NICHING EVOLUTIONARY ALGORITHMS FOR MULTIMODAL AND DYNAMIC OPTIMIZATION

YU LING

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Master of Engineering

2010

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Associate Professor Ponnuthurai Nagaratnam Suganthan, for his kind guidance and financial support during my postgraduate study. He has led me into the kingdom of research. His continuous encouragement and valuable advices are most important for the completion of this work.

I am grateful to the technical staff in Machine Learning Lab and all my fellow students who have created a friendly environment and have shared with me their knowledge and experience. I would also like to thank the staff at Parallel and Distributed Computing Centre in the School of Computer Engineering for providing great simulation facilities.

I am greatly indebted to my parents for their endless love and support for my study in Singapore. I would like to dedicate this thesis to them.

Table of Contents

List of Figures

List of Tables

Summary

Many optimization functions have complex landscapes with multiple global or local optima. In order to solve such problems, niching evolutionary algorithms were introduced. The "niching" concept in evolutionary algorithms was brought from the ecological "niches". It describes the roles that different individuals take when there are several optima to pursue. Niching gives growth to diverse promising species in the population, making it possible to locate multiple optima in a multimodal landscape.

In this thesis, a literature review on evolutionary algorithms and several classes of niching methods is presented. After that, a simulation-based comparative study is carried out using hybrid composition test functions with multiple global optima. Three popular niching techniques with binary genetic algorithms are examined for their searching ability, accuracy and computation speed in solving the hybrid composition problems. The number of functions evaluations is employed as the main performance measure. It has been observed that the performance of the niching methods varies with problems, while methods that belong to the same class have shared characteristics.

Based on these observations, an ensemble of niching algorithms is proposed. Despite the fact that various niching methods have been investigated in the last few decades, no effort has been reported on combining different niching methods in a single algorithm. With the ensemble of different niching methods in one algorithm, the populations with different niching methods can lead the evolution to distinct paths and keep a high level of diversity. Exchange of solutions among the niching populations expedites the ensemble algorithm in finding multiple optimal solutions. Results have shown that the proposed algorithm is either comparable to or better than any of the

composing single niching methods in terms of its searching ability on every test problem.

In addition, this thesis includes the application of niching methods to multimodal optimization problems in a dynamic environment. A recent version of evolutionary programming, named unbiased evolutionary programming, is modified by introducing a schedule to adjust the strategy parameter and the local search towards the most improving directions. To make the algorithm adaptable to environmental changes, two niching external archives are used. They serve as short-term and long-term memories respectively and enhance the diversity of the population. The proposed algorithm was accepted for CEC 2009 Competition on Evolutionary Computation in Dynamic and Uncertain Environments and has yielded satisfactory results on a majority of the test problems.

Chapter 1

Introduction

1.1 Motivations

Multimodal optimization has been a subject of interest for many years. As opposed to unimodal problems, multimodal problems exhibit the feature of having multiple global or local optima. There are generally two scenarios of multimodal optimization. The first one aims at finding the global optimum when the objective function has plenty of local optima. In order to do so, a search algorithm should be able to explore a vast proportion of the search area and avoid being trapped in a local optimum. In the second case, there exist several equally good solutions, which raises the issue of how an algorithm can capture them all.

Many different algorithms and techniques have been developed to deal with multimodal problems. Evolutionary Algorithms (EAs) are a popular category. Inspired by biological evolution processes such as selection, reproduction, and mutation, EAs are known to be robust global search algorithms for optimization and machine learning [1], [2]. Moreover, the introduction of niching and speciation in EAs adds their strength in handling multimodal optimization problems [3]. Now, not only can an EA provide us with one global optimum, but also it can effectively identify several global and local optima of a function.

Despite of the fact that various niching methods have been proposed and used with the evolutionary algorithms, evaluations of such methods are in most cases limited to empirical comparisons with regard to a few test functions which share a low level of

complexity. Before we can apply the niching evolutionary algorithms to more challenging environments, we still have to prove their strengths using sophisticated test problems and suitable performance metrics. Observations of the characteristics of these algorithms will help us to analyze how to improve them, or to combine certain algorithms to achieve a better effect.

Another benefit that the niching methods are offering us is the increased diversity of the population of candidate solutions. As an EA evolves, it will converge to a point in the search space. Sometimes a suboptimal solution will dominate the population, preventing the algorithm from further evolution. Such circumstances of premature convergence are unfavourable, especially when some forms of changes in the optimization environment are concerned. In a changing environment, a successful optimization algorithm should not only be able to locate the optimum, as it does in the static sense, but also be capable of detecting the environmental changes and tracking the new optimum. Therefore, it is essential to develop an algorithm that can maintain a diverse population and is not as prone to converge prematurely.

1.2 Objectives

The main objectives of the research are:

- To improve the searching ability of evolutionary algorithms in order to meet the requirement of locating multiple optima;
- To develop new niching algorithms that are suitable for a wider range of multimodal optimization problems;

- To introduce diversity enhancing measures that can prevent stagnation in the search process; and
- To extend the use of niching evolutionary algorithms to high-dimensional multimodal optimization and dynamic optimization.

1.3 Major Contributions of the Thesis

The major contributions of this thesis are summarized as follows:

- Review of literature on evolutionary algorithms and niching methods;
- Empirical comparison of niching methods on hybrid composition functions with novel evaluation criteria;
- Design of ensembles of niching algorithms (ENA) in the scope of genetic algorithms (GA) that preserve diversity of the populations and benefit from the best composing method; and
- Application of niching methods in the external memories of evolutionary programming (EP) to solve multimodal optimization problems in a dynamic environment.

1.4 Organization of the Thesis

The remainder of the thesis is organized as follows. Chapter 2 presents a literature review covering several evolutionary algorithms, including genetic algorithms,

evolutionary programming, and evolution strategies (ES). Existing niching and speciation methods as well as spatially structured evolutionary algorithms (SSEAs) are described. In Chapter 3, performances of niching methods are examined using a set of hybrid composition test functions. Chapter 4 introduces the ensemble of niching algorithms with its construction, procedures, implementation details and strengths. Experimental results of ENA and the single methods are analyzed in this chapter. Chapter 5 presents evolutionary programming with the use of niching in external memories for dynamic optimization. Chapter 6 concludes the thesis and lists a few areas for future research.

Chapter 2

Literature Review

2.1 Evolutionary Algorithms

Evolutionary Algorithms are a class of probabilistic search and optimization algorithms which incorporates aspects of natural selection and survival of the fittest. Different from the traditional search methods, they make use of a collection of candidate solutions, named the *population*, that evolve according to rules of selection, recombination, mutation and survival, referred to as genetic operators. Each candidate solution is called an *individual* in the population. A *fitness* value, which is usually linked directly to the objective function, is assigned to every individual, and it determines whether the solution can reproduce itself or survive into the next *generation*, as it is called in an EA. In every generation, selection acts as a force to increase the quality of the population. Often individuals with a higher fitness have a higher chance to be selected as parents. Recombination and mutation then create the necessary diversity and promote the exploration of new and promising areas of the search space. The processes make the population gradually converge to the optimal point. Figure 2.1 shows the typical procedures of an EA.

EAs have many advantages over classical optimization techniques. They are especially useful when the mathematical properties such as the derivatives of the objective function cannot be readily obtained. They perform well on complex, discontinuous, and “noisy” fitness landscapes where plenty of local optima exist and yet have higher chance of attaining the global optimum.

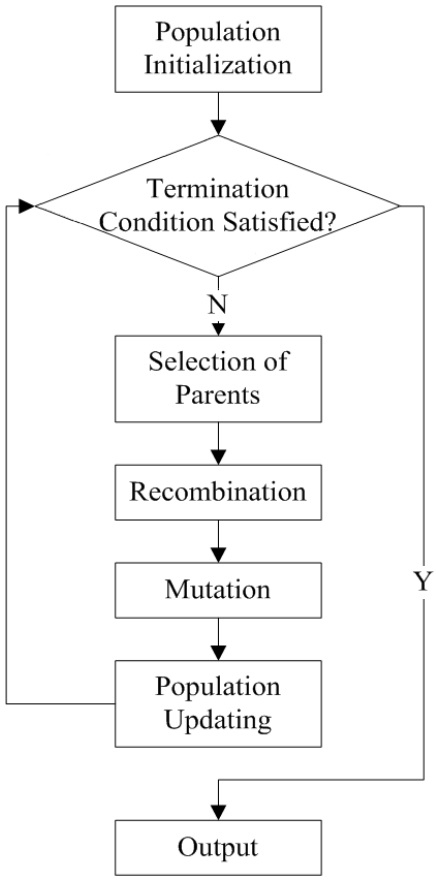

Figure 2.1 Flowchart of a General Evolutionary Algorithm

The mainstreams of evolutionary algorithms include genetic algorithms, evolutionary programming, and evolution strategies. They were developed independently of each other. From the early nineties on, these techniques are unified as different types of EAs. Successful applications of them have been seen in a variety of optimization problems, such as routing, scheduling, travelling salesman's problem, knapsack problems, image processing and signal analysis, data mining, and financial forecasting. The initial formulations of EP, ES and GAs are for solving unconstrained problems, but with the help of many constraint handling methods, EAs are powerful techniques for solving constrained problems as well.

2.1.1 Genetic Algorithms

The origin of genetic algorithms dates back to the 1950s [4], [5], when computer simulations of evolution started. In the 1970s, genetic algorithms were formally introduced by J. H. Holland in his book *Adaptation in Natural and Artificial Systems* (1975) [6]. They incorporate biological evolution processes such as inheritance, selection, recombination (also called crossover), and mutation. With these characteristics, the algorithms are able to explore a far greater range of potential solutions to a problem than conventional search methods.

Two techniques of population control are currently used in the field of serial genetic algorithms: generational and steady-state. The former one, which is probably the most widely used, replaces the entire population at once, while the latter replaces only a few members at a time [7]. The difference in selection strength and genetic drift often leads to different performance behaviors between the two techniques.

A canonical generational genetic algorithm usually has the following steps.

Population Initialization:

In this step, a population is formed by a number (known as the population size) of randomly generated candidate solutions in the search space. Each individual of the population is coded into a specific type of representation, e.g. float, binary or Gray, and the fitness values are calculated.

Termination Condition:

Usually a maximum number of generations or a maximum number of function evaluations or desired objective value to reach is used as the termination condition. If

the condition is met, the algorithm will go to the output step. Otherwise, it will iterate through the loop that contains the genetic operators.

Selection of Parents:

Individuals can either be selected through a fitness-based process, where fitter solutions are more likely to be selected, or be chosen randomly for creating offspring of the population. The most common and well-studied selection methods are roulette wheel selection and tournament selection.

Roulette wheel selection chooses members from the population based on a probability that is proportional to their fitness. Thus, positive fitness values are required for this method. Parents are selected according to their fitness. The better the fitness of the individual, the greater the chance it will be selected.

Tournament selection involves running "tournaments" among a few individuals chosen at random from the population. The winner of each tournament (the one with the best fitness) is selected for crossover. The selection pressure of this method can be easily adjusted by changing the tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected.

Crossover and Mutation:

After selecting a pair of parents, the algorithm will generate two offspring by certain crossover and mutation schemes. When binary coding is employed, one-point, two-point and uniform crossovers are commonly used with bit-wise mutation operation.

One-point crossover is a typical crossover operator in GA. It chooses a random slicing position along the chromosomes of both parents. All data beyond that point in either

organism string is swapped between the two parents. The resulting chromosomes are two offspring. An instance of one-point crossover is as follows. If the parents have binary strings 00001111 and 01011010, and the crossover point is chosen to be after the fourth bit, then the offspring are 00001010 and 01011111.

Two-point crossover calls for two points to be selected on the parent chromosomes. Everything between the two points is swapped between the parents, rendering two offspring.

In uniform crossover, every bit of the offspring can be taken from either parent with a fixed probability, typically 0.5.

For bitwise mutation, every bit has the same mutation rate, say, 0.01, and it changes from 0 to 1 or from 1 to 0 if a randomly generated value (between 0 and 1) is less than the mutation rate. For example, an individual may change from 00001100 to 00001101 under the mutation rate 0.1.

Population Updating:

For every offspring, the new fitness value is calculated. If the offspring generation totally replaces the parent generation regardless of their fitness values, it is a non-elitist way, and the best individual in the current generation may not be guaranteed a place in the next generation.

In an elitist manner, the best individual is copied to the population in the next generation. The rest are chosen in classical way. Elitism prevents losing the best found solution to date. A variation is to replace the worst solution in the offspring by the best in the parent population.

Output:

Users can define what should be displayed and saved. The recorded information can include the best individual with its fitness value, the population average fitness value, the standard deviation of fitness values and so on.

2.1.2 Evolutionary Programming

Evolutionary programming was first proposed by L. J. Fogel in 1960s as an alternative method for generating artificial intelligence [8]. Later, since the middle of 1980s, it has been developed to solve more general tasks including prediction problems, numerical and combinatorial optimizations, and machine learning [9], [10]. As a member of EAs, the basic idea of EP is also the imitation of the natural evolution. Since this approach models organic evolution at the level of species, the original EP does not rely on any kind of recombination or crossover. There is no information exchange among individuals in EP. Mutation is the only way to generate offspring.

Current studies involving evolutionary programming usually make use of self-adaptation of the strategy parameter in EP [11]–[13]. According to the description in [11], the self-adaptive EP is implemented as follows:

1) Generate the initial population of μ individuals, and set k=1. Each individual is taken to be a pair of real-valued vectors (x_i, η_i), i=1,2,…,μ, where x_i is the vector of objective variables and η_i is a vector of standard deviations (also known as strategy parameters) corresponding to Gaussian mutations.

2) Evaluate the fitness value for each individual (x_i, η_i) of the population based on the objective function $f(x_i)$.

3) Each parent (x_i, η_i) creates an offspring (x_i', η_i'), by:

$$x_i'(j) = x_i(j) + \eta_i(j)N_j(0,1) \tag{2-1}$$

$$\eta_i'(j) = \eta_i(j)\exp(\tau' N(0,1) + \tau N_j(0,1)) \tag{2-2}$$

for j=1,2,…,n, where n is the number of dimensions, $N(0,1)$ denotes a normally distributed one-dimensional random number with mean zero and standard deviation one. $N_j(0,1)$ indicates that the random number is re-sampled anew for each value of j. The factors τ and τ' are commonly set to $(\sqrt{2\sqrt{n}})^{-1}$ and $(\sqrt{2n})^{-1}$. The order of (2-1) and (2-2) may be swapped to improve the performance. Note that the offspring generated by such rules can be out of the variable domain of the problem. Hence, boundary checks should be performed.

4) Calculate the fitness of each offspring (x_i', η_i').

5) Conduct pair-wise comparison over the union of parents (x_i, η_i) and offspring (x_i', η_i'), $\forall i$=1,2,…,μ. For each individual, q opponents are chosen uniformly at random from the combined parents and offspring population. For each comparison, if the individual's fitness is no smaller than the opponent's, it receives a "win."

6) Select the μ individuals out of (x_i, η_i) and (x_i', η_i'), $\forall i$=1,2,…,μ, that have the most wins to be parents of the next generation.

7) Stop if the halting criterion is satisfied; otherwise, set k=k+1 and go to Step 3.

Another trend to improve EP is to modify or change the mutation operator. Instead of the traditional Gaussian mutation, Cauchy mutation and Lévy mutation were also applied to EP, and they were shown to turn out better performance when searching in a large neighbourhood and in the case of functions having many local optima respectively [14], [15].

2.1.3 Evolution Strategies

Evolution strategies were proposed in the 1960s and developed further along the 1970s and later by I. Rechenberg and H.-P. Schwefel [16], [11]. Evolution strategies focus on solving real-valued parameter optimization problems and the evolution is usually modeled on the phenotypic level, which means no coding is needed for the candidate solutions. The primary operators of ES are mutation, recombination and selection.

Despite their independent development over 30 years, ES and EP share many similarities. Both of them use multivariate zero mean Gaussian mutation to create offspring. The similarities extend to the use of self-adaptive methods for the mutation. Most of the theoretical results on convergence developed for ES or EP also apply directly to the other.

The main differences between ES and EP are in selection and recombination. EP typically uses stochastic selection via a tournament. Each individual in the population faces competition against a predefined number of opponents and receives a “win” if its fitness value is at least as good as its opponent’s. Individuals with the least wins are eliminated. In contrast, ES typically uses deterministic truncation selection in

which the worst individuals are wiped out from the population based directly on their fitness rankings. Since EP is an abstraction of evolution at the species level and recombination does not occur between species, there are no recombination mechanisms used in EP. On the contrary, ES is an abstraction of evolution at the level of individuals, and thus some forms of recombination are reasonable.

The simplest ES involves only one point (parent) and the result of its mutation (offspring). The parent and the offspring compete for survival. Only if the offspring's fitness is higher, it becomes the parent of the next generation. Otherwise the mutant is disregarded. This is a (1+1)-ES. More generally, λ mutants can be generated and compete with the parent, called (1+λ)-ES. In a (μ, λ)-ES, μ parents produce $\lambda > \mu$ offspring but only the best μ offspring survive, while the current parents are always disregarded. Contemporary derivatives of ES often use a population of μ parents and also recombination as an additional operator (called (μ/ρ+, λ)-ES) [17]. This is believed to make them less prone to get stuck in local optima. On the other hand, mutation strategy parameter control can be realized to favor previously selected mutation steps in the future, and this results in a completely derandomized self-adaptation scheme, called *covariance matrix adaptation* (CMA), and the new version of ES, CMA-ES [18], [19].

2.2 Niching Methods in Evolutionary Algorithms

The idea of niching is inspired by the way organisms evolve in nature. In ecology, a "niche" can be used to describe the role which a species takes to deal with the distribution of resources and competition, while at the same time altering those factors.

An example is Darwin's Finches, a group of finch species on the Galápagos Islands [20]. Faced with the need to search for alternative food source, they adapted themselves to the environment and eventually a great variety of beak sizes and shapes resulted. Since within a niche, individuals are forced to share the same available food source, the species grow into different niches, to avoid the conflict for the resources.

In evolutionary computation, niching involves the formation of subgroups within a population where each subgroup targets a specific task such as discovering one peak in a multimodal function. Compared with the simple evolutionary algorithm, the uniqueness of niching algorithms lies in the fact that they preserve not only the highly-fit individuals, but also weaker individuals so long as they belong to groups without the highly-fit ones. This gives the population an opportunity to pass the genetic information of such individuals to their offspring, and it ensures that this information does not become extinct quickly. By maintaining a reasonably balanced variety of genetic information, niching procedures allow the population to simultaneously focus on more than one region in the search space, which is essential to discover several optima in a single run.

Since the earliest niching approach, *preselection*, was proposed by Cavicchio [21] in 1970, several niching methods such as crowding [22]–[28], sharing [29]–[36], and clearing [37]–[39] have emerged.

2.2.1 Crowding and Restricted Tournament Selection

The crowding methods encourage competition for limited resources among similar individuals in the population. They follow the analogy that dissimilar individuals tend

to occupy different niches, so that they typically do not compete. The end result is that in a fixed-size population at equilibrium, new members of a particular species replace older members of that species, and the overall number of members of a particular species does not change [23]. They make use of a distance measure (either in genotype or phenotype space) to determine similarity between individuals and inserts new members into the population by replacing similar ones. Belonging to this category are *crowding* [22], *deterministic crowding* [23], [24], *restricted tournament selection* (RTS) [25], and so on.

Crowding [22] was introduced in De Jong's dissertation in 1975. The algorithm allows only a fraction of the population to reproduce and die in each generation. A percentage of the population, specified by the *generation gap*, is chosen via the fitness-proportionate roulette wheel selection to go through reproduction. For every newly-created individual, *CF* (a number called crowding factor) individuals are randomly taken from the population and the most similar to the new individual (either in a genotypic or a phenotypic sense) is replaced. Fitness values are not involved in the replacement procedure.

Mahfoud [23], [24] proposed deterministic crowding as an improvement for crowding. The algorithm dispenses with the fitness-proportionate selection of crowding, and randomly selects two parents from the population instead to perform crossover and mutation. To add back selection pressure and minimize replacement error that can occur in preselection, the two offspring replace their closest parent if the offspring has higher fitness. No parameters in addition to those of a simple EA are used in the algorithm. The replacing scheme is as shown in Figure 2.2.

Let ***c1***, ***c2*** be the offspring created by ***p1***, ***p2***,
if distance(***p1***,***c1***)+distance(***p2***,***c2***)
  <distance(***p1***,***c2***)+distance(***p2***,***c1***)
  if fitness(***c1***)>fitness(***p1***), replace ***p1*** with ***c1***;
  if fitness(***c2***)>fitness(***p2***), replace ***p2*** with ***c2***.
else
  if fitness(***c2***)>fitness(***p1***), replace ***p1*** with ***c2***;
  if fitness(***c1***)>fitness(***p2***), replace ***p2*** with ***c1***.
end

Figure 2.2 Replacement in Deterministic Crowding

Restricted tournament selection [25] adapts tournament selection (see Section 2.1.1) for multimodal optimization. Like deterministic crowding, it selects two parents from the population to generate two offspring by performing crossover and mutation. After that, for each offspring, the algorithm chooses a random sample of *w* (window size) individuals from the population and determines which one is the nearest to the offspring, by either Euclidean (for real variables) or Hamming (for binary variables) distance measure. The nearest member within the w individuals will compete with the offspring to determine who has higher fitness. If the offspring wins, it is allowed to enter the population by replacing its opponent. As more and more highly fit individuals enter the population, this method should force tougher competition for spots to be held between members of the same niche, but at the same time it allows other niches to flourish. The RTS is shown in Figure 2.3.

Many algorithms follow the crowding idea. There are *multi-niche crowding* [26], in which both the selection and replacement steps are modified with some type of crowding, and *q-nearest neighbors replacement* [27], [28], which introduces a competition between the offspring and the worst of its *q* nearest neighbors, among others.

```
N=size of population;
for i=1:size of offspring,
  choose w random numbers from 1 to N without replacement as selected_indices;
  opponent_index=selected_indices(1);
  for j=2:w,
    if distance between offspring(i) and population(selected_indices(j)) <
        distance between offspring(i) and population(opponent_index)
      opponent_index= selected_indices(j);
    end
  end
  if fitness(population(opponent_index)) < fitness(offspring(i))
    population(opponent_index) = offspring(i);
  end
end
```

Figure 2.3 The RTS

2.2.2 Sharing and Clustering

In sharing methods, similar individuals are penalized by a reduction in their fitness values, as they share the limited resource of that niche. The standard *fitness sharing* [29] uses a sharing radius which is defined beforehand and usually fixed throughout the niching process. The population member i's fitness, called the shared fitness, is given by:

$$f_{shared}(i) = \frac{f_{original}(i)}{\sum_{j=1}^{N} sh(d_{ij})} \tag{2-3}$$

with the sharing function

$$sh(d_{ij}) = \begin{cases} 1-(\frac{d_{ij}}{\sigma_{share}})^{\alpha}, & \text{if } d_{ij} < \sigma_{share} \\ 0, & \text{otherwise} \end{cases} \tag{2-4}$$

where N is the population size, d_{ij} is the distance between individuals i and j, σ_{share} is the sharing radius, and α (usually equals to 1) is a constant parameter describing the sharing level. This method derates the payoff in densely-populated regions, thus, it encourages search in unexplored regions, and causes subpopulations to form.

The clustering methodology is also incorporated into niching methods to help form the niches and avoid the need for estimation of σ_{share}. In [30], the formation of the niches is based on the adaptive *MacQueen's K-Means* algorithm. The algorithm begins with k seed points taken as the best k individuals. Using a minimum allowable distance d_{min} between niche centers, a few clusters are formed from the seed points. The remaining population members are then added to these existing clusters or are used to form new clusters based on parameters d_{min} and d_{max}. The fitness is calculated based on the distance d_{ic} between the individual and its niche center only, which makes the algorithm computationally faster than the standard fitness sharing approach. The final fitness of an individual is calculated by the relation:

$$F_i = \frac{f_i}{n_c\left(1-\left(\frac{d_{ic}}{2d_{\max}}\right)^{\alpha}\right)} \tag{2-5}$$

where n_c is the number of individuals in the niche containing the individual i, d_{max} is the maximum distance allowed between an individual and its niche center, and α is a constant.

Sharing methods are of great interest to many researchers. Modifications and extensions of sharing, especially with regard to the identification and maintenance of the niches, can be found in various studies [31]–[36]. The recent *dynamic fitness sharing* (DFS) [36] allows a dynamic identification of the niches and applies the

sharing mechanism to each niche separately. It also incorporates a species elitist strategy that keeps the best individual in every niche, called species masters. The forming of niches is obtained by ordering the population at every generation according to the raw fitness of the individuals. Then, the best individual is chosen as a species master. Its species can be defined as a subset of individuals in the population which have a distance from the species master less than the niche radius and do not belong to other species. The process is iterated by choosing the first not marked individual as a master for a new species and terminates when all the individuals in the current population have been analyzed. It is different from the standard fitness sharing in that each individual belongs to exactly one niche in the population, and only niche members other than the masters are penalized by equation (2-3), where d_{ij} involves the niche member *i* and its "neighbours" in the same niche.

2.2.3 Clearing

Clearing (CLR) [37] is an alternative to the sharing methods. Instead of sharing resources between all individuals of a niche as in the fitness sharing scheme, clearing attributes them only to the best members of each niche and eliminates the other individuals within the same niche. The remaining individuals in the mating pool generate offspring.

The clearing procedure firstly rearranges the population in descending order according to their fitness values. At the second step, it picks out one individual at a time from the top of the list until the niche capacity fills, and removes all the individuals worse than the selected ones within the specified clearing radius distance, defined either in genotype for binary parameter functions or in phenotype for real

valued functions. After that, it comes to the next individual on the list and repeats the second step. This continues until all the individuals in the population are examined. Figure 2.4 describes the clearing procedure. When the clearing procedure is finished, the algorithm will choose parents from the surviving individuals and create offspring to make up the pre-specified population size.

```
initialize mating pool with descendingly sorted population;
i=1;
while i <= size of mating pool,
 j=i+1;
 while j<= size of mating pool,
  calculate distance d between the i^th and the j^th member in
       the mating pool;
  if d < clearing radius
   remove the j^th member from the mating pool;
  end
  j=j+1;
 end
  i=i+1;
end
```

Figure 2.4 The Clearing Procedure

This clearing procedure illustrates the situation that only one member survives in every niche, i.e., the capacity of every niche is 1.

In addition to the methods listed above, there are many other niching methods. For example, sequential niching [40], [41] involves iterating the GA but avoids re-searching on the same peak of a fitness landscape by applying a fitness derating function to the raw fitness function so that fitness values are depressed in the regions of the problem space where solutions have already been found. Aiming at finding unevenly spread optima, the GAS (S stands for species) method [42] dynamically creates a subpopulation structure using a radius function instead of a single radius, and a "cooling" method similar to simulated annealing. Species conserving techniques [43], [44] are based on the concept of dividing the population into several species, each of which is built around a dominating individual called the species seed. The

algorithm protects promising solutions by means of conserving the species seeds. Hierarchical niching [45], [46], on the other hand, organizes the subpopulations into different fitness levels and maintains continuing search at all (absolute) fitness levels. The generation openings are filled by individuals imported from lower levels or generated by mutating other individuals of the same level. The emerging advanced methods include localized niching [47], [48] which is applied to the neighborhoods, sub-structural niching [49] which maintains diversity at the sub-structure level rather than at individual level, adaptive schemes [50]–[53], hybrid methods [54]–[57], and niching mechanism applied to combinatorial optimization problems [58].

2.3 Spatially Structured Evolutionary Algorithms

Closely related to the niching methods are the spatially structured evolutionary algorithms, in which members of a population evolve in relatively independent *demes* [59]. Structured populations are populations in which any given individual has its own neighborhood, which is smaller, sometimes much smaller, than the size of the population. The concept of a metric space and an associated distance is not always needed. More often, distances between individuals can be given by the network itself, as measured along the path that links the two individuals in a mathematical graph. Selection and mating are restricted to spatially close individuals only, i.e. only those that are in the same neighborhood can interact to produce offspring.

An SSEA is essentially an implementation of the parapatric speciation concept. It can be used alone or in conjunction with a particular niching method as well [48]. The

principal structures of populations that have been used in EAs include the island model and the cellular model.

2.3.1 The Island Model

The island model [60] is a popular and efficient way to implement a genetic algorithm on both serial and parallel machines. The basic idea is to divide a large population into several smaller ones. In a parallel implementation of an island model, each machine executes a standard genetic algorithm and maintains its own subpopulation for search. The subpopulations periodically exchange a portion of their members in a process called migration. Two parameters are involved in the island model. One is the migration interval, expressed by the number of generations (or evaluations) between migrations, and the other is the migration size, which is the number of individuals or the percentage to migrate.

The various islands maintain some degree of independence and thus explore different regions of the search space while at the same time sharing information by means of migration, which can be seen as a way to sustains the population diversity and improve the search quality [61].

2.3.2 The Cellular Model

In the cellular model, the individuals making up the population are disposed according to a regular lattice topology [62], [63]. This kind of spatially structured EAs features slow diffusion of solutions through the over-lapped small neighborhoods of

the population, leading to a more explorative behavior than conventional EAs, while exploitation takes place inside each neighborhood by genetic operations.

A canonical cellular evolutionary algorithm (cEA) starts with the cells in a random state and proceeds by successively updating them using evolutionary operators, until a termination condition is satisfied. Since the population is usually structured in a regular grid of d dimensions (d=1,2,3...) with a neighborhood defined on it, updating a cell in a cEA means selecting parents in the individual's neighborhood, applying genetic operators to them, and finally inserting the offspring into the equivalent places of the current population following a given replacement policy.

According to this canonical cEA, all the cells can be updated in parallel, yielding the so-named synchronous cEA. The alternative is the asynchronous cEA, in which the cells are updated one at a time in some sequential order.

Chapter 3

Evaluation of Niching Methods Using Hybrid Composition Functions

Many real-world global optimization problems require us to search for more than one good solution. After obtaining several good solutions, we can analyze them to choose the most favorable one associated with stable operating conditions. Stability of operating conditions can be assessed by identifying whether the local shape of the objective function is somewhat a plateau which implies that minor variations in the parameter values are unlikely to change the objective function value significantly and such kind of situation is preferred in practice.

Simple genetic algorithms often lack the ability to maintain a diverse population, which is essential to locating multiple optima, thus, niching methods are introduced. Researchers proposed various kinds of niching techniques, among which crowding and fitness sharing are the most common. In this chapter, genetic algorithms with popular niching methods are studied in the framework of hybrid composition functions. We take a closer look at three niching methods: deterministic crowding (DC), restricted tournament selection (RTS), and clearing (CLR) which have shown good performance in previous studies [64]–[66].

The performance of the three niching genetic algorithms is compared using a set of multimodal composition test functions originally proposed for the special session on real parameter optimization at the 2005 IEEE Congress on Evolutionary Computation (CEC 2005). The number of function evaluations is employed as the main control

parameter for an unbiased comparison instead of using the generation count as done frequently in the previous comparative studies. Results are given in tables and graphs to show the searching ability, accuracy, and computation time requirement of each method.

3.1 Test Functions

Current comparisons of niching methods are limited to a few test functions which share a low level of complexity. However, we are interested in the performance of niching genetic algorithms in more challenging environments. Moreover, from previous results, we cannot ensure that similar performance will generalize to more general and complicated problems.

Our test suite is composed of ten bivariate hybrid composition functions. Originally, they were used in the Special Session on Real-Parameter Optimization at the Congress on Evolutionary Computation 2005 for locating the global optimum only. Now we make use of their multimodality and high complexity to test the niching methods. The test functions are described one by one in this section. Mathematical formulas and properties of these functions can be found in [67]. Before we can incorporate these functions, some modifications are needed. Firstly, as we focus on the use of genetic algorithms, a linear transformation is done to change the minimization benchmark functions to maximization problems. In this way the objective function values can be directly used as fitness values, i.e. better individuals have higher function values. At the same time, a number is added to each existing function to ensure that most of the solution candidates have positive fitness values.

Secondly, since the functions have rugged landscapes with numerous local optima, we have to make our desired global optima stand out for the algorithms to follow. We set *bias*=0 in all the functions so that every basic function in the composition function offers a global optima with the same fitness value, and we can pursue the ten randomly generated global optima instead of just one global optimum in the original benchmark functions.

A. H1

H1 is generated from F_{15} (Hybrid Composition Function 1) in [67] by transforming the value $F(x)$ to $3000–F(x)$. It is a multimodal, scalable function with a huge number of local optima and it combines the properties of Rastrigin's Function, Weierstrass Function, Griewank's Function, Ackley's Function, and the Sphere Function. The variable domain is $[–5, 5]^2$. The global optima are located at (3.3253, –1.2835), (–2.2465, 3.9382), (1.7378, –4.4943), (–1.5504, –4.3339), (–2.7358, 0.4853), (–1.8717, 1.8285), (–3.9243, –2.7541), (–4.4362, –1.0142), (–1.4198, 3.6078), and (–2.4369, –2.3015).

B. H2

H2 is generated from F_{16} (Rotated Hybrid Composition Function 1) in [67] by transforming the value $F(x)$ to $3000–F(x)$. All settings are the same as H1 except the linear transformation matrices are with condition number 2 rather than identity matrices. The variable domain and the global optima are also the same as those of H1.

C. H3

H3 is generated from F_{17} (Rotated Hybrid Composition Function 1 with Noise in Fitness) in [67] by transforming the value $F(x)$ to $4000–F(x)$. Gaussian noise is added

to confuse the algorithms. The variable domain and the global optima are also the same as those of H1.

D. H4

H4 is generated from F_{18} (Rotated Hybrid Composition Function 2) in [67] by transforming the value $F(x)$ to $3500–F(x)$. It is a multimodal, scalable function with a large number of local optima and it combines the properties of Ackley's Function, Rastrigin's Function, the Sphere Function, Weierstrass Function, and Griewank's Function. The variable domain is $[–5, 5]^2$. The global optima are located at (1.5953, 2.644), (–0.4694, –0.9504), (1.5059, 1.4645), (2.8727, –0.8312), (4.4658, –3.9845), (–3.6435, –2.0476), (–1.1745, 3.0297), (0.1917, 1.5042), (3.3343, –0.873), and (0, 0).

E. H5

H5 is generated from F_{19} (Rotated Hybrid Composition Function 2 with a Narrow Basin for one of the Global Optima) in [67] by transforming the value $F(x)$ to $3000–F(x)$. All settings are the same as H4 except for the narrow basin for one of the global optima. The variable domain and the global optima are also the same as those of H4.

F. H6

H6 is generated from F_{20} (Rotated Hybrid Composition Function 2 with one of the Global Optima on the Bounds) in [67] by transforming the value $F(x)$ to $3000–F(x)$. All settings are the same as H4 except one of the global optima is set on the bounds. The variable domain is $[–5, 5]^2$. The global optima are located at (1.5953, 5), (–0.4694, –0.9504), (1.5059, 1.4645), (2.8727, –0.8312), (4.4658, –3.9845), (–3.6435, –2.0476), (–1.1745, 3.0297), (0.1917, 1.5042), (3.3343, –0.873), and (0, 0).

G. H7

H7 is generated from F_{21} (Rotated Hybrid Composition Function 3) in [67] by transforming the value $F(x)$ to $5000–F(x)$. It is a rotated, scalable, non-separable, and multimodal function with numerous local optima and it combines the properties of Rotated Expanded Scaffer's F6 Function, Rastrigin's Function, F8F2 Function, Weierstrass Function, and Griewank's Function. The variable domain is $[–5, 5]^2$. The global optima are located at (1.2141, –0.01), (4.0818, 1.5216), (0.6052, –3.9738), (–2.6639, –2.1802), (3.0023, 0.5371), (–3.7921, 1.9074), (–0.7006, 0.3496), (1.261, 4.0312), (–2.9174, –1.2577), and (–1.272, –1.2908).

H. H8

H8 is generated from F_{22} (Rotated Hybrid Composition Function 3 with High Condition Number Matrix) in [67] by transforming the value $F(x)$ to $10000–F(x)$. All settings are the same as H7 except the linear transformation matrix for each component function has a high condition number. The variable domain is $[–5, 5]^2$. The global optima are located at (1.2141, –0.01), (4.0818, 1.5216), (0.6052, –3.9738), (–2.6639, –2.1802), (3.0023, 0.5371), (–3.7921, 1.9074), (–0.7006, 0.3496), (1.261, 4.0312), (–2.9174, –1.2577), and (–1.272, –1.2908).

I. H9

H9 is also a variant of H7. This time we set five optima on the bounds as well as transforming the objective value $F(x)$ to $5000–F(x)$. The variable domain is $[–5, 5]^2$. The global optima are located at (1.2141, 5), (4.0818, 1.5216), (0.6052, 5), (–2.6639, –2.1802), (3.0023, –5), (–3.7921, 1.9074), (–0.7006, –5), (1.261, 4.0312), (–2.9174, 5), and (–1.272, –1.2908).

J. H10

H10 is generated from F_{24} (Rotated Hybrid Composition Function 4) in [67] by transforming the objective value $F(x)$ to $5000–F(x)$. It is a rotated, scalable, non-separable, and multimodal function with a great many local optima and it combines the properties of Weierstrass Function, Rotated Expanded Scaffer's F6 Function, F8F2 Function, Ackley's Function, Rastrigin's Function, Griewank's Function, Non-Continuous Expanded Scaffer's F6 Function, Non-Continuous Rastrigin's Function, High Conditioned Elliptic Function, and the Sphere Function with Noise in Fitness. The variable domain is $[–5, 5]^2$. The global optima are located at the points (–2.933, –3.034), (2.7566, 2.6439), (0.7973, –2.6471), (–3.358, 1.2959), (–2.128, –2.5922), (1.2224, 4.1029), (–4.3712, 1.7946), (1.2632, 2.2135), (–0.9932, 4.2521), and (2.2338, –1.5576).

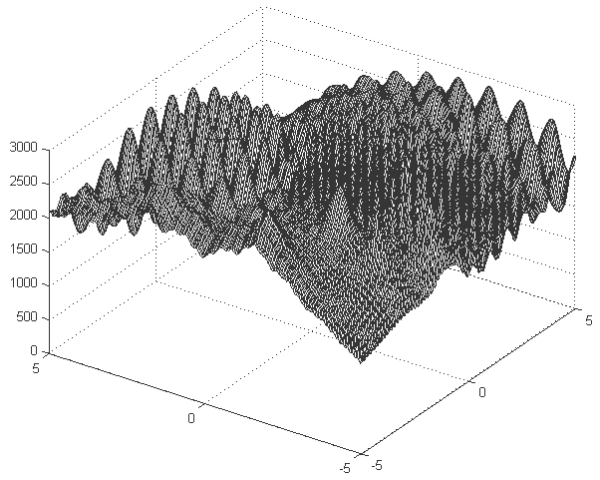

Figure 3.1 Landscape of H1

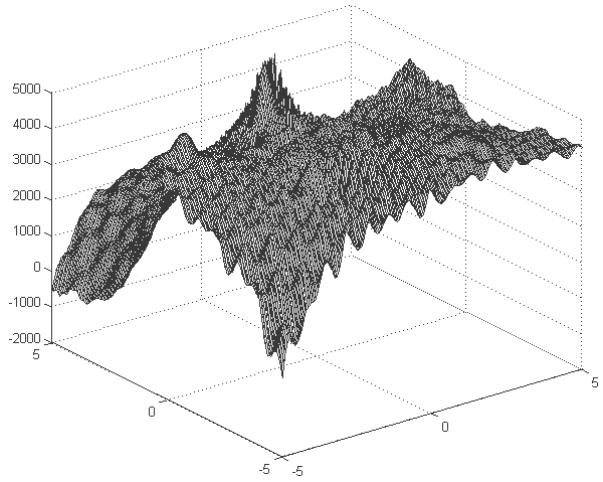

Figure 3.2 Landscape of H7

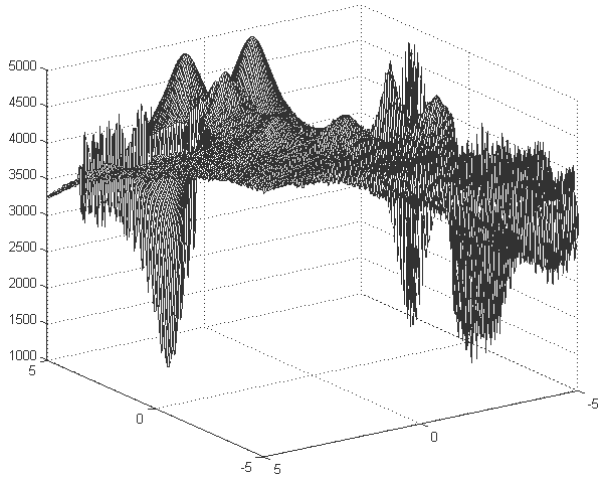

Figure 3.3 Landscape of H10

3.2 Evaluation Criteria

Most research concerning niching genetic algorithms examine their performance after a certain number of evolution generations or iterative loops. However, the number of offspring created and consequently the number of function evaluations consumed

during such a period may vary as niching mechanisms differ in each method. A method may be considered to be giving better results by creating more offspring and consuming more function evaluations than other methods. In this thesis, we compare the performance of niching methods on the basis of creating (consuming) the same number of offspring (objective function evaluations), which makes it easier to arrive at a fair judgment of the niching algorithms.

We use the number of function evaluations as the main control parameter. In this way, we hope that the results can be relevant for large scale, complex and computationally expensive real-world problems in which function evaluations take the most of the processing time. Specifically speaking, we set an equal number of 200,000 function evaluations to each of the three niching methods, and compare their performances in locating the ten global optima both at the end and in the possess of the evolutionary search.

We adopt the performance metric *success rate* (SR) from [68] to determine the ability of niching methods in locating the optima. Considering the variable range in the problems, an optimum is considered to be found if there exists a solution in the population within the Euclidean distance 0.01 to that optimum. The success rate is the number of optima found by the algorithm divided by the total number of optima.

Another metric we employ is the *maximum peak ratio* (MPR) originally defined in [31] and also used in [69]. Here we only include the optima obtained by the algorithm and the related actual optima in calculating the MPR defined as:

$$MPR = \frac{\sum_{i=1}^{q} \tilde{f}_i}{\sum_{i=1}^{q} \hat{F}_i} \qquad (3\text{-}1)$$

where q is the number of optima found, $\{\tilde{f}_i\}_{i=1}^{q}$ denotes the optima obtained by the algorithm and $\hat{F}_i$ is the corresponding actual optimum of $\tilde{f}_i$. The MPR indicates the algorithm's accuracy. In addition, the total time of completing a single run which includes 200,000 function evaluations is recorded.

3.3 Experimental Setup

We examine 25 independent runs of each niching method. Mean, median and standard deviation values of SR, MPR, and time taken (denoted as TT in tables) after 200,000 function evaluations are compared. In each run, we trace the change of SR and MPR every 100 function evaluations. We assign a uniform population size of 500 for every niching method in every test problem. Binary coding is used with single-point crossover and bit-wise mutation. The mutation rate is also the same at 0.05. For each variable dimension, a 17-bit binary string will be generated, which in turn gives a decimal precision of 10^{-4}.

Specific parameters for these niching methods include the window size w in RTS and the clearing radius σ in CLR. The former is fixed to 25 (or 5% of total individuals), and the latter, 0.4. These values were chosen as the best of 5 possible choices according to mean SR (see Table 3.1). The computers we use for simulation have configurations as shown in Table 3.2.

Table 3.1 Niching Options for RTS and CLR

Window Size in RTS	Clearing Radius in CLR
5	0.1
10	0.2
20	0.4
25	1
50	2

Table 3.2 Computer Configuration

CPU	P4, 3000 MHz
Memory Total	4GB
Operating System	Linux
Programming	Matlab 7.4 (R2007a)

3.4 Comparison of Results

Table 3.3–3.12 display the mean, median and standard deviation values of SR, MPR, and TT of the three niching methods, DC, RTS, and CLR. For every performance metric, the best average values obtained are highlighted. In terms of SR, two-sided, pair-wise *t*-tests are conducted for each test problem using the significance level of 0.05 to test if the results are statistically significant. The outcome of the *t*-tests is shown in Table 3.13. Among the 10 test problems, DC is better than RTS in 6 cases, and they are equal in the rest 4 cases. DC and CLR run close to each other according to the statistics. CLR has an advantage to RTS in 5 cases and they are equal in the rest

5 cases. There are 3 test problems in which no significant difference can been seen among the three methods. There are 2 cases (H7 and H9) in which the three methods have different performance to each other, with DC winning the first and CLR winning the second.

Table 3.3 Simulation Results – H1

		DC	RTS	CLR
SR	Mean	0.952	0.716	**0.968**
	Median	1	0.7	1
	StdDev	0.058595	0.085049	0.062716
MPR	Mean	0.999679	**0.999913**	0.999553
	Median	0.99974	0.99995	0.99956
	StdDev	0.000261	0.000102	0.000187
TT *(s)*	Mean	**1528.48**	1793.724	3028.004
	Median	1526	1793.7	3026.8
	StdDev	8.555846	11.23433	53.43477

Table 3.4 Simulation Results – H2

		DC	RTS	CLR
SR	Mean	0.824	0.664	**0.832**
	Median	0.8	0.7	0.8
	StdDev	0.092556	0.07	0.085245
MPR	Mean	0.998643	**0.99983**	0.998847
	Median	0.99854	0.99992	0.99889
	StdDev	0.000726	0.000237	0.000739
TT *(s)*	Mean	**1859.44**	2368.852	3485.38
	Median	1817.8	2404.5	3441.7
	StdDev	58.6198	128.0413	138.0013

Table 3.5 Simulation Results – H3

		DC	RTS	CLR
SR	Mean	**0.812**	0.656	0.8
	Median	0.8	0.7	0.8
	StdDev	0.088129	0.076811	0.11547
MPR	Mean	0.999132	**0.999821**	0.99879
	Median	0.99913	0.99989	0.99873
	StdDev	0.000496	0.000224	0.000666
TT	Mean	**1956.368**	2200.752	3424.812
	Median	2013.1	2234.4	3419.1
(s)	StdDev	113.405	94.3373	89.24367

Table 3.6 Simulation Results – H4

		DC	RTS	CLR
SR	Mean	**0.776**	0.684	0.716
	Median	0.8	0.7	0.7
	StdDev	0.066332	0.055377	0.102794
MPR	Mean	0.999593	**0.999727**	0.999244
	Median	0.9998	0.9998	0.99954
	StdDev	0.000376	0.000271	0.000955
TT	Mean	**1967.844**	2249.24	3424.892
	Median	2005	2309.5	3403.7
(s)	StdDev	137.0726	132.7032	101.4654

Table 3.7 Simulation Results – H5

		DC	RTS	CLR
SR	Mean	**0.684**	0.672	**0.684**
	Median	0.7	0.7	0.7
	StdDev	0.068799	0.045826	0.154596
MPR	Mean	**0.999694**	0.99968	0.996752
	Median	0.99996	0.99984	0.99825
	StdDev	0.000418	0.000282	0.004354
TT *(s)*	Mean	**1821.096**	2089.024	3475.232
	Median	1817.7	2086.1	3442.1
	StdDev	7.760986	7.92345	157.1549

Table 3.8 Simulation Results – H6

		DC	RTS	CLR
SR	Mean	**0.768**	0.752	0.744
	Median	0.8	0.8	0.7
	StdDev	0.09	0.07703	0.12936
MPR	Mean	**0.99952**	0.99922	0.99872
	Median	0.99974	0.99941	0.9992
	StdDev	0.00066	0.00084	0.00134
TT *(s)*	Mean	**2019.4**	2244.44	3651.59
	Median	2014.6	2265.7	3624.9
	StdDev	98.9929	104.539	134.861

Table 3.9 Simulation Results – H7

		DC	RTS	CLR
SR	Mean	**0.924**	0.788	0.856
	Median	0.9	0.8	0.9
	StdDev	0.059722	0.088129	0.09609
MPR	Mean	0.999578	**0.999819**	0.999324
	Median	0.99963	0.99989	0.99941
	StdDev	0.000179	0.000144	0.000697
TT	Mean	**1628.22**	1904.256	2843.144
	Median	1632.8	1885.7	2834.3
(s)	StdDev	29.27095	55.3006	21.65506

Table 3.10 Simulation Results – H8

		DC	RTS	CLR
SR	Mean	**0.604**	0.58	0.588
	Median	0.6	0.6	0.6
	StdDev	0.02	0.057735	0.072572
MPR	Mean	**0.999998**	0.99998	0.999956
	Median	1	1	0.99999
	StdDev	1.2E-05	3.49E-05	0.000131
TT	Mean	**1453.704**	1732.696	3081.796
	Median	1449.5	1730.7	3117.7
(s)	StdDev	13.47515	13.76364	144.7041

Table 3.11 Simulation Results – H9

		DC	RTS	CLR
SR	Mean	0.872	0.836	**0.928**
	Median	0.9	0.8	0.9
	StdDev	0.067823	0.06377	0.073711
MPR	Mean	0.999542	**0.999909**	0.99928
	Median	0.99954	0.99997	0.99942
	StdDev	0.000308	0.00018	0.000688
TT *(s)*	Mean	**1466.396**	1955.388	3046.512
	Median	1464.7	1967.5	3045.5
	StdDev	14.29567	70.24106	148.4629

Table 3.12 Simulation Results – H10

		DC	RTS	CLR
SR	Mean	0.604	0.628	**0.652**
	Median	0.6	0.6	0.7
	StdDev	0.045461	0.09363	0.096264
MPR	Mean	0.99988	**0.999987**	0.999554
	Median	1	1	0.99975
	StdDev	0.000397	3.02E-05	0.000554
TT *(s)*	Mean	**1617.252**	1932.308	3248.192
	Median	1614.1	1911.5	3222.8
	StdDev	55.42285	60.00345	92.45776

Table 3.13 Statistical Test Results

	DC vs. RTS	DC vs. CLR	RTS vs. CLR
H1	1	0	-1
H2	1	0	-1
H3	1	0	-1
H4	1	1	0
H5	0	0	0
H6	0	0	0
H7	1	1	-1
H8	0	0	0
H9	1	-1	-1
H10	0	-1	0

Meaning of values: 0 – there is no difference between the two methods; 1 – the former is better; -1 – the latter is better.

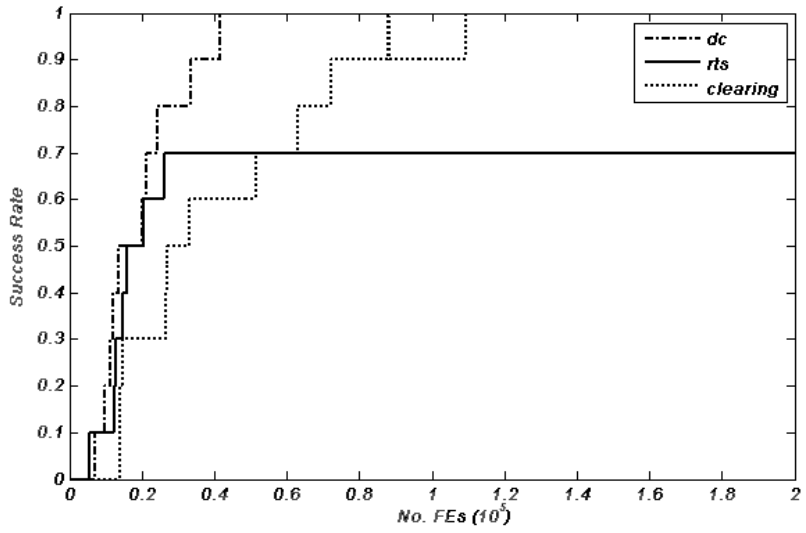

Figure 3.4 SR vs. No. of FEs – H1

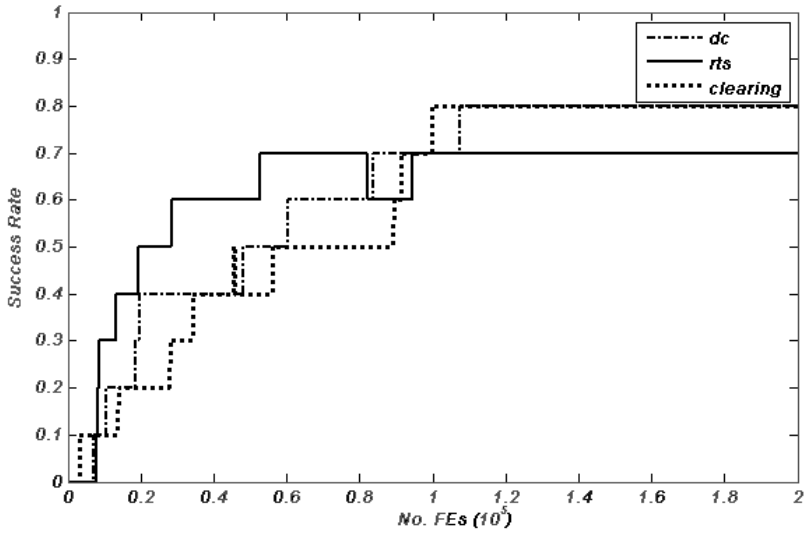

Figure 3.5 SR vs. No. of FEs – H2

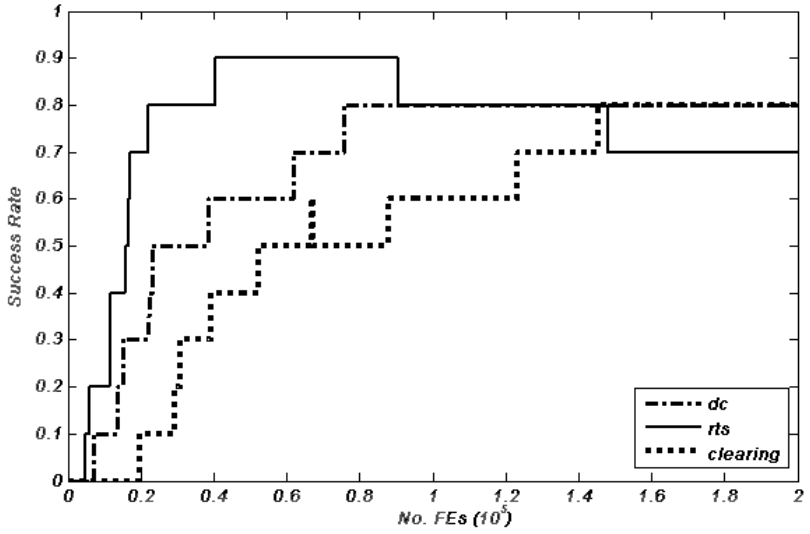

Figure 3.6 SR vs. No. of FEs – H3

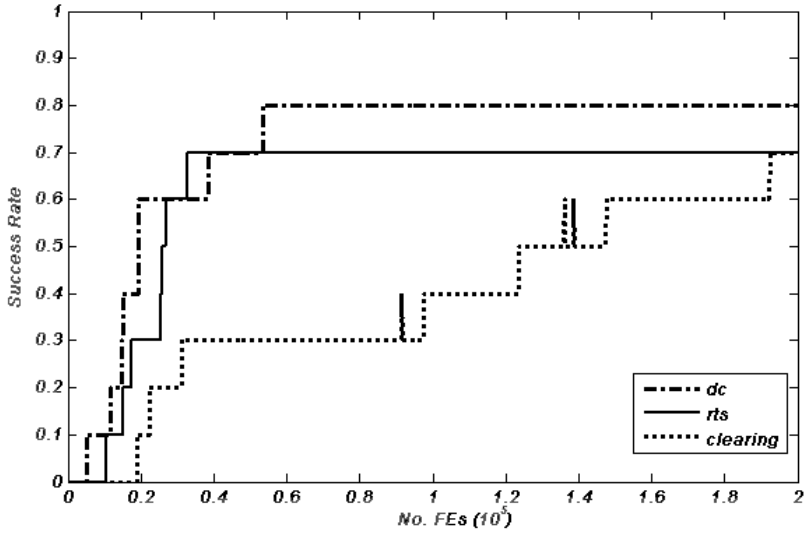

Figure 3.7 SR vs. No. of FEs – H4

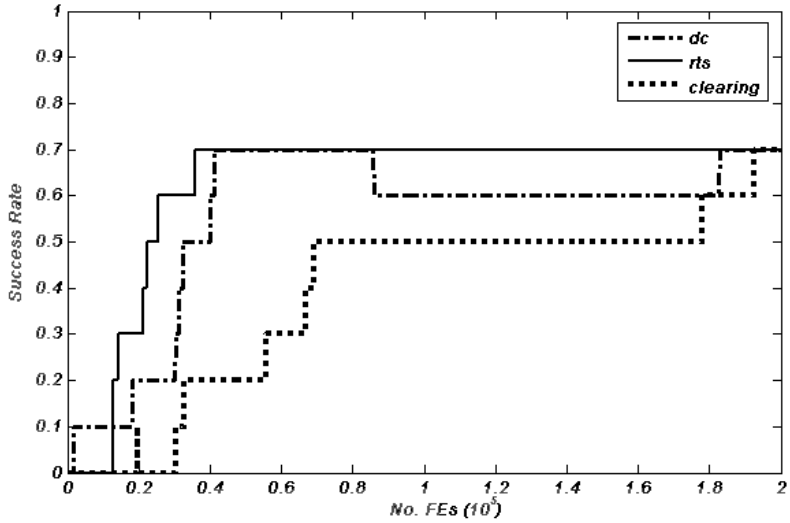

Figure 3.8 SR vs. No. of FEs – H5

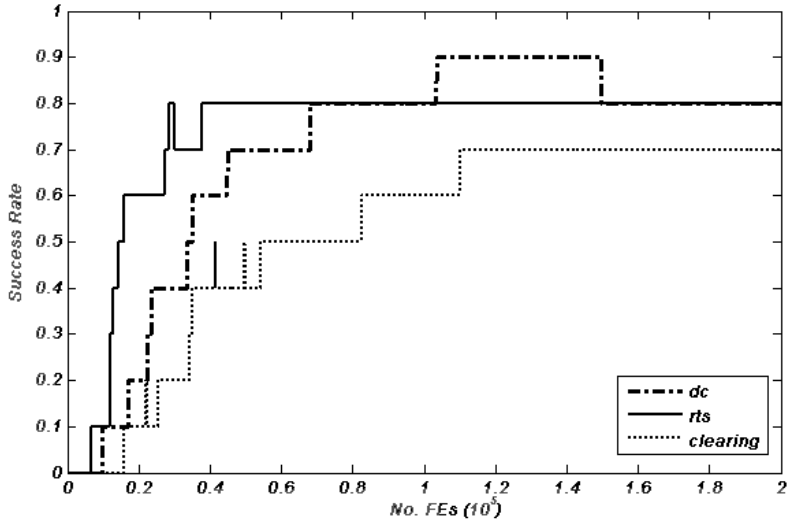

Figure 3.9 SR vs. No. of FEs – H6

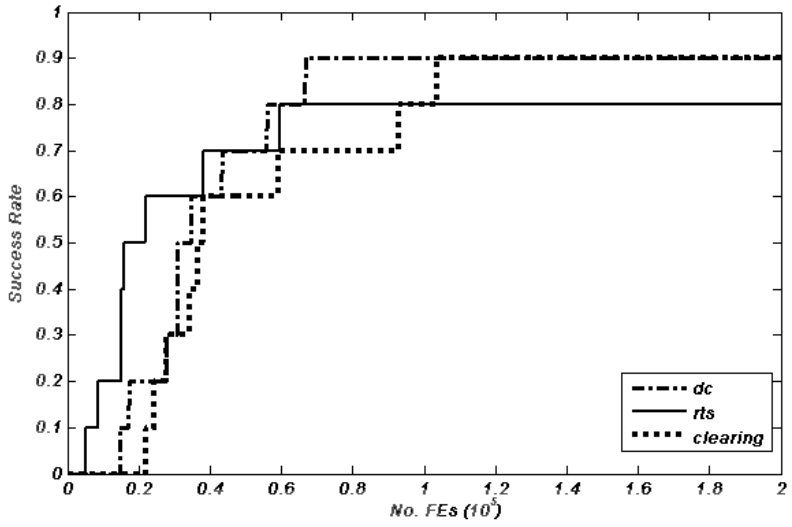

Figure 3.10 SR vs. No. of FEs – H7

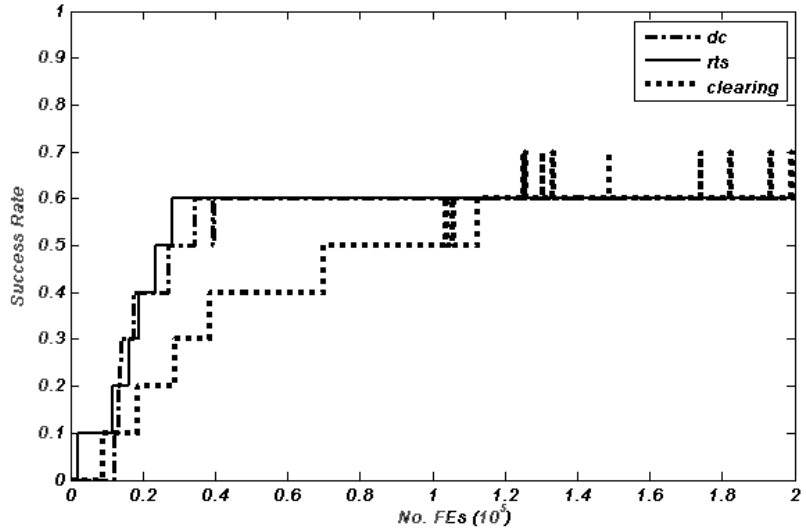

Figure 3.11 SR vs. No. of FEs – H8

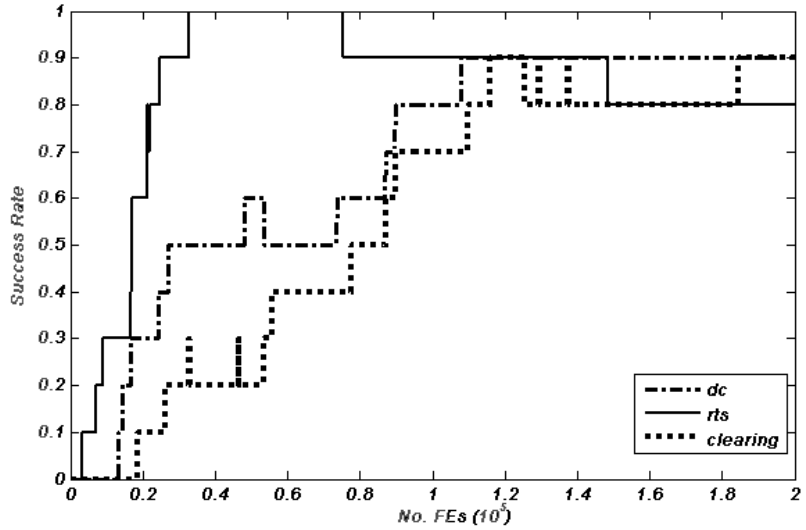

Figure 3.12 SR vs. No. of FEs – H9

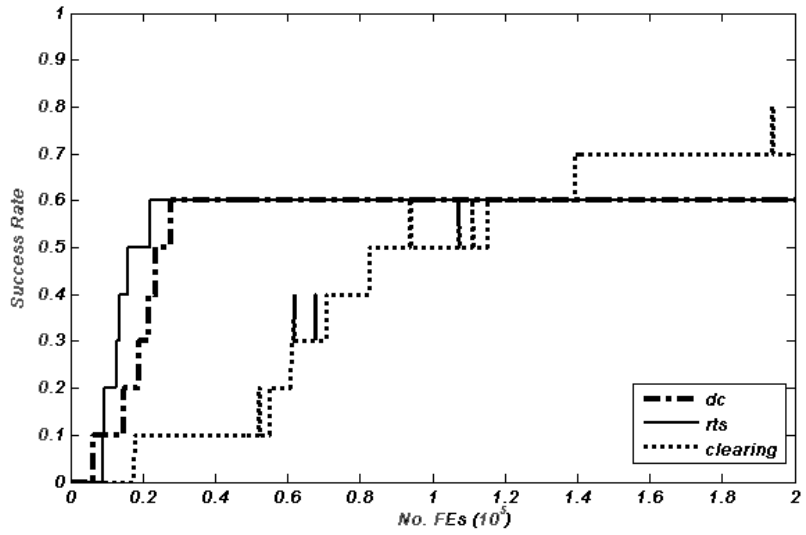

Figure 3.13 SR vs. No. of FEs – H10

It can be observed that the success rate may occasionally go down in the performance graphs. That represents the situation when a solution appears within the tolerance we define in Section 3.2 in a niche and is then replaced by another individual from another already identified niche with a higher fitness. The RTS suffers significantly from this problem as can be seen in Figure 3.12.

3.5 Discussion

From the tables and figures, we can see that all three niching methods have had a chance to make the best performance in one of the test problems. With respect to the success rate, restricted tournament selection has a faster convergence compared to the other two methods. Clearing maintains a gradual growth in success rate and in H10 it outperforms deterministic crowding and restricted tournament selection by finding more global optima within 200,000 function evaluations.

The maximum peak ratio is very close to 1 in all cases, which partially results from the small tolerance value of 0.01. The two crowding methods, deterministic crowding and restricted tournament selection win narrowly against clearing in terms of MPR.

If we consider the computation time, DC is the fastest. RTS is approximately requires 20% more time while the clearing requires more than 100% additional time when compared with the RTS. That is due to the computational cost of each algorithm. DC only compares offspring with their parents, regardless of the population size, so the computational cost of DC is competitive when the population is large. RTS has a computational cost of $O(wN)$, with w equalling the window size and N equalling the population size. In clearing, the number of distance comparisons is approximately $O(mN)$, where m is the number of niches. However, for a specific test problem, the number of niches in clearing is not known in advance, and it can be large if the clearing radius is chosen relatively small. Therefore, taking into consideration the success rate, MPR and the computation time (TT), we can conclude that the RTS and DC have an advantage over clearing on this problem set.

Further investigation involving higher-dimensional problems is needed to characterize the performance of niching methods in a more explicit manner. Another aspect to investigate would be to develop methods to retain all identified niches instead of dropping them subsequently as the RTS was suffering in Figure 3.12.

Chapter 4

Ensemble of Niching Algorithms

Although niching algorithms have been investigated for almost four decades as effective procedures to obtain several good and diverse solutions of an optimization problem, no effort has been reported on combining different niching algorithms to form an effective ensemble of niching algorithms. In this chapter, an *ensemble of niching algorithms* (ENA) is proposed. The concept is illustrated by two instantiations of ENAs each of which is realized using four different parallel populations. The offspring of each population is considered by all parallel populations. The instantiations of the ENAs are tested on a set of 16 real and binary problems and compared against the single niching methods with respect to searching ability and computation time. Results confirm that the proposed method is as good as or better than the best single method in the ENA on every test problem. Moreover, comparison with four state-of-the-art niching algorithms demonstrates the competitiveness of our proposed ENA.

4.1 General Considerations

As mentioned in Chapter 3, in real-world optimization problems, sometimes we are not satisfied with only one optimal solution. The demand for multiple solutions is more prominent when there exist several near optimal answers to a problem. Niching genetic algorithms are used to solve these problems as they attempt to maintain a

diverse population and are not as prone to converge prematurely as simple genetic algorithms.

Over the last few decades, various niching methods have been proposed. However, so far, to solve a certain problem, only one method is used at a time. It is often the case that we have to try out several niching methods and tune their parameters to find the best method and its best set of parameters to solve a given problem. Irrespective of the exhaustiveness of the parameter tuning, no one method can be the best for all problems. The various niching methods divide the population into niches where the same selection and survival criteria are implemented. Instead of that, we can establish several populations, each of which can employ a distinct niching algorithm. As algorithm performance is in direct proportion to the algorithms design for the specific problem, a mixture of niching methods will ensure acceptable average performance on a broad variety of problems. The existing niching methods offer us a wide choice of combinations in this scenario.

Given a certain problem, we are unable to predict which niching method will be the best to solve it as different methods have different characteristics. Some methods may be good at climbing the peaks of the optima after the basins of attractions are located. Other methods may good at exploring the search space. Hence, it is natural to consider using an ensemble of niching algorithms and parameters in order to benefit from the strengths of each algorithm. Apart from this, interaction rules can be set so that all the algorithms can always learn from others to improve their weaknesses.

4.2 Construction of an Ensemble

Here we introduce the ensemble of niching algorithms. It combines several existing niching methods into a unified framework for searching for multiple solutions within a multimodal fitness landscape. It is realized by using a separate population for each niching method. The offspring created by every population are considered by all other populations for inclusion in their populations according to the selection rules of the respective populations.

4.2.1 Parallel Populations vs. Sub-populations

In the ENA, we use parallel populations. In other words, every niching instance has its own population. The advantages of sub-populations have been discussed in many studies. For example, in the island model [60], sub-populations of genetic algorithms are formed by the speciation tree. Each GA is usually identical in its genetic operators and according to the migration rules information is exchanged between the sub-populations. The sub-population measure prevents the algorithm from converging to a single point of the search space. Unlike the island model, ENA has totally different niching methods and each of the niching method has its own population. As island models were developed primarily to maintain diversity, not every offspring (or function evaluation) is considered by every sub-population. In the ENA, every offspring is considered by every niching algorithm for possible inclusion in its own population according to its own selection rules. Usage of ensemble of different niching algorithms simultaneously has never been realized by island model or any other hybridization approaches.

4.2.2 Selecting Individual Algorithms

Now we come to a practical question. How do we choose the individual niching methods in an ENA? As is mentioned in Section 4.1, niching algorithms have their own features, so there is hardly a universally best choice. In this sense, the selected individual niching algorithms should be competitive and complementary to each other.

Observing the categories of niching methods, we can choose several methods from different categories or several instances of a method with distinct parameter settings. In this chapter, two instances of ENA are presented. ENA1 uses four separate populations of two instantiations of the RTS (restricted tournament selection) and two instantiations of the CLR (clearing) algorithms. ENA2 is formed by two instantiations of the RTS and two instantiations of the DFS (dynamic fitness sharing) algorithms.

As both RTS and CLR have shown good performance in previous studies [64]–[66], we select them for the first instantiation of ENA. More importantly, these methods differ in realizing the niching concepts, which thereby makes them more likely to follow distinct evolution paths. The second instantiation is decided similarly, which combines RTS and another efficient niching method, DFS, from the sharing category.

As is common in the crowding methods, the “niche” concept is rather implicit in RTS. There are no niche radii as in fitness sharing, clustering, and clearing. Competition is held among similar individuals. CLR keeps only the best individuals in every niche and eliminates the others, resulting in a fast evolution pace. DFS is moderate when compared with the CLR. It balances the selection pressure of the population by imposing penalty function on its members. Individuals in dense niches will get more

punishment in fitness; however, the weaker individuals are not eliminated directly. In DFS, fitness values play a role in the selection procedure, whereas in RTS and CLR, what matters is just the relative order of the fitness, not the exact values. Thus, whether the slopes of the peaks are steep or shallow have a greater impact on DFS. Although we demonstrate the ENA using the above configurations, the idea of ensemble can be generally applied to other combinations of niching algorithms. It is absolutely possible to form other effective ENAs using other configurations.

4.2.3 Modifications to Existing Niching Methods

Several modifications are made in the selected niching methods to enhance their competitiveness and promote reliable performance. We observe that the original CLR method generates a variable numbers of offspring in each generation. When the clearing radius is chosen too large, the majority of the population will be eliminated in the clearing process, which means many offspring should be generated from the limited number of parents. In this way, we cannot guarantee the quality of offspring. On the other hand, when the clearing radius is chosen too small, few population members will be eliminated, which yields very slow evolution pace. Unfortunately, in most cases, we are not informed of the correct choice of the clearing radius to ensure that a relatively stable number of offspring being generated. Motivated by such considerations, we introduce the use of a variable population size so that fixed number of offspring can be guaranteed. The procedures of our modified CLR is basically the same as the original CLR method, but in every generation we update the population size to be equal to the remaining number of individuals after clearing together with the fixed number of offspring generated in each generation.

In the DFS method, we adopt equation (2-5) to replace equation (2-3) for the penalty of fitness values. As equation (2-5) only calculates the individual's distance from its niche center (or species master, in the context of the DFS method), it should lessen some computation time in terms of distance calculations and improve the efficiency of the method. Inspired by [70], we use stochastic universal sampling (SUS) instead of roulette wheel to further reduce bias in the selection phase.

4.2.4 Interactions between the Populations

Another part that makes ENA different from the previous algorithms is the interaction scheme between the populations. Instead of randomly picking a fixed proportion of the population as migrants as is done in the island model, ENA permits every newly created individual to be considered by all the populations. All offspring created in the current generation will be stored in an array, with an index vector indicating which population created them. Whether an offspring generated by population A can enter population B is determined by the population updating strategy used in population B.

For example, in ENA1, which is composed of RTS1, RTS2, CLR1, and CLR2, we gather offspring from RTS2, CLR1, and CLR2, and let them go through the tournament replacement process in RTS1. If a certain offspring i wins over the nearest member j in the randomly selected w_1 individuals from RTS1, then i will replace j. Note that the order of creating new individuals and replacement is different in the RTS populations and the CLR populations. In the CLR populations, some individuals are cleared first to make room for the new comers. The new individuals are definitely kept, at least in the current population. Hence, the approach for inserting offspring from other populations into the CLR populations differs from that in the RTS

populations. The offspring will be added into the mating pool in the next generation and the clearing process decides whether the offspring from other population survive or not. In the DFS populations of ENA2 also offspring from other populations are integrated in the next generation. The sharing fitness is calculated for each member of the DFS population and every offspring from other populations. Then the roulette wheel selection is performed to select parents.

Figure 4.1 shows a possible migrating route of an offspring in the ENA. The offspring *A* is rejected by its own population in generation *i*, but is accepted by population #3, and is selected as parent to create offspring *A'* in generation *i*+1. Then *A'* is accepted by populations #2 and #3 as parent in the next generation.

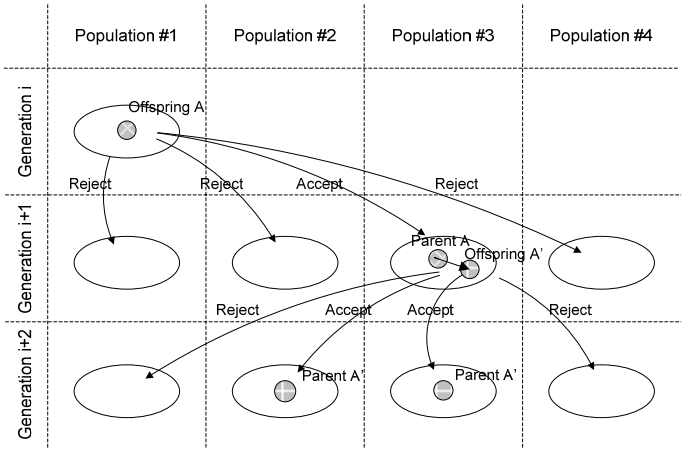

Figure 4.1 Interactions between the Populations

Having passed the updating rules, an offspring that is successfully accepted by a population is either of high fitness or far from the existing peaks. In the former case, the offspring helps to climb the peak. In the latter case, it unveils a new search area. Even though an offspring requires only one fitness function evaluation, it has the opportunity to take part in the evolution process in every population. Therefore, we attempt to benefit the most from each function evaluation.

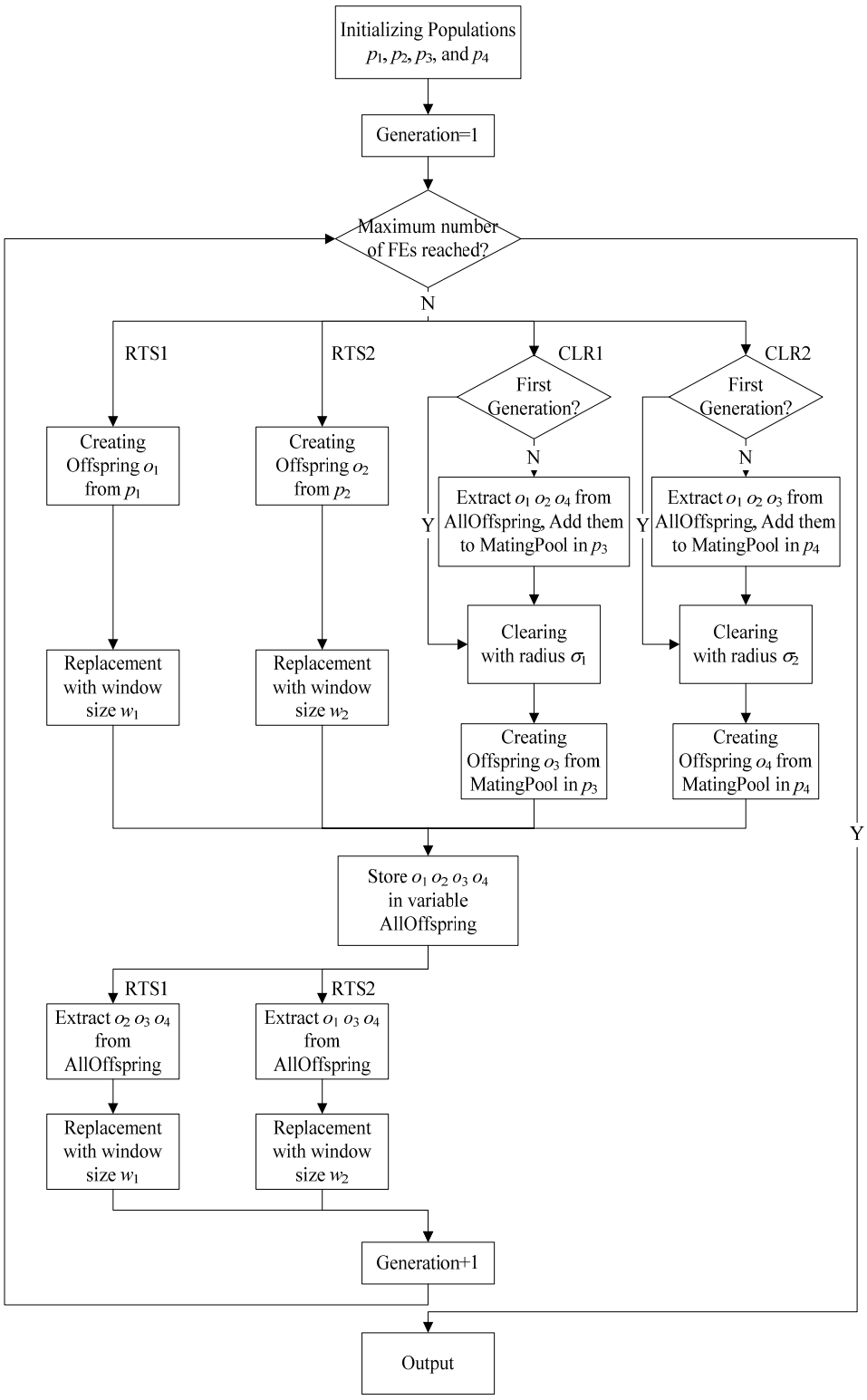

Figure 4.2 Flowchart of the Ensemble of Niching Algorithm (ENA1)

4.2.5 The Overall Procedure

During the initialization stage, the populations are randomly and separately generated in the search space for all niching methods. After that, the populations go through the niching process and the genetic operations. For instance in ENA1, random selection, crossover, mutation and restrict tournament replacement are conducted in the RTS populations. In the CLR populations, all individuals form a mating pool and undergo a clearing process. The CLR algorithm produces new individuals through crossover and mutation operations to fill the vacancies after clearing.

The ENA then stores all the offspring created in the current generation and implement the interaction scheme described above. This process completes one generation and this process is iterated until the maximum number of function evaluations (FEs) is reached. The flowchart of ENA1 is presented in Figure 4.2. The procedures of ENA2 are also similar except that it has RTS1, RTS2, DFS1 and DFS2.

4.3 Test Functions

The test suite consists of 16 problems. The first ten test functions of the suite are real-variable functions and the others are binary-variable functions. Many of them have been widely used in previous studies. All the functions are multimodal with 5 to 32 desired optima. F1 to F6 are either 1-dimensional or 2-dimensional, and F7 to F10 are higher-dimensional problems. The binary test problems typically have a length of around 30 bits.

Table 4.1 Summary of Test Functions

Function	Number of	Variable	Total Bit Length*
F1	5	Real (1-D)	14
F2	4	Real (2-D)	34
F3	25	Real (2-D)	42
F4	10	Real (2-D)	28
F5	20	Real (2-D)	28
F6	18	Real (2-D)	36
F7	20	Real (3-D)	51
F8	12	Real (4-D)	68
F9	32	Real (5-D)	80
F10	10	Real (10-D)	140
F11	32	Binary	30
F12	32	Binary	30
F13	27	Binary	24
F14	32	Binary	30
F15	32	Binary	30
F16	32	Binary	40

* As the niching methods are implemented using the standard genetic algorithm, binary coding is used to translate the variables into bit strings if the test function is in real domain.

F1 is a 1-dimensional function with unequally spaced peaks of non-uniform height. It is one of the most commonly used test functions proposed by Goldberg and Richardson [29] and investigated by Deb [71], Deb and Goldberg [72], and many other researchers [73], [40]. F2 is the modified *Himmelblau*'s function in Deb's study [71]. F3 is the *Shekel's Foxholes* function used in De Jong's dissertation [22]. F4 and F5 are two instances of *Hump* functions [66] with optima that can be randomly generated or defined by users. F6 is *Shubert* function [43] with a linear transformation. F7 and F8 are 3-dimensional and 4-dimensional peak functions [74]. In addition to

these functions, we construct the 5-dimensional and 10 dimensional *product* functions F9 and F10, to test the performance of niching algorithms in a non-separable, high-dimensional environment.

The binary-variable functions F11-F13 are massively multimodal and deceptive functions used in various studies [75], [76], [24], [37], [48]. The global optima are isolated, and at the same time there are a huge number of local optima. They can be hard problems for niching methods as most points in the search space are placed on a misleading path towards a local optimum only. We construct three more binary-variable functions F14-F16 with similar properties. A summary of the properties of these functions is shown in Table 4.1. Mathematical definitions and peak locations are as follows. All functions are maximization problems.

F_1: $F_1(x) = \exp(-2(\ln 2)(\frac{x-0.08}{0.854})^2)\sin^6(5\pi(x^{\frac{3}{4}}-0.05))$. The variable domain is [0, 1]. The optima are located at the x values of 0.0797, 0.2463, 0.4495, 0.6791, and 0.9302.

F_2: $F_2(x_1,x_2) = \frac{2186-(x_1^2+x_2-11)^2-(x_1+x_2^2-7)^2}{2186}$. The variable domain is $[-6, 6]^2$. The four global optima are located at (3, 2), (3.5844, –1.8481), (–3.7793, –3.2832), and (–2.8051, 3.1313), of the same height of 1.

F_3: $F_3(x_1,x_2) = 500 - \frac{1}{0.002+\sum_{i=0}^{24}\frac{1}{1+i+(x_1-a(i))^6+(x_2-b(i))^6}}$, where $a(i) = 16(\mathrm{mod}(i,5)-2)$, $b(i) = 16(\lfloor i/5 \rfloor - 2)$. Both variables are restricted to the range [–65.536 65.535], and the optima are located at (16i, 16j), where i and j can take any of the integers from [–2, 2] including zero.

F_4 and F_5: Let N be the total number of peaks, $\{\mathbf{x}_k\}_{k=1}^N$ be the set of peaks, $\{h_k\}_{k=1}^N$ be

the respective heights, $\{r_k\}_{k=1}^N$ be the radii of the humps, and $\{\alpha_k\}_{k=1}^N$ be the parameters with respect to the steepness, then $F_{4,5}(\mathbf{x}) = \max\{f_k(\mathbf{x}) \mid k = 1,2,\cdots,N\}$, where

$$f_k(\mathbf{x}) = \begin{cases} h_k(1 - (\frac{\|\mathbf{x} - \mathbf{x}_k\|}{r_k})^{\alpha_k}), & \text{if } \|\mathbf{x} - \mathbf{x}_k\| \le r_k \\ 0, & \text{otherwise} \end{cases}.$$

For F4, $N = 10$, $\{h_k\}_{k=1}^N = 1$, $\{r_k\}_{k=1}^N$ are 0.01, 0.03, 0.01, 0.01, 0.004, 0.01, 0.02, 0.006, 0.01, and 0.008 respectively, and $\{\alpha_k\}_{k=1}^N$ are 1, 0.5, 0.25, 0.25, 1, 0.5, 1, 2, 0.5, and 3 respectively. The variable domain is $[0, 1]^2$. The optima are located at $\{\mathbf{x}_k\}_{k=1}^N$, which are randomly initialized within $[0, 1]^2$.

For F5, $N = 20$, $\{h_k\}_{k=1}^N = 1$, $\{r_k\}_{k=1}^N = 0.05$, and $\{\alpha_k\}_{k=1}^N$ are 1, 1, 1, 1, 1, 1, 1, 1, 1, 0.75, 1.5, 1, 1, 1, 2, 1, 1, 1, 0.75, and 0.25 respectively. The optima are located at $\{\mathbf{x}_k\}_{k=1}^N$, which can be randomly initialized within the variable domain $[0, 1]^2$. To add the difficulty of this function, we make several optimal points close to each other while others are farther away.

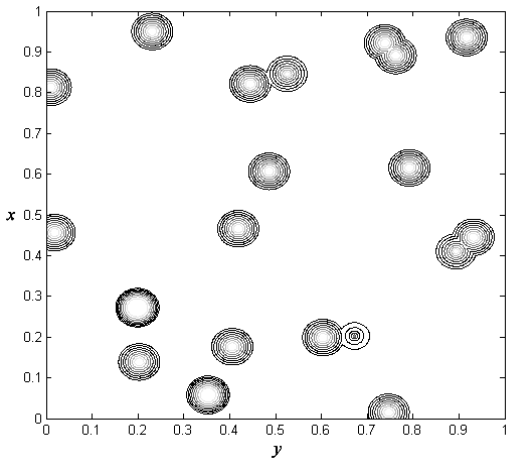

Figure 4.3 Contour Map of F5

F_6: $F_6(x_1, x_2) = 250 - (\sum_{i=1}^{5} i\cos((i+1)x_1 + i))(\sum_{j=1}^{5} j\cos((j+1)x_2 + j))$. The variable domain is $[-10, 10]^2$. A total 18 global optima are located at the points (–7.7083, –7.0835), (–7.0835, –7.7083), (–1.4251, –7.0835), (–0.8003, –7.7083), (4.8581, –7.0835), (5.4829, –7.7083), (–7.7083, –0.8003), (–7.0835, –1.4251), (–1.4251, –0.8003), (–0.8003, –1.4251), (4.8581, –0.8003), (5.4829, –1.4251), (–7.7083, 5.4829), (–7.0835, 4.8581), (–1.4251, 5.4829), (–0.8003, 4.8581), (4.8581, 5.4829), and (5.4829, 4.8581) respectively.

F_7 and F_8: $F_{7,8}(\mathbf{x}) = \max_{i=1}^{m}(H_i/(1 + W_i \cdot \sqrt{\frac{\|\mathbf{x} - \mathbf{x}_i^*\|^2}{n}}))$, where m is the number of peaks, n is the number of dimensions, **H**, **W**, and $\mathbf{x}^*$ denote the peak height, width and position respectively. For F7, n=3, m=20, **H**=20, **W**=2. The variable domain is $[-5, 5]^3$. The optima are located at $\mathbf{x}^*$, which are randomly initialized within $[-5, 5]^3$. For F8, n=4, m=12, **H**=30, **W**=1. The optima are located at $\mathbf{x}^*$, which are randomly initialized within the variable domain $[-5, 5]^4$.

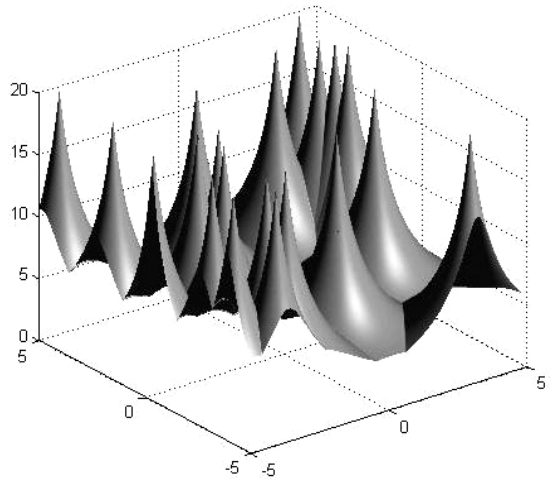

Figure 4.4 Landscape of F7 illustrated in 3-D Space

F_9: $F_9(\mathbf{x}) = \prod_{i=1}^{5} \min(\tilde{x}_i^2, (\tilde{x}_i - 2)^2, (\tilde{x}_i + 2)^2)$, where $\tilde{\mathbf{x}} = \mathbf{x} \cdot A$, and

$A = \begin{pmatrix} \cos(\pi/6) & 0 & 0 & -\sin(\pi/6) & 0 \\ 0 & \cos(\pi/4) & 0 & 0 & \sin(\pi/4) \\ 0 & 0 & 1 & 0 & 0 \\ \sin(\pi/6) & 0 & 0 & \cos(\pi/6) & 0 \\ 0 & -\sin(\pi/4) & 0 & 0 & \cos(\pi/4) \end{pmatrix}$. The variable domain of this function is $[-2, 2]^5$. The optima are located at $\mathbf{x}^* \cdot A^{-1}$, where every dimension of $\mathbf{x}^*$ may have the value 1 or –1.

F_{10}: $F_{10}(\mathbf{x}) = \max_{i=1}^{10}((\prod_{j=1}^{10}(1-x_j)) \cdot (1-2|x_i - 0.5|)/(1-x_i))$. The variable domain is $[0, 1]^{10}$. The 10 optima are located at $\mathbf{x}^*$, where all dimensions take zero value except that one dimension equals 0.5.

F_{11}: In F11, the variables are 30 binary bits and the fitness value is the sum of five unitation functions $u(x)$, each of which is operated on 6 consecutive bits of the string.

$u(x) = \begin{cases} 1, & x = 0 \text{ or } x = 6 \\ 0, & x = 1 \text{ or } x = 5 \\ 0.360384, & x = 2 \text{ or } x = 4 \\ 0.640576, & x = 3 \end{cases}$, where x is the number of 1's in the substring. The optima of F11 are located at bit strings that have the form $\underbrace{c_1 c_1 \cdots c_1}_{6\,c_1\text{'s}} \underbrace{c_2 c_2 \cdots c_2}_{6\,c_2\text{'s}} \underbrace{c_3 c_3 \cdots c_3}_{6\,c_3\text{'s}} \underbrace{c_4 c_4 \cdots c_4}_{6\,c_4\text{'s}} \underbrace{c_5 c_5 \cdots c_5}_{6\,c_5\text{'s}}$, where $\{c_i\}_{i=1}^5$ are either 0 or 1, and the optimal fitness is 5.

F_{12}: $F_{12} = 5(\frac{F_{11}}{5})^{15}$. The optima and optimal fitness values are as in F11.

F_{13}: In F13, the variable length is 24 bits. The overall fitness is the sum of the fitness values acquired from the three 8-bit substrings. For substrings, a top fitness 10 is assigned to global substrings 00000000, 10001100, and 01001010, while others get a fitness value that equals to the Hamming distance from its nearest global substring.

F_{14}: F14 is a variant of F13. It has a string length of 30 bits. Every 6 bits form a substring. The global strings (as described in F13) are 010011 and 100001, which receive fitness value 6. There are 32 global optima, with the fitness value of 30.

F_{15}: In F15, the variable is a 30-bit binary string and the fitness value is the sum of five unitation functions $u(x)$, each of which is operated on 6 bits of the string. $u(x) = \begin{cases} 1 - 0.2x, & x \le 5 \\ 6, & x > 5 \end{cases}$, where x is the number of 1's in the substring. The optima are located at $\underbrace{c_1 c_1 \cdots c_1}_{6\,c_1\text{'s}} \underbrace{c_2 c_2 \cdots c_2}_{6\,c_2\text{'s}} \underbrace{c_3 c_3 \cdots c_3}_{6\,c_3\text{'s}} \underbrace{c_4 c_4 \cdots c_4}_{6\,c_4\text{'s}} \underbrace{c_5 c_5 \cdots c_5}_{6\,c_5\text{'s}}$, where $\{c_i\}_{i=1}^{5}$ are either 0 or 1, and the optimal fitness is 5.

F_{16}: In F16, the variable string has the length of 40 bits and is divided into five 8-bit substrings. The highest fitness, 1, is assigned to the global substrings 10100000 and 01111110. When the substring is other than 10100000 and 01111110, its fitness is calculated using the expression $u(x) = \begin{cases} 0.6, & x = 0,4,8 \\ 0.3, & x = 1,3,5,7 \\ 0, & x = 2,6 \end{cases}$, where x is the number of 1's in the substring. Optimal strings are the concatenations of 10100000 or 01111110.

4.4 Parameterization and Evaluation Criteria

In the experiments, we aim to show that the conceptual advantages of ENA translate into practical advantages in performance compared with single niching algorithms. For this purpose, we examine the searching ability and the computation cost of both the ensemble methods and their constituting niching methods on the basis of using the same number of function evaluations. For functions F1-F6 and F11-F16, we set the

total number of function evaluations to be 50,000. For higher dimensional F7-F10, the number of function evaluations is 200,000.

The initial population size for each of the four constituting niching methods in both ENA1 and ENA2 is 500 for F1-F6 and F11-F16, and 1000 for F7-F10. For the sake of fair comparisons, we test the single niching methods with an initial population of 2000 in problems F1-F6 and problems F11-F16, and an initial population of 4000 in problems F7-F10. In every generation, the same number of offspring is generated in each of the populations in the ensemble, and the total number of offspring generated in every generation in the ensembles remains equal to that in the four single constituting methods.

Two instances of every niching method are used in the ensembles with different parameter values. For RTS, the niching parameter is the window size in tournament selection. Ideally it should be comparable to the number of optima we want to identify. In practice, a larger value is used in case some incorrect peaks compete with the desirable ones. We set it to 25 and 100 for the two instances respectively. It does not differ in test functions. In CLR and DFS, the niching parameter is the clearing or sharing radius. We choose the two instances of parameters to be 1% of the sum of variable ranges in every dimension and 5% of the sum of variable ranges in every dimension for real-variable problems. For binary-variable problems the parameters are 2 and 3 bits. Binary coding is used to the decimal precision of 10^{-4} for real parameter problems. We use one-point crossover and bit-wise mutation. The mutation rate is 0.05 for RTS, 0.02 for CLR, and 0.01 for DFS. A translation of fitness by adding 0.1 is always done for DFS, as the selection phase requires the fitness be positive. It should be noted that ENA does not have extra parameters in addition to

those used in the constituting niching methods. The above parameters are used for all test problems.

We compare the two ensemble algorithms with the single methods after consuming the total number of function evaluations. We examine 30 independent runs of each niching method. *Success rate* (SR) [68] is adopted to evaluate the searching ability of niching methods in locating the optima. An optimum is considered to be found if there exists a solution in the population within the tolerated Euclidean distance to that optimum. Considering the variable range in the problems, we set the tolerance values for real-variable functions as in Table 4.2. These tolerance values will provide us the satisfactory precision of the solutions. For binary bit string functions, an optimum is said to be found only when the variable reaches the exact point in the search space, i.e. the tolerance is always zero. The success rate is the number of optima found by the algorithm divided by the total number of optima.

Table 4.2 Tolerance Values for Real-Variable Test Problems

Test Problem	Tolerance
F1	0.00005
F2	0.02
F3	0.1
F4	0.15.*radius (as in [66])
F5	0.15.*radius
F6	0.02
F7	0.05
F8	0.05
F9	0.05
F10	0.02

In addition, the total time of completing a single run which includes the total number of function evaluations (which is 50,000 for problems F1-F6 and F11-F16, and 200,000 for problems F7-F10) is also recorded.

4.5 Results

4.5.1 Success Rate

Table 4.3 and 4.4 show the testing results consisting of averages and standard deviation values of SR with regard to all 16 functions. By reviewing the tables, we can observe that the performance of a single niching method varies with test problems. For instance, RTS2, with its larger window size, has a high success rate in the 25-peak test problem F3, and it has satisfactory performance in most other low-dimensional real-variable functions, but it cannot maintain its performance in high-dimensional problems and binary problems. For the 10-D test function F10, it cannot locate even one optimum. RTS1 is well qualified for F1 and F2, but not for F7-16, though it yields considerably higher success rates than RTS2 in F13 and F15. The clearing methods, CLR1 and CLR2, sometimes exhibit good ability to locate the optima, especially for binary problems. However, they cannot handle F7-10 as well, despite that CLR1 gets a 90% success rate for F10. The performance of CLR2 is inferior in F6. It only manages to locate half of the optima because its clearing radius is too large for this problem and as a result some good solutions are wrongly cleared. The DFS algorithms exhibit similar performance trends compared to the CLR methods, which is resulted from the way of niche formation. Owing to the moderate punishment on fitness, the DFS algorithms are slightly better than the CLR algorithms when success rates are considered. In the 5-D test problem F9 and the 10-D F10, DFS1 and DFS2

can still detect most of the optima. The 16 test problems offer us a variety of scenarios to examine the proposed ensemble of niching algorithms.

Two-sided, pair-wise *t*-tests between the ensembles and the single methods are conducted. We set the null hypothesis H_0 as "there is no difference between ENA and other method in terms of the SR values." Accordingly, the alternative hypothesis H_1 is "The two methods are significantly different." A significance level of 0.05 is implemented. This gives us a statistical view of the situations. Generalizing the results we find that out of the total 128 cases, there are 97 cases in which the ensemble method is at an advantage. And there are 31 cases in which no significant difference can been seen. There are no cases in which the ensemble is inferior. The outcome of the *t*-tests is presented in Table 4.5. It shows that the proposed ENA is more effective to find the optimal solutions for most of the test problems. At all the times, it is performing better than or as good as the best single niching method.

When any of the single niching methods is enough to solve the problem, as is for F5, the ensembles are equally competitive. When only one or two single algorithm is qualified, as is for F10, the ensembles are able to follow the best single algorithm. And when all the single methods have difficulties to find the peaks, as is for F8, both ENA1 and ENA2 outperform the single methods and achieve high success rates. Using the Table 4.5, we may calculate the number of problems on which a method is the best. Ties will cause the point to be divided among the equally good algorithms. ENA1 scores 11 wins over its composing methods out of 16 problems, and ENA2 has 9.92 wins. Accordingly, CLR1, being the best of the single methods in ENA1, only has 2.17 wins, and DFS2, the best of the single methods in ENA2, has 2.58 wins, as shown in Table 4.6.

Table 4.3 Comparison of Success Rates (Part A: F1 – F10)

Test Problem	Niching Method	SR Average	SR StdDev	Niching Method	SR Average	SR StdDev
F1	RTS1	1	0	RTS1	1	0
	RTS2	1	0	RTS2	1	0
	CLR1	0.82	0.0610257	DFS1	0.8733333	0.0980265
	CLR2	0.8933333	0.1014833	DFS2	0.9933333	0.0365148
	ENA1	1	0	ENA2	1	0
F2	RTS1	1	0	RTS1	1	0
	RTS2	0.9416667	0.1260017	RTS2	0.9416667	0.1260017
	CLR1	0.6	0.2330458	DFS1	0.425	0.255542
	CLR2	0.8916667	0.1420802	DFS2	0.6916667	0.181983
	ENA1	1	0	ENA2	1	0
F3	RTS1	0.8506667	0.08658	RTS1	0.8506667	0.08658
	RTS2	0.9626667	0.0457077	RTS2	0.9626667	0.0457077
	CLR1	0.2853333	0.0937985	DFS1	0.936	0.0453036
	CLR2	0.96	0.0297113	DFS2	0.9946667	0.0173669
	ENA1	0.996	0.0122051	ENA2	0.9933333	0.015162
F4	RTS1	0.6933333	0.1460593	RTS1	0.6933333	0.1460593
	RTS2	0.6533333	0.1252125	RTS2	0.6533333	0.1252125
	CLR1	0.5066667	0.1638614	DFS1	0.6333333	0.1184187
	CLR2	0.6833333	0.1205829	DFS2	0.7433333	0.1072648
	ENA1	0.87	0.0702213	ENA2	0.86	0.0968468
F5	RTS1	0.98	0.0249136	RTS1	0.98	0.0249136
	RTS2	1	0	RTS2	1	0
	CLR1	1	0	DFS1	1	0
	CLR2	0.8183333	0.0307474	DFS2	0.94	0.0380562
	ENA1	1	0	ENA2	1	0
F6	RTS1	0.9870367	0.0315738	RTS1	0.9870367	0.0315738
	RTS2	0.9314803	0.0595912	RTS2	0.9314803	0.0595912
	CLR1	0.833332	0.0899342	DFS1	0.8999993	0.0830446
	CLR2	0.5055557	0.0223656	DFS2	0.5629633	0.0896199
	ENA1	1	0	ENA2	0.998148	0.0101438
F7	RTS1	0.925	0.0583539	RTS1	0.925	0.0583539
	RTS2	0.9083333	0.0630863	RTS2	0.9083333	0.0630863
	CLR1	0.5116667	0.0925532	DFS1	0.5483333	0.1054411
	CLR2	0.885	0.0842308	DFS2	0.9533333	0.0392458
	ENA1	0.9883333	0.0252003	ENA2	0.9966667	0.0182574
F8	RTS1	0.1444448	0.0977066	RTS1	0.1444448	0.0977066
	RTS2	0.0611113	0.0654101	RTS2	0.0611113	0.0654101
	CLR1	0.0722226	0.0682764	DFS1	0.0500001	0.0517884
	CLR2	0.1527787	0.123958	DFS2	0.2027794	0.1130462
	ENA1	0.8	0.0807055	ENA2	0.802777	0.0773208
F9	RTS1	0.1489583	0.0709215	RTS1	0.1489583	0.0709215
	RTS2	0.1010417	0.058425	RTS2	0.1010417	0.058425
	CLR1	0.3	0.0979999	DFS1	0.8989583	0.0685063
	CLR2	0.175	0.0689554	DFS2	0.9791667	0.0351388
	ENA1	1	0	ENA2	1	0
F10	RTS1	0	0	RTS1	0	0
	RTS2	0	0	RTS2	0	0
	CLR1	0.9033333	0.0808717	DFS1	0.94	0.1037238
	CLR2	0	0	DFS2	1	0
	ENA1	1	0	ENA2	1	0

Table 4.4 Comparison of Success Rates (Part B: F11 – F16)

Test Problem	Niching Method	SR Average	SR StdDev	Niching Method	SR Average	SR StdDev
F11	RTS1	0.1520833	0.0798762	RTS1	0.1520833	0.0798762
	RTS2	0.0770833	0.051075	RTS2	0.0770833	0.051075
	CLR1	0.9989583	0.0057054	DFS1	1	0
	CLR2	1	0	DFS2	1	0
	ENA1	1	0	ENA2	1	0
F12	RTS1	0.15	0.0588366	RTS1	0.15	0.0588366
	RTS2	0.0614583	0.043023	RTS2	0.0614583	0.043023
	CLR1	1	0	DFS1	1	0
	CLR2	1	0	DFS2	1	0
	ENA1	1	0	ENA2	1	0
F13	RTS1	0.6444447	0.1008923	RTS1	0.6444447	0.1008923
	RTS2	0.2543207	0.1000123	RTS2	0.2543207	0.1000123
	CLR1	1	0	DFS1	1	0
	CLR2	0.9987653	0.0067625	DFS2	1	0
	ENA1	1	0	ENA2	1	0
F14	RTS1	0.2364583	0.0930952	RTS1	0.2364583	0.0930952
	RTS2	0.1135417	0.0515671	RTS2	0.1135417	0.0515671
	CLR1	1	0	DFS1	1	0
	CLR2	0.9989583	0.0057054	DFS2	0.9989583	0.0057054
	ENA1	1	0	ENA2	1	0
F15	RTS1	0.5822917	0.0846851	RTS1	0.5822917	0.0846851
	RTS2	0.384375	0.0876807	RTS2	0.384375	0.0876807
	CLR1	0.9041667	0.0591411	DFS1	0.9770833	0.0258651
	CLR2	0.9197917	0.068013	DFS2	0.9322917	0.0506446
	ENA1	1	0	ENA2	1	0
F16	RTS1	0.00625	0.0151323	RTS1	0.00625	0.0151323
	RTS2	0	0	RTS2	0	0
	CLR1	0.984375	0.0355358	DFS1	0.9864583	0.037323
	CLR2	0.9541667	0.0732801	DFS2	0.9770833	0.03664
	ENA1	0.9947917	0.0118453	ENA2	1	0

Besides the merged use of several niching methods that improves the performance of the ensemble algorithms in locating multiple optima of a problem, geographical distribution of the populations also add to the advantage of the proposed algorithm, as it forms a natural speciation similar to the effects of spatially structured evolutionary algorithms.

Table 4.5 *t*-Test Results

Test Problem	ENA1 vs RTS1	ENA1 vs RTS2	ENA1 vs CLR1	ENA1 vs CLR2	ENA2 vs RTS1	ENA2 vs RTS2	ENA2 vs DFS1	ENA2 vs DFS2
F1	0	0	1	1	0	0	1	0
F2	0	1	1	1	0	1	1	1
F3	1	1	1	1	1	1	1	0
F4	1	1	1	1	1	1	1	1
F5	1	0	0	1	1	0	0	1
F6	1	1	1	1	1	1	1	1
F7	1	1	1	1	1	1	1	1
F8	1	1	1	1	1	1	1	1
F9	1	1	1	1	1	1	1	1
F10	1	1	1	1	1	1	1	0
F11	1	1	0	0	1	1	0	0
F12	1	1	0	0	1	1	0	0
F13	1	1	0	0	1	1	0	0
F14	1	1	0	0	1	1	0	0
F15	1	1	1	1	1	1	1	1
F16	1	1	0	1	1	1	0	1

Meaning of values: 0 – there is no difference between ENA and other method; 1 – ENA is better; -1 – the other method is better.

Table 4.6 Best Performance Count

Group	Niching Method	Number of Wins over Methods in the Same Group
1	RTS1	0.8333
	RTS2	0.6667
	CLR1	2.1667
	CLR2	1.3333
	ENA1	**11**
2	RTS1	0.75
	RTS2	0.5833
	DFS1	2.1667
	DFS2	2.5833
	ENA2	**9.9167**

4.5.2 Computation Time

As we apply the same population size and same number of function evaluations to the ensembles and the single methods, we are able to make a fair comparison in terms of computation time among the methods. Table 4.7 summarizes the time required for all the algorithms to complete a single run of each test problem. It is obvious that the clearing method CLR2 with larger clearing radius has a significant advantage. In this method, more individuals are eliminated every iteration, which reduces the distance calculations between individuals and speeds up the whole process. The computation speed of ENA1 and ENA2 is comparable with CLR1, which follows the fastest CLR2. DFS1 and DFS2 require longer processing time, and the RTS methods are the slowest.

Table 4.7 Comparison of Computation Time

Test Problem	RTS1	RTS2	CLR1	CLR2	DFS1	DFS2	ENA1	ENA2
F1	255.49	244.07	5.9186	4.9216	39.117	38.115	117.51	79.902
F2	337.35	345.08	60.009	9.4768	234.55	97.864	179.47	143.37
F3	539.87	544.75	110.96	23.801	75.977	61.241	188.68	141.10
F4	473.94	477.44	50.563	13.886	133.88	47.189	240.55	119.52
F5	481.26	481.92	52.100	14.533	117.35	56.602	235.15	121.54
F6	496.09	482.79	67.705	22.045	195.22	64.866	259.46	140.25
F7	3585.8	3634.1	1180.8	99.835	2801.5	862.99	1588.6	1491.1
F8	4031.5	4047.2	1361.9	171.25	2762.4	875.37	1704.5	1785.8
F9	3654.8	3467.1	1936.1	456.39	3067.5	1587.2	1597.2	2586.9
F10	7208.4	6344.6	3418.1	3107.5	7090.0	5417.0	3799.1	4237.1
F11	291.98	300.74	184.58	165.52	329.59	315.65	115.27	128.65
F12	276.42	304.78	141.49	134.35	284.52	155.09	108.11	132.00
F13	286.16	285.47	107.00	104.66	300.79	258.36	123.77	115.73
F14	311.60	315.25	145.30	116.18	283.03	262.13	129.83	139.06
F15	314.53	322.01	129.11	124.59	420.78	256.72	106.70	133.47
F16	347.94	352.41	199.39	192.24	595.69	591.81	187.08	183.28
Total	22893	21949	8151.2	4761.2	18732	10948	10681	11679

The values are the average computation time for one run, in seconds.

On the other hand, we must be aware of the fact that the test functions are not computational expensive themselves. One fitness evaluation of these functions is usually less than 10^{-4} seconds. However, in many real-world problems, the computation of fitness can be very time-consuming. In aerodynamic design optimization for example, it takes over 10 hours to perform one fitness evaluation when it involves a 3-D computational fluid dynamics (CFD) simulation [77]. Due to ENA's superior searching capability, it shall be beneficial to use the ENA in such circumstances where the computation of fitness takes the most of the processing time.

4.5.3 Effective Function Evaluations

By assigning a fixed number of function evaluations to the ENA and all single methods in the comparison, we accept the fact that each niching algorithm in the ENA obtains fewer function evaluations because of its use of several algorithms at the same time. Yet, through the interaction mechanism among different populations, there will be good solutions (for certain algorithms) joining the other populations so that the overall algorithm efficiency is promoted.

Tracing the number of offspring that go into the next generation in any population in the ENA, we find that both in-population evolution and inter-population interaction play an important part along the whole search process. Figure 4.5–4.16 plot the number of offspring created in one population of the ensemble and accepted by any of the four populations. From them we can see how the function evaluations are utilized by the algorithm.

Taking the case of ENA1 on F1 for example, in most of the time RTS1 (population #1) and RTS2 (population #2) are generating more successful individuals that pass the survival rules of the populations. It is consistent with our experimental results of the single methods on this problem. For ENA1 on F2, in the early stage, the CLR1 population is generating more good solutions that are accepted by the ENA populations, owing to its elitist selection scheme, but later RTS1 and RTS2 also contribute to the evolution and keep the populations diverse. In the case of F3, we see that the different niching methods are playing an equal part in ENA1, whereas in ENA2, the function evaluations assigned to DFS2 are gaining a more significant effect.

Resulting from the fitness construction of F4 and F5, the number of surviving offspring for these two functions is much less than that of other test functions. Large areas of the search space receive zero fitness, which makes it hard for good solutions to appear. As compensation, the total number of accepted offspring is steady throughout the whole evolution process, with RTS1 and DFS2 taking the lead for ENA1 and ENA2 respectively.

In the case of ENA1 on F10, we see that CLR1 is obviously performing better than the other niching populations, as the function evaluations used in CLR1 are turned to useful offspring in the algorithm, which is again consistent with the search results of the single methods. The instance of ENA2 on F10 shows a little bit different scenario. In this test case, DFS1 is contributing the most initially and afterwards DFS2 takes on the responsibility to bring better offspring to the populations.

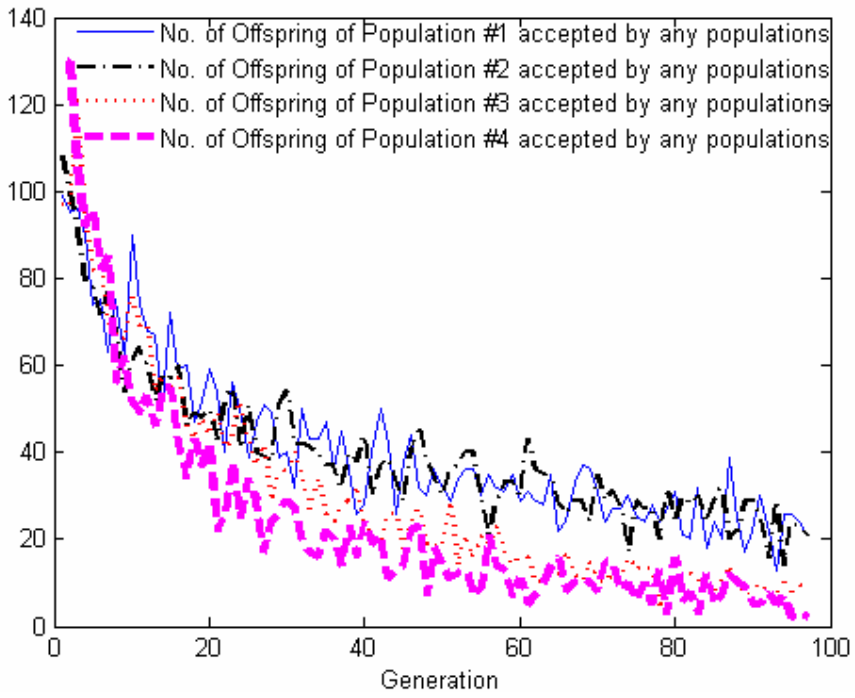

Figure 4.5 Effective Function Evaluations (F1, ENA1)

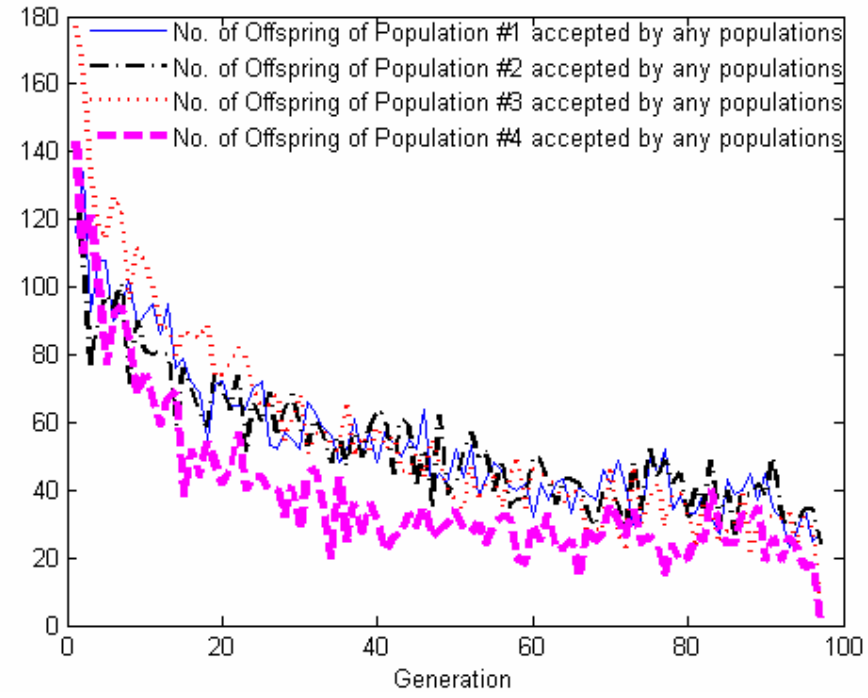

Figure 4.6 Effective Function Evaluations (F2, ENA1)

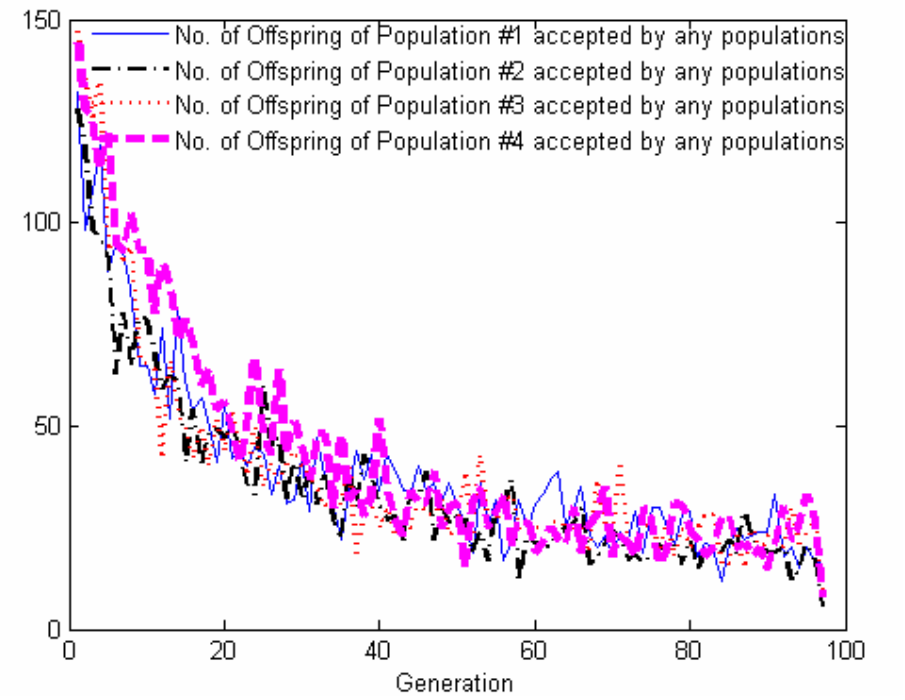

Figure 4.7 Effective Function Evaluations (F3, ENA1)

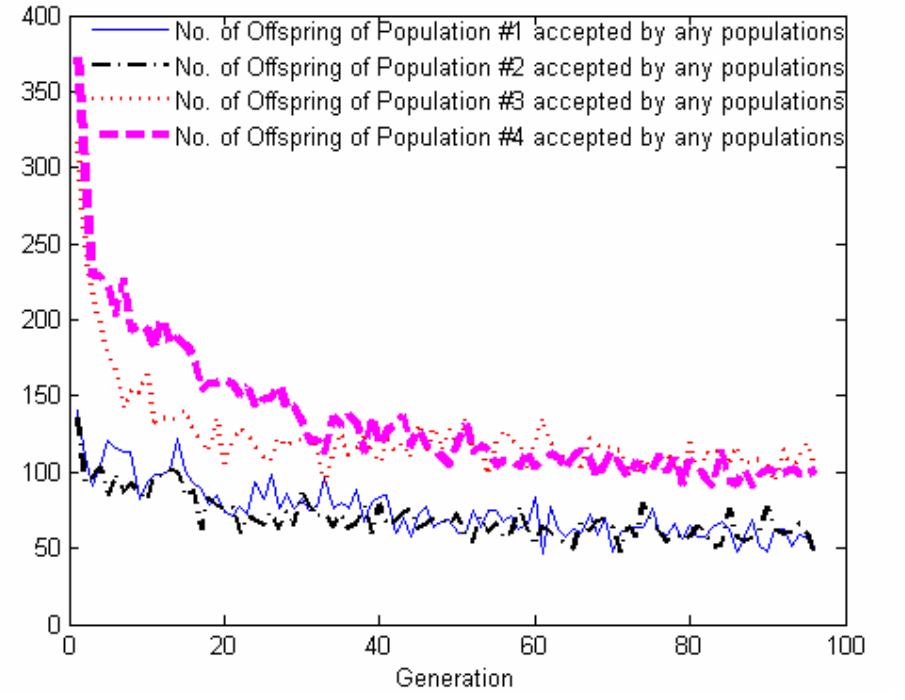

Figure 4.8 Effective Function Evaluations (F3, ENA2)

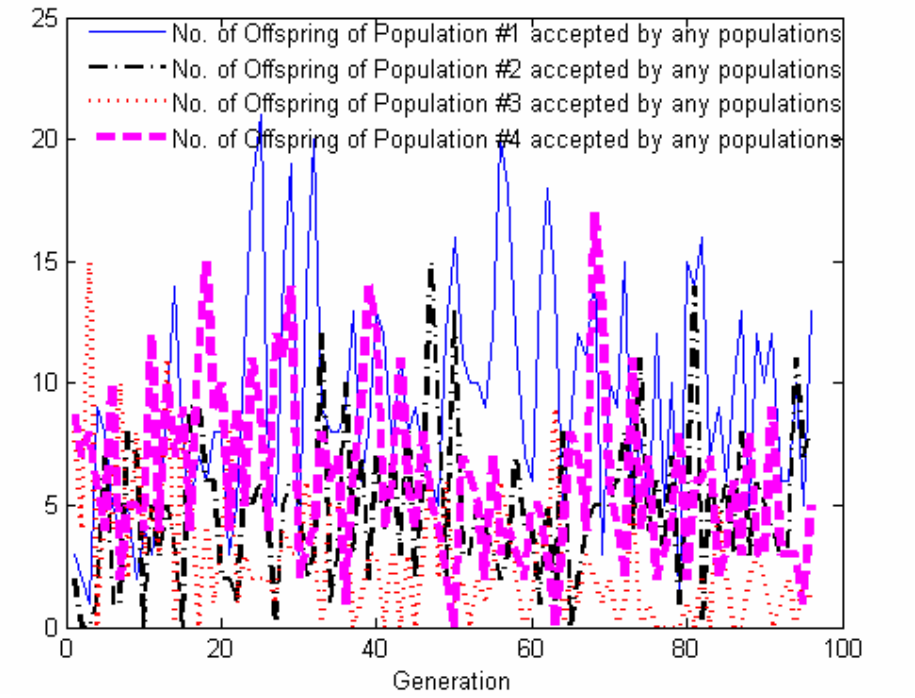

Figure 4.9 Effective Function Evaluations (F4, ENA1)

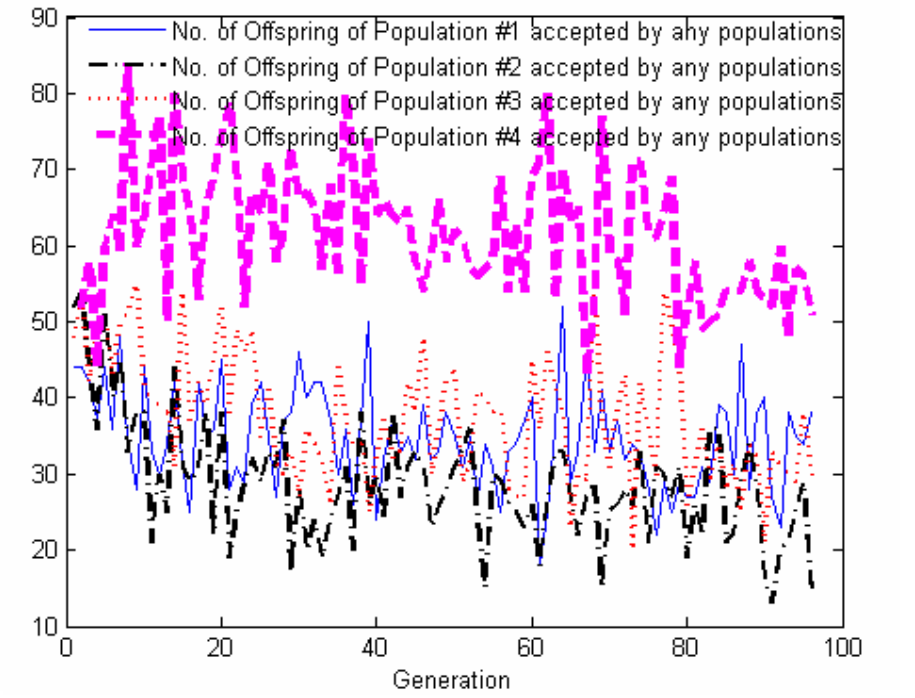

Figure 4.10 Effective Function Evaluations (F4, ENA2)

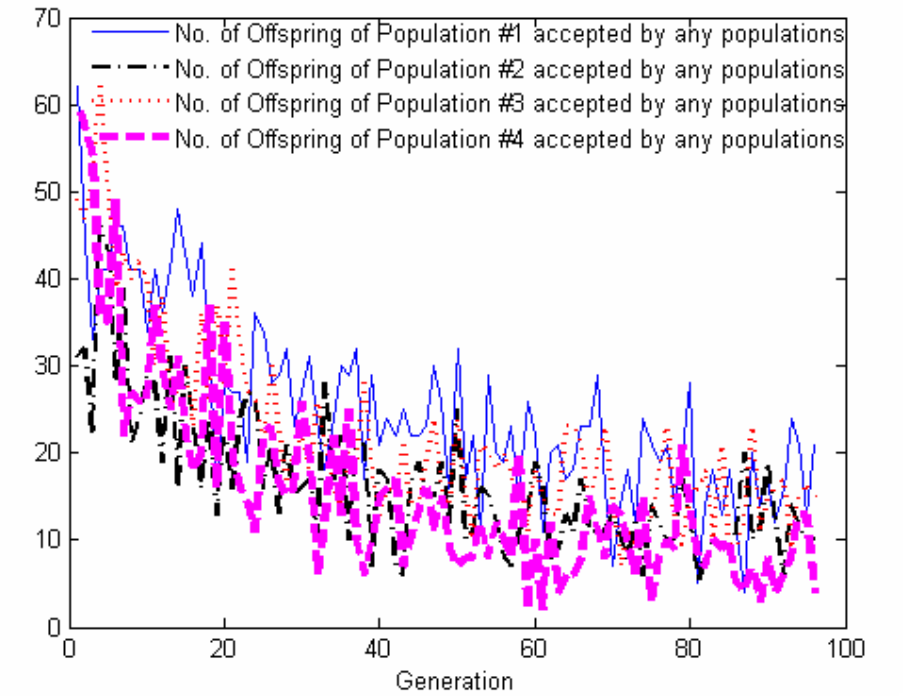

Figure 4.11 Effective Function Evaluations (F5, ENA1)

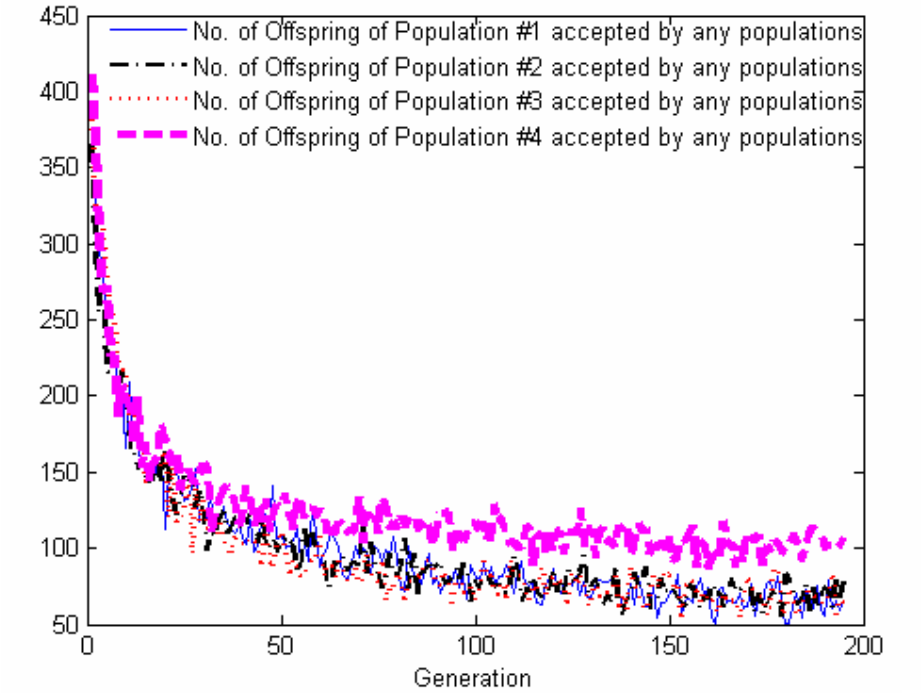

Figure 4.12 Effective Function Evaluations (F7, ENA2)

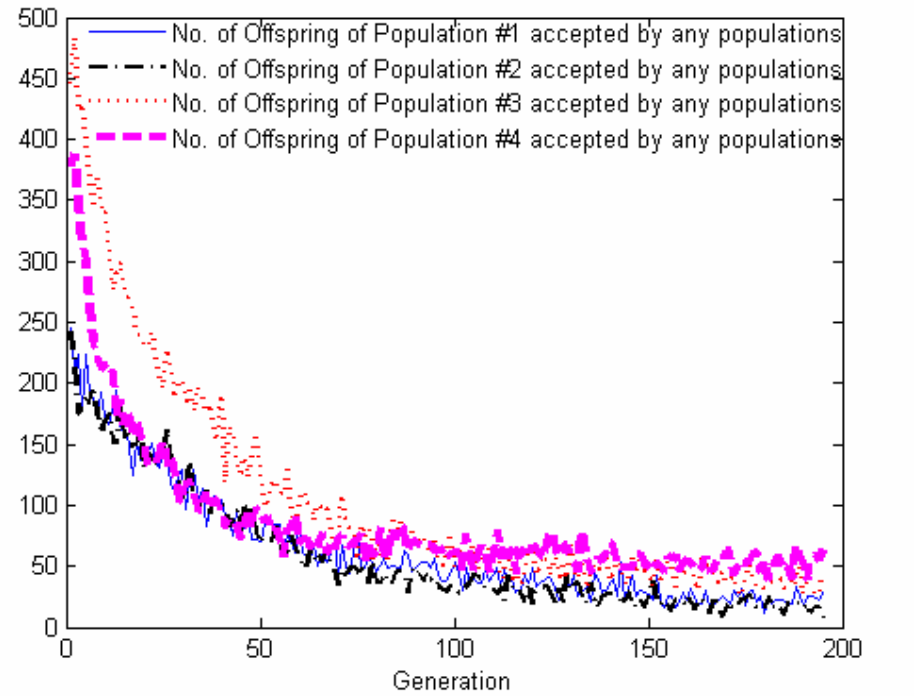

Figure 4.13 Effective Function Evaluations (F10, ENA1)

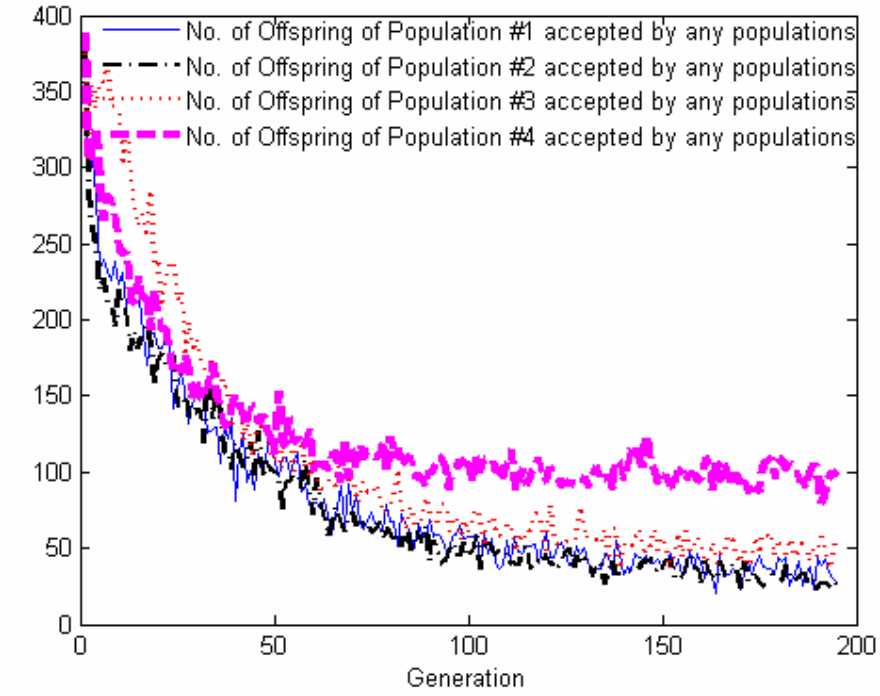

Figure 4.14 Effective Function Evaluations (F10, ENA2)

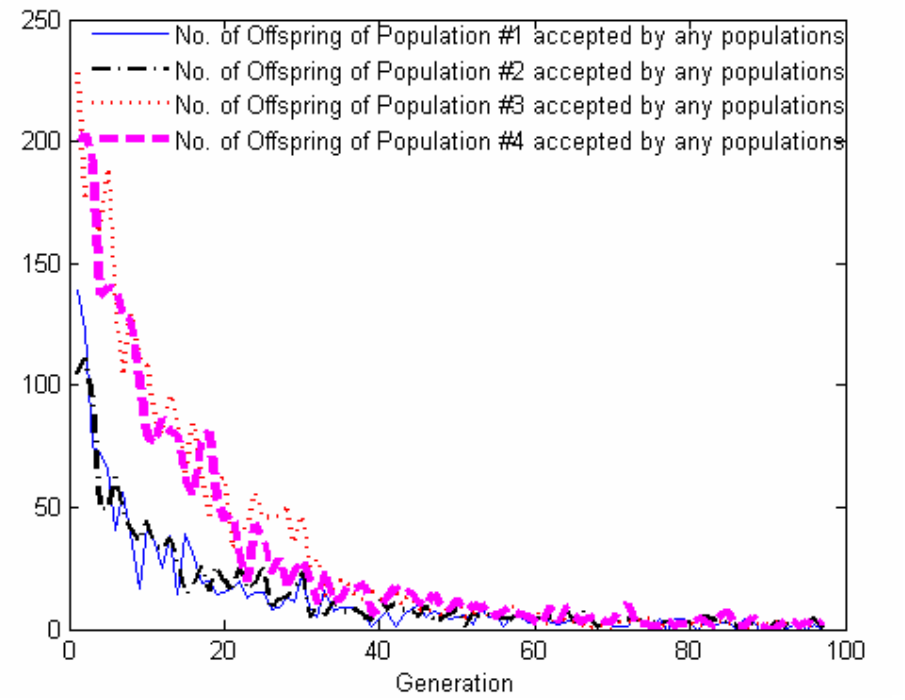

Figure 4.15 Effective Function Evaluations (F12, ENA1)

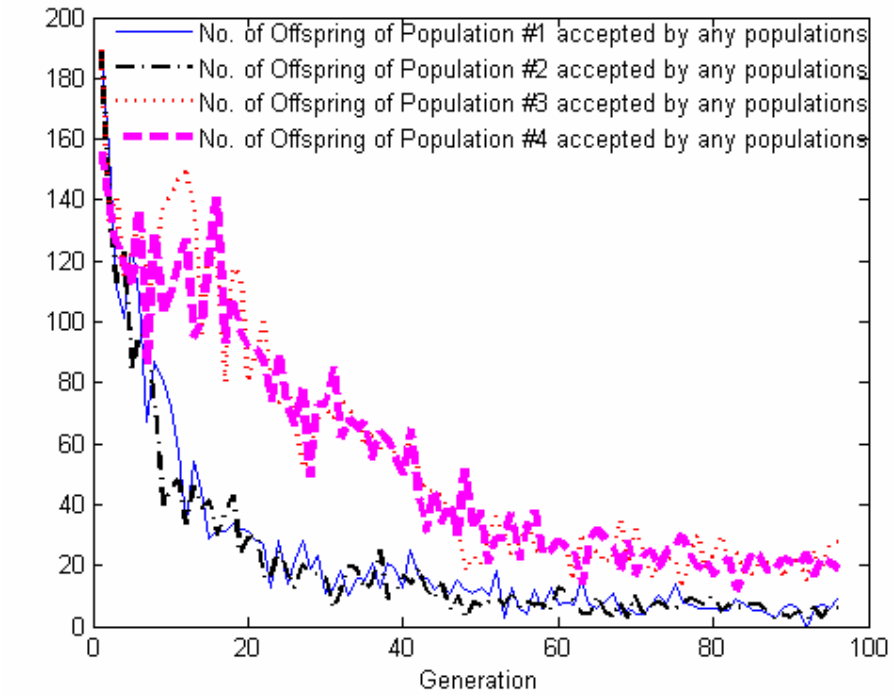

Figure 4.16 Effective Function Evaluations (F12, ENA2)

The figures of effective function evaluations give us a way to justify the roles of the single methods in the ENA. As we find out in most cases, the ENA can be helped by the best composing method in various phases of the search progress.

4.5.4 Strengths of ENA

From the above results, we see that the ensemble of niching algorithms has the following strengths:

1. Due to the different characteristics of the niching algorithms, different niching algorithms can be the best for solving different problems, as is shown from the SR values in Table 4.3 and 4.4. When solving a particular problem, the best algorithm for that particular problem will in general generate offspring of better quality which may enter the population of other niching algorithms that are not the best for solving this problem. Thereby the best niching algorithm for solving a particular problem will dominate other niching algorithms and guide the weaker algorithms to evolve well too. More offspring generated from the best single method are accepted in the ENA, as is described by Figure 4.3–4.14 in Section 4.5.3.

2. Our ensemble algorithm encourages an offspring to stay, in any population that can benefit. An offspring may be rejected by the population where it is created, but it may enter other populations. By passing the updating rule of a particular population in the ENA, the offspring verifies itself as a useful member to that population.

3. Creating a new individual involves a function evaluation. The ensemble algorithm makes the most out of the every new individual and every function evaluation as each population considers every offspring for possible inclusion in it according to its own selection rule.

4. ENA requires less parameter tuning as the migration between the populations using different niching algorithms together with selection cause a homeostatic simulation of a good parameter choice.

On the other hand, we encourage the use of ENA in the circumstances when we are not sure which niching algorithm will achieve the best performance. If we have the knowledge that a certain niching algorithm will be the most suitable one for a multimodal optimization task, it is not necessary to use ENA.

4.6 Comparison with State-of-the-Art Niching Algorithms

From the experimental results, we have demonstrated the advantage of ENA over its composing single methods. Apart from that, we are going to review four recently proposed niching algorithms and compare ENA with its predecessors. The widely used benchmark functions in our test suite allow us to make a direct comparison of the results. For simplicity's sake, ENA1 is selected as the representative of the ensembles.

4.6.1 Discussion on Parameter Tuning

Generally speaking, fine-tuning the niching parameters for every problem separately will give better outcome. But, here we only aim to show the suitability of ENA to solve these problems. Therefore, the same parameter values are used to solve all problems. The trade-off between computation cost of parameter tuning and algorithm efficiency is left to be explored in the future. On the other hand, we observe that ENA is not much sensitive to the niching parameters as single methods. Taking F6 as an example, we see the clear influence of parameters to the results of two clearing methods. CLR2 misses half of the optima when its clearing radius is too large. ENA does not suffer from this problem thanks to the other niching populations that also maintain the good solutions. RTS would wrongly delete the optima if window size is too small as in F3. However, when we combine different parameters in the ENA, we cancel out the bad effects, because different parameter combinations help speed-up the convergence and retain the identified optimal individuals simultaneously.

4.6.2 Comparison of Results

In [43], a species conserving genetic algorithm (SCGA) was proposed for multimodal function optimization. By preserving a set of individuals called species seeds, the technique offers the promise of being an effective and efficient method for finding multiple global optima. However, for the 2-D Shubert function (F6), using the best choice of population size out of 15 settings, SCGA needs an average 35647 FEs to locate all the 18 optima, while ENA needs only 24313 FEs on average based on 30 independent runs. SCGA counts “best individuals” as individuals with fitness higher

than $(f_{\max} - f_{\min}) \times r_f$ with r_f=0.95, while ENA achieves higher precision with r_f>0.997 using 30% less FEs.

Another method, a sharing scheme based on niche identification techniques (NIT) was introduced in [35]. It reported that for function $f(x) = (1-x)\sin^4(\frac{5\pi}{(1+x)^4})$ (comparable to our F1) with 5 optima, NIT can locate 10, 10, 10, 9, and 5 times each optimum in 10 runs requiring 10,000 FEs in each run. In our experiments, ENA discovers the 4th and 5th optima after 4873 FEs and 6906 FEs on average, respectively. In the worst cases, the 4th and 5th optima are discovered after 7800 and 12,700 FEs respectively. The percentage that the 4th and 5th optima are discovered within 10000 FEs by ENA is 100% and 83.33%. Each one is better than the corresponding values of 9/10 and 5/10 that of NIT.

In [47], spatially-structured evolutionary algorithms (SSEA) and localized niching were presented. The paper offers a specific implementation of localized niching called local clearing (LC), which takes the traditionally global operation of clearing and applies it to each location in space. Two test functions, F11 and F13 are used to examine the proposed methods. Ring SSEA without clearing fails to find all optima. LC is able to discover and maintain all 32 peaks in the search space with less than 24000 FEs while ENA uses 23683 FEs on average. For F13, ENA outperforms LC by consuming only an average of 18563 FEs to locate all the optima, whereas LC needs up to 30000.

Finally, we compare with the real-coded sequential niching memetic algorithm (SNMA) proposed in [41], since both SNMA and ENA were tested on the modified *Himmelblau*'s function F2. To locate all four optima of F2, SNMA uses up to 31416

FEs on average, whereas ENA only needs 22770. These comparisons offer us a better view of ENA in relation to the state-of-the-art niching algorithms.

4.7 Conclusion

We have proposed an ensemble approach in the field of niching genetic algorithms for solving multimodal optimization problems. The ensemble of niching algorithms possesses several niching algorithms with their own parallel populations. In contrast to most other genetic algorithms with parallel populations, ENA employs different niching schemes and every function evaluation is used by every population. The offspring of every population can be included in other niching populations by going through elimination process of the specific niching population. This maximizes the usage of function evaluations, thereby enabling the ENA to evolve all the populations to effectively search for more optimal solutions.

It is seen from the results presented above, that the proposed ENA is capable of locating more optima than any of its composing single niching method in most cases. This establishes its strength in the field of multimodal function optimization. Although we incorporated four niching algorithms with four parallel populations, it is possible to construct different ENA instantiations by using different niching algorithms. As the ENA introduces no extra parameters in addition to the parameters in the single methods, ENA with more populations is also possible.

Chapter 5

Evolutionary Programming with Ensemble of Niching Memories for Dynamic Optimization

This chapter presents the evolutionary programming with an ensemble of memories to deal with optimization problems in dynamic environments. The proposed algorithm modifies a recent version of evolutionary programming by introducing a simulated-annealing-like dynamic strategy parameter as well as applying local search towards the most improving directions. Diversity of the population is enhanced by an ensemble of external archives that serves as short-term and long-term memories. The archive members also act as the basic solutions when environmental changes occur. The algorithm is tested on a set of 6 multimodal problems with a total 49 change instances provided for the CEC 2009 Competition on Evolutionary Computation in Dynamic and Uncertain Environments and the results are presented.

5.1 Optimization in a Dynamic Environment

In recent years, evolutionary optimization in dynamic environments has attracted much interest among researchers. A successful dynamic optimization algorithm should not only be able to locate the optimum, as it does in the static sense, but also be capable of detecting when the environment changes and tracking the new optimum.

This can be a challenging problem, as in most cases, especially when the environment has been static for some time, the population tends to converge to the best solution,

and thereby loses its adaptability. In order to be flexible enough to respond to environmental changes, the algorithm needs to explore several potentially good regions in the search space, and at the same time, maintain the pace of fine search for the optimal solution.

We have chosen evolutionary programming (EP) as the main algorithm to perform the task of optimization in dynamic environments on account of its fast convergence ability. Therefore our essential task is to increase the diversity of the population and ensure it does not get trapped into a single optimal solution so that it cannot evolve further. We use an ensemble of memory-based archives to introduce diverse individuals into the population. Moreover, we examine the strategy parameter in EP and propose a dynamic scheme that is designed to make the EP more suitable for optimization in dynamic environments.

5.2 Unbiased Evolutionary Programming with Dynamic Strategy Parameter

In a recent study [78], a new version of EP called *unbiased evolutionary programming* (UEP) was introduced and was shown to have markedly improved performance for high-dimensional optimization. We will take this version to solve the dynamic optimization problems. Other than generating Gaussian deviates in axial directions, a set of angles is randomly generated in UEP. These are resolved into a direction vector in the co-ordinate system, which is then multiplied by a Gaussian deviate.

Several steps are added to the self-adaptive EP mentioned in Chapter 2. Firstly, an additional parameter ϕ_i is assigned to each of the population member, making it a triple (x_i, ϕ_i, η_i), i=1,2,…,μ. Each x_i represents an n-dimensional vector of objective variables. ϕ represents an (n−1)-dimensional vector of angles, initialized randomly from a uniform distribution between 0 and 2π, and η_i represents a vector of standard deviations. After evaluating the fitness score of each individual as in classical EP, create a set of μ offset vectors, Δx_i, i=1,2,…,μ, as follows:

a) For each value of i, set *prevSin*=1. Then for each value of j, j=1,2,…,n−1, set:

$$\Delta x_i(j) = \cos(\phi_i(j)) \times prevSin \tag{5-1}$$

$$prevSin \leftarrow \sin(\phi_i(j)) \times prevSin \tag{5-2}$$

Finally set $\Delta x_i(n) = prevSin$.

b) Randomly permute the elements of Δx_i. This is necessary because the above efficient method of resolving angles into Cartesian components generates the co-ordinates in monotonically decreasing order.

When generating the offspring (x_i', ϕ_i', η_i'), $\Delta x_i(j)$ is multiplied to the increment of $x_i(j)$, which makes equation (2-1) as:

$$x_i'(j) = x_i(j) + \eta_i(j) N_j(0,1) \Delta x_i(j) \tag{5-3}$$

$\phi_i'(j)$ is again randomly generalized from a uniform distribution between 0 and 2π, and $\eta_i'(j)$ is derived by equation (2-2). The other steps are as in the classical self-adaptive EP.

To make the UEP more efficient in searching for the optimum, we introduce a local search method to be applied to the algorithm. In every generation, the fitness of each offspring will be compared to that of its parent. From the largest increments of the fitness values we obtain the most-improving directions. These directions are used to perform the local search, which is described by the following equation:

$$x_i'' = x_i' + F(x_i' - x_i) \tag{5-4}$$

where $x_i' - x_i$ offers one of the most-improving direction and F is a constant (set to 0.85 in our experiments).

Observing the self-adaptive EP described in Chapter 2, we find that the parameter η has a tendency to get near to either infinity or zero as generation number increases. The former situation is unfavourable, as solutions vibrate severely even at the end of the evolution. On the contrary, we may want to have a larger mutation deviation at the beginning, and, as population evolves, transfer the population from exploration to fine search near the good solutions with small mutation deviation. Annealing of the strategy parameter η will just serve the purpose.

Simulated annealing (SA) was proposed as a method for solving discrete optimization problems as well as single and multiple objective optimization problems in various fields [79]–[83]. Initial temperature is chosen such that it can capture the entire solution space. Cooling schedule determines functional form of the change in temperature required in SA. The earliest annealing schedules have been based on the analogy with physical annealing. They set initial temperature high enough to accept all transitions and used a proportional temperature. Three important cooling schedules are logarithmic, Cauchy and exponential. It was shown in [84] that the classical

cooling schedules are all equivalent, with no clearly better annealing schedule than the logarithmic schedule to ensure convergence towards the set of optima with a probability of one.

We use a variant of the exponential form of cooling schedule to adjust the parameter in EP, which is explained by the following equation:

$$\eta = T_0 \exp(\sqrt{\frac{1}{n}} - \sqrt{\frac{t}{n}}) \cdot (r + 0.5) \qquad (5\text{-}5)$$

where T_0 denotes the initial temperature which is set to 5 for our experiments, n is the dimension number, t stands for time or number of generations, and r is a uniformly distributed random variable in the interval of [0, 1]. This formula applies to the η value in every dimension of every individual in the population. A sample of the parameter values over 1000 generations is shown in Figure 5.1.

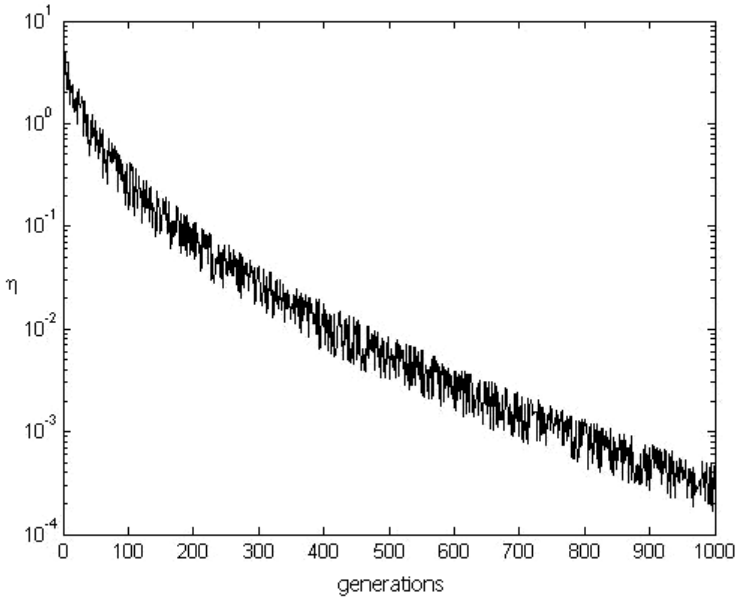

Figure 5.1 Semi-log Plot of η

The above scheme enables EP to have an exponentially decreasing deviation for mutations with certain perturbation. It is a simple and explicit form, which is very useful in dynamic environments. The strategy parameter at any time depends only on the initial temperature, but not on the previous values. This makes it easy for the algorithm to adapt to environmental changes. On detecting a change, the strategy parameter will be re-initialized.

5.3 Niching External Memories and Environmental Change Handling

In order to make the algorithm be suitable for optimizing dynamic problems, we will make some further modifications. Although we can always regard each change as the arrival of a new problem, it is more efficient if we can make use of the previous knowledge about the search space, assuming that the changes are not too radical. There are many ways to deal with environmental changes [85], [86]. For example, after a change is detected, the mutation rate can be raised to increase the diversity of the population [87], or else, niching methods can be used to spread out the population so that it may adapt to changes more easily [88]. Memory-based approaches [74], [89] have also been proposed to recall information from past generations, which is especially useful when the new optimum is not far from previous locations. In addition, we may apply multi-population techniques to track multiple peaks in the fitness landscape [90], [91].

Here we propose a diversity enhancement method which is implemented not only after the environmental changes but also during the whole run. It is realized using an

ensemble of two memory archives that store solutions found by the algorithm in different times before the current generation. The first archive, which represents the short-tem memory, serves for the regular updating, and the second, which represents the long-term memory, is used when premature convergence is identified. When both cannot help and the population totally loses diversity, we restart the search.

5.3.1 Archive Operations

The first archive is renewed every 10 generations, i.e. it stores all the offspring created in the last 10 generations. A niching method, clearing [37], is applied to the archive when it has gathered all the members. This is to ensure the individuals with high fitness in the archive are distributed evenly in the search space instead of crowding in one area. After the clearing procedure, we may assume the archive to have a certain level of diversity and we can directly make use of the members to replace the population in order to allow further exploration.

Different from the standard clearing procedure, we number all the individuals sequentially within a niche according to their fitness values. The individual with the highest fitness is assigned the number “1”. After the numbering, we sort the archive from number “1” to the largest number so that the individuals with the highest fitness in every niche are on the top of the list and the individuals with the second highest fitness in every niche are next and so on.

Although we are going to replace the whole population with the top individuals from the sorted archive, we may still find some consistency of evolution as all the archive members are from the last 10 generations and there should surely be a certain number

of them from the last generation. Our experiments have revealed that if we keep part of the population and replace the other part with the archive members, there will be some unnecessary redundancy. When a change is detected, we also count on the first archive to obtain the starting points for the next round.

The second archive relates to a higher level of diversity, of which we use the standard deviation of fitness values as the measure. It is generated in the same way as the first archive but only in the very beginning of the search and right after every detected change when the population is most diverse. In other words, we just maintain this archive until the next change occurs.

When the standard deviation of fitness in the population is successively very small, say, less than a predefined threshold for the last 50 generations, we shall replace 95 percent of the population with the members from the second archive, so that reasonably good and diverse members are stored. This means that for the 95 percent of the population the evolution is dated back to the very beginning of the current environmental state. We still keep 5 percent of the elites so that they can continue fine search without any impact.

In an effort to avoid repeatedly using the same individuals in the second archive, we use a probabilistic scheme when picking the archive members. As we have mentioned earlier, the archive is already properly sorted, so we pick the individuals from the archive sequentially with a fixed probability of 0.8. The individuals that are chosen out will be put to the bottom of the archive.

5.3.2 Detection of Changes

The detection of changes plays an important part in our algorithm. We must firstly detect the changes effectively so that we can take relevant measures. As is said in [85], we may use the deterioration of the population performance or the time-averaged best performance as an indicator; or we may re-evaluate several individuals and if the fitness of at least one individual has changed, we recognize it as a change.

As far as our experiments are concerned, we adopt the second strategy. The experimental results show that it is very efficient provided that the environmental change has an influence on every individual. The "interesting points" we focus on are the top three solutions in first archive (after clearing) as they are going to get into the population as soon as a change is detected. We do so with the purpose to decrease the wastage of function evaluations.

5.3.3 When Population Totally Loses Diversity

There are times that the population will lose diversity totally. All the individuals are converging to a single point. This can be disadvantageous as in a dynamic environment the optimum will probably move to another location sooner or later.

In our experiments, we found that partial restart is not enough to stop this situation. Reference [88] has also suggested that the main problem of partial restart is that it is very hard for the newly introduced solutions to establish themselves when the population contains highly fit individuals. Therefore when we find that the standard deviation of fitness is extremely small, say, less than another predefined threshold, we

are going to reinitialize all the population except the best one. Moreover, we set the strategy parameter η to the initial value and restart the "time" according to it.

5.4 Generalized Dynamic Benchmark Generator (GDBG)

We use the test suite called the Generalized Dynamic Benchmark Generator, which is provided for CEC 2009 Competition on Evolutionary Computation in Dynamic and Uncertain Environments.

5.4.1 Framework of the GDBG System

Dynamic optimization problems (DOPs) can be defined as follows:

$$F = f(x, \phi, t) \tag{5-6}$$

where F is the optimization problem, f is the cost function, x is a feasible solution in the solution set $\mathbf{X}$, t is the real-world time, and ϕ is the system control parameter, which determines the solution distribution in the fitness landscape.

In the GDBG system, the dynamism results from a deviation of solution distribution from the current environment by tuning the system control parameters. It can be described as follows:

$$\phi(t+1) = \phi(t) \oplus \Delta\phi \tag{5-7}$$

where $\Delta\phi$ is a deviation from the current system control parameters. Then, we can get the new environment at the next moment t+1 as follows:

$$f(x,\phi,t+1) = f(x,\phi(t) \oplus \Delta\phi,t) \tag{5-8}$$

There are six change types of the system control parameters in the GDBG system. They are small step change, large step change, random change, chaotic change, recurrent change, and recurrent change with noise. The framework of the six change types are described as follows:

small step:

$$\Delta\phi = \alpha \cdot \|\phi\| \cdot r \cdot \phi_{severity} \tag{5-9}$$

large step:

$$\Delta\phi = \|\phi\| \cdot (\alpha \cdot sign(r) + (\alpha_{\max} - \alpha) \cdot r) \cdot \phi_{severity} \tag{5-10}$$

random:

$$\Delta\phi = N(0,1) \cdot \phi_{severity} \tag{5-11}$$

chaotic:

$$\phi(t+1) = A \cdot (\phi(t) - \phi_{\min}) \cdot (1 - (\phi(t) - \phi_{\min}) / \|\phi\|) \tag{5-12}$$

recurrent:

$$\phi(t+1) = \phi_{\min} + \|\phi\|(\sin(\frac{2\pi}{P}t + \varphi) + 1) / 2 \tag{5-13}$$

recurrent with noise:

$$\phi(t+1) = \phi_{\min} + \|\phi\|(\sin(\frac{2\pi}{P}t + \varphi) + 1) / 2 + N(0,1) \cdot noisy_{severity} \tag{5-14}$$

where $\|\phi\|$ is the change range of ϕ, $\phi_{severity}$ is a constant number that indicates change severity of ϕ, $\phi_{\min}$ is the minimum value of ϕ, $noisy_{severity} \in (0,1)$ is noisy severity in recurrent with noisy change. $\alpha \in (0,1)$ and $\alpha_{\max} \in (0,1)$ are constant values, which are set to 0.04 and 0.1 in the GDBG system. A logistics function is used in the chaotic change type, where A is a positive constant between (1.0, 4.0). If ϕ is a vector, the initial values of the items in ϕ should be different within $\|\phi\|$ in the chaotic change. P is the period of recurrent change and recurrent change with noise, φ is the initial phase, r is a random number in (-1, 1), $sign(x)$ returns 1 when x is greater than 0, returns -1 when x is less than 0, and returns 0 otherwise. $N(0,1)$ denotes a normally distributed one dimensional random number with mean zero and standard deviation one.

In some real world problems, the number of variables may change online over the time processes. For example, in dynamic scheduling problem one machine might break down, or in the routing problem you may have to add a new node into an already-planned route. On the other hand, the performance of EAs may deteriorate quickly as the dimensionality increases due to the increased problem complexity and search space. In GDBG, the number of dimensions changes as follows:

$$D(t+1) = D(t) + sign \cdot \Delta D \qquad (5\text{-}15)$$

where ΔD is a predefined constant, for which the default value is 1. $sign = 1$ initially, to allow the number of dimensions to increase continually, and when $D(t) = Max\_D$, $sign = -1$, opening a decreasing phase. If $D(t) = Min\_D$, $sign = 1$ again. Max_D and Min_D are the maximum and minimum number of dimensions. When the number of dimension decreases by 1, just the last dimension is removed from the fitness landscape, and the fitness landscape of the left dimensions does not

change. When the number of dimension increases by 1, a new dimension with random value is added into the fitness landscape. Dimensional change only happens following the non-dimensional change.

5.4.2 Benchmark Instances

The two benchmark instances are: dynamic rotation peak benchmark generator (DRPBG) and dynamic composition benchmark generator (DCBG).

A. Dynamic Rotation Peak Benchmark Generator

Given a problem $f(x,\phi,t)$, $\phi=(\vec{H},\vec{W},\vec{X})$, where $\vec{H}$, $\vec{W}$, and $\vec{X}$ denote the peak height, width, and position respectively. The function $f(x,\phi,t)$ is defined as follows:

$$f(x,\phi,t)=\max_{i=1}^{m}(\vec{H}_i(t)/(1+\vec{W}_i(t)\cdot\sqrt{\sum_{j=1}^{n}\frac{(x_j-\vec{X}_j^i(t))^2}{n}})) \tag{5-16}$$

where *m* is the number of peaks, and *n* is the number of dimensions.

$\vec{H}$ and $\vec{W}$ change as follows:

$$\vec{H}(t+1)=DynamicChanges(\vec{H}(t)) \tag{5-17}$$

$$\vec{W}(t+1)=DynamicChanges(\vec{W}(t)) \tag{5-18}$$

where in the height change, $height\_severity$ should read $\phi\_h_{severity}$ according to (5-9)–(5-11), and $\|\phi\_h\|$ is height range. Accordingly, $width\_severity$ and width range should read $\phi\_w_{severity}$ and $\|\phi\_w\|$ in the width change.

A rotation matrix $R_{ij}(\theta)$ is obtained by rotating the projection of $\vec{x}$ in the plane *i*–*j* by an angle θ from the *i*-th axis to the *j*-th axis. The peak position $\vec{X}$ is changed by the following algorithm:

Step 1. Randomly select *l* dimensions (*l* is an even number) from the *n* dimensions to compose a vector $r = [r_1, r_2, \cdots, r_l]$.

Step 2. For each pair of dimension *r*[*i*] and dimension *r*[*i*+1], construct a rotation matrix $R_{r[i],r[i+1]}(\theta(t))$, $\theta(t) = DynamicChanges(\theta(t-1))$.

Step 3. A transformation matrix $A(t)$ is obtained by:

$$A(t) = R_{r[1],r[2]}(\theta(t)) \cdot R_{r[3],r[4]}(\theta(t)) \cdots R_{r[l-1],r[l]}(\theta(t))$$

Step 4. $\vec{X}(t+1) = \vec{X}(t) \cdot A(t)$

where the change severity of θ ($\phi\_\theta_{severity}$) is set to 1, and the range of θ should read $\|\phi\_\theta\|$, $\phi\_\theta \in (-\pi, \pi)$. For the value of *l*, if *n* is an even number, $l = n$; otherwise $l = n - 1$.

For recurrent changes, ϕ_θ is within $(0, \pi/6)$.

B. Dynamic Composition Benchmark Generator

The dynamic composition functions are extended from the static composition functions developed by Suganthan *et al.* [67]. The composition function can be described as:

$$F(x, \phi, t) = \sum_{i=1}^{m} (w_i \cdot (f_i'((x - \vec{O}_i(t) + O_{iold}) / \lambda_i \cdot \vec{M}_i) + \vec{H}_i(t))) \qquad (5\text{-}19)$$

where the system control parameter $\phi = (\vec{O}, \vec{M}, \vec{H})$, $F(x)$ is the composition function, $f_i(x)$ is the i-th basic function used to construct the composition function. m is the number of basic functions, $\vec{M}_i$ is orthogonal rotation matrix for each $f_i(x)$, $\vec{O}_i(t)$ is the optimum of the changed $f_i(x)$ caused by rotating the landscape at the time t. O_{iold} is the optimum of the original $f_i(x)$ without any change, the O_{iold} is 0 for all the basic functions used in this chapter. The weight value w_i for each $f_i(x)$ is calculated as:

$$w_i = \exp(-sqrt(\frac{\sum_{k=1}^{n}(x_k - o_i^k + o_{iold}^k)^2}{2n\sigma_i^2}))$$

$$w_i = \begin{cases} w_i & \text{if } w_i = \max(w_i) \\ w_i \cdot (1 - \max(w_i)^{10}) & \text{if } w_i \neq \max(w_i) \end{cases}$$

$$w_i = w_i / \sum_{i=1}^{m} w_i$$

where σ_i is the converge range factor of $f_i(x)$, whose default value is 1.0, λ_i is the stretch factor for each $f_i(x)$, which is defined as:

$$\lambda_i = \sigma_i \cdot \frac{X_{\max} - X_{\min}}{x_{\max}^i - x_{\min}^i}$$

where $[X_{max}, X_{min}]^n$ is the search range of $F(x)$ and $[x_{\max}^i, x_{\min}^i]^n$ is the search range of $f_i(x)$.

In (5-19), $f_i^{'}(x) = C \cdot f_i(x) / \left| f_{\max}^i \right|$, where C is a predefined constant, which is set to 2000, and $f_{\max}^i$ is the estimated maximum value of $f_i(x)$, which is estimated as:

$$f_{\max}^i = f_i(x_{\max} \cdot M_i)$$

Table 5.1 Details of the Basic Benchmark Functions

Name	Function	Range
Sphere	$f(x)=\sum_{i=1}^{n} x_i^2$	[–100, 100]
Rastrigin	$f(x)=\sum_{i=1}^{n}(x_i^2-10\cos(2\pi x_i)+10)$	[–5, 5]
Weierstrass	$f(x)=\sum_{i=1}^{n}(\sum_{k=0}^{k_{\max}}[a^k\cos(2\pi b^k(x_i+0.5))])-n\sum_{k=0}^{k_{\max}}[a^k\cos(\pi b^k)]$ $a=0.5, b=3, k_{\max}=20$	[–0.5, 0.5]
Griewank	$f(x)=\frac{1}{4000}\sum_{i=1}^{n}(x_i)^2-\prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}})+1$	[–100, 100]
Ackley	$f(x)=-20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2})-\exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i))+20+e$	[–32, 32]

In the composition DBG, $\vec{M}$ is initialized using the above transformation matrix construction algorithm and then remains unchanged. The dynamism of the system control parameter $\vec{H}$ and $\vec{O}$ are changed as the parameters $\vec{H}$ and $\vec{X}$ in DRPBG.

For both DRPBG and DCBG, chaotic change of peak locations directly operates on the value of each dimension instead of using rotation matrix to simulate chaotic systems in real applications.

Five basic benchmark functions are used in the GDBG system. Table 5.1 shows the details of the five functions.

5.4.3 Problem Definition and Parameter Settings

Overview of the test functions on real space

F_1: Rotation peak function

F_2: Composition of Sphere function

F_3: Composition of Rastrigin's function

F_4: Composition of Griewank's function

F_5: Composition of Ackley's function

F_6: Hybrid composition function

For all test functions:

Dimension: n(fixed)=10; n(changed) $\in$ [5, 15]

Search range: $x \in [-5, 5]^n$

Change frequency: $frequency$=10,000$\times n$FES

The number of changes: num_change=60

Period: p=12

Severity of recurrent change with noise: $noisy_{severity}$=0.8

Chaotic constant: A=3.67

Chaotic initialization: If ϕ is a vector, the initial values of the items in ϕ should be randomly generated using uniform distribution within $\|\phi\|$ in (5-12).

Step severity: α=0.04

Maximum of α: α_{max}=0.1

Height range: $h \in [10, 100]$

Initial height: *initial_height*=50

Height severity: $\phi\_h_{severity}$=5.0

For all composition functions:

The number of basic functions m=10

Coverage range factor: σ_i=1.0, i=1,2, $\cdots$,n

C=2000

F_1: Rotation Peak Function

The number of peaks: m=10, 50

Width range: $w \in [1, 10]$

Width severity: $\phi\_w_{severity}$=0.5

Initial width: *initial_width*=5

Properties:

- ♠ Multimodal
- ♠ Scalable
- ♠ Rotated
- ♠ The number and locations of local optima are artificially controlled
- ♠ $x \in [-5, 5]^n$, global optimum $x^*(t) = \vec{O}_i$, $F(x^*(t)) = H_i(t)$, $H_i(t) = \max \sum_{j=1}^{m} H_j$

F_2: Composition of Sphere Function

Basic functions: f_1–f_{10}=Sphere function

Properties:

- ♠ Multimodal
- ♠ Scalable
- ♠ Rotated
- ♠ 10 local optima
- ♠ $x \in [-5, 5]^n$, global optimum $x^*(t) = \vec{O}_i$, $F(x^*(t)) = H_i(t)$, $H_i(t) = \min \sum_{j=1}^{m} H_j$

F_3: Composition of Rastrigin's Function

Basic functions: f_1–f_{10}=Rastrigin's function

Properties:

- ♠ Multimodal
- ♠ Scalable
- ♠ Rotated
- ♠ A huge number of local optima
- ♠ $x \in [-5, 5]^n$, global optimum $x^*(t) = \vec{O}_i$, $F(x^*(t)) = H_i(t)$, $H_i(t) = \min \sum_{j=1}^{m} H_j$

F_4: Composition of Griewank's Function

Basic functions: f_1–f_{10}=Griewank's function

Properties:

- ♠ Multimodal
- ♠ Scalable
- ♠ Rotated
- ♠ A huge number of local optima
- ♠ $x \in [-5, 5]^n$, global optimum $x^*(t)= \vec{O}_i$, $F(x^*(t))=H_i(t)$, $H_i(t)=\min \sum_{j=1}^{m} H_j$

***F*$_5$: Composition of Ackley's Function**

Basic functions: f_1–f_{10}=Ackley's function

Properties:

- ♠ Multimodal
- ♠ Scalable
- ♠ Rotated
- ♠ A huge number of local optima
- ♠ $x \in [-5, 5]^n$, global optimum $x^*(t)= \vec{O}_i$, $F(x^*(t))=H_i(t)$, $H_i(t)=\min \sum_{j=1}^{m} H_j$

***F*$_6$: Hybrid Composition Function**

Basic functions: f_1–f_2=Sphere function, f_3–f_4=Ackley's function, f_5–f_6=Griewank's function, f_7–f_8=Rastrigin's function, f_9–f_{10}=Weierstrass function

Properties:

- ♠ Multimodal
- ♠ Scalable
- ♠ Rotated
- ♠ A huge number of local optima
- ♠ Different function properties are mixed together
- ♠ Sphere functions give two flat areas to the function landscape
- ♠ $x \in [-5, 5]^n$, global optimum $x^*(t) = \vec{O}_i$, $F(x^*(t)) = H_i(t)$, $H_i(t) = \min \sum_{j=1}^{m} H_j$

5.4.4 Evaluation Criteria

Problems: Function F_1–F_6

Dimension: n=10, [5–15]

Runs/problem/change type: 20

Max_FES/change: $10{,}000 \times n$

Sampling frequency: s_f=100

Initialization: uniform random initialization within the search space

Global optimum: All problems have the global optimum within the given bounds and there is no need to perform search outside of the given bounds for these problems.

Non-dimensional change detection: Algorithm should detect the *non-dimensional* change by itself instead of informing the algorithm when a *non-dimensional* change occurs.

Dimensional change detection: Algorithm should be informed when a *dimensional* change occurs.

Termination: Terminate when reaching *num_change*.

1) Record absolute function error value $E^{last}(t)=|f(x_{best}(t))-f(x^*(t))|$ after reaching Max_FES/change for each change.

For each change type of each function, present the following values for $x_{best}(t)$ over 20 runs:

Average best, average mean, average worst values and STD.

Average best (Avg_best)= $\sum_{i=1}^{runs} \min_{j=1}^{num\_change} E_{i,j}^{last}(t)/runs$

Average mean (Avg_mean)= $\sum_{i=1}^{runs} \sum_{j=1}^{num\_change} E_{i,j}^{last}(t)/(runs \times num\_change)$

Average worst (Avg_worst)= $\sum_{i=1}^{runs} \max_{j=1}^{num\_change} E_{i,j}^{last}(t)/runs$

STD= $\sqrt{\frac{1}{runs \times num\_change - 1} \sum_{i=1}^{runs} \sum_{j=1}^{num\_change} (E_{i,j}^{last}(t) - Avg\_mean)^2}$

2) Convergence Graphs (or Run-length distribution graphs)

Convergence graphs for each problem for dimension n=10. The graph would show the median performance of the relative value $r(t)$ of $f(x_{best}(t))$ and $f(x^*(t))$ for total runs with termination by the Total_FES.

For maximization function F_1, $r(t)=f(x_{best}(t))/f(x^*(t))$; for minimization function F_2–F_6, $r(t)=f(x^*(t))/f(x_{best}(t))$.

5.5 Experiments and Results

5.5.1 Experimentation

As discussed in Section 5.4, there are 6 multimodal test functions. The first test problem F_1 is the dynamic peak generator with 10 and 50 peaks. The other problems F_2–F_6 are dynamic composition functions. Seven change instances are examined. They are small-step changes (T_1), large-step changes (T_2), random changes (T_3), chaotic changes (T_4), recurrent changes (T_5), recurrent changes with noise (T_6), and random changes with changed dimensions (T_7). They make a total 49 test instances. For every test instance, 60 changes are involved in each run and a problem will be tested on 20 independent runs.

We use the population size of 100 for all the test problems. 20% of the function evaluations are assigned to local search with 5 directions applied to 4 individuals, and the rest of function evaluations are for generating offspring by the UEP algorithm. The tournament size q for the selection phase of EP is 10. Other parameters of our algorithm include the interval of archive updating, the initial temperature of η, the clearing radii in the archives, and the thresholds with regard to premature convergence and complete loss of diversity. The first three are set to be 10

generations, 6, and 5 respectively. The thresholds can be set to approximately 1/100 and 1/1000 of the initial fitness deviation for the cases of premature convergence and loss of diversity.

5.5.2 Results

All the environmental changes are successfully detected in our experiments. The absolute function errors (See Appendix A) after reaching maximum number of fitness evaluations for all the changes are recorded. Average best, average mean, average worst values and standard deviations of the error values are listed in Table 5.2–5.7. Figure 5.2–5.8 show the convergence graphs for each problem (median run) for dimension n=10.

We can see that the proposed algorithm performs well in all the change instances of test functions F_2, F_4, and F_5. It is also very effective in change instances T_1, T_4, T_5, and T_6 of function F_1, but not equally advantaged in change instances T_2, T_3, and T_7. It has some difficulties in solving problem F_6, and it is not able to trace the changing optimum in F_3. While it is quite natural that the problems with small-step changes can be solved smoothly, it is interesting to see that the algorithm can handle chaotic changes as well. And we would also like to note that the recurrent changes are not meant to be easy for the algorithm because the optimal solution is definitely moving to another peak (in the sense of maximization problems) after every change. Nonetheless, the performance of the algorithm is not obviously flawed in such circumstances.

Table 5.2 Error Values Achieved for F_1

Peaks (m)	Errors	T_1	T_2	T_3	T_4	T_5	T_6	T_7
10	Avg_best	0.0054	0.0044	0.0043	0.0057	0.0110	0.0104	0.0032
	Avg_worst	35.009	51.032	47.041	13.96	47.763	54.099	52.805
	Avg_mean	5.7109	10.658	10.87	1.5033	8.2954	8.232	13.123
	STD	9.6761	13.851	13.499	3.0008	13.227	13.102	14.96
50	Avg_best	0.0063	0.0053	0.0050	0.0058	0.0197	0.0164	0.0037
	Avg_worst	26.538	50.227	44.899	13.497	21.09	27.041	43.844
	Avg_mean	5.7391	13.285	15.896	1.4109	2.2653	3.1577	12.703
	STD	6.8424	12.944	13.365	2.4466	4.239	5.6002	11.917

Table 5.3 Error Values Achieved for F_2

Errors	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Avg_best	0.1266	0.1383	0.13615	0.132	0.12985	0.1195	0.0896
Avg_worst	38.758	45.346	29.778	32.751	34.247	35.26	38.11
Avg_mean	6.2147	7.2236	4.9885	4.2067	3.5058	3.478	6.7124
STD	9.6292	11.024	8.245	7.5828	7.3318	7.5956	10.274

Table 5.4 Error Values Achieved for F_3

Errors	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Avg_best	0.1996	0.17025	0.17975	0.20085	0.16635	0.1431	0.1243
Avg_worst	512.53	504.83	501.49	555.05	507.77	506.33	492.96
Avg_mean	151.98	140.47	136.67	164.96	95.123	107.54	107.69
STD	190.71	182.71	183.86	216.4	151.82	158.36	160.33

Table 5.5 Error Values Achieved for F_4

Errors	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Avg_best	0.13325	0.1386	0.13335	0.13045	0.13	0.1118	0.0814
Avg_worst	37.581	47.009	36.414	34.924	31.496	35.28	46.404
Avg_mean	6.601	8.1906	7.1991	5.0355	3.121	3.5162	8.3141
STD	10.032	11.923	10.145	8.3325	6.6867	7.3484	11.558

Table 5.6 Error Values Achieved for F_5

Errors	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Avg_best	0.20075	0.18235	0.19615	0.2484	0.2035	0.184	0.17445
Avg_worst	44.887	54.133	36.438	39.928	55.669	56.092	56.026
Avg_mean	7.9042	10.091	7.2867	6.2507	8.2195	7.9011	10.779
STD	11.287	13.28	10.201	10.116	13.016	12.911	13.533

Table 5.7 Error Values Achieved for F_6

Errors	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Avg_best	0.1493	0.14815	0.15235	0.15565	0.1404	0.12375	0.13875
Avg_worst	94.921	73.431	58.111	50.242	100.68	84.709	237.13
Avg_mean	17.303	18.732	16.005	11.753	26.311	24.558	25.231
STD	22.801	19.006	17.397	13.594	29.318	26.794	48.063

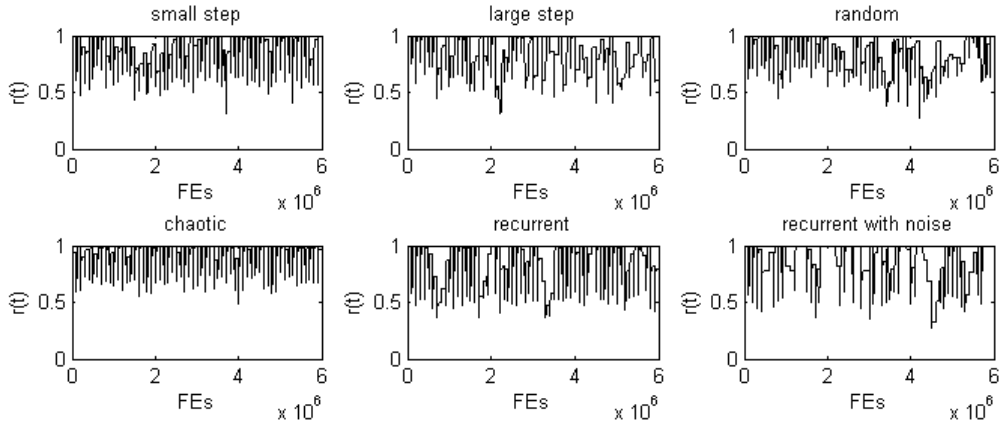

Figure 5.2 Convergence Graph for F_1 (10 peaks)

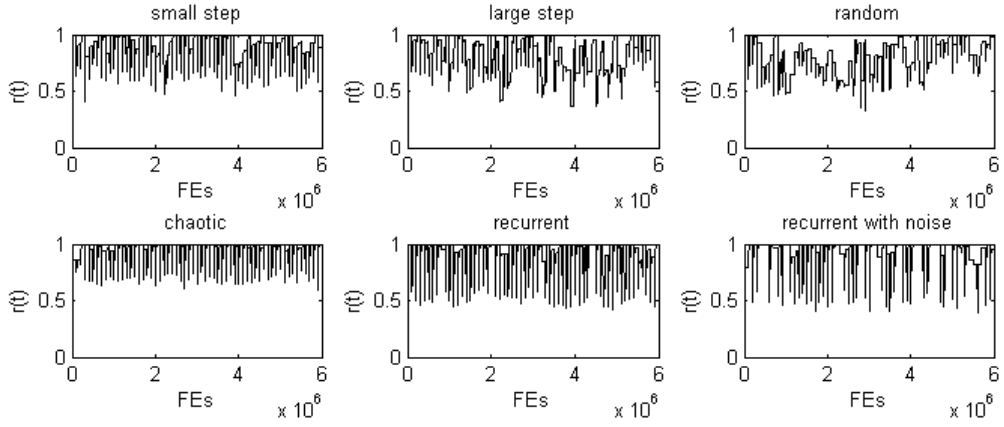

Figure 5.3 Convergence Graph for F_1 (50 peaks)

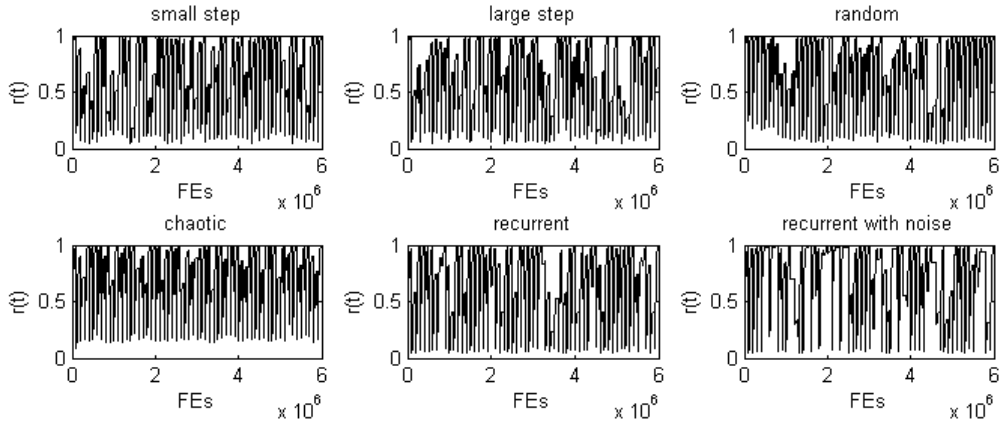

Figure 5.4 Convergence Graph for F_2

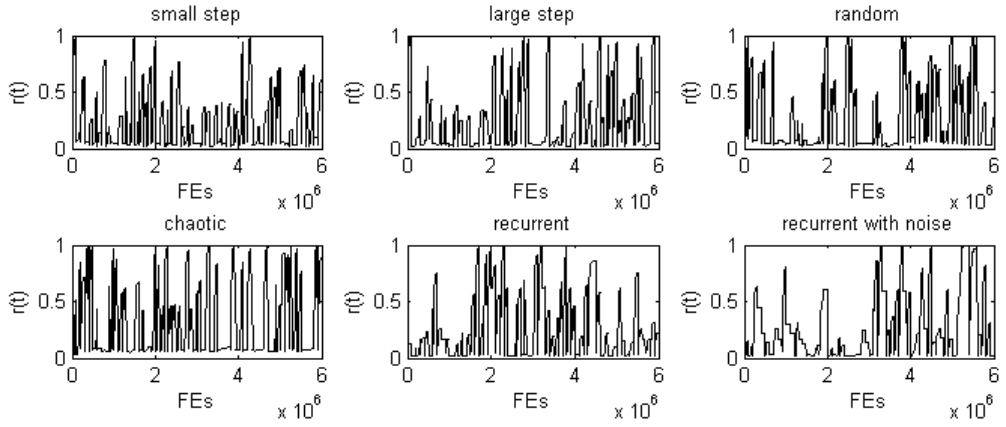

Figure 5.5 Convergence Graph for F_3

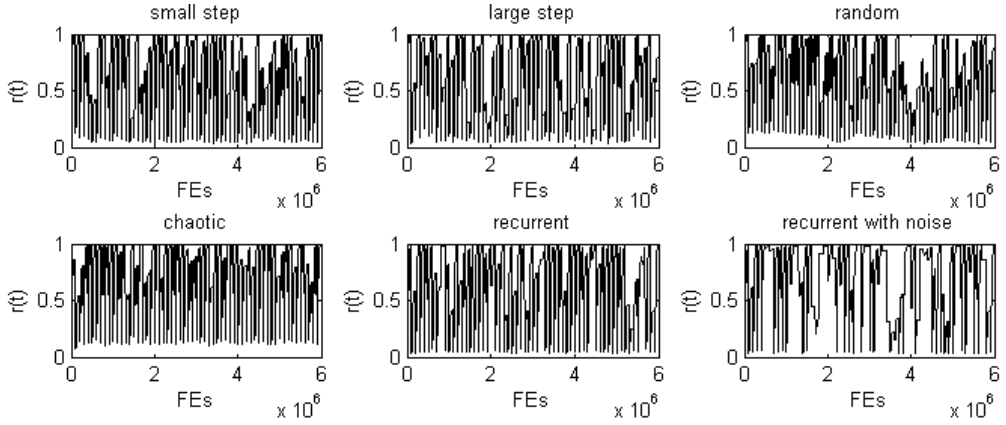

Figure 5.6 Convergence Graph for F_4

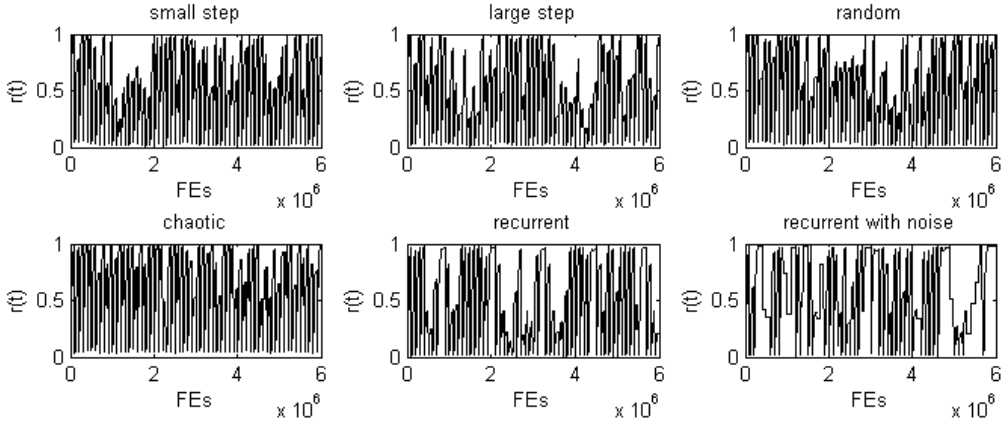

Figure 5.7 Convergence Graph for F_5

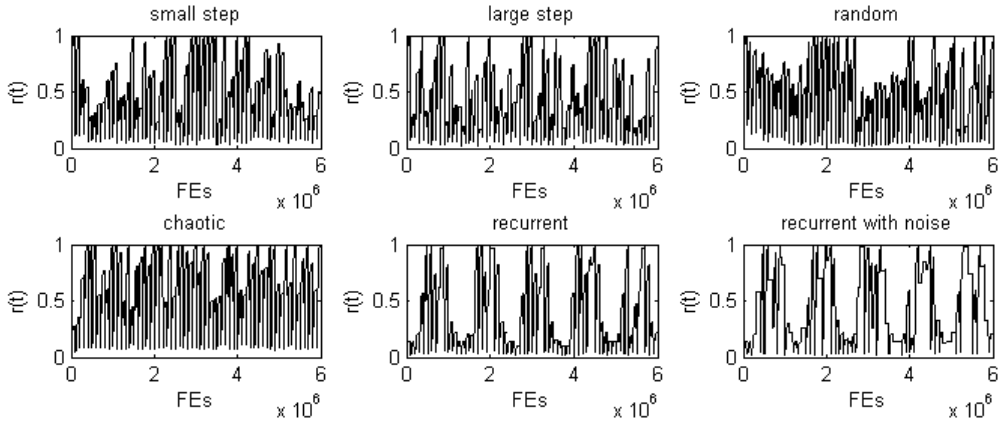

Figure 5.8 Convergence Graph for F_6

5.6 Conclusion

In this chapter, we proposed the evolutionary programming with an ensemble of external memories for dynamic multimodal optimization. The niching method which we apply to the memory populations helps us to select competitive and diverse solutions to be supplied to the evolving population. Though we use EP as the main algorithm, our implementation of the memory archives is general enough to be applied with other algorithms such as genetic algorithms and particle swarm optimizers.

We tested the proposed algorithm on the benchmarks of CEC 2009 Competition on Dynamic Optimization. It has showed suitability for the majority of the test problems, managing to trace the changing optimum in most of the time. It can be further improved to get better performance in dynamic optimization.

Chapter 6

Conclusions and Recommendations

6.1 Conclusions

This thesis focuses on the use of niching evolutionary algorithms for solving multimodal optimization problems.

Firstly, a brief review on evolutionary algorithms was presented along with several categories of niching methods. Three popular niching methods were experimentally investigated with respect to multimodal hybrid composition functions. The searching ability, accuracy, and computation time of the three methods were compared using the number of function evaluations as the main control parameter. It was shown that the performance of the niching methods varied with problems and every method had its own niche.

In Chapter 4, an ensemble of niching algorithms was proposed. To my knowledge, it was the first to employ different niching methods in one algorithm in a parallel way. Detailed description of the proposed ENA, including the selection of individual algorithms, the interactions between offspring generated by different niching methods, and the procedures were given. It has been demonstrated that when faced with an optimization problem, one can combine several existing niching methods that are competitive and complementary to each other to achieve better searching ability. Simulation results have confirmed that the proposed algorithm can be comparable to or better than the single niching methods on all the test problems.

In Chapter 5, multimodal optimization was investigated in a dynamic environment. A simulated-annealing-like dynamic strategy parameter was introduced to the unbiased evolutionary programming. The exponentially decreasing value of the parameter enabled the algorithm to have a vast search at the beginning and alter to fine search later. The property that it did not depend on information of previous generation made the parameter itself easy to adapt to environmental changes. A niching method was used in the external archives that regularly brought in diverse solution candidates to the population, preventing EP from getting stuck into local convergence. These modifications helped the proposed algorithm to respond efficiently to the challenges posed by the changing environment.

6.2 Recommendations for Further Research

Based on the research work conveyed in this thesis, I recommend several promising research directions that can be explored further.

- Developing niching evolutionary algorithms to locate the global optimum in multimodal functions:

  Up to now, the main effort is focused on the optimization problems that require us to locate multiple optima simultaneously. There are also interesting global optimization problems that have multimodal landscapes with numerous local optima. Measures have to be taken to ensure the population to be diverse enough, since getting stuck in one of the local optima is often an issue to be tackled when using the EAs. Techniques such as parallel populations in the ENA can also be applied.

- Extending the proposed ENA to allow more benefit from its best composing niching method:

  It is shown in Chapter 4 that ENA is able to locate more optima than its composing single niching methods for the 16 real-valued and binary test problems. The information regarding which niching method is more suitable for which problem is not exploited. In such circumstances, the same number of function evaluations is assigned to every composing method in the ENA. However, as the populations evolve in the algorithm, one may get some clue on which population is performing better. This involves developing a performance measure (or indicator), and then, according to this indicator one may assign the number of function evaluations differently to drive it faster in the best search directions.

- Extending the proposed ENA to other evolutionary algorithms;

- Applying niching methods in the both main population and the external archives for dynamic optimization; and

- Comparing niching techniques with spatial techniques.

Author's Publications

JOURNAL

E. L. Yu and P. N. Suganthan, "Ensemble of Niching Algorithms," *Information Sciences*, accepted, DOI: 10.1016/j.ins.2010.04.008. (Chapter 4)

CONFERENCE

L. Yu and P. N. Suganthan, "Empirical Comparison of Niching Methods on Hybrid Composition Functions," in *Proc. IEEE Congress on Evolutionary Computation*, Hong Kong, 2008, pp. 2194-2201. (Chapter 3)

E. L. Yu and P. N. Suganthan, "Evolutionary Programming with Ensemble of Explicit Memories for Dynamic Optimization," in *Proc. IEEE Congress on Evolutionary Computation*, Norway, 2009, pp. 431-438. (Chapter 5)

TECHNICAL REPORT

C. Li, S. Yang, T. T. Nguyen, E. L. Yu, X. Yao, Y. Jin, H.-G. Beyer, and P. N. Suganthan, "Benchmark Generator for CEC'2009 Competition on Dynamic Optimization," Technical Report, University of Leicester, University of Birmingham, Nanyang Technological University, October 2008. (Section 5.4)

References

[1] T. Bäck, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*, Oxford University Press US, 1996.

[2] K. A. De Jong, *Evolutionary Computation: A Unified Approach*, MIT Press, Cambridge, MA, 2006.

[3] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Evolutionary Computation 2: Advanced Algorithms and Operators*, Taylor & Francis, 2000.

[4] A. S. Fraser, "Simulation of genetic systems by automatic digital computers. I. Introduction," *Australian J. of Biol. Sci.*, vol. 10, 1957, pp. 484–491.

[5] A. S. Fraser, "Simulation of genetic systems by automatic digital computers. II. Effects of linkage on rates of advance under selection," *Australian J. of Biol. Sci.*, vol. 10, 1957, pp. 492-499.

[6] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

[7] G. J. E. Rawlins, *Foundations of Genetic Algorithms, Volume 1*, Morgan Kaufmann, 1991.

[8] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*, John Wiley & Sons, 1966.

[9] D. B. Fogel, *System Identification through Simulated Evolution: A Machine Learning Approach to Modeling*, Ginn, 1991.

[10] D. B. Fogel, *Evolutionary Computations: Toward a New Philosophy of Machine Intelligence*, Wiley-IEEE Press, 1995.

[11] H.-P. Schwefel, *Numerical Optimization of Computer Models*, John Wiley & Sons, 1981.

[12] D. B. Fogel, L.J. Fogel and J.W. Atmar, "Meta-evolutionary programming," in *Proc of the 25th Asilomar Conf. on Signals, Systems and Computers*, Pacific Grove, CA, 1991, pp. 540–545.

[13] K.-H. Liang, X. Yao, and C. S. Newton, "Adapting self-adaptive parameters in evolutionary algorithms," *Applied Intelligence*, vol. 15, No. 3, November 2001, pp. 171–180.

[14] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary Computation*, vol. 3, No. 2, Jul. 1999 pp. 82–102.

[15] C.-Y. Lee and X. Yao, "Evolutionary programming using mutations based on the Lévy probability distribution," *IEEE Transactions on Evolutionary Computation*, vol. 8, No. 1, Feb. 2004, pp. 1–13.

[16] I. Rechenberg, *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*, Dr.-Ing. Thesis, Technical University of Berlin, 1971. Reprinted by Frommann-Holzboog, Stuttgart, 1973.

[17] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies: a comprehensive introduction," *Natural Computing*, Vol. 1, No.1, 2002, pp. 3–52.

[18] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation," in *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, Piscataway, New Jersey, 1996, pp. 312–317.

[19] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, Vol. 9, No. 2, 2001, pp. 159–195.

[20] D. Lack, *Darwin's Finches*, Edition 2, Cambridge University Press, 1983.

[21] D. J. Cavicchio, *Adaptive Search Using Simulated Evolution*, Doctoral Dissertation, University of Michigan, Ann Arbor, 1970.

[22] K. A. De Jong, *An analysis of the behavior of a class of genetic adaptive systems*, Doctoral Dissertation, University of Michigan, 1975.

[23] S. W. Mahfoud, "Crowding and preselection revisited," in *Parallel Problem Solving from Nature (PPSN-2)*, 1992, pp. 27–37.

[24] S. W. Mahfoud, *Niching methods for genetic algorithms*, PhD Thesis, University of Illinois at Urbana-Champaign, 1995. IlliGAL Report No. 95001.

[25] G. Harik, "Finding multiple solutions in problems of bounded difficulty," University of Illinois at Urbana-Champaign, 1994, IIIiGAL Report No. 94002.

[26] W. Cedeño, V. R. Vemuri, and T. Slezak, "Multi-niche crowding in genetic algorithms and its application to the assembly of DNA restriction-fragments," *Evolutionary Computation*, Volume 2, Issue 4, 1994, pp. 321–345.

[27] M. Li and J. Kou, "A novel type of niching methods based on steady-state genetic algorithm," *Lecture Notes in Computer Science*, Vol. 3612, 2005, pp. 37–47.

[28] M. Li and J. Kou, "Crowding with nearest neighbors replacement for multiple species niching and building blocks preservation in binary multimodal functions optimization," *Journal of Heuristics*, Vol. 14, No. 3, 2008, pp. 243–270.

[29] D. E. Goldberg and J. J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," *Genetic algorithms and their application: Proc. Of 2nd Int. Conf. on Genetic Algorithms*, 1987, pp. 41–49.

[30] X. Yin and N. Germay, "A fast genetic algorithm with sharing scheme using cluster analysis methods in multi-modal function optimization," *Proc. of Int. Conf. on Artificial Neural Nets and Genetic Algorithms*, 1993, pp. 450–457.

[31] B. L. Miller and M. J. Shaw, "Genetic algorithms with dynamic niche sharing for multimodal function optimization," *Proc. of IEEE Int. Conf. on Evolutionary Computation*, New York, USA, 1996, pp. 786–791.

[32] D. E. Goldberg and L. Wang, "Adaptive niching via coevolutionary sharing," *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science*, 1998, pp. 21–38.

[33] J. Gan and K. Warwick, "A genetic algorithm with dynamic niche clustering for multimodal function optimization," *Proc. of the 4th Int. Conf. on Artificial Neural Nets and Genetic Algorithms*, 1999, pp. 248–255.

[34] D. Gong, F. Pan, S. Xu, "Adaptive niche hierarchy genetic algorithm," in: *TENCON '02: Proceedings of IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering*, 2002, pp. 39–42.

[35] C.-Y. Lin, W.-H. Wu, "Niche identification techniques in multimodal genetic search with sharing scheme," *Advances in Engineering Software*, Vol. 33, No. 11-12, 2002, pp. 779–791.

[36] A. Della Cioppa, C. De Stefano, and A. Marcelli, "Where are the niches? dynamic fitness sharing," *IEEE Transactions on Evolutionary Computation*, Vol. 11, No. 4, 2007, pp. 453–465.

[37] A. Pétrowski, "A clearing procedure as a niching method for genetic algorithms," *Proc. of the IEEE Int. Conf. on Evolutionary Computation*, New York, USA, 1996, pp. 798–803.

[38] C.-G. Lee, D.-H. Cho, H.-K. Jung, "Niching genetic algorithm with restricted competition selection for multimodal function optimization," *IEEE Transactions on Magnetics*, Vol. 35, No. 3, 1999, pp. 1722–1725.

[39] J.-K. Kim, D.-H. Cho, H.-K. Jung, C.-G. Lee, "Niching genetic algorithm adopting restricted competition selection combined with pattern search method," *IEEE Transactions on Magnetics*, Vol. 38, No. 2, 2002, pp. 1001–1004.

[40] D. Beasley, D. R. Bull, R. R. Martin, "A sequential niche technique for multimodal function optimization," *Evolutionary Computation*, Vol. 1, No. 2, MIT Press, 1993, pp.101–125.

[41] J. E. Vitela, O. Castaños, "A real-coded niching memetic algorithm for continuous multimodal function optimization," *Proc. IEEE Congress on Evolutionary Computation*, Hong Kong, 2008, pp. 2170–2177.

[42] M. Jelasity and J. Dombi, "GAS, a concept on modeling species in genetic algorithms", *Artificial Intelligence*, Vol. 99, No. 1, February 1998, pp. 1–19.

[43] J.-P. Li, M. E. Balazs, G. T. Parks, and P. J. Clarkson, "A species conserving genetic algorithm for multimodal function optimization," *Evolutionary Computation*, Vol. 10, No. 3, 2002, pp. 207–234.

[44] K.-C. Wong, K.-S. Leung, M.-H. Wong, "An evolutionary algorithm with species-specific explosion for multimodal optimization," *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, Montreal, Québec, Canada, 2009, pp. 923–930.

[45] J. Hu and E. Goodman, "Robust and efficient genetic algorithms with hierarchical niching and a sustainable evolutionary computation model," *Lecture Notes in Computer Science*, Vol. 3102, 2004, pp. 1220–1232.

[46] J. Hu, E. Goodman, K. Seo, Z. Fan, R. Rosenberg, "The hierarchical fair competition (HFC) framework for sustainable evolutionary algorithms," *Evolutionary Computation*, Vol. 13, No. 2, 2005, pp. 241–277.

[47] G. Dick, "A comparison of localised and global niching methods," in: *17th Annual Colloquium of the Spatial Information Research Centre (SIRC 2005: A Spatio-temporal Workshop)*, 2005, Dunedin, New Zealand, pp. 91–101.

[48] G. Dick and P. Whigham, "Spatially-structured sharing technique for multimodal problems," *Journal of Computer Science and Technology*, Vol. 23, No. 1, Jan. 2008, pp. 64–76.

[49] K. Sastry, H. A. Abbass, David E. Goldberg, D. D. Johnson, "Sub-structural niching in estimation of distribution algorithms," *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, Washington DC, USA, 2005, pp. 671–678.

[50] R. Roy, I. C. Parmee, "Adaptive restricted tournament selection for the identification of multiple sub-optima in a multi-Modal function," *Lecture Notes in Computer Science*, Vol. 1143, 1996.

[51] E. Dilettoso and N. Salerno, "A self-adaptive niching genetic algorithm for multimodal optimization of electromagnetic devices," *IEEE Transactions on Magnetics*, Vol. 42, No. 4, April 2006, pp. 1203–1206.

[52] S. Tabandeh, C. Clark, and W. Melek, "A genetic algorithm approach to solve for multiple solutions of inverse kinematics using adaptive niching and clustering,"

in *Proc. of 2006 IEEE Congress on Evolutionary Computation*, Vancouver, Canada, 2006.

[53] O. M. Shir, M. Emmerich, T. Bäck, "Self-adaptive niching CMA-ES with mahalanobis metric," *Proc. IEEE Congress on Evolutionary Computation*, Singapore, 2007, pp. 820–827.

[54] W. F. Sacco, M. D. Machado, C. M. N. A. Pereira, R. Schirru, "The fuzzy clearing approach for a niching genetic algorithm applied to a nuclear reactor core design optimization problem," *Annals of Nuclear Energy* 31, 2004, pp. 55–69.

[55] F. Streichert, G. Stein, H. Ulmer, and A. Zell, "A clustering based niching EA for multimodal search spaces," *Lecture Notes in Computer Science*, Vol. 2936, 2004, pp. 293–304.

[56] W. Sheng, S. Swift, L. Zhang, X. Liu, "A weighted sum validity function for clustering with a hybrid niching genetic algorithm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 35, No. 6, 2005, pp. 1156–1167.

[57] L. Wei, M. Zhao, "A niche hybrid genetic algorithm for global optimization of continuous multimodal functions," *Applied Mathematics and Computation*, Vol. 160, 2005, pp. 649–661.

[58] Y. Nagata, "Niching method for combinatorial optimization problems and application to JSP," *Proc. IEEE Congress on Evolutionary Computation*, Vancouver, BC, Canada, 2006, pp. 2822–2829.

[59] M. Tomassini, *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time (Natural Computing Series)*, Springer-Verlag New York, Inc., 2005.

[60] M. Bessaou, A. Pétrowski and P. Siarry, "Island model cooperating with speciation for multimodal optimization," *Lecture Notes in Computer Science*, Vol. 1917, 2000, pp. 437–446.

[61] D. Whitley, S. Rana, R. B. Heckendorn, "The island model genetic algorithm: on separability, population size and convergence," *Journal of Computing and Information Technology*, vol. 7, no 1, 1999, pp. 33–47.

[62] B. Manderick and P. Spiessens, "Fine-grained parallel genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms*, 1989, pp. 428–433.

[63] D. Whitley, "Cellular genetic algorithms," in *Proc. of the Fifth International Conference on Genetic Algorithms (ICGA)*, California, CA, USA, 1993, pp. 658.

[64] B. Sareni, L. Krähenbühl, and A. Nicolas, "Niching genetic algorithms for optimization in electromagnetics, I fundamentals," *IEEE Trans. Magnetics*, Vol. 34, No. 5, September 1998, pp. 2984–2987.

[65] E. Pérez, F. Herrera and C. Hernández, "Finding multiple solutions in job shop scheduling by niching genetic algorithms," *Journal of Intelligent Manufacturing*, Vol. 14, 2003, pp. 323–339.

[66] G. Singh and K. Deb, "Comparison of multi-modal optimization algorithms based on evolutionary algorithms," *Proc. of Genetic and Evolutionary Computation Conf.*, Seattle, 2006, pp. 1305–1312.

[67] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. -P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization," Technical Report, Nanyang Technological University, Singapore and KanGAL Report No. 2005005, May 2005.

[68] J. Gan and K. Warwick, "A variable radius niching technique for speciation in genetic algorithms," in *Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2000)*, San Francisco, USA, 2000, pp. 96–103.

[69] O. M. Shir and T. Bäck, "Niching in evolution strategies," in *Proc. Of Conference on Genetic and Evolutionary Computation*, 2005.

[70] B. Sareni and L. Krähenbühl, "Fitness Sharing and Niching Methods Revisited," *IEEE Transactions on Evolutionary Computation*, Vol. 2, No. 3, Sep 1998, pp. 97–106.

[71] K. Deb, "Genetic algorithms in multimodal function optimization," Master's Thesis, TCGA Report No. 89002, Department of Engineering Mechanics, University of Alabama, 1989.

[72] K. Deb and D. E. Goldberg, "An investigation of niche and species formation in genetic function optimization," *Proc. of International Conference on Genetic Algorithms*, Morgan Kaufmann, 1989, pp. 42–50.

[73] J.-P. Watson, "A performance assessment of modern niching methods for parameter optimization problems," *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, 1999, pp. 702–709.

[74] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," *Proc. 1999 Congress on Evolutionary Computation*, 1999, Vol. 3, pp. 1875–1882.

[75] D. E. Goldberg, K. Deb, J. Horn, "Massive multimodality, deception and genetic algorithms", in R. Männer, B.Manderick (Eds.) *Parallel Problem Solving from Nature* 2, North Holland, 1992, pp. 37–46.

[76] J. Horn, D. E. Goldberg, "Genetic algorithm difficulty and the modality of fitness Landscapes" In L. D. Whitley & M. D. Vose (eds), *Foundations of Genetic Algorithms* 3, Morgan Kaufmann. San Francisco, CA, 1995, pp. 243–269.

[77] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Computing*, Vol. 9, No. 1, 2005, pp. 3–12.

[78] C. MacNish and X. Yao, "Direction matters in high-dimensional optimisation," in *Proc. Congress on Evolutionary Computation*, Hong Kong, 2008, pp. 2372–2379.

[79] B. Suman and P. Kumar, "A survey of simulated annealing as a tool for single and multiobjective optimization," *J. of the Operational Research Society*, vol. 57, 2006, pp. 1143–1160.

[80] S. E. Carlson and R. Shonkwiler, "Annealing a genetic algorithm over constraints," in *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics*, vol. 4, 1998, pp. 3931–3936.

[81] J. M. Górriz, C. G. Puntonet, J. D. Morales and J. J. delaRosa, "Simulated annealing based-GA using injective contrast functions for BSS," *Lecture Notes in Computer Science*, Springer, vol. 3514, 2005, pp. 585–592.

[82] L. Fang, P. Chen, and S. Liu, "Particle swarm optimization with simulated annealing for TSP," *Proc of the 6th Conference on 6th WSEAS Int. Conf. on Artificial Intelligence*, *Knowledge Engineering and Data Bases*, vol. 6, Corfu Island, Greece, 2007, pp. 206–210.

[83] S. Bandyopadhyay, S. Saha, U. Maulik, and K. Deb, "A simulated annealing-based multiobjective optimization algorithm: AMOSA," *IEEE Trans. Evolutionary Computation*, vol. 12, No. 3, June 2008, pp. 269–283.

[84] E. Triki, Y. Collette, and P. Siarry, "A theoretical study on the behavior of simulated annealing leading to a new cooling schedule," *European J. of Operational Research*, vol. 166, 2005, pp. 77–92.

[85] J. Branke, *Evolutionary Optimization in Dynamic Environments*, Springer, 2002.

[86] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments—a survey," *IEEE Trans. Evolutionary Computation*, vol. 9, No. 3, June 2005, pp. 303–317.

[87] H. G. Cobb, "An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments," Naval Res. Lab., Washington, DC, Tech. Rep. AIC-90-001, 1990.

[88] W. Cedeño and V. R. Vemuri, "On the use of niching for dynamic landscapes," in *Proc. Int. Conf. Evol. Comput.*, 1997, pp. 361–366.

[89] S. Yang, "Explicit memory schemes for evolutionary algorithms in dynamic environments," *Evolutionary Computation in Dynamic and Uncertain Environments*, vol. 51, 2007.

[90] R. K. Ursem, "Multinational GAs: multimodal optimization techniques in dynamic environments," in *Proc. Genetic and Evolutionary Computation Conf.*, 2000, pp. 19–26.

[91] M. Wineberg and F. Oppacher, "Enhancing the GA's ability to cope with dynamic environments," in *Proc. Genetic Evol. Comput. Conf.*, D. Whitley et al., Eds., 2000, pp. 3–10.