and randomized algorithm more appropriate than a sequential and static approach. The results of RMEA, when compared with that of *SMEA*, make this fact apparent.

4.6.2 Operators

RMEA uses neighbors and strangers for recombination and crossover operators. It therefore keeps track of the neighbors and strangers throughout the evolution for each individual of the population. Besides, the standard deviation of mutation is also carefully decided either from the deviation of other individuals along an attribute or from the magnitude of the attribute being mutated. To examine whether all these extra calculations have contributed to the exploitations and explorations, we now introduce another variant of RMEA, named as naïve-RMEA, which neither use neighbors/strangers nor controls the mutation standard deviation. To conduct recombination or crossover, naïve-RMEA selects participants randomly across the entire population. Besides, it always employs gaussian $N(0,1)$ distribution for mutation. Naïve-RMEA still consists of a conventional stage similar to CEP, and both exploitation and exploration stages. The only difference between the exploitation and exploration stages is that, the exploitation stage accepts better individuals only, while the exploration stage, by some hill climbing moves, may accept immediate worse individuals, which have potentials to prove better in the future. Naïve-RMEA is compared with RMEA on the 23 functions. Results from the following tables 4.12, 4.13 indicate that, naïve-RMEA has performed much worse than RMEA on all the functions, except slightly better performance only for f_{12} and f_{13}. This letdown of naïve-RMEA is by order of magnitude for $f_1, f_2, f_3, f_4, f_9, f_{10}$ and f_{11}. All these results exemplify the contributions of neighbors, strangers and controlling of the standard deviation of mutation behind the effective exploitations, explorations and thus, superior performance by RMEA.

Table 4.12
Comparison between RMEA and naïve-RMEA on unimodal functions f_1–f_7. Results are averaged over 50 runs.

Function	Generations	Algorithm	Mean Best	Better Performance
f_1	1500	NAÏVE-RMEA	6.95×10^{-5}	RMEA
		RMEA	1.06×10^{-12}	
f_2	2000	NAÏVE-RMEA	3.26×10^{-4}	RMEA
		RMEA	2.66×10^{-11}	
f_3	5000	NAÏVE-RMEA	1.64×10^{-9}	RMEA
		RMEA	1.05×10^{-45}	
f_4	5000	NAÏVE-RMEA	9.66×10^{-2}	RMEA
		RMEA	6.81×10^{-10}	
f_5	20000	NAÏVE-RMEA	78.19	RMEA
		RMEA	20.89	
f_6	1500	NAÏVE-RMEA	0 (450 generations)	RMEA
		RMEA	0 (350 generations)	
f_7	3000	NAÏVE-RMEA	7.51×10^{-51}	RMEA
		RMEA	1.24×10^{-51}	

Table 4.13
Comparison between RMEA and naïve-RMEA on a number of multimodal functions. Results are averaged over 50 runs.

Function	Generations	Algorithm	Mean Best	Better Performance by
f_8	9000	NAÏVE-RMEA	-10259	RMEA
		RMEA	-11322	
f_9	5000	NAÏVE-RMEA	9.74×10^{-13}	RMEA
		RMEA	5.34×10^{-51}	
f_{10}	1500	NAÏVE-RMEA	4.62×10^{-3}	RMEA
		RMEA	1.18×10^{-7}	
f_{11}	2000	NAÏVE-RMEA	1.19×10^{-8}	RMEA
		RMEA	1.01×10^{-20}	
f_{12}	1500	NAÏVE-RMEA	1.62×10^{-5}	NAÏVE-RMEA
		RMEA	2.77×10^{-4}	
f_{13}	1500	NAÏVE-RMEA	2.63×10^{-3}	NAÏVE-RMEA
		RMEA	5.99×10^{-3}	
f_{14}	100	NAÏVE-RMEA	1.992	RMEA
		RMEA	1.19	
f_{21}	100	NAÏVE-RMEA	-7.27	RMEA
		RMEA	-7.58	
f_{22}	100	NAÏVE-RMEA	-8.52	RMEA
		RMEA	-9.41	
f_{23}	100	NAÏVE-RMEA	-8.03	RMEA
		RMEA	-9.96	

4.6.3 Stages

Since conventional, exploration and exploitation stages employ different operators, they affect the population fitness in different ways. An interesting question is: what are the roles of these stages in evolution, especially in terms of fitness improvement and diversity preservation? This issue is introduced in this section, and further elaborated in the next section. To examine the effect of the different stages on fitness, RMEA is parameterized in three different ways with different sets of values for k_1, k_2 and k_3. In the first setting, which has been named as RMEA$_{conv}$, the conventional stage has been set much lengthier than the other two stages by using $k_1 = 50$ with $k_2 = k_3 = 10$. In the second and third settings, which have been named as RMEA$_{explore}$ and RMEA$_{exploit}$, more exploration and exploitation are allowed by setting $k_2 = 50$ (with $k_1 = k_3 = 10$) and $k_3 = 50$ (with $k_1 = k_2 = 10$) respectively.

Table 4.14 shows that RMEA$_{explore}$ performs better than RMEA$_{conv}$ and RMEA$_{exploit}$ with 6 out of the 9 tested functions. Since all other parameters are kept identical in all the experiments, the better performance of RMEA$_{explore}$ indicates that the effectiveness of the exploration stage for optimization. This is because the exploration stage tries to balance between the population diversity and solution quality first by applying explorative operators to reach new regions of the search space and then by applying hill-climbing steps to discover better, yet diverse individuals. This means exploration is not a basic operation, rather is a compound operation that uses several hill-climbing steps after the application of an explorative operator. However, exploration may not be suitable rather harmful for simple functions where the number of local minima is small. This is possibly the reason for the worse performance of RMEA$_{explore}$ with respect to RMEA$_{exploit}$ for f_{21}-f_{23}. RMEA$_{exploit}$ performs best for these functions. The worse result of RMEA$_{explore}$ on some functions suggests that RMEA should employ moderate and equal/nearly equal proportion of conventional, exploration and exploitation stages to achieve robustness. Besides, this is also indicative that, a single setting of the parameters, with some extreme choices for parameter values, usually never performs better for all the problems. Instead, a moderate choice of parameter values, (e.g. $k_1 = k_2 = k_3 = 30$) usually proves to be more robust. However, for all the experiments presented in this section, RMEA performs sufficiently well, which proves again the extra-ordinary effectiveness of RMEA, and the robustness of its parameters.

Table 4.14
Effects of varying the proportion of conventional, exploration and exploitation operations. Exploration stages exhibit exceptionally better performance for the high dimensional functions, f_1-f_{13}, while exploitation stages exhibit superior performance for the low dimensional functions, f_{14}-f_{23}.

Function	RMEA$_{exploit}$ with k_1, k_2, k_3 = 10,10,50	RMEA$_{conv}$ with k_1, k_2, k_3 = 50,10,10	RMEA$_{explore}$ with k_1, k_2, k_3 = 10,50,10	Best
f_1	2.49×10^{-7}	3.22×10^{-4}	8.11×10^{-26}	RMEA$_{explore}$
f_3	2.19×10^{-2}	1.46×10^{-3}	2.35×10^{-28}	RMEA$_{explore}$
f_7	3.95×10^{-24}	6.95×10^{-20}	5.58×10^{-73}	RMEA$_{explore}$
f_{10}	3.63×10^{-4}	5.56×10^{-3}	8.23×10^{-14}	RMEA$_{explore}$
f_{11}	3.13×10^{-12}	5.02×10^{-9}	1.85×10^{-36}	RMEA$_{explore}$
f_{13}	5.93×10^{-3}	1.86×10^{-2}	4.83×10^{-3}	RMEA$_{explore}$
f_{21}	-9.78	-7.21	-8.81	RMEA$_{exploit}$
f_{22}	-10.25	-8.45	-8.77	RMEA$_{exploit}$
f_{23}	-10.41	-9.91	-9.31	RMEA$_{exploit}$

4.6.4 Evolution of Diversity and Fitness

This section introduces a new measure of population diversity, named as Neighborhood Diversity (*ND*), and examines how *ND* is evolved throughout the evolution. During each generation, the neighborhood size, N is set uniformly at random from 1% to 5% of the population size. As mentioned previously, RMEA maintains a list of N nearest neighbors and N farthest strangers across the population for each individual. Suppose,

M = the size of the population, which is static throughout the evolution

$N \sim [1\%...5\%]$ = size of the neighborhood at the current generation

S_i = the set of neighbors (nearest individuals) for individual i

d_{ij} = distance (euclidean distance) between individuals i and j

The neighborhood diversity, *ND* of the population at the current generation is defined as,

$$ND = \frac{1}{M}\frac{1}{N}\sum_{i=1}^{M}\sum_{j\in S_i} d_{ij}$$

Since the exploitative or explorative operations involve individuals from the same or different neighborhoods, the proposed measure of population diversity, *ND* is directly affected by the exploitative and explorative operators. To examine how *ND* is evolved, RMEA is executed on a random selection of 9 functions. While RMEA evolves the population, the population *ND* is calculated in each generation and plotted against the generations for the different functions, and presented in Figure 4.3(a), (b). The graphs show that *ND* undergoes periodic rise and fall under the impact of the recurring exploration and exploitation stages. The same graphs for CEP show only gradual decrease of *ND* with the progress of the evolutionary process. If the population diversity falls by a significant amount, it may lead the search process to be trapped into poor local optima. Therefore, loss of diversity has been mentioned as the principal cause for premature convergence [10]. However, RMEA successfully prevents such unrestrained fall of diversity with periodic reduction and restoration of diversity, as illustrated in the graphs.

Consider the graph of function f_4 in details with logarithmic y-axis in Figure 4.3(b). While CEP exhibits a gradual decrease of diversity to as low as 0.001, RMEA nicely allows the diversity to rise and fall in the range of 0.1 to 10, which is high in comparison to CEP. At the beginning of the evolutionary process, CEP starts to drop population diversity slower than RMEA, and it maintains more diversity than RMEA up to about 800 generations. Afterwards, the diversity by CEP falls below that of RMEA and it continues to drop more and more by CEP as its evolutionary process progresses. On the other hand, RMEA, after an initial quick lose of diversity (due to the exploitative operations), periodically restores nearly the same level of sufficient diversity by its recurring explorative operations. Each time the diversity falls during the exploitation stage, it is nicely raised in the subsequent exploration stage, without any difficulty or irregularity, till the end of the evolution. RMEA displays a similar periodic behavior with diversity for the functions f_2, f_6, f_7, f_{10}, f_{11}, f_{12} and f_{13}, while CEP drops the diversity significantly and it never shows any attempt to recover the diversity.

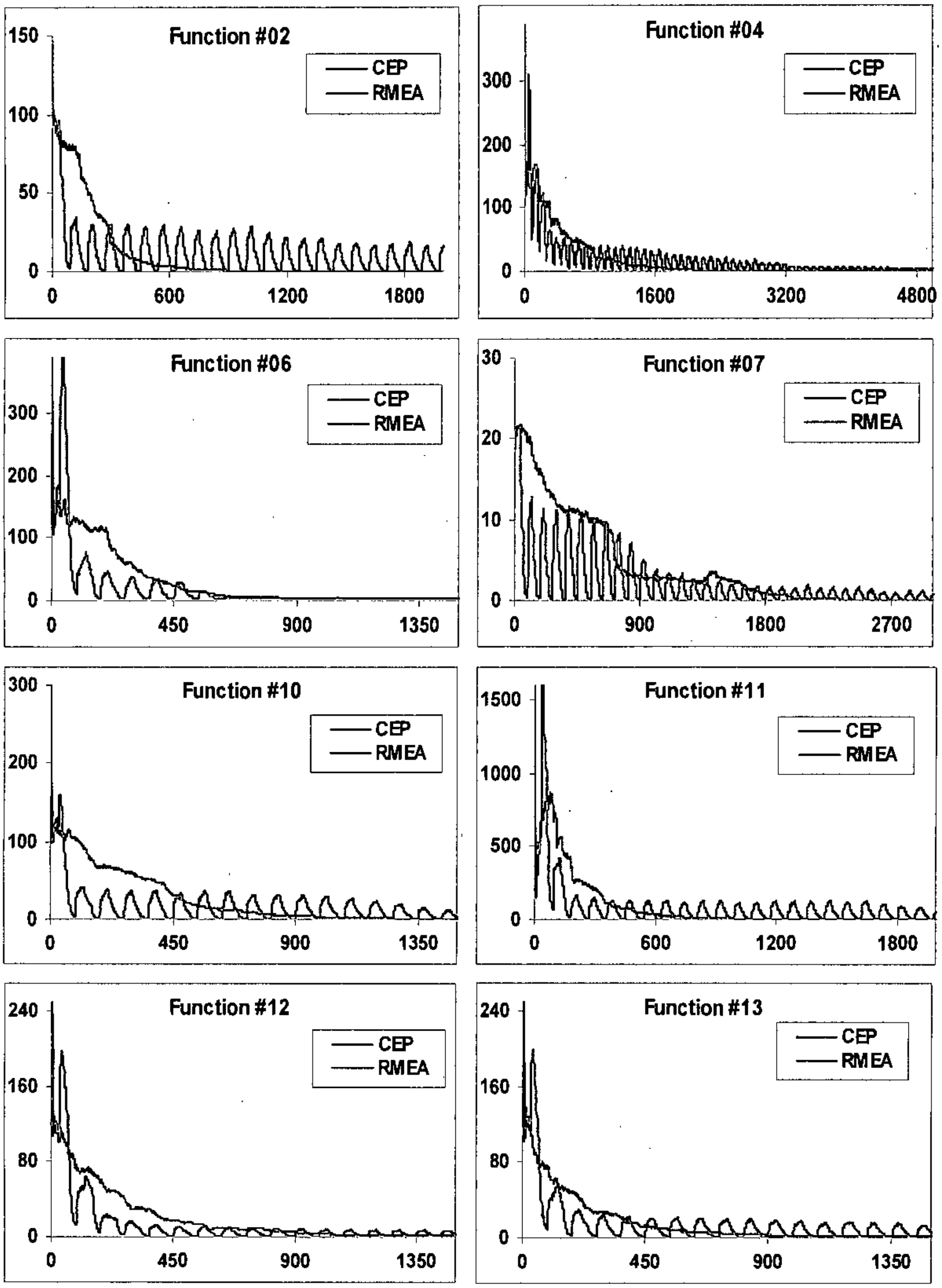

Figure 4.3(a): Evolution of neighborhood diversity with generations, for functions f_2, f_4, f_6, f_7, f_{10}, f_{11}, f_{12} and f_{13}. Graphs exhibit that diversity falls drastically by CEP during the late generations, while RMEA shows periodic rise and fall of diversity. Similar results are observed for rest of the functions.

103111

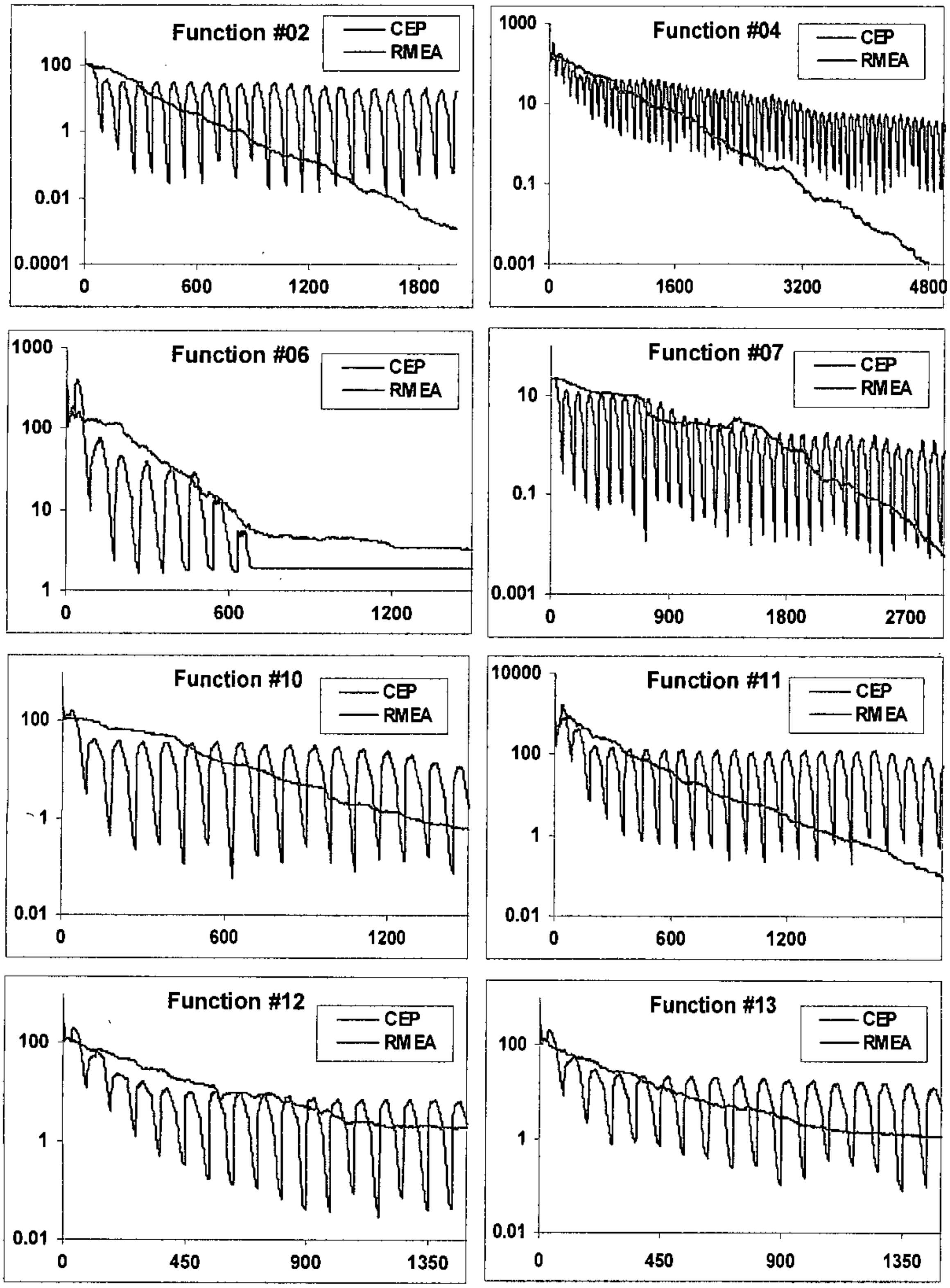

Figure 4.3(b): The Neighborhood diversity vs. generation graphs, reproduced with logarithmic scale along y-axis. The small values of neighborhood diversity during the late generations are magnified with the logarithmic scale. Similar results are observed for rest of the functions.

In contrast, RMEA periodically restores the lost diversity to a sufficient amount (e.g., 10 in f_4 or 100 in f_{11}). Similar results are found for all other functions, which reveal the sufficient diversity-preserving potential of RMEA.

Not only does RMEA show periodic characteristics with diversity, but also a regular and periodic optimization pattern is observed in the progression of the fitness values. To illustrate the fact, some graphs from the figures 4.1, 4.2 are redrawn in Figure 4.4, but for a smaller number of generations. The graphs display the fitness vs. generation curves for the first 350 generations of some functions from f_1–f_{13}. All of them demonstrate quite similar optimization pattern. Different color is used to represent the optimizations carried by the different stages. It is noticeable that both exploration and exploitation stages participate significantly in the optimization process, while the conventional stage seems to have relatively weaker fitness gain. The exploration stage initially shows higher fitness gain than the exploitation stage, which is apparent for the functions f_1, f_2, f_7, f_{11} and f_{12}. This is natural, because explorations are easier than exploitations at the beginning of the search process. Afterwards, both explorations and exploitations perform nearly equal role, except for f_7, for which exploitations prove to be more effective afterwards. It is noticeable that the recurring nature of RMEA is nicely reflected in the progression of fitness curve for all the functions, which indicates that the periodic nature of the algorithm properly matches with the characteristics of the fitness landscape, consisting of re-occurring peaks and valleys. Therefore, these functions pose no difficulty for the algorithm. It is quite appealing when the properties of an evolutionary system are directly reflected in the progression of the evolution.

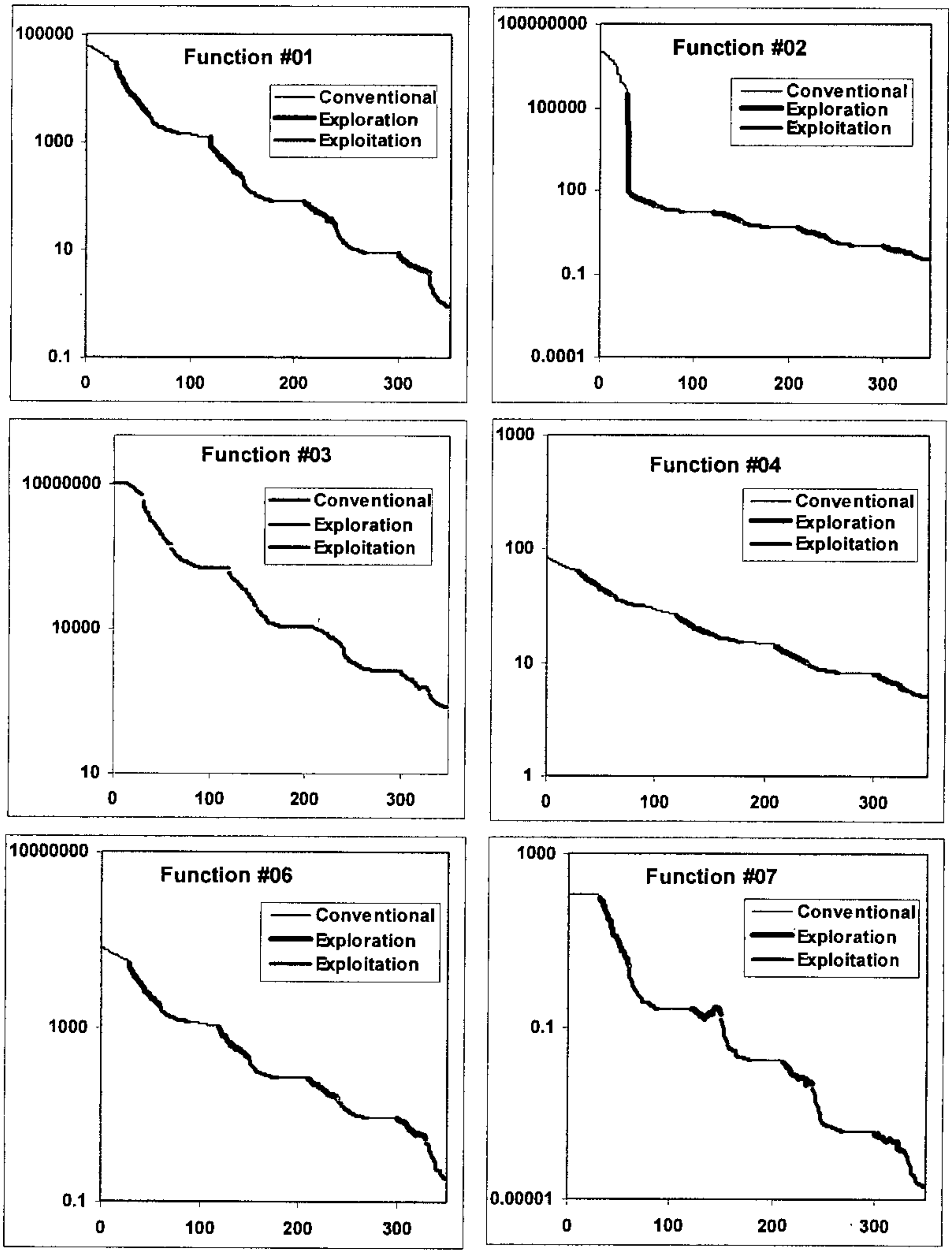

Figure 4.4(a): Optimization conducted at different rates by the three different stages for unimodal functions. The horizontal and vertical axes show no. of generations and the function values respectively. Graphs exhibit that, exploration stage contributes maximum in optimization, while the conventional stage contributes the least.

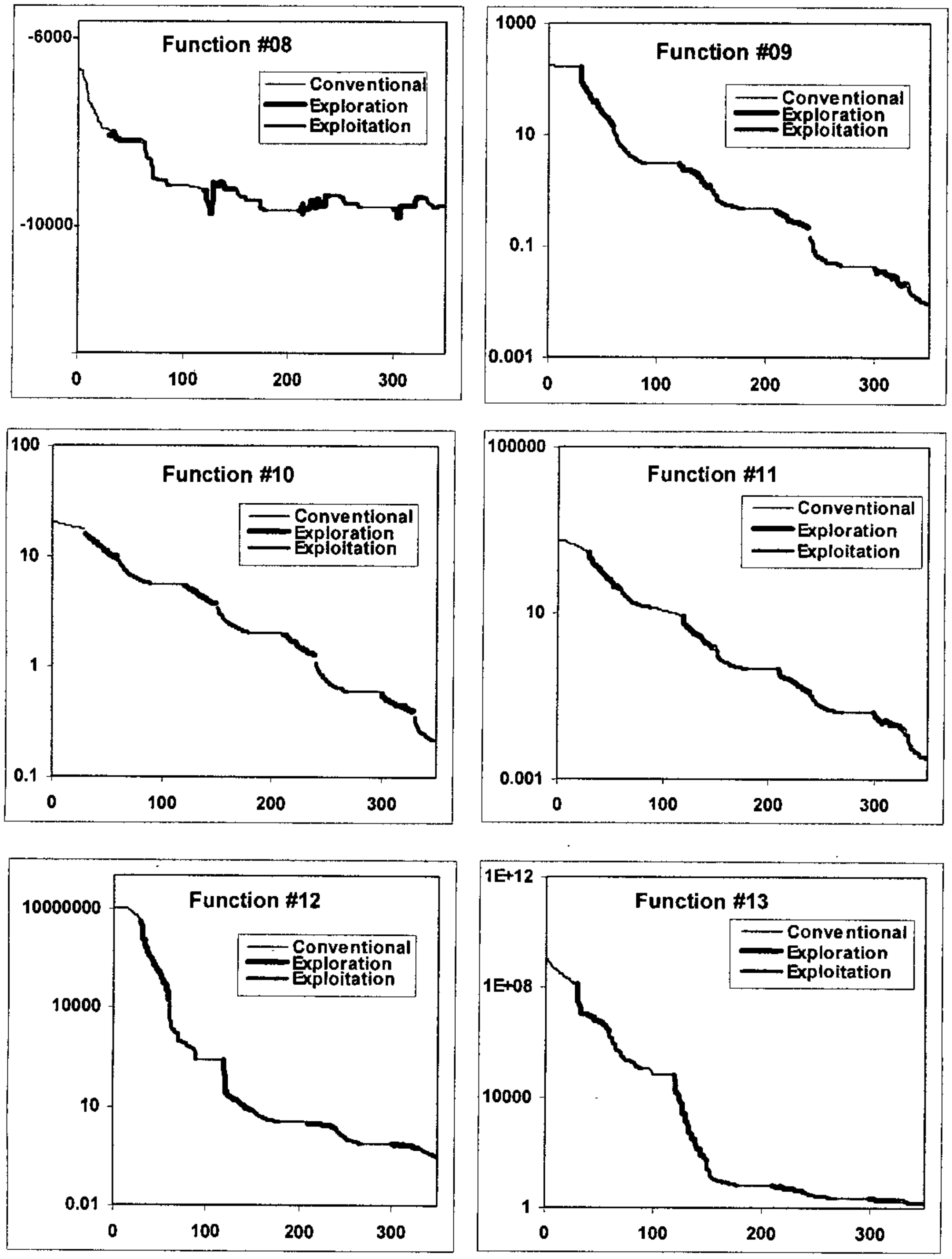

Figure 4.4(b): Optimization conducted at different rates by the three different stages for. multimodal functions. The horizontal and vertical axes show no. of generations and the function values respectively. Graphs exhibit that, exploration stage contributes maximum in optimization, while the conventional stage contributes the least.

4.7 Adaptation of Parameters

Three user-specific parameters k_1, k_2 and k_3, which define the lengths of the conventional, exploration and exploitation stages, control the principal characteristics of RMEA. These parameters, however, may cause some difficulty for an inexperienced user to specify. In addition, fixed values of these parameters throughout the entire evolution may not be appropriate for some problems. In this section, an adaptive scheme is proposed which automatically adjusts these parameters based on the effectiveness of the stages. This new scheme is entitled as Ada-RMEA, reflecting its adaptive ability to adjust the parameters.

The central idea behind the proposed adaptation scheme is using the relative gain (in fitness) per generation (*RGPG*) of a particular stage which is an indication of its effectiveness in the evolution. Suppose, a particular stage starts execution just after the *i*-th generation and ends at $(i+N)$-th generation. During this span of N generations, the function is minimized from $f^{(i)}$ to $f^{(i+N)}$. Thus, the optimization achieved by the stage is $f^{(i)} - f^{(i+N)}$, which may be either positive or negative to indicate improvement or deterioration, respectively (since minimization is being considered). If the amount of improvement is expressed as a ratio to the original starting value $f^{(i)}$, we get the relative gain as $[f^{(i)} - f^{(i+N)}] / f^{(i)}$. When the relative gain is averaged over the N generations, we get the *RGPG* as:

$$(1/N) * [f^{(i)} - f^{(i+N)}] / f^{(i)}$$

The *RGPG* values are calculated for the conventional, exploration and exploitation stages, which are expressed as $RGPG_{\text{conv}}$, $RGPG_{\text{expr}}$ and $RGPG_{\text{expt}}$. Their arithmetic mean, $RGPG_{\text{mean}}$ is also calculated. The basic idea for adaptation is, if a stage exhibits higher *RGPG* than the average $RGPG_{\text{mean}}$, its length is increased. Otherwise, its length is decreased. The stage lengths: k_1, k_2 and k_3 may be adapted using the following formulae:

$$k_i^{(t+1)} = k_i^{(t)} + \Delta k_i^{(t)}; \quad i = 1,2,3$$

$$\text{where,}\ \Delta k_1^{(t)} = RGPG_{conv} - RGPG_{mean},$$

$$\Delta k_2^{(t)} = RGPG_{\text{expr}} - RGPG_{mean}$$

$$\Delta k_3^{(t)} = RGPG_{\text{expt}} - RGPG_{mean}$$

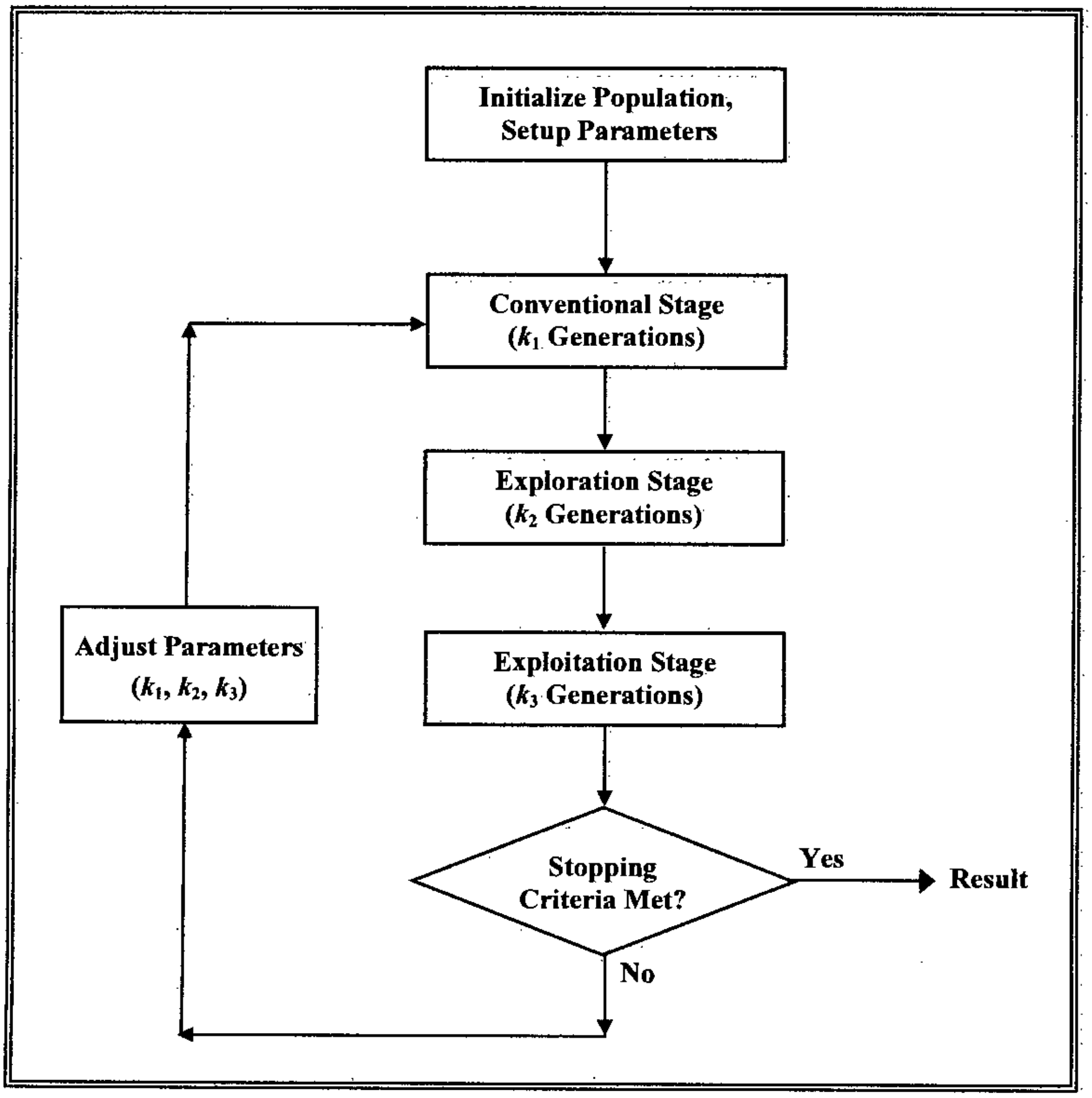
Initialize Population, Setup Parameters
Conventional Stage (k_1 Generations)
Exploration Stage (k_2 Generations)
Exploitation Stage (k_3 Generations)
Adjust Parameters (k_1, k_2, k_3)
Stopping Criteria Met?
Yes
Result
No

Figure 4.5: Adaptive recurring multistage evolutionary algorithm (Ada-RMEA)

These formulae ensure that if a particular stage shows higher *RGPG*, its length is increased. However, to ensure continuity and stability of the evolutionary process, two further considerations are employed, too. Firstly, the values of k_1, k_2, and k_3 are always maintained within the range [10...50], since too short or too long execution of a particular stage may not be beneficial for the optimization process (Table 4.10). Secondly, k_1, k_2, k_3 are not allowed to be changed by more than 10% of their previous values at a single step. This regulation ensures smoother changes of the behavior of the evolutionary process. The flowchart of Ada-RMEA is presented in Figure 4.5, with its pseudo-code in algorithm 4.1.

```
1. Initialize Parameters. Generate Initial Population.
2. for $k_1$ generations              [Conventional Stage]
      Mutate each individual
      Execute tournament among parents and offspring to get the next generation
3. for $k_2$ generations              [Exploration Stage]
     for each individual, I
         Update the set of neighbors of I
         Randomly choose one of the six explorative operators
         Apply the operator on I to obtain a new individual I′
         Perform some uphill steps from I′ to decide whether to accept or reject it.
4. for $k_3$ generations              [Exploitation Stage]
     for each individual, I
        Update the set of neighbors of I
        Randomly choose one of the six exploitative operators
        Apply the operator on I to obtain a new individual I′
        Accept I′ if it has better fitness than I. Otherwise Reject.
5. If the best solution found is acceptable or the maximum number of generations has
   been elapsed, conclude RMEA and output the best individual of the last generation.
   Otherwise continue.
6. Adjust the values of $k_1$, $k_2$ and $k_3$ based on the relative gain per generation (RGPG)
   of the conventional, exploration and exploitation stages.
7. Return to step 3 and start another cycle of conventional, exploration and
   exploitation stages.
```

Algorithm 4.1: Adaptive recurring multistage evolutionary algorithm

4.7.1 Experimental Results

This section presents the results of applying Ada-RMEA on the suite of 23 functions. Initial values of k_1, k_2, and k_3 are set to 30. To make fair comparison with RMEA, all other parameters are kept identical. Table 4.15 compares the results of Ada-RMEA with RMEA on unimodal functions, f_1- f_7. Results show that Ada-RMEA always performs better than RMEA for these functions, with the only minor exception of f_5. This proves that, emphasizing the stages with better fitness gain promotes the performance of RMEA further. Table 4.16 compares Ada-RMEA on high dimensional multimodal functions, f_8-f_{13}. Again, it is noticeable that Ada-RMEA performs better than RMEA. Table 4.17 shows the comparison results on the low dimensional multimodal functions, f_{14}- f_{23}. These results are somewhat similar to RMEA and sometimes better for Ada-RMEA, especially for the relatively complex functions in this family: f_{21}, f_{22}, and f_{23}.

Table 4.15
Comparison between RMEA and Ada-RMEA on unimodal functions, f_1–f_7.
Results represent the mean of the best function values, averaged over 50 runs.

Function	Generations	Algorithm		BETTER
		RMEA	ADA-RMEA	
f_1	1500	1.06×10^{-12}	1.20×10^{-15}	ADA-RMEA
f_2	2000	2.66×10^{-11}	1.21×10^{-12}	ADA-RMEA
f_3	5000	1.06×10^{-45}	4.56×10^{-68}	ADA-RMEA
f_4	5000	6.81×10^{-10}	1.97×10^{-18}	ADA-RMEA
f_5	20000	20.89	22.88	RMEA
f_6	1500	0	0	SIMILAR
f_7	3000	1.24×10^{-51}	1.18×10^{-64}	ADA-RMEA

Table 4.16
Comparison between RMEA and Ada-RMEA on high dimensional multimodal functions, f_8–f_{13}. Results represent the mean of the best function values, averaged over 50 runs.

Function	Generations	Algorithm		BETTER
		RMEA	ADA-RMEA	
f_8	9000	-11332.6	-11503.5	ADA-RMEA
f_9	5000	5.34×10^{-51}	8.66×10^{-58}	ADA-RMEA
f_{10}	1500	1.18×10^{-7}	3.71×10^{-8}	ADA-RMEA
f_{11}	2000	1.01×10^{-20}	1.69×10^{-21}	ADA-RMEA
f_{12}	1500	2.77×10^{-4}	1.95×10^{-4}	ADA-RMEA
f_{13}	1500	5.99×10^{-3}	4.33×10^{-3}	ADA-RMEA

Table 4.17
Comparison between RMEA and Ada-RMEA on low dimensional multimodal functions, f_{14}–f_{23}. Results represent the mean of the best function values, averaged over 50 runs.

Function	Generations	Algorithm		BETTER
		RMEA	ADA-RMEA	
f_{14}	100	1.19	1.08	ADA-RMEA
f_{15}	4000	$3.25\text{x}10^{-4}$	3.25×10^{-4}	SIMILAR
f_{16}	100	-1.03	-1.03	SIMILAR
f_{17}	100	0.398	0.398	SIMILAR
f_{18}	100	3.00	3.00	SIMILAR
f_{19}	100	-3.86	-3.86	SIMILAR
f_{20}	200	-3.38	-3.38	SIMILAR
f_{21}	100	-7.58	-10.22	ADA-RMEA
f_{22}	100	-9.41	-9.58	ADA-RMEA
f_{23}	100	-9.96	-10.21	ADA-RMEA

4.7.2 Effects of Adaptation on Stage Lengths

This section describes the effect of adaptation on the lengths of the conventional, exploration and exploitation stages. Initially, an equal length is used for each stage. Figure 4.5 shows the lengths of the different stages being changed with generations by the adaptation scheme. The adaptive process lengthens more effective stages, and shortens the weaker stages. As illustrated earlier, the exploration stage exhibits the highest fitness gain, while the conventional stage shows the least fitness gain for most of the functions. So k_1, the length of the conventional stage tends to decrease, while k_2, being the exploration stage length, starts to increase from the very beginning. For example, for function f_2, the evolution starts with k_1=k_2=k_3=30, while it ends with k_1=19, k_2=40 and k_3=29. This shows that the length of the exploration stage has eventually become double of the conventional stage. For function f_3, evolution ends with k_1= 10, k_2=44 and k_3=18, which means the exploration stage completely dominates the other two stages in the optimization process. The exploration stage also outperforms the others for function f_4, as the evolution ends with k_1= 10, k_2=50 and k_3=30.

However, all the three stages contribute significantly for some functions. For example, for the function f_7, the evolution ends with k_1= 26, k_2=32 and k_3=32. Similar scenario is found for f_6 and f_9, where the exploration stage takes away a few generations from the conventional stage. However, a completely opposite scenario is found in case of f_8 for which the evolution ends with k_1= 49, k_2=10 and k_3=50. So, exploration stage suffers miserably in this function. As we have observed, the exploration stage operates against the ongoing optimization performed by the conventional and exploitation stages for this function. Each time the conventional and exploitation stages minimize f_8 by some amount, the exploration stage starts execution and deteriorates the solutions to some extent. So, the adaptive RMEA truncates its stage length to the minimum (i.e., 10). However, in almost all the functions, the exploitation stage contributes nearly, or slightly less than one-third of the total optimizations. For example, for f_{12}, the exploration and exploitation stages end with identical strength (k_1=16, k_2=33, k_3=33). Also for f_{13}, exploitation stage contributes more than the others (k_1=23, k_2=24, k_3=34).

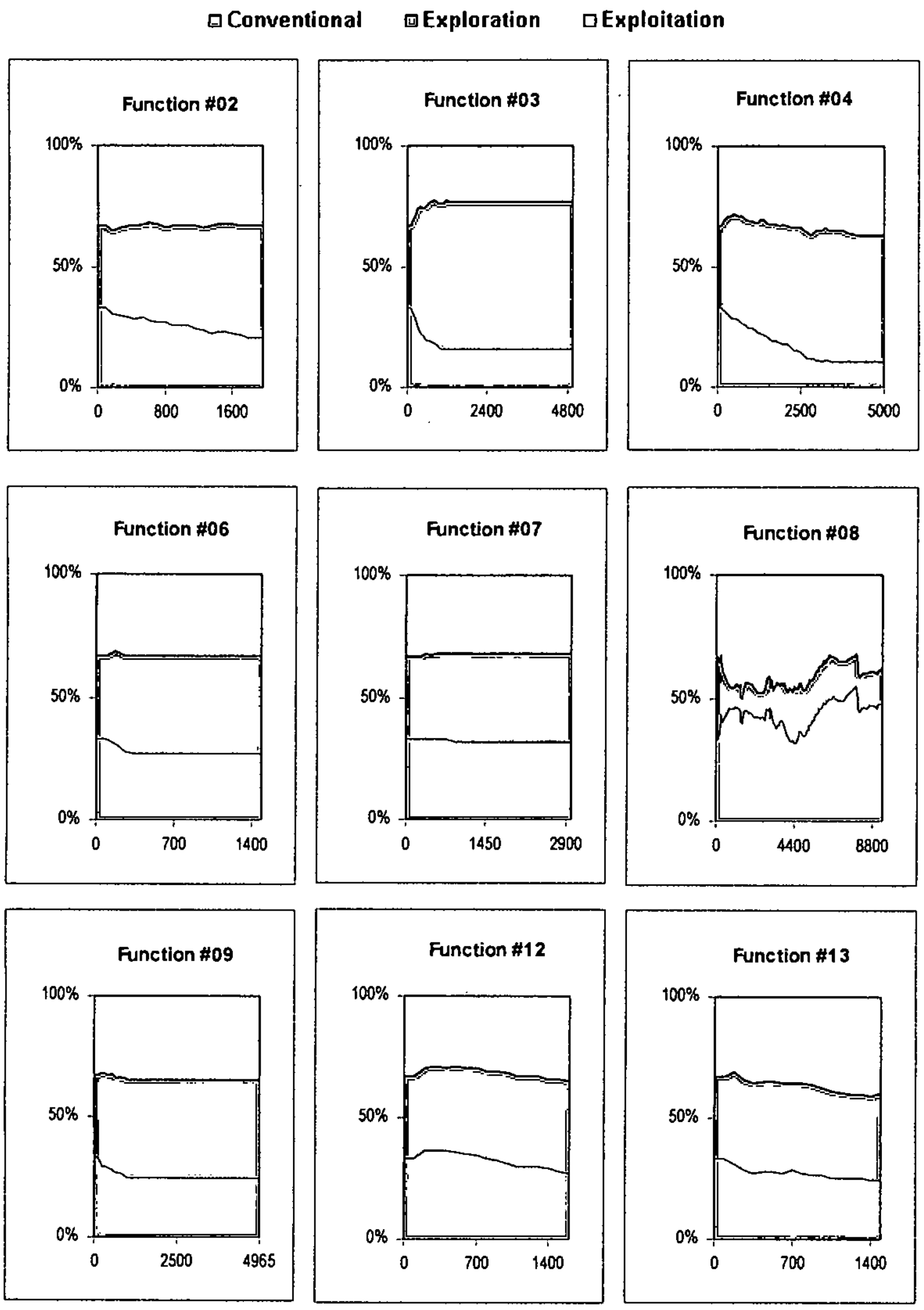

Figure 4.6: Effects of adaptation on lengths of the conventional, exploration and exploitation stages. Vertical axis shows the proportion of conventional, exploration and exploitation stages. Horizontal axis shows no. of generations.

In summary, all the stages contribute somewhat more or less in the minimization process. This is quite natural, because searching is a complex task and each stage has its particular strengths (and limitations, too) to overcome the barriers and continue the search in order to reach the global optimum. The results presented in this section clearly establish the significance and necessity of each of the conventional, exploration and exploitation stages during the search process.

Chapter 5
Conclusions and Future Works

5.1 Conclusions

RMEA introduces a novel framework for evolutionary algorithms to unravel the conflicting goals of exploitation and exploration during evolution. RMEA has demonstrated very promising results, outperforming several other existing algorithms on a number of benchmark problems. However, worse performance with only a few functions is nothing unusual, because no evolutionary approach can perform better for every problem [21]. Such an inspiring performance from RMEA is well-desired, because RMEA employs quite a different mechanism than the others. However, RMEA needs to perform some extra computations to keep track of a set of neighbors and strangers for every individual, and to perform extra fitness evaluations during the hill-climbing steps after exploration.

RMEA realizes the possibility of distributing the conflicting goals of exploitation and exploration across its recurring stages. Since RMEA repeats its stages again and again, it does not need to make an explicit decision of ideal switching from one stage to another. Instead, its recurring stages ensure that the different phases of evolution are automatically distributed across the generations. While most algorithms seem to stagnate during the late generations, RMEA still continues optimization at a graceful rate. This is because the alternating and repeating stages ensure better immunity from stagnation.

The framework presented by RMEA is generic enough to be effectively extended to many other existing algorithms. As every EA has to maintain a population of potential solutions, it may readily introduce the participation of neighbors and strangers in the genetic operations. In addition, the algorithm may define its own specific exploitative and explorative operators, tailored particularly for the problem to be solved. However, the hill climbing steps after each explorative operation require extra evaluations of fitness and may increase the computational complexity, especially for some real world problems that require considerable computations for fitness evaluations.

Three parameters – k_1, k_2, k_3 define the proportion of conventional, explorative and exploitative operations during evolution. RMEA is not critically sensitive to these parameters. We have tested with several sets of values from the range of [10...100], and RMEA always performed well. However, moderate and equal values are preferred for these parameters (e.g., k_1=k_2=k_3=30). The amount of hill climbing steps after exploration has been made dependant on the amount of exploration caused by the immediate explorative operator. The neighborhood size is set randomly to 1%~5% of the population size during the advent of each generation. All these choices are just 'rational' and they are not meant to be optimal. RMEA is quite robust with all these parameter values. In case when the user has no prior problem-specific knowledge, the default values may be safely assumed as they have performed well with every problem we tested. However, we have also presented an adaptive variant of RMEA, i.e., Ada-RMEA, which adjusts the values of k_1, k_2, and k_3 based on the effectiveness of the different stages. Ada-RMEA shows better results than RMEA because it extends more effective stages and shortens weaker stages. More importantly, Ada-RMEA requires no problem specific knowledge from the user.

5.2 Future Works

i) **Incorporating the fitness landscape information to guide the algorithm:** The algorithm considers all individuals across the search space in similar fashion, without considering its position and prospects within the fitness landscape. The properties of the fitness landscape are also ignored for defining neighbors and strangers, exploitations, explorations, and for determining the necessary number of uphill steps (i.e., k_4) after explorative operations. Some approaches, e.g. [13], have reported significant improvements by guiding the algorithm using the information of the fitness landscape. So, it remains to be seen how the fitness landscape information can be incorporated into the algorithm for exploitation, exploration, hill climbing and for defining neighbors and strangers. We hope to explore this possibility in future.

ii) **Dynamic Size for Neighborhoods and Population**: The current implementation of RMEA uses the same number of neighbors and strangers for each individual throughout the search space. This ignores the actual distance among the individuals. Besides, a static population size is also used throughout the entire evolution. But the different phases of a search process usually requires varying amount of support from the population. Maintaining a dynamic population size has been proved effective for some previous works [40]. So, it remains to be seen whether dynamic size for the neighborhoods and population can produce better performance for RMEA.

iii) **Regarding Both Fitness and Diversity for Adaptation:** The adaptive variant of RMEA, i.e., ada-RMEA considers only fitness improvements by the different stages to adapt the stage lengths, without considering their effects on diversity. The reinsertion scheme is also solely based on fitness. So, a further possibility of research with RMEA is to employ both fitness and diversity information in order to adapt the parameters and stage lengths of RMEA, and to test how the adaptive system performs on some benchmark problems which demand preservation of adequate population diversity throughout the evolution.

iv) **Combining RMEA with Other Algorithms:** RMEA employs generic and simple genetic operations, e.g., recombination, crossover and mutation for both exploitation and exploration. But there exists several kinds of algorithms, e.g., greedy and machine learning algorithms, which may better exploit the current population information to constitute a more effective exploitation (and exploration) stage. So, there are lots of possibilities to check how different algorithms may be employed for exploitation and exploration to improve the performance of RMEA further.

v) **Parallel Implementation of RMEA:** RMEA is particularly suitable for parallel implementation. The conventional, exploration and exploitation stages may run in parallel as separate independent modules, while a central control module may get feedback from them from time to time in order to initiate some important procedure, like exchange of individuals between modules or recombination of the best individuals from different modules. This may significantly improve the performance of RMEA, especially for complex optimization problems.

vi) **Evaluate RMEA on Other problems:** RMEA has displayed successful results on numerical optimization problems. There exist numerous complex problems, including many real world ones. So it remains to be seen how RMEA performs on other problems. We hope to explore this possibility in the future.

References

[1] Yang, S. and Yao, X., "Experimental study on population-based incremental learning algorithms for dynamic optimization problems", Soft Computing, Vol-9, pp 815–834, 2005.

[2] Miller, J. F., Thomson, P. and Fogarty, T., "Designing electronic circuits using evolutionary algorithms. arithmetic circuits: a case study", Proceedings of the Genetic Algorithms and Evolution Strategies in Engineering and Computer Science, Quagliarella, D., Periaux, J., Poloni, C. and Winter, G., Eds. New York: Wiley, 1998.

[3] Koza, J. R., "Evolving the architecture of a multi-part program in genetic programming using architecture-altering operations", Proceedings of the Fourth Annual Conference on Evolutionary Programming, San Diego, CA, pp 695–717, 1995.

[4] He, J., Yao, X. and Li, J., "A Comparative Study of Three Evolutionary Algorithms Incorporating Different Amount of Domain Knowledge for Node Covering Problems", IEEE Transactions on Systems, Man, and Cybernetics, Part C, Vol-35, pp 266–271, 2005.

[5] Au, W. -H., Chan, K. C. C. and Yao, X., "Data Mining by Evolutionary Learning for Robust Churn Prediction in the Telecommunications Industry", IEEE Transactions on Evolutionary Computation, Vol-7, pp 532–545, 2003.

[6] Fürnkranz, J. and Kubat, M., Eds. Machines that Learn to Play Games, Nova Scientific Publishers, Huntington, NY, pp 11–59, 2001.

[7] Gordon, T. G. W. and Bentley, P. J., "On Evolvable Hardware", Soft Computing in Industrial Electronics, Ovaska, S. and Sztandera, L., Eds. Physica-Verlag, Heidelberg, pp 279–323, 2002.

[8] Taylor, T. and Massey, C., "Recent developments in the evolution of morphologies and controllers for physically simulated creatures", Artificial Life, Vol-7, No. 1, pp 77-87, 2001.

[9] Bäck, T., Fogel, D. B. and Michalewicz, Z., Eds. Handbook on Evolutionary Computation, Oxford University Press, New York, 1997.

[10] McPhee, N. and Hopper, N., "Analysis of Genetic Diversity through Population History", Proceedings of the Genetic and Evolutionary Computation Conference, Florida, Vol-3, pp 1112–1120, 1999.

[11] McKay, R., "Fitness sharing in genetic programming", Proceedings of the Genetic and Evolutionary Computation Conference, Las Vegas, Nevada, pp 435–442, 2000.

[12] Ursem, R. K., "Diversity guided Evolutionary algorithm", Proceedings of the Parallel Problem Solving from Nature (PPSN) VII, Vol-2439, Merelo, J. J., Adamidis, P., Schwefel, H. P., Eds. Granada, Spain, pp 462–471, 2002.

[13] Ursem, R. K., "Multinational Evolutionary Algorithms," Proceedings of the Congress of Evolutionary Computation (CEC), Angeline, P. J., Michalewicz, Z., Schoenauer, M., Yao, X. and Zalzala, A., Eds. pp 1633–1640, 1999.

[14] Thomsen, R., Rickers, P. and Krink, T., "A Religion-Based Spatial Model For Evolutionary Algorithms", Proceedings of the Parallel Problem Solving from Nature (PPSN) VI, Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J. J. and Schwefel, H. P., Eds. pp 817–826, 2000.

[15] Mahfoud, S., "Crowding and preselection revisited", Technical Report 92004, Illinois Genetic Algorithms Laboratory (IlliGAL), 1992.

[16] Goldberg, D. E. and Richardson, J., "Genetic Algorithms with Sharing for Multimodal Function Optimization", Proceedings of the International Conference on Genetic Algorithms (ICGA), Grefenstette, J. J., Ed. Cambridge, MA, pp 41-49, 1987.

[17] Cobb, H. G. and Grefenstette, J. F., "Genetic algorithms for tracking changing environments", in Proceedings of the International Conference on Genetic Algorithms, Forrest, S., Ed. Urbana-Champaign, IL, pp 523-530, 1993.

[18] Greenwood, G. W., Fogel, G. B. and Ciobanu, M., "Emphasizing Extinction in Evolutionary Programming", Proceedings of the Congress of Evolutionary Computation, Angeline, P. J., Michalewicz, Z., Schoenauer, M., Yao, X. and Zalzala, A., Eds. pp 666–671, 1999.

[19] Shimodaira, H., "A Diversity Control Oriented Genetic Algorithm (DCGA): Development and Experimental Results", Proceedings of the Genetic and Evolutionary Computation Conference, Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M. and Smith, R. E., Eds. pp 603–611, 1999.

[20] Oppacher, F. and Wineberg, M., "The shifting balance genetic algorithm: Improving the GA in dynamic environment", Proceedings of the Genetic and Evolutionary Computation Conference, Vol-1, Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M. and Smith, R. E., Eds. pp 504–512, 1999.

[21] Wolpert, D. H. and Macready, W. G., "No free lunch theorems for optimization", IEEE Transactions on Evolutionary Computation, Vol-1, No. 1, pp 67–82, 1997.

[22] Yao, X., Liu, Y. and Lin, G., "Evolutionary Programming Made Faster", IEEE Transactions on Evolutionary Computation, Vol-3, No. 2, 1999.

[23] Back, T., Evolutionary algorithms in theory and practice, Oxford, England: Oxford University Press, 1996.

[24] Goldberg, D.E., Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA, 1989.

[25] Fogel, L.J., Owens, A.J. and Walsh, M.J., Artificial Intelligence through simulated evolution, New York, John Wiley & Sons, 1966.

[26] Booker, L.B., Goldberg, D.E. and Holland, J.H., "Classifier systems and genetic algorithms", Artificial Intelligence, Vol-40, pp 235–282, 1989.

[27] Koza, J.R., Genetic programming: On the programming of computers by means of natural selection, Cambridge, MA: MIT Press, 1992.

[28] Mühlenbein, H. and Schlierkamp-Voosen, D., "Predictive Models for the Breeder Genetic Algorithm: Continuous Parameter Optimization", Evolutionary Computation, Vol-1, No. 1, pp 25–49, 1993.

[29] Bäck, T., "Optimal mutation rates in genetic search", Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA), Forrest, S., Ed. pp 2–8, 1993.

[30] Bäck, T., Evolutionary Algorithms in Theory and Practice - Evolution Strategies, Evolutionary Programming, Genetic Algorithms, Oxford, New York: Oxford University Press, 1996.

[31] Mühlenbein, H., "The Breeder Genetic Algorithm - a provable optimal search algorithm and its application", IEE Colloquium, Applications of Genetic Algorithms, Digest No. 94/067, London, March 15, 1994.

[32] Schwefel, H. –P., Numerical Optimization of Computer Models, Chichester, UK: John Wiley & Sons, 1981.

[33] Rechenberg, I., "Evolutionsstrategie '94", Werkstatt Bionik und Evolutionstechnik, Vol-1, frommann-holzboog, Stuttgart: Germany, 1994.

[34] Fogel, D. B., Evolutionary computation: toward a new philosophy of machine intelligence, New York, IEEE Press, 1995.

[35] Ostermeier, A., Gawelczyk, A. and Hansen, N., "A derandomized approach to self-adaptation of evolution strategies", Evolutionary Computation, Vol-2, No. 4, pp 369–380, 1994.

[36] Ostermeier, A., Gawelczyk, A. and Hansen, N., "Step size adaptation based on non-local use of selection information," in Proceedings of the Parallel Problem Solving from Nature (PPSN) III, Davidor, Y., Schwefel, H. –P., Männer, R., Eds. Berlin: Springer, pp 189–198, 1994.

[37] Hansen, N., Ostermeier, A. and Gawelczyk, A., "On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation", in Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA), Eshelman, L., Ed. Pittsburgh, PA, pp 57-64, 1997.

[38] Thomsen, R. and Rickers, P., "Introducing Spatial Agent-Based Models and Self-Organised Criticality to Evolutionary Algorithms", Master's thesis, University of Aarhus, Denmark, 2000.

[39] Krink, T., Thomsen, R. and Rickers, P., "Applying Self-Organised Criticality to Evolutionary Algorithms", in Proceedings of the Parallel Problem Solving from Nature (PPSN) VI, Vol-1917, Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J.J. and Schwefel, H. -P., Eds. pp 375–384, 2000.

[40] Lorena, L.A.N. and Furtado, J.C., "Constructive Genetic Algorithm for Clustering Problems", Evolutionary Computation, Vol-9, No. 3, pp 309-327, 2001.

Appendix A

Benchmark Functions

A. Sphere Model

$$f_1(x) = \sum_{i=1}^{30} x_i^2, \qquad -100 \le x_i \le 100$$

$$\min(f_1) = f_1(0,0,\ldots,0) = 0$$

B. Schwefel's Problem 2.22

$$f_2(x) = \sum_{i=1}^{30} |x_i| + \prod_{i=1}^{30} |x_i|, \qquad -10 \le x_i \le 10$$

$$\min(f_2) = f_2(0,0,\ldots,0) = 0$$

C. Schwefel's Problem 1.2

$$f_3(x) = \sum_{i=1}^{30} \left(\sum_{j=1}^{i} x_j \right)^2, \qquad -100 \le x_i \le 100$$

$$\min(f_3) = f_3(0,0,\ldots,0) = 0$$

D. Schwefel's Problem 2.21

$$f_4(x) = \max_i \{|x_i|,\ 1 \le i \le 30\}, \qquad -100 \le x_i \le 100$$

$$\min(f_4) = f_4(0,0,\ldots,0) = 0$$

E. Generalized Rosenbrock's Function

$$f_5(x) = \sum_{i=1}^{29} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(x_i - 1\right)^2 \right], \qquad -30 \le x_i \le 30$$

$$\min(f_5) = f_5(1,1,\ldots,1) = 0$$

F. Step Function

$$f_6(x) = \sum_{i=1}^{30} \left(\lfloor x_i + 0.5 \rfloor \right)^2, \qquad -100 \le x_i \le 100$$

$$\min(f_6) = f_6(0,0,\dots,0) = 0$$

G. Quartic Function i.e. Noise

$$f_7(x) = \sum_{i=1}^{30} ix_i^4 + random[0, 1), \qquad -1.28 \le x_i \le 1.28$$

$$\min(f_7) = f_7(0,0,\dots,0) = 0$$

H. Generalized Schwefel's Problem 2.26

$$f_8(x) = -\sum_{i=1}^{30} x_i \sin\left(\sqrt{|x_i|}\right), \qquad -500 \le x_i \le 500$$

$$\min(f_8) = f_8(420.9687, 420.9687, \dots, 420.9687) = 0$$

I. Generalized Rastrigin's Function

$$f_9(x) = \sum_{i=1}^{30} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right], \qquad -5.12 \le x_i \le 5.12$$

$$\min(f_9) = f_9(0,0,\dots,0) = 0$$

J. Ackley's Function

$$f_{10}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{30} x_i^2}\right) - \exp\left(\frac{1}{30}\sum_{i=1}^{30}\cos 2\pi x_i\right) + 20 + e$$

$$-32 \le x_i \le 32$$

$$\min(f_{10}) = f_{10}(0,0,\dots,0) = 0$$

K. Generalized Griewank Function

$$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{30}\left(x_i^2\right) - \prod_{i=1}^{30}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \qquad -600 \le x_i \le 600$$

$$\min(f_{11}) = f_{11}(0,0,\dots,0) = 0$$

L. Generalized Penalized Functions

$$f_{12}(x) = \frac{\pi}{30}\left\{10\sin^2(\pi y_1) + (y_{30} - 1)^2 + \sum_{i=1}^{29}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right]\right\}$$
$$+ \sum_{i=1}^{30} u(x_i, 10, 100, 4)$$

$$-50 \le x_i \le 50, \quad \min(f_{12}) = f_{12}(1,1, \ldots, 1) = 0$$

$$f_{13}(x) = 0.1\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{29}(x_i - 1)^2\left[1 + \sin^2(3\pi x_{i+1})\right] + (x_{30} - 1)^2\left[1 + \sin^2(2\pi x_{30})\right]\right\} + \sum_{i=1}^{30} u(x_i, 5, 100, 4)$$

$$-50 \le x_i \le 50, \quad \min(f_{13}) = f_{13}(1,1, \ldots, 1) = 0$$

where, $y_i = \frac{1}{4}(x_i + 1)$ and $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \le x_i \le a \\ k(-x_i - a)^m, & x_i < -a \end{cases}$

M. Shekel's Foxholes Function

$$f_{14}(x) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}\right]^{-1}$$

$$-65.536 \le x_i < 65.536, \quad \min(f_{14}) = f_{14}(-32, -32) \approx 1$$

where, $(a_{ij}) = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \ldots\ldots\ldots 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \ldots\ldots\ldots 32 & 32 & 32 \end{pmatrix}$

N. Kowalik's Function

$$f_{15}(x) = \sum_{i=1}^{11}\left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2, \qquad -5 \le x_i \le 5$$

$$\min(f_{15}) \approx f_{15}(0.1928,\ 0.1908,\ 0.1231,\ 0.1358) \approx 0.0003075$$

O. Six-Hump Camel-Back Function

$$f_{16}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1 x_2 - 4x_2^2 + 4x_2^4, \qquad -5 \le x_i \le 5$$

$$x_{\min} = (0.08983, -0.7126),\ (-0.08983, 0.7126)$$

$$\min(f_{16}) = -1.0316285$$

Table A.1
Coefficients for Kowalik's function, f_{15}

i	a_i	b_i^{-1}
1	0.1957	0.25
2	0.1947	0.5
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

P. Branin Function

$$f_{17}(x)=\left(x_2-\frac{5.1}{4\pi^2}x_1^2+\frac{5}{\pi}x_1-6\right)^2+10\left(1-\frac{1}{8\pi}\right)\cos(x_1)+10,$$

$-5 \le x_1 \le 10,\ 0 \le x_2 \le 15$

$x_{\min}=(-3.142,\ 12.275),\ (3.142,\ 2.275),\ (9.425,\ 2.425)$

$\min(f_{17})=0.398$

Q. Goldstein-Price Function

$$f_{18}(x)=\left[1+(x_1+x_2+1)^2\left(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2\right)\right]$$
$$\times\left[30+(2x_1-3x_2)^2\left(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2\right)\right]$$

$-2 \le x_i \le 2, \quad \min(f_{18})=\min(0,-1)=3$

R. Hartman's Family

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

with n = 3, 6 for f_{19} and f_{20}, respectively. $0 \le x_j \le 1$. The coefficients: c_i and a_{ij} are defined in Tables A.2 and A.3, for functions f_{19} and f_{20}, respectively. The global minimum is equal to -3.86 for f_{19} and -3.32 for f_{20}, which are reached at the point (0.114, 0.556, 0.852) and (0.201, 0.150, 0.477, 0.275, 0.311, 0.657) respectively.

Table A.2
Coefficients for Hartman function, f_{19}

i	a_{ij}, j = 1, 2, 3			c_i	p_{ij}, 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

Table A.3
Coefficients for Hartman function, f_{20}

i	a_{ij}, j = 1, …, 6						c_i	p_{ij}, j = 1, …, 6					
1	10	3	17	3.5	1.7	8	1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.05	10	17	0.1	8	14	1.2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	3	3.5	1.7	10	17	8	3	0.2348	0.1415	0.3522	0.2883	0.3047	0.6650
4	17	8	0.05	10	0.1	14	3.2	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

S. Shekel's Family

$$f_{23}(x) = -\sum_{i=1}^{m}\left[(x-a_i)(x-a_i)^T + c_i\right]^{-1}$$

with m = 5, 7, 10 for f_{21}, f_{22} and f_{23}, respectively.

$0 \le x_j \le 10$. These three functions have 5, 7 and 10 local minima respectively.

For $1 \le i \le m$, $x_{\text{local\_opt}} \approx a_i$ and $f\left(x_{\text{local\_opt}}\right) \approx 1/c_i$.

The coefficients: c_i and a_{ij} are defined in Table A.4.

Table A.4
Coefficients for Shekel functions f_{20}, f_{22}, f_{23}

i	a_{ij}, j = 1, …, 4				c_i
1	4	4	4	4	0.1
2	1	1	1	1	0.2
3	8	8	8	8	0.2
4	6	6	6	6	0.4
5	3	7	3	7	0.4
6	2	9	2	9	0.6
7	5	5	3	3	0.3
8	8	1	8	1	0.7
9	6	2	6	2	0.5
10	7	3.6	7	3.6	0.5