A Probabilistic Cooperative-Competitive Hierarchical Search Model

by

Wong Yin Bun, Terence

A thesis submitted in partial fulfilment of the
requirements for the degree of
Master of Philosophy

in the

Division of Computer Science and Engineering
Graduate School
of
The Chinese University of Hong Kong

August 1998

ADVISORS

Professor Leung, Kwong Sak & Professor King, Irwin Kuo Chin

To me

Abstract

A Probabilistic Cooperative-Competitive Hierarchical Search Model

by

Wong Yin Bun, Terence

Master of Philosophy

The Chinese University of Hong Kong

Stochastic searching methods have been widely applied to areas such as global optimization and combinatorial optimization problems in a vast number of disciplines. To name a few, science, engineering, and operations researches. Representatives of these methods are *Simulated annealing (SA)*, Evolution type algorithms like *Genetic algorithms/programming (GA/GP)*, *Evolution programming/strategy (EP/ES)*, and so on. Their developments are all inspired from nature: physical annealing process, genetic, evolution, and ecology. Interestingly enough, they put little emphasis on the importance of the past searching information and the property of the landscape at the time of searching.

Motivated by this, a new probabilistic searching model is developed. On the structural aspect, the search space is divided into finite number of n-dimensional partitions. These partitions are then organized into a hierarchy. Subordinates are said to be representing the finer details of the landscape while the superordinates are said to be representing the gross structure of the landscape. This structural organization of the search space provides a foundation for the development of algorithms exploiting the dynamic viewing of landscape. On the algorithmic side, a population-based bottom-up self-feedback algorithm coupled with two key ideas stemmed from the nature: *Cooperation* and *Competition* is adopted. Although they are not new ideas in computational intelligence, they were used separately without emphasing their complementary nature.

In this thesis, the results on evaluating this algorithm empirically are presented. Numerical function optimization is used as the test bed owing to the simplicity and the ease of manipulation. Numerical functions of different characteristics are used to show its robustness. The first part of this thesis covers the current research done in the field. The second part of this thesis present the basic algorithm and illustrating its behavior. The third part of the thesis concentrates on the use of cooperation and competition to equip the basic algorithm. Finally, we show with experimental results that our model is versatile enough to work cooperatively with genetic algorithms.

合作與競爭並存之樹狀搜尋方法

黃彥斌
哲學碩士

計算機科學與工程學系
香港中文大學

隨機搜尋方法經已被廣泛地利用於不同學術領域之廣域性優化上，例如，自然科學，工程學，及運籌學。具代表性之隨機搜尋方法有模擬退火法，模擬遺傳法，模擬進化法。總的來說，這些方法之發展皆是啟發自大自然：自然退火法，遺傳學，及生態學等。

這些方法皆存有一個共同的特點。他們忽略了從過去搜尋得來的資料及問題本身之地勢之變動性。因此，本論文將會談論一個針對此問題而開發的隨機搜尋方法。從結構上來說，整個多維搜尋空間會被切割成若干個次元相同的小空間。不斷重複相同的切割，但每次小空間的大小不一。若把較大的空間放在高位，而較小的放在低位，這樣一樹狀結構便能構成。此結構之重要性在於建立基礎與將要開展之方法。從方法的層面看，它是一個多個體、從零碎到整體、及自我反饋的方法。再者，此方法採納了兩個重要之要素：合作與競爭。雖然它們並不是新意念，它們大多數被獨立地採用。因此，它們的互補能力將被論述。

此論文之第一部分概括了與此論題有關之研究。此論文之第二部分說明了此方法之基本要素，並展示其表現。此論文之第三部分集中說明合作與競爭怎樣增強在第二部分提出的基本方法。此論文之第四部分，亦即最後的一部分，將會介紹一個把本方法與模擬遺傳法結合的模型，及顯示其好處。

ACKNOWELDGEMENTS

Here, I especially want to thanks those who have helped, supported and influenced me, those who have given me chances to explore, those who were patient in waiting for my improvement, those who have provided excellent research environment and facilities, in particular computing facilities, those who have introduced me the technically excellent operating system — Linux, and those who have understood and have backed me up psychologically.

They are the Computer Science and Engineering department of my University, my supervisors Professor Leung Kwong Sak and Professor King Irwin Kuo Chin, the staff members of the department, my MPhil. colleagues — Mr. Hsieh Arthur, Mr. Cheung King Lum, Mr. Szeto Tom, Mr. Cheung Chi Chiu, Ms. Chan Polly, Mr. Wong Jason, Mr. Lam Sze Kin, Mr. Cheung Shing Kwong, Mr. Wong Yuk Chung, Mr. Fong Cedric, Mr. Kwok Chan Man, Ms. Leung Mary, and Mr. Chu Kam Wing, the course lecturer of my tutorial class Professor Moon Yiu Sang and Professor Leung's research fellow Professor Xu Zong Ben. Last but not least, I whole-heartedly thank my family, in particular my parents.

I sincerely thank for their help and support throughout my MPhil study!

This article was processed using the LaTeX macro package.

Table of Contents

List of Figures

List of Tables

Part I

Preliminary

CHAPTER 1

Introduction

Stochastic searching is a category of techniques commonly used in tackling large-scaled global optimization problems in many disciplines such as engineering, science and operations research. VLSI layout design, scheduling [31], resource/task allocation [32, 29], network optimization [54], optimal management [23] and even the complex optimizations of physical and biological systems are typical applications. Global optimization is to find an optimal set of objective variable instances in a search space constructed by enumerating the objective variables. Here is a definition of global optimization:

> Given $\mathcal{D} : \mathcal{X} \rightarrow \mathcal{O}$, where $\mathcal{X}$ is the set of all instances of the domain input variables (objective variables), $\mathcal{O}$ is the set of all output of the domain, and a criterion/evaluation $\mathcal{F} : \mathcal{O} \rightarrow \mathcal{R}$, where $\mathcal{R}$ is set of all 'scores' of the domain outputs, a global optimization problem $\mathcal{G} \circ \mathcal{F} \circ \mathcal{D}$ is defined as finding $x \in \mathcal{X}$ such that $r \in \mathcal{R}$ is maximized/minimized.

Usually, a fitness landscape/cost surface is said to be formed by the objective variables and their evaluated scores. This landscape/surface is usually huge and high dimensional, making the optimization task difficult. The difficulty is further increased by the high ruggedness (modality) of the landscape.

Different stochastic searching methods are proposed to handle global optimization problems. To name a few, Monte Carlo method (commonly known as pure random search), greedy descent/ascent with random multi-start methods, simulated annealing, and evolutionary methods. All these methods explore the search space directly to locate the optimal solution. Monte Carlo method is the simplest one which explores the landscape by taking samples from the solution space one in each iteration in a random fashion. Since this method is purely random, the global convergence of this method is guaranteed by keeping the best-so-far solution. This method is quite slow, though global optimal is guaranteed given infinite long time. Contrary to the Monte Carlo method, greedy descent/ascent methods are very fast local deterministic optimization technique exploiting local advantages only. Since the moves made by these methods are always

'downhill' or 'uphill', they end up at the local optima only. Hence, an iterative random restart strategy is used whenever a local optimum is reached, in order to jump out of the local optimum. Simulated annealing is a method simulating the physical annealing process that a sample obtained with less score is accepted with a certain probability (acceptance probability) while higher score samples are accepted with probability 1.0. The acceptance probability is subject to change monotonically by a cooling schedule. Again, global convergence is guaranteed given infinite amount of time. Evolutionary methods are originated from biological genetics and evolution that in each iteration, a new population is generated by successively applying genetic operators on the individuals of the old population. The standard set of genetic operators consists of crossover, mutation, and selection operators. Basically, all these methods keep solving the *static* landscape representing the whole search space of the problem at the highest resolution, and search without memorizing the past information obtained.

In this thesis, a new stochastic searching model is presented. It quantizes the continuous search space into partitions, which are organized into a binary hierarchy. Partitions located upper in the hierarchy represent larger portions of the landscape, and hence the gross details of the landscape. Partitions located lower in the hierarchy represent the landscape in finer details. By doing so, the benefits are twofold. Firstly, the rugged problem landscape can be smoothed, as the hierarchy allows different levels of resolution. The difficulty due to the ruggedness can be decreased. Secondly, it provides a basis to implement algorithms which dynamically change the 'view' of the landscape on the way of searching. The property of the landscape at different resolutions can be very different. For example, at low resolution (the macroscopic view), the landscape may be roughly unimodal, but at high resolution (the microscopic view), it may be highly multimodal. Thus, it is reasonable to adopt appropriate searching strategies at different resolutions. This involves the development of an algorithm capable of zooming at different resolutions dynamically. In this thesis, a stochastic bottom-up (population-based cooperative) self-feedback algorithm with memory developed to serve this purpose is presented. Moreover, this algorithm is designed to be general enough to rely on minimal *a prior* information about the problem. Hence, information is going to be gathered, memorized and used on the course of searching. The information gathering process is achieved cooperatively by a population of agents which sample the solution space simultaneously, in order to reduce the stochastic error.

When facing the high-dimensional problems and deceptive problems, the model is further equipped with two complementary strategies: *cooperation* and *competition*. By cooperation, we mean the cooperation among various objective variables to seek an optimum. Simply speaking, an objective variable instance, the presence of which can promote the score of a solution gets higher *cooperative fitness* than the one which cannot. Under this strategy, the searching is no longer a simple landscape traversal,

but a number of cooperative single-objective-variable traversal. In other words, the global optimization problem becomes finding a set of objective variable instances which have the highest cooperative fitness. However, it is a single-optimum-seeking method that only one global solution will be found. It would fail in the deceptive problems. Briefly, deceptive problem is defined as the presence of one or more than one deceptive attractors which mislead the search algorithm that they are the true global solution(s). From this definition, we can see that deceptive problem consists of a minimum of two optima. Tackling this kind of problem, redundancy and competition are introduced. By redundancy, we mean the presence of more than one population (subpopulation) to gather the landscape information separately. By competition, we mean the competition among these subpopulations. In the presence of competition, different subpopulations are forced to search diversely to occupy different areas of the landscape. The occupancy of a certain area of the landscape by a subpopulation is said to mask out that area. If there are several subpopulations, some of them will be attracted to and mask out the deceptive optima, leaving the global optimum prominent.

1.1 Thesis themes

1.1.1 Dynamical view of landscape

Observer staying outside of the searching process would conclude that he got the landscape of the problem. What they get is the *static* view of the whole problem space. However, as the algorithm converges, the region of interest gets more restricted and finer details of the landscape should be revealed. In other words, if we define landscape as the view of the algorithm on the problem at particular time, the landscape is said to be dynamically changing. In the rest of this thesis, we keep using this meaning for the word landscape. On referring to the original and the widely accepted meaning of landscape, '*static landscape*' is used instead. An immediate implication of this definition of landscape is that the property of the landscape is also changing dynamically. This is one of the motivations of this work.

1.1.2 Bottom-up self-feedback algorithm with memory

The search property of the memoryless algorithms would stay unchanged regardless of the changes in the landscape view. Take genetic algorithm as an example, the transition matrix which defines the probability transiting from one configuration to its neighbor stays unchanged throughout the evolution. That means how the algorithm act given a particular landscape is static and the information about the landscape is assumed to be available. This motivates the second theme of this thesis: gathering of landscape information on the course of searching, and using this information (feedback)

to further guide the searching. The use of bottom-up approach (cooperative work of a population of agents) is to increase the robustness.

1.1.3 Cooperation and competition

High-dimensional problems pose a challenge to the searching algorithms. Besides the exponential raise in the search space size, the dependency among dimensions decreases the extend of reducing search space size by heuristic. In the context of genetics, the phenotype of two genes can only be revealed when they come together. Existing technique dedicated to exploit this dependency is cooperation. In evolutionary computation, fitness is the sole measure to qualify the individuals in a population. Fitness in the context of the problem can be defined differently. To accomplish the dependency of dimensions, the fitness measure should be defined in such a way to promote the cooperative improvement.

Deception is another challenge to searching algorithms. This is a concept of relative sense. Roughly, a problem can either be deceptive or non-deceptive. It depends on the algorithm to be used. Pure random search is an exception, since it is the only one stochastic algorithm which can be classified as non-heuristic in nature. Any other algorithms are said to be heuristic and is bound to be deceived. Competition which originates in nature is employed in evolutionary computation to deal with this challenge. To fit into the model developed based on the first two themes mentioned above, a kind of competition is developed sharing some similarities to the *sharing methods* in genetic algorithms. One of the advantages of the developed competition model is the removal of the *niche radius* parameter.

1.1.4 Contributions to genetic algorithms

The motivation of this work comes from the missing characters found on some existing stochastic algorithms (GAs, SA, and MGDs). Thus, the model is designed to provide these missing characters. It is believed that the integration of the developed model and any one of the evaluated algorithms is beneficial. The last theme of this thesis is going to integrate them so as to gain the strengths from both.

1.2 Thesis outline

Chapter 2 The research of stochastic searching/optimization algorithms and related techniques are so mature that a complete survey involves huge amount of the relevant literatures. Hence, in this chapter, we only provide reviews on those aspects which are relevant to our main theme in large extend: several commonly used optimization algorithms namely pure random search, simulated annealing, multistart greedy approaches,

and genetic algorithms are discussed.

Chapter 3 The first half of the chapter is devoted to the model developed to serve the purpose mentioned previously: how the search space is partitioned and organized in a hierarchical way; how the self-feedback algorithm is designed with dynamic landscape view in mind; and what essential enhancement features are used. In another half of the chapter, experimental results are presented to illustrate the behaviors of the algorithm, giving some basic ideas of how the algorithm can be tuned. Finally, the strengths and the weaknesses of the algorithm is discussed.

Chapter 4 and 5 These two chapters covers two complementary techniques, which are cooperation and competition. The reasons why these two techniques are employed are stated separately in these two chapters. Experimental results dedicated to illustrate the usefulness of them are shown.

Chapter 6 Instead of contrasting the features of our model to the existing techniques, we look for the possibility of integration by making use of their similarities. In this chapter, we present how integration of our model and genetic algorithms (GA) is made possible. Experimental results are presented to show that the integration is beneficial.

1.3 Contribution at a glance

1.3.1 Problem

- Many problems in engineering, science, operation research, scheduling, planning and so on can be considered as optimization problems.
- Stochastic methods to optimization is found to be effective in handling large-scale problem with lots of local optima. Examples are simulated annealing, genetic algorithms, and multi-start greedy methods.
- These methods, however, have common drawbacks:
  1. They search (optimize) the solution space at the *highest resolution.*
  2. They search (optimize) *without memorizing past information* (or with weak memory only).
- The objective of this research is to tackle these drawbacks by providing a new model and methodology.

1.3.2 Approach

To achieve the stated objective, a new model called *Probabilistic Cooperative-Competitive Hierarchical Search (pccBHS)* is proposed. The main features are as follows:

- Partitioning of search space and organizing the partitions into a hierarchical structure.
- Exploitation of the partition hierarchy by an iterative stochastic bottom-up algorithm. The algorithm features feedback and memory.
- Incorporation of cooperation and competition *simultaneously* to cater for high-dimensionality and deceptiveness, in addition to making use of their complementary nature.

1.3.3 Contributions

- Provided 1) *a basis for resolution control*, 2) *search space smoothing* and 3) *search space reduction* by hierarchically organizing the search space and algorithmically utilizing the hierarchy.
- Introduced memory into stochastic search.
- Enhanced canonical GA by integrating the model with the GA.

CHAPTER 2

Background

2.1 Iterative stochastic searching algorithms

Iterative stochastic search is one of the important paradigms in global optimization. *Pure random search* (PRS), *multistart/singlestart greedy descent/ascent*, *simulated annealing* (SA) [39, 38] and *evolutionary methods* [33, 26, 21, 10, 40, 4] are well-known approaches of this paradigm. Before discussing their characteristics and the internal details, we show in Figure 2.1 the overview of this paradigm.

2.1.1 The algorithm

Search algorithms of this paradigm consist of four main parts sequenced in a cyclic way—the *generation* of new candidate solutions, the *evaluation* of the new solutions, the *selection* of solutions from both the current and the new solutions, and the *checking for termination conditions.*

- **Generation of new candidate solutions**

  The first part of the paradigm is the generation of new candidate solution set given a current one. New solution can be generated in two ways: (1) transforming the current solution(s) by specially designed operators, and (2) picking up from the solution space regardless of the what the current solution(s) is/are. Among the mentioned approaches, it is only the PRS that belongs to the later case. For evolutionary approaches, new solutions are generated by means of various genetic operators such as crossover and mutation. SA lies somewhat in-between PRS and evolutionary approaches that new solution is generated randomly similar to PRS, but it is only the solutions that belong to the neighborhood of the current solution are considered. It should be noted that the generation can be biased if heuristic about the problem to be solved is available (see [1, 2]).

- **Evaluation of the new candidate solutions**

Algorithm ITERATIVE STOCHASTIC SEARCH
Initialize $\mathcal{S}$
While *true*
$\mathcal{S}' \leftarrow$ generate a new solution set of size N given $\mathcal{S}$
$\mathcal{R}' \leftarrow$ evaluate each of the solutions in the new solution set $\mathcal{S}'$.
$\mathcal{S} \leftarrow$ select totally N solutions from both $\mathcal{S}$ and $\mathcal{S}'$ based on $\mathcal{R}$ and $\mathcal{R}'$.
if stopping criteria are met **then**
stop
End if
End while
End algorithm

FIGURE 2.1: ITERATIVE STOCHASTIC SEARCH

The second part of the paradigm is the evaluation of the new candidate solution set. This evaluation process reveals the quality of the solutions by giving score to each solution in the population. Strictly speaking, the evaluation process consists of two separate processes. The first of these two processes, which is compulsory, is to evaluate how good the solutions are in solving the objective problem. This process generates what we call *raw fitness*. Raw fitness is usually problem-specific and is meaningful within the context of the problem. Giving solely the raw fitness without associating it with a problem has no idea of how good the solution is. The second of the two processes which is optional (but commonly used) is to assign fitness values $f \in [0.0, 1.0]$ to the solutions. The range of this fitness is arbitrary, but the usual practise is to choose a range from 0.0 to 1.0 with 0.0 representing the worst and 1.0 the best. However, this fitness assignment process is employed only in evolutionary approaches which use fitness proportionate selection scheme, because algorithms such as SA and greedy descent/ascent do not normally need a quantified solution quality in the third process (discussed later): the selection process. In evolutionary approaches, there are various fitness assignment methods [5], such as *fitness ranking*, *fitness shifting*, and *fitness windowing*.

- **Selection of right candidates for the next iteration**

  The third part of the paradigm is a process which selects right candidates from both the new and the current solution sets. With the presence of the selection force, any mentioned search approaches will be able to converge to any optimum, as it is the ultimate convergence force generator. It should be noted that selection does not generate new solutions. It just provides a bias towards some solution instances.

Thus, designing an appropriate selection scheme for the problem at hand would become the design to compromise the convergence speed and the solution quality, i.e. the balance between exploration and exploitation. A scheme with strong bias (highly greedy) would increase the convergence speed and at the same time reduce the diversity. Premature convergence would be resulted. A scheme with weak bias would slow down the convergence speed, favoring diversification and hence the chance to converge to the global optimum. In both the evolutionary and greedy descent/ascent approaches, the convergence force is fixed once the scheme is chosen. However, it is not the case for SA. The cooling schedule used in it is a selection scheme which increases the greediness monotonically.

- **Checking for termination conditions**

  The final part of the paradigm is to check for the termination condition. Törn and Žilinskas [58] has given a detailed description of how stopping condition can be designed. As the name suggested, it is used to decide when the algorithm is going to be terminated. The design of the termination condition is crucial to the performance of all iterative stochastic searching algorithms in practise. It is because this kind of algorithms have no solution quality guarantee in finite time, so conditions have to be defined to determine the computation that is allowed to expense. The common termination condition is *or*-ing any of the following commonly used conditions:

  - **Condition 1**: The specified number of iterations is reached.
  - **Condition 2**: The prescribed error bound is reached.
  - **Condition 3**: The prescribed solution is reached.
  - **Condition 4**: The best-so-far solution persists for a fixed number of iterations.
  - **Condition 5**: The expectation (probability) of improving the best-so-far solution is too low to be realized by a reasonable amount of computation.

2.1.2 Stochasticity

As the name suggested, iterative stochastic algorithms are modeled in a probabilistic way. Generally speaking, the 'decisions' made in the algorithm, such as selecting, accepting and rejecting candidate solutions, are done in a probabilistic way. Since randomness is introduced in the algorithms, different results would be obtained given the same condition (except the random number seed). In devising general algorithm to tackle global optimization problems, we consider the target optimization problems as black-boxes with little or even no problem-specific information. It is very likely that the

problem to be solved are multimodal. Stochasticity would be a better approach over deterministic methods (such as greedy descent/ascent), owing to the fact that it is more capable of and effective in jumping out of the local optima.

- In simulated annealing, stochasticity is used in generating new candidate solution. In the unbiased case, all solutions belonging to the neighborhood of the current solution have equal chance of being selected. In the biased case, heuristic is used to favor for some solutions. Besides, the selection of the new generated candidate solution or the keeping of the current solution is subject to the current acceptance probability. The acceptance rule is generally specified as follows. Denote x and x' as the current and the new solutions,

$$\begin{aligned} Prob\{\text{keeping x}\} &= exp\left(\frac{-\{c(x') - c(x)\}}{T_g}\right). \\ Prob\{\text{accepting x'}\} &= 1 - Prob\{\text{keeping x}\} \end{aligned} \tag{2.1}$$

  where $c(.)$ gives the cost/fitness of the solution, T_g is the *temperature* at the iteration g.

- In evolutionary approaches, stochasticity is extensively used in the genetic operators. Crossover operators which mimic the exchange of genetic materials between two chromosomes determine crossover points (positions where crossover are taking place) in a probabilistic way. For example, in one-point crossover, two chromosomes picked out from the population will cross with each other with probability P_χ—the crossover probability—at a position along both chromosomes. If the chromosome length is l, crossover point may be any one from 1 to $l-1$. In the simplest case, this crossover point is chosen in a random uniform way. It is only when the heuristic is available or the way the problems are formulated that crossover points are chosen non-uniformly. Mutation is another operator that generates new chromosomes by transforming the genes of the chromosomes in the population. The transformation is determined by a mutation probability P_μ. It is usually set equal to $1/l$ where l is the chromosome length such that each chromosome of the population will be transformed into any one of its neighborhood probabilistically. Selection is another genetic operator with probabilistic nature. As discussed before, it provides a convergence force towards the desired area of the search space. Operating in a deterministic way would immediately turns it into a greedy method, so all selection operators, such as proportionate selection and tournament selection, are designed in such a way that the probability of a chromosome being selected depends on the chromosome fitness.

- The basic optimization method used in single-start/multi-start greedy descent/ascent can only find local optimal point owing to their greedy nature. In order to find

the global optimal point without exhaustive search, they randomly generate new starting point whenever local optimum is reached.

2.2 Fitness landscapes and its relation to neighborhood

2.2.1 Direct searching

When the searching is achieved by transforming a (set of) sample(s) from the sample space into another (set of) sample(s) by means of operators in a successive way (see Figure 2.1), it is said to be a *direct search* method. Greedy descent/ascent, simulated annealing, and evolutionary methods all belong to this paradigm. It should be noted that pure random search does not strictly have any operator to transform the samples obtained to samples elsewhere is the sample space, but it is also classified as a direct search method. It is because it is also a method that searches for the global solution by explicit sample space navigation. Törn and Žilinskas [58] also classifies random search methods (pure random search and its variants), simulated annealing and descent/ascent methods as direct search methods owing to the fact that these methods utilize only local information (function value, for function optimization problems). To elaborate, given the obtained samples, these algorithms traverse the sample space by using only the information available to generate samples in the next iteration. For simple (not necessarily easy) problems, such as function optimization, the samples plus their function values are the local information to be used by the direct search methods. The samples are used in the transformational process in generating new samples. While their function values are used in the acceptance (selection) process. Continuously repeating this process produces a trajectory from the initial sample points to the final one. The samples are said to be navigated explicitly by the operator(s) on the search space. Hence, search methods that are iterative, having operators to transform samples and the use of local information are direct search methods.

2.2.2 Exploration and exploitation

One of the central issues in direct search methods is the compromise between two contradictory search strategies: exploration and exploitation. Exploration means visiting of the untouched areas of the search space. It is essential to all search algorithms looking for global solution. Given finite time, there is no guarantee on no better result that can be found in the next (several) iterations. Usually, methods of this kind guarantee infinite time global optimality only [46]. In practical applications, constrained by the maximum allowed computation time, the optimal strategy is to maximize the chance of locating the global solution. To achieved this, maximizing the search coverage, i.e., exploration, is the most common way employed by the direct search methods to max-

imize the solution quality using no or minimal *a-prior* knowledge about the problem. Although this strategy is crucial to the problem-solving in global sense, the speed of generating better solution is rather slow. Exploitation, which is a greedy strategy, means making use of the information currently available to search for the best possible solution locally. The local search component of the searching algorithms are all considered as exploitation. For instances, hillclimbing, crossover operators of genetic algorithms, and greedy descent/ascent in single-start and multi-start random search are all components providing exploitation features. Exploitation can locate the local best solution very fast (compared to the exploration strategy). However, this strategy deals nothing to the optimality in global sense, unless the problem consists of one single optimum. We can see that the strengths of one strategy is the weakness of the another strategy. They have to be compromised to benefit from both. In fact, the compromise is difficult to make and is problem-dependent. Research was done in revealing the properties of the various implementations of exploration and exploitation, providing guidelines for practical use only.

2.2.3 Fitness landscapes

Given a search problem, a landscape representing all the possible outputs of the problem can be constructed. In appendix B, there shows several landscapes of the test problems used in this thesis. For one-dimensional problems (with one objective variable), their landscapes are lines with x-axis representing the objective variable values and y-axis representing the corresponding function values. For two-dimensional problems (with two objective variables), their landscapes are surfaces. The landscape of a problem tells several pieces of information. It illustrates not only the mapping of function variables and function values, but also the characteristics of the problem in relevance to searching, such as the modality and the deceptiveness. This information is normally used to describe the difficulty of the search problems. Generally, the higher the modality and the higher the deceptiveness of the problem, the more difficult the problem is. Modality describes the ruggedness of the landscape. Smooth landscape usually composes of one (unimodal) or a few modals (multimodal). Landscape that is highly or even massively multimodal consists of many or numerous modals. Figure 2.2 shows examples of landscapes of different modalities.

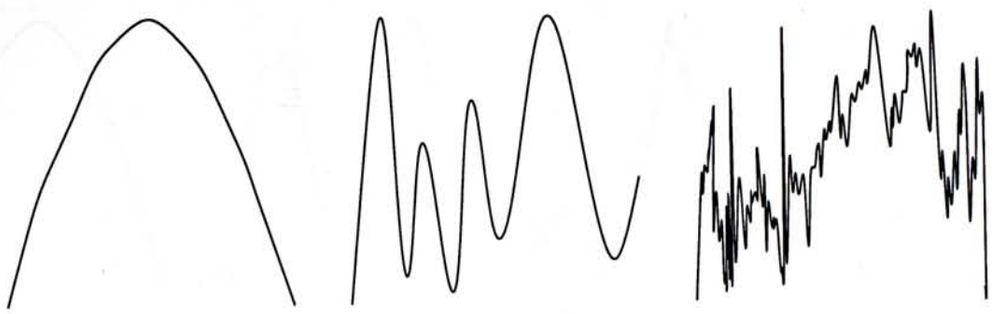


FIGURE 2.2: LANDSCAPES OF DIFFERENT MODALITIES. (A) UNIMODAL. IT CONSISTS OF A SINGLE OPTIMUM. (B) MULTIMODAL. IT CONSISTS OF MORE THAN ONE OPTIMA. (C) HIGHLY (MASSIVELY) MULTIMODAL. IT CONSISTS OF NUMEROUS OPTIMA. IT SHOULD BE NOTED THAT THERE IS NO CLEAR CUTTING POINTS BETWEEN THE THREE CATEGORIES OF MODALITIES.

Search problems with landscapes of high modality are difficult to solve by direct search methods. Recall that direct search methods search for the optimal solution by traversing the landscape explicitly. In the presence of many local optima, extensive exploration is required to avoid trapping in the local optima (due to exploitation). As mentioned before, exploration is a slow process, hence long searching time is expected to be spent to attain solution of reasonably high quality. Deceptive problems are those having local optima which are more favorable than the global optima. Since all practical searching algorithms are heuristic algorithms that some of them are more suitable in solving some kinds of problems, they would easily be deceived if the heuristic used is inappropriate. Besides the modality and the deceptiveness of the landscapes, the general shape of the landscape would also pose difficulty to search algorithms. To be more precise, some algorithms are more suitable to solve problems with landscape in certain shapes while some other algorithms are suitable for landscapes of another shape. In [7], landscapes are roughly classified into four different categories according to their shapes (see Figure 2.3):

- **Category 1** Big-valley with one single optimum.
- **Category 2** Smooth surface (with mid fluctuations) composed of several big and distinct local optima.
- **Category 3** Surface with many well-correlated local optima.
- **Category 4** Golf-hole (inverted) like.

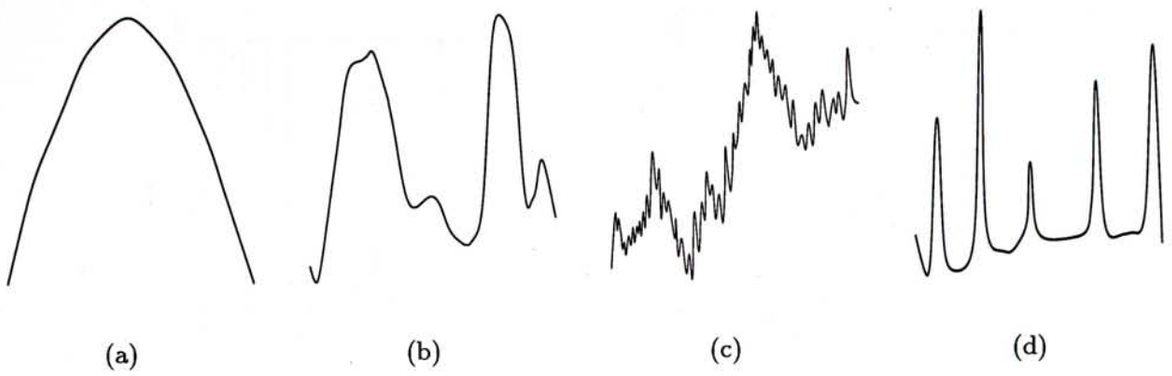
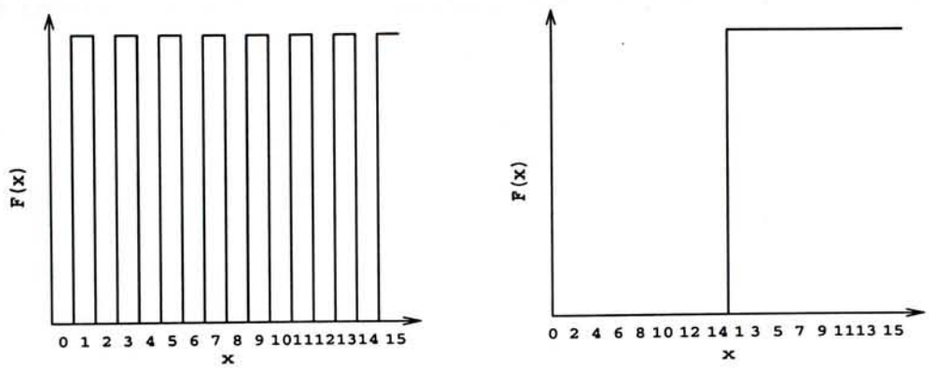


FIGURE 2.3: FOUR KINDS OF LANDSCAPES CLASSIFIED ACCORDING TO SHAPE. (A). UNIMODAL. (B). SMOOTH SURFACE COMPOSED OF SEVERAL BIG AND DISTINCT LOCAL OPTIMA. (C). SURFACE WITH MANY WELL-CORRELATED LOCAL OPTIMA (THE CORRELATION IS NOT SHOWN IN PRECISE AND ACCURATE WAY). (D). GOLF-HOLE (INVERTED) LIKE.

It is obvious that greedy methods, such as simple hillclimbing, are the most effective method in tackling problems in category 1, since the landscape has one single optimum that hillclimbing algorithms guarantee finite time optimality. It is because any better samples are definitely nearer to the global optimum. Although exploitation-only method (which is the common practise in hillclimbing algorithms) is enough in solving the problem, exploration is believed to be useful at the early stage of an optimization run when the initial sample(s) is/are poor. Landscapes of the second category favor for multistart strategies, since they consists of a few distinct and connected optima which make a few restarts to reach the true global optimum. Landscapes of the third category favor for simulated annealing and evolutionary methods, but not greedy methods. The presence of the numerous local optima makes greedy methods impossible to reach the global solutions without exhausting the search space. Simulated annealing and evolutionary methods are different. Simulated annealing is capable of escaping local optima by the random jump at high temperature and home in the (possibly) global optimum at the final stage of the cooling schedule. The forth category is the hardest to all searching algorithms. As the name suggested, landscapes of this category have uncorrelated optima resting on a relatively flat surface. The experience (in whatever way) obtained in the course of searching would have little helps in the future search. In other words, heuristics are not applicable in this kind of problems. Exhaustive search and random search are the last resorts.

Strictly speaking, the possible outputs of a problem is simply a set and the elements of which are unordered. Jones [37] pointed out that the formation of a landscape requires an ordering of the set in some way, otherwise, the landscape of a problem is undefined. So far, we have assumed the presence of the ordering, although it is not



(a) (b)

FIGURE 2.4: (A). LANDSCAPE SHOWN WITH ASCENDING x ORDER. (B). LANDSCAPE SHOWN WITH ASCENDING ODD x ORDER AND THE ASCENDING EVEN x ORDER.

important to know what the ordering is as far as the previous discussion is concerned. For numerical problems such as function optimization, the numerical ascending order is normally adopted. However, it is possible to have more than one ordering for a set and hence, more than one landscape for a single problem. In Figure 2.4, we show two different landscapes representing the same problem by two different orderings.

2.2.4 Neighborhood

Neighborhood is an important concept in iterative stochastic search. It lays the basis for the design and operation of the search operators. It also defines a landscape for the problem. In fact, neighborhood is essential to the performance of a searching algorithm. It is because neighborhood defines the landscape and hence, the modality, deceptiveness and the various landscape properties. The landscape of a problem can be multimodal (see Figure 2.4(a)), and at the same time be unimodal (see Figure 2.4(b)) depending on how neighborhood is defined. Simply speaking, given the set $\mathcal{X}$ of all solutions constituting the whole sample space, defining neighborhood is to define for each solution $x \in \mathcal{X}$ a solution set $\psi \subseteq \mathcal{X}$ which are admissible from x in one operation. The neighborhood defined for the problem shown in Figure 2.4(a) is $\psi_a = \{R_{\mathcal{X}}(x - 1), R_{\mathcal{X}}(x + 1)\}$ while that in Figure 2.4(b) is a bit complicated. It is basically $\psi_b = \{R_{\mathcal{X}}(x - 2), R_{\mathcal{X}}(x + 2\}$. Special handling of the boundary cases as required. The function $R_{\mathcal{X}}(.)$ is a remainder function.

Given the sample space of size V^n, there are V^n number of neighborhoods. The neighborhoods are overlapped meaning that some members of a neighborhood are the members of other neighborhoods. The cardinalities $|\psi_a|$ and $|\psi_b|$ are 2. In many cases, the cardinalities are larger than 2, particularly when the problems are of high dimension.

- **Pure random search** The neighborhood defined for pure random search is simple: $\psi = \mathcal{X}$, i.e., all samples are admissible in one single step.

- **Greedy decent/ascent** The neighborhood definition for greedy descent/ascent methods are the same as the generic one described before. The neighborhood size of a usual definition equals the dimensionality of the problem, i.e., $\psi = n$. Things become complicated when it goes to the multistart case. The size is n for the greedy part while that for the multistart is $\mathcal{X}$.

- **Simulated annealing** The neighborhood size depends on how the neighborhood is defined.

- **Evolutionary methods** Evolutionary methods are more complicated. Some members like genetic algorithms and genetic programming, use more than one operator in each iteration. The operators are crossover and mutation. Some practical implementations even use more than these. Designing an operator means defining a neighborhood for the whole sample space, and hence the landscape. The presence of more than one operator in action means introducing more than one neighborhood/landscape. Fortunately the operators are operated in a sequential manner with the input and output drawn from the same set of samples. Thus, the operators can be added together generating one single landscape only.

2.3 Species formation methods

Speciation can be thought of as a phenomenon of the formation of groups of individuals having distinct characteristics to the individuals in other groups. The formation of species is made possible by the genetic differentiation plus the selection force from the environment. The idea of speciation is borrowed by GA community to cater for the problems with landscapes that are multi-modal or deceptive in nature. A number of speciation algorithms were proposed, namely, *crowding* [11], *deterministic crowding* [43], *sharing* [28, 44], and *dynamic niching* [45].

2.3.1 Crowding methods

Crowding

Crowding [11] is a competition model to maintain population diversity to prevent premature convergence rather than a species formation method. It is an extension to the preselection method of Cavicchio [9]. Different varieties of individuals are maintained in the population, but the differentiation among individuals in not very distinct. Crowding is achieved as follows: for each offspring, parents from a set of size CF (crowding

factor) chosen randomly from the old population are compared for similarity. The most similar one with lower fitness than the offspring is replaced by the offspring. The rationale behind this method is that individuals which are similar are said to share similar niche and compete for the same resources. Expelling the similar individuals with lower fitness away from the niche effectively maintains a high quality and diverse population. Although less fit (and similar) individuals are kept replacing, only mild speciation effect is produced and it is empirically shown to be not effective in solving multimodal problems [12]. The computational complexity of this method is $O(nm)$.

2.3.2 Deterministic crowding

Developed by Mahfond [43] to improve the high computational requirement of DeJong's crowding method. One of the weaknesses of DeJong's crowding method is the expensive computation used in the replacement of parents by offsprings. The replacement method of deterministic crowding is that only parents and their direct descendants are compared for similarity and replaced, when the descendants have higher fitness. Generally, there are two offsprings that are produced for each pair of parents and hence two different parent-offspring pairings are possible. The one that has higher total similarity is used for replacement. The computational complexity is reduced from nm to $2n$.

2.3.3 Sharing method

With sharing (firstly introduced by Holland [33] and expanded by Goldberg and Richardson [28]), species are formed according to the *similarity* among individuals in the population and the pre-set *niche radius*. Similar to all other speciation methods presented so far, it is achieved by the competition of limited resources among individuals within the same niche. Individuals sharing the same (or similar) features, either genotypically or phenotypically, belong to the same niche and their fitness should be shared based on the following sharing function.

$$Sh(d_{i,j}) = \begin{cases} 1 - (\frac{d_{i,j}}{\sigma_{share}})^{\alpha}, & \text{if } d_{i,j} < \sigma_{share} \\ 0, & \text{otherwise} \end{cases}$$

where $d_{i,j}$ is the similarity distance between individuals i and j. This similarity distance is defined such that the more similar they are, the shorter the distance between them. This sharing function defines a symmetric niche area (e.g. circular niche area for two-dimensional function) with σ_{share} the radius and individual i the center. The parameter α allows the sharing function to have different shape. However, there is no literature reporting research result about this parameter and intuitively, this parameter should have limited usefulness and is normally set to 1.0.

Based on this sharing function, a set of individuals can be identified as sharing the same niche with an individual. Summing up all the sharing values of this set of

individuals become a *niche count.* This niche count determines the resulting fraction of the fitness remained:

$$f_{sh_i} = \frac{f_{raw_i}}{m_i} \tag{2.2}$$

We can see that the resulting shared fitness depends on i) how many individuals falling in the niche of the individual in consideration; and ii) the similarity distance of those individuals.

Regardless of the popularity and effectiveness of the sharing method, there are several issues that should be mentioned.

1. Estimation of the number of peaks of the function is difficult. It accounts for the availability of domain knowledge. This leads directly to the difficulty of determining niche radius σ_{share}.

2. The single fixed-value σ_{share} implies that niches are all of same size and at the same resolution level. In fact, most of the functions that people are interested in are highly unstructured with peaks of different sizes positioned irregularly.

3. The computational complexity required in calculating shared fitness and determining niche count is $O(n^2)$. These computation expenses sustain throughout the evolution.

2.3.4 Dynamic niching

Dynamic niching is developed by Miller and Shaw [45] to reduce computational requirement of the standard sharing method. The two assumptions that are used in the standard sharing method apply in this method: (1) the number of niches q can be estimated; (2) niches are separated (non-overlapped) and located at the same resolution level. The model is based on an observation on GA with sharing that individuals tend to populate the niches as time passes by. This phenomenon can be explained by the fact that once the niche radius σ_{share} is set, the number of niches is more or less fixed. Dynamic niching identifies these niches and uses these niches to categories the individuals, i.e., either belonging or not belonging to the niches. In the standard sharing method, all individuals in the population are considered as niche centers. For population of size N, there are fixed number of N niches. Similarity measurement will be taken by every pair of individuals. The complexity due to this exhaustive similarity measurement is $O(n^2)$. Based on the observation mentioned above, dynamic niche sharing firstly identifies q niches at the time concerned (that is why this method is named), and categorizes the individuals which belongs to the niches using the same similarity measurement. Since the number of niches is now q but not N, the complexity will be dropped to $O(nq)$,

$q \leq N$. A greedy approach is used to identify the niches (see Algorithm 2.1 for detail). Instead of having one niche count for each individual, there is one common niche count for each dynamic niche. Standard fitness reduction is then applied to each of the dynamic niches using their respective common niche counts. Those individuals which cannot be categorized are catered for by the standard sharing method.

Algorithm 2.1 Dynamic niche sharing: Greedy dynamic peak identification. The aim of this part of the algorithm is to identify a dynamic niche set, i.e., the set of top q number of peaks.

Algorithm Dynamic niching: Greedy dynamic peek identification

/* P^s is the sorted population.
* $\text{Sort}_{<,F}(P)$ sorts P in decreasing F order.
* q is a pre-set parameter indicating the number of dynamic niches.
* σ_{share} is the pre-set niche radius.
* $\mathcal{Y}$ is the dynamic niche set, the elements of which are individuals of the current population.
* $d_{a,b}$ is the similarity distance between a and b.
*/

$P^s \leftarrow \text{Sort}_{<,F}(P)$ /* Sort in decreasing fitness order */
$y = 0, \mathcal{Y} = \emptyset$ /* Initial dynamic niche set is null */
$i = 1$
Loop
  $j = 1$
  Loop
    if $d_{P_i^s, \mathcal{Y}(j)} > \sigma_{share}$ then
      Insert P_i^s into $\mathcal{Y}$
    end if
    $j = j + 1$
    $y = y + 1$
  until $j = N$ **or** $P_i^s \in \mathcal{Y}$
  $i = i + 1$
until $y = q$ **or** $i > N$

end Algorithm

The method is more efficient than the standard sharing method. The computational requirement in each generation is those required to identify the dynamic niche set plus the similarity measurements needed to categorize the population. Both of them have $O(nq)$ complexity. Generally, the initial population should be more or less evenly

distributed over the sample space. Most of them should be categorized as not belonging to the dynamic niches. Thus, the overall computational complexity should be $O(n^2)$ as if the standard sharing. Later, clusters begin to form and more and more individuals are being classified as belonging to the dynamic niches. The overall computational complexity should be dropped to $O(nq)$.

2.4 Summary

In this chapter, centering around three important algorithms, namely greedy descent, simulated annealing, and evolutionary algorithms, a brief overview of iterative stochastic searching algorithm is given. Iterative stochastic searching algorithm is a probabilistic algorithm which does not guarantee an optimal solution in finite time. There are three main components, namely the generation, the evaluation, the selection of candidate solutions and the checking of the fulfillment of the termination conditions. The important issue of iterative stochastic search lies on the idea of fitness landscape and its relation to neighborhood is presented.

Part II

Probabilistic Binary Hierarchical Search

CHAPTER 3

The basic algorithm

In this chapter, a stochastic searching model called *pBHS* (*Probabilistic Binary Hierarchical Search*) [41] is presented. Structurally, it transforms the optimization problem into a selection problem by quantizing and organizing the continuous search space into a *binary hierarchy* of partitions. Solving an optimization problem becomes locating the partition in which the optimal solution is resided through a series of branch selections in a top-down manner. Algorithmically, it is an iterative stochastic bottom-up searching algorithm with feedback and memory. In each iteration, samples obtained from the sample space are assembled together forming part of the past experience (memory). The memory is used in the next iteration as a searching guide to obtain new samples. Besides being a searching guide, the memory also acts as a convergence speed moderator–*adaptive remembrance scheme*. Using this iterative stochastic algorithm, information about the static landscape is accumulated. The behavior and the performance of this basic model is illustrated with the use of function optimization problems.

3.1 Introduction

Global optimization approaches under the category of stochastic methods such as simulated annealing (SA) [39, 38] and evolutionary algorithms (EAs) [33, 26, 21, 6] and those under the category of heuristic search methods such as multistart greedy descent strategies (MGDs) [58, 52, 34] have several characteristics: (i) for SA and MGDs, they try to find the global optimal solution by searching the large sample space in the finest detail; and (ii) they search without memorizing past global information. These characteristics could in some instances be undesirable. Motivated by their shortcomings, we provide our model as both an alternative and/or a complementary approach.

Direct searching for global solution Simulated annealing (SA) traverses a huge sample space from one configuration to another neighboring configuration to search

for the global optimal solution. Searching of a large sample space in this manner is inefficient, although global optimality is theoretically guaranteed given infinite amount of time. MGDs search for the global optimal solution simply by pushing every configuration picked out (randomly) to the reachable local optimum. The best local optimum obtained is treated as the global one. Theoretically, to the limit of trying all samples in the sample space, MGDs guarantee global optimality. Since the heuristic used in MGDs is local optimization technique, MGDs will have good performance particularly when the fitness landscapes of the problems are smooth and have few optima.

Lacking of good global information The difficulty of global optimization lies on a facts that global information is not available. Operators in SA, which move the existing configurations to the neighbors, can only exploit local advantages. The situation is slightly better for methods of population-based searching paradigm such as EAs, since they have a population of configurations. The occupancy of the population in the sample space is the result of past convergence force acted upon the population. Although the occupancy reflects to a certain extend the global picture, they are merely the sample points in the space recording limited past searching experience. Hence the reliability of which in obtaining global solution is rather limited. Moreover, methods that are dedicated to local optimization such as gradient descent and the variants are used together with multistart strategy hoping that one of the initial points can reach the global optimum. Unfortunately, method to determine initial points in order to maximize the expectation of getting the best (not necessarily the optimum) obtained solution is quite rare. Distributing uniformly the initial points on the search space becomes the best available strategy to maximize the coverage and hence to maximize the expectation of global optimization. We consider such approach rather pessimistic, despite its robustness and simplicity. Instead of waiting for the solution, we suggest that global information should be collected in the course of solution finding.

To deal with these limitations, we organize the sample space into a binary hierarchy separating the huge space into pieces of manageable size and adopt a paradigm called *emergent computation* [22]. Figure 3.1 illustrates the idea of emergent computation. The main theme of this paradigm is the emergence of global properties or patterns from the collective behaviors of many local interactions[1]. This paradigm paves the way for us to obtain the inaccessible global information.

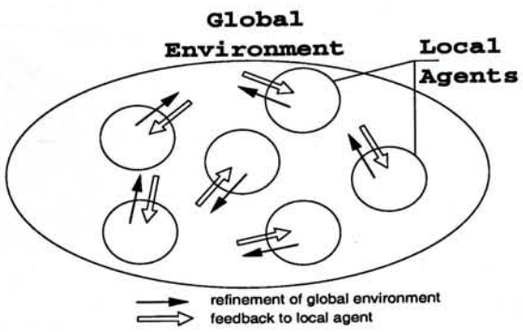


FIGURE 3.1: DYNAMICS OF EMERGENT COMPUTATION

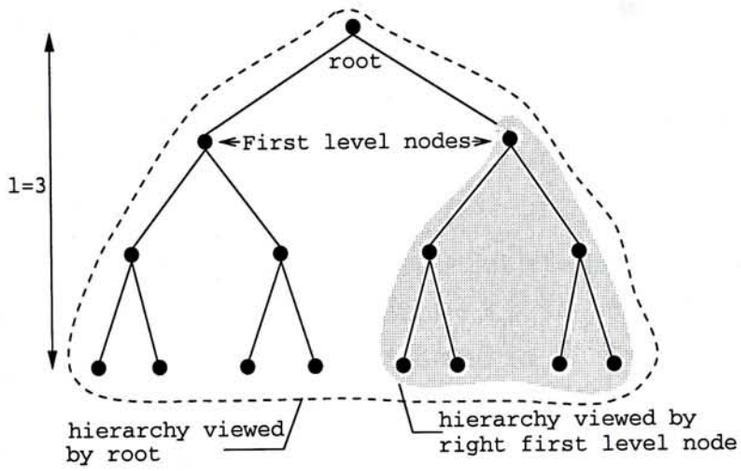


FIGURE 3.2: SEARCH SPACE REDUCTION (FIRST VIEW)

3.2 Search space reduction with binary hierarchy

Given a balanced binary hierarchy (Figure 3.2) of l levels[2], there are l number of branch layers, $l+1$ number of node layers and 2^l number of leaf nodes. Each non-leaf node has two branches radiated out. To locate a leaf node, we go through l number of branches starting at the root. If we need to make a decision on which branch to traverse next, we will have to make l number of such decisions. Since a branch of the hierarchy leads to a unique non-overlapping sub-hierarchy below it, after making a decision on the branch to go, in principle we just need to consider the corresponding sub-hierarchy in the next decision. Making decision at the root, we face the whole hierarchy with 2^l number of leaf nodes (the outer region enclosed by the dashed line in Figure 3.2). On deciding the subsequent branch at any one of the first level nodes, we just face 2^{l-1} (i.e. half of the whole hierarchy) number of leaf nodes (the shaded region in Figure 3.2). It is because half of the total leaf nodes are already pruned in the previous decision. It is clear that the size of the hierarchy we are facing is diminishing with the decision made

[1] Our current investigation modified the paradigm slightly in such a way to suit our problems at hand. Details are covered in the rest of the paper.

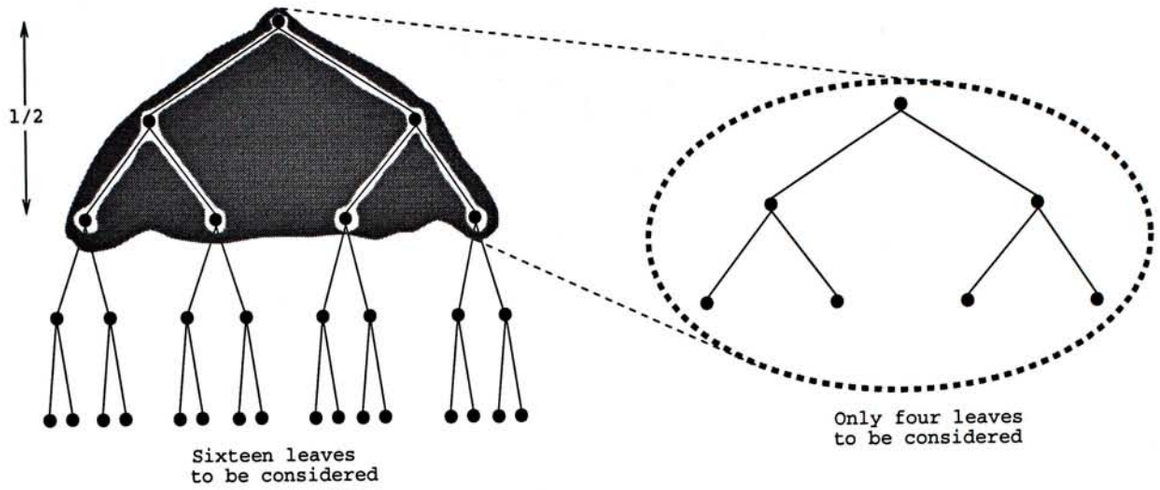


FIGURE 3.3: SEARCH SPACE REDUCTION (SECOND VIEW)

towards the bottom.

Viewing the hierarchy in another way, if we cut the hierarchy into 2 halves longitudinally at node level $\lfloor l/2 \rfloor$ as shown in Figure 3.3, the number of leaf nodes faced by all sub-hierarchies at the upper half are reduced to $2^{\lfloor l/2 \rfloor}$. Those in the lower half are, however, kept unchanged as mentioned before. In general, if we cut the hierarchy successively at each node level in a top-down manner, total number of 'leaf nodes' faced are $2l$. It can be seen that the apparent size of the hierarchy can be reduced drastically.

The formation of such a hierarchy basically defined $l + 1$ number of resolution levels of the solution landscape. Node level upper in the hierarchy represent a coarse landscape revealing the general macroscopic view, while node level lower in the hierarchy represent a fine landscape revealing the detail. This resolution hierarchy allows an algorithm designed to concentrate on the searching at the lower resolution, which is easier, locating the promising area first and to drive into the precise optimum later at the higher resolution when it is converging.

3.3 Search space modeling

In this section, we express our problem in terms of unconstrained function optimization (subject to bound constraint only). Given a n-dimensional continuous real-value function

$$\begin{aligned} F : X \longrightarrow \mathbb{R} \qquad & \text{where} \quad X \subseteq \mathbb{R}^n \\ & \text{and} \quad x^l \leq X \leq x^u \end{aligned} \tag{3.1}$$

to optimize, we need to find $x^* \in X$ such that $F(x^*)$ is maximized (or minimized).

[2]We define a 'level' as a layer of branches but not as a layer of nodes.

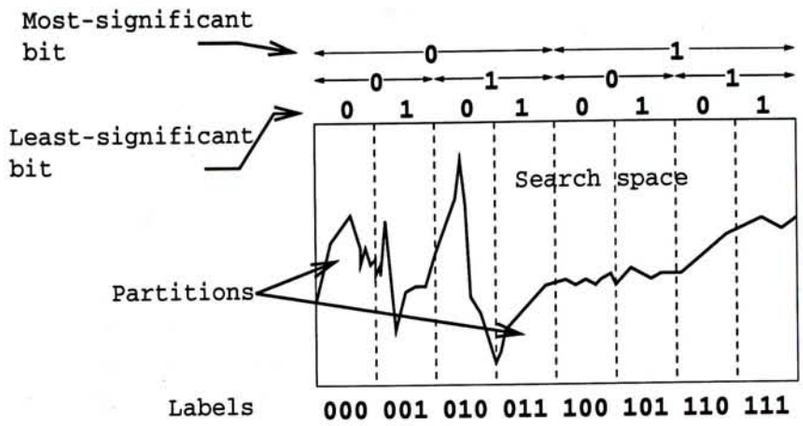


FIGURE 3.4: LABELING OF PARTITIONS: PARTITIONING

Depending on the required solution precision (precision of variables in x^*), we quantize the n-dimensional search space into V^n number of partitions (each of them is of n-dimension) of equal size. By creating this sample space with V^n partitions, the optimization problem can then be modeled as a searching and approximation problem with V^n number of choices. Imposing a restriction on V that it should be equal to 2^l where $l \in \mathbb{N}$, a binary number labeling scheme is then introduced to label the partitions. For the sake of simplicity, we restrict our consideration to the one-dimensional case first. Extension to the n-dimensional case, which is fairly straightforward, will be done afterwards.

Denoting S as the set of all binary strings of length l in the form of $b_{l-1}\, b_{l-2} \cdots b_0$, where $b_i \in \{0, 1\}$, we can label the partitions of the sample space of the one-dimensional function by assigning consecutive binary strings from 0 to $V-1$ to consecutive partitions as illustrated in Figure 3.4. For instance, if l equals to 3, the partitions are represented sequentially as 000, 001, 010, ..., 110, and 111 in an increasing x direction. Based on this labeling scheme, we noticed that the one-dimensional search space is not only divided into V partitions, but also a hierarchy of partitions with each bit demarcating the partition inherited from the immediate more-significant bit into two halves (see Figure 3.5). (The digits at the upper part of Figure 3.4 show the partition hierarchy formed by the different significant bits). The top layer (the most significant bit) consists of two bit-values which represent the right and the left half of the whole sample space. The second layer consists of four bit-values representing the four partitions divided from the two in the previous layer. Partitioning in this way allows us to treat each partition as a sequence of bits so that finding an optimal partition can be done by optimizing each bit. Then the problem becomes so simple that it accounts for just a series of l selections between 0 and 1.

Dealing with the n-dimensional functions, we simply apply the same labeling scheme to each of the variables in x. Then, having n number of variables, there are n number

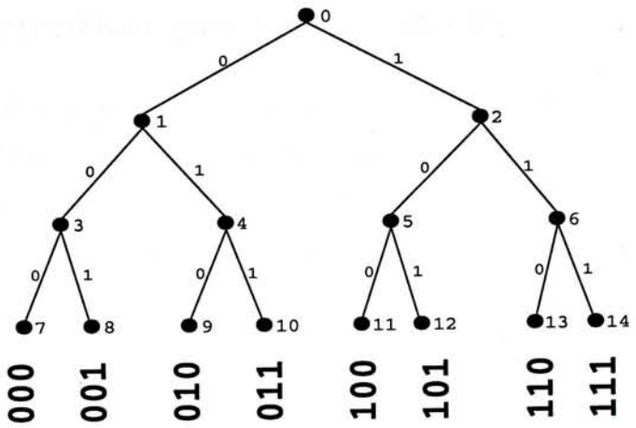


FIGURE 3.5: LABELING OF PARTITIONS: (B) FORMATION OF HIERARCHY

of such separate binary hierarchies. The optimization problem would then become n simultaneous series of l selections.

To locate the optimal solution, we need a way to explore the hierarchies. Exploring in an exhaustive way can give the optimal solution, but computationally it is unrealistically expensive. While exploring in a random fashion has an unacceptably low chance of getting the optimal solution. Probabilistic search, then, becomes a reasonable choice. To do the probabilistic search, we give scores to the states of each bit b_i. Since we are considering a binary system, two scores $a_{m,k}$ and $a_{m,k+1}$, $k = 2(l-1-i)$ are assigned one to each state indicating how well the states perform in that bit position in the past. Using these scores, a reasonable bit-value selection scheme (probabilistic search) becomes possible. We now restate our problem as follows:

The original problem is to find $x^* \in X$ where $X \subseteq \mathbb{R}^n$ such that

$$\forall x \in X \bullet \begin{cases} F(x^*) \geq F(x) & \text{if Maximization,} \\ F(x^*) \leq F(x) & \text{if Minimization.} \end{cases} \tag{3.2}$$

After the transformation, it becomes a problem to find probabilistically an optimal vector of binary strings $s^* \in S^n$ to where x^* belongs:

$$\begin{aligned} & \max \; Prob(\text{select } s^*) \\ = \; & \max \prod_{m=0}^{n-1} Prob(\text{select } s^*_m) \\ = \; & \max \prod_{m=0}^{n-1} \prod_{i=l-1}^{0} Prob(\text{select } b^*_{m,i}) \end{aligned} \tag{3.3}$$

where $s^*_m \in S$ is the m-th component in vector s^*,
$b^*_{m,i}$ is the i-th significant bit of binary string s^*_m.
It can then be re-formulated as finding $b^*_{m,i}$ such that for $0 \leq m < n$ and $0 \leq i < l$,

$$b^*_{m,i} = \arg \max_k \{ a_{m,k} : k = 2(l-1-i) + b_{m,i} \}. \tag{3.4}$$

3.4 The information processing cycle

To solve the problem formulated in the last section, we present in this section an iterative algorithm based on an information processing cycle characterized by a population of homogeneous searching agents and a searching environment. Specifically, a population of N binary string vector s each with n number of components s_m are generated and tested for optimality in each iteration. In the population, there might be some information related to the global picture of the objective function that we can extract. The information is gathered in each iteration of search to a reliable extend that the agents, based on this global information, can produce the optimal solution.

3.4.1 Local searching agents

Each agent is designed to generate in each time step n number of binary strings through n sequences of bit-value selection probabilistically. We treat the set of scores $a_{m,k}(t) \in [0.0, 1.0]$ at time t stated in Eq. (3.4) as our global information accumulated up to time t. For each function variable x_m, we define a vector

$$A_m(t) = [\ a_{m,0}(t)\ \ a_{m,1}(t)\ \ a_{m,2}(t)\ \ \cdots\ \ a_{m,2l-1}(t)\ ] \tag{3.5}$$

composed of $2l$ number of $a_{m,k}(t)$ (two consecutive $a_{m,k}(t)$ for one bit in binary string of length l). For an n-dimensional problem, the whole set of scores would be

$$A(t) = [\ A_0(t)\ \ A_1(t)\ \ \cdots\ \ A_{n-1}(t)\ ]^T. \tag{3.6}$$

In order to make the selection possible, a correspondence is drawn between $A_m(t)$ and our binary string s_m. Every non-overlapping pair of two consecutive $a_{m,k}(t)$ is used to represent a single bit. For instance, elements $a_{m,0}(t)$ and $a_{m,1}(t)$ correspond to the most-significant bit b_{l-1}, $a_{m,2}(t)$ and $a_{m,3}(t)$ correspond to the second most-significant bit b_{l-2} and so on. For each such pair of elements, we dedicate the former one as the score for $b_i = 0$ and the later one as the score for $b_i = 1$. For instance, $a_{m,0}(t)$ is the score of 0 in bit b_{l-1} and $a_{m,1}$ is the score of 1 in bit b_{l-1}. Figure 3.6 shows the correspondence of a binary string and $A_m(t)$. In fact, it is not necessarily that $A_m(t)$ and the correspondence be defined as above. Different applications can have different definitions.

Specifically, the generation of a binary string starts at the most-significant bit and proceeds one bit at a time towards the least-significant one, carrying the meaning of dividing the search space into half successively following the sample space hierarchy. The probability of selecting a bit-value at the i-th bit $b_{m,i}$ of the m-th string s_m is defined as follows:

$$Prob\ (b_{m,i} = \kappa) = \begin{cases} a_{m,k}(t), & \kappa = 0, \\ 1 - a_{m,k}(t), & \kappa = 1. \end{cases} \tag{3.7}$$

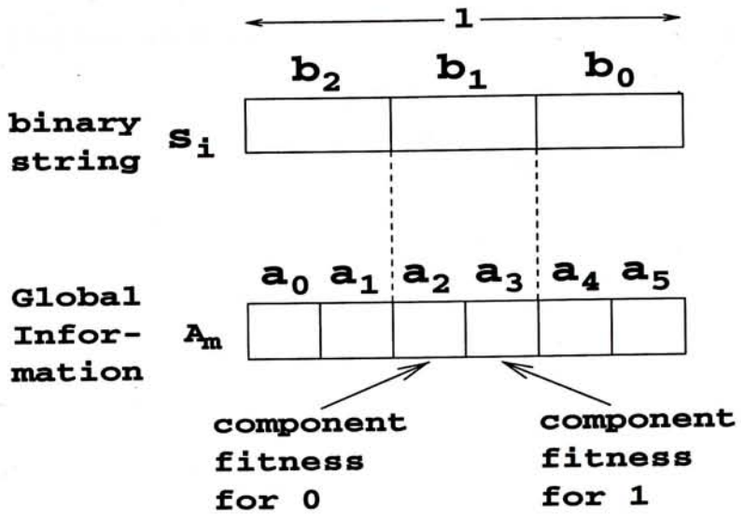


FIGURE 3.6: CORRESPONDENCE OF BINARY STRING AND THE RETAINED COMPONENT FITNESS LIST

As shown in Eq. (3.7), the selection of bit-value depends in a straightforward way on the respective global information complying with Eq. (3.3) and Eq. (3.4). The larger the $a_{m,k}$ value, the higher the chance the corresponding bit-value is selected.

After generating the binary strings $s_m, 0 \leq m < n$ for all function variables x_m, we have an n-dimensional partition picked out. Since the ultimate goal is to optimize the original function stated in Eq. (3.2), practically, we need a function value x from the partition for evaluation. The function value x for the partition is chosen according to:

$$x_m = \frac{s_m}{V}(x_m^u - x_m^l) + x_m^u, \tag{3.8}$$

i.e., the minimum x in the region. Unless specified otherwise, we use x_m and s_m interchangeably. Now we have a means of evaluating the partition by evaluating the representative instead.

3.4.2 Global environment

Given a reliable global information A^*, the searching agents described in the above section should be able to find s^* with probability approaching 1.0 fulfilling Eq. (3.3), i.e., $Prob(\text{select } s^*) \approx 1$. The question is how to make A^* reliable? We approach this question as follows.

Every binary string generated will be evaluated to give a function value $F(x)$. This function value is the raw fitness of the binary string vector. Assuming that the good performance of a binary string vector is contributed by the underlying components of each constituting binary strings, we assign the raw fitness of the binary string vector to the constituting components. The previously defiNed correspondence between A_m and a binary string basically treats each bit as a single constituting component. Then, two

vectors of length l for the raw fitness values of both states gained by a binary string are defined. We denote $u_m = [\, u_{m,0}\ u_{m,1}\ \cdots\ u_{m,l-1}\,]$ as a vector indicating the raw fitness of the bits with bit-values equal to 0 for s_m and $w_m = [\, w_{m,0}\ w_{m,1}\ \cdots\ w_{m,l-1}\,]$ as the vector indicating the raw fitness of bits with bit-values equal to 1. Fitness assignment to the states of each component is as follows:

For the m-th binary string s_m of the solution x, and $0 \leq i < l$,

$$\begin{cases} u_{m,i} = F(x) \text{ and } w_{m,i} = 0 & \text{if } b_{m,l-1-i} = 0, \\ u_{m,i} = 0 \text{ and } w_{m,i} = F(x) & \text{if } b_{m,l-1-i} = 1. \end{cases} \tag{3.9}$$

It is obvious that a single sample is not reliable enough in terms of getting the global view. Hence, we distribute a population of searching agents trying different partitions simultaneously. Their raw fitness values are added together forming another quantity called *component fitness*. The more partitions are tried, the more reliable the component fitness values are. The assembling is done in the following way.

For a population of size N, we have two sets of N raw fitness vectors u_m and w_m. Summation of all the same components of the N vectors of the respective sets gives the component fitness for the respective states. Denoting $u_{j,m,i}$ and $w_{j,m,i}$ as the raw fitness values for states 0 and 1 in the $(l-1-i)$-th bit gained from evaluating the j-th binary string in the population for variable x_m respectively, the component fitness values for both states of the $(l-1-i)$-th bit $b_{m,j,(l-1-i)}$ resulted from the population are:

$$U_{m,i} = \frac{\sum_{j=0}^{N-1} u_{m,j,i}}{|\Omega_{m,i,0}|}, \qquad W_{m,i} = \frac{\sum_{j=0}^{N-1} w_{m,j,i}}{|\Omega_{m,i,1}|}. \tag{3.10}$$

where $\Omega_{m,i,\kappa} = \{j \in \{0,1,\ldots,N-1\} : b_{m,j,l-1-i} = \kappa\}$ and $\kappa = \{0,1\}$.

While U_m and W_m are vectors with l number of vector components:

$$U_m = [\, U_{m,0}\ U_{m,1}\ \cdots\ U_{m,l-1}], \tag{3.11}$$

$$W_m = [\, W_{m,0}\ W_{m,1}\ \cdots\ W_{m,l-1}]. \tag{3.12}$$

Vectors U_m and W_m are normalized such that $U_{m,i} + W_{m,i} = 1$, $0 \leq i < l$. Putting U_m and W_m together, we obtain a vector of combined component fitness with the same structure as A_m:

$$H_m = [\, U_{m,0}\ W_{m,0}\ U_{m,1}\ W_{m,1}\ \cdots\ U_{m,l-1}\ W_{m,l-1}\,]. \tag{3.13}$$

Using this current component fitness values to make decision, the searching agents should be able to produce better binary strings, as they now have an immediate past searching experience to rely on. Continuously using the newly produced component fitness means forgetting the past searching experience except the immediate one. Instead of forgetting completely the past, we retain all the past information. The past component

fitness values for the m-th function variable are retained as follows: denote $h_{m,k}(t-1)$ as the k-th component of H_m at time $t-1$, $0 \leq i < l$,

$$a_{m,k}(t) \;=\; \beta_{m,i}(t-1) \cdot a_{m,k}(t-1) \;+\; (1-\beta_{m,i}(t-1)) \cdot h_{m,k}(t-1) \qquad (3.14)$$

where $k = 2(l-1-i)$ for state 0, and $k = 2(l-1-i)+1$ for state 1.

Practically, we keep every antagonistic pair inside $A_m(t)$ normalized: $a_{m,k}(t) + a_{m,k+1}(t) = 1$. The newly introduced quantity $\beta_{m,i}(t)$ is called *remembrance*. It determines the fraction of the past collected information $a_{m,k}(t-1)$ to be retained in the generation t. It is defined in such a way that different bits can have different remembrance values. There are two reasons why different bits should have different remembrances:

1. Intuitively, the more significant bits controlling larger common partitions should have more reliable information collected than the less significant bits controlling smaller shattered partitions given same number of samples tried. Losing more past information to accommodate for the new one at the more significant bits to increase the speed of convergence becomes plausible. Hence, the more significant the bit, the smaller the remembrance it should be.

2. The hierarchical structure has an advantage on search space reduction (see section 3.2). Briefly speaking, reduction occurs at a level of the hierarchy when sufficient information is collected at all upper levels. For instance, if the most-significant bit $b_{m,l-1}$ collected enough information, either $a_{m,0}(t)$ or $a_{m,1}(t)$ will have very high value. Say if $a_{m,1}(t)$ has a higher value, it is highly probable that the right partition contains the global optimum. Searching should then be concentrated on that region. In other words, the size of the search space is reduced by half, suggesting a smaller remembrance value be used to speed up the convergence.

Therefore, we devised an *adaptive remembrance scheme* to speedup the convergence. Let τ denote a threshold value above which means converged and vice versa and β denote the minimum allowed remembrance. Suppose the r-th bit $b_{m,r}$ of binary string s_m for function variable x_m is the first bit encountered starting from the most significant side that satisfies the following:

$$\mid 0.5 - a_{m,2(l-1-r)}(t) \mid \;>\; \tau \quad \vee \quad \mid 0.5 - a_{m,2(l-r)}(t) \mid \;<\; \tau. \qquad (3.15)$$

Then the remembrance value used in each bit of s_m is set according to:

$$\beta_{m,i}(t) = \begin{cases} \beta(t) & l > i \geq r, \\ \dfrac{r-i+\beta(t)}{r-i+1} & r > i \geq 0. \end{cases} \qquad (3.16)$$

This scheme, basically, keeps the remembrance for the converged bits (b_{l-1} to b_{r+1}) constant at β, while interpolates the rest from β to $(r+\beta)/(r+1)$. Figure 3.7 shows the remembrance settings at difference stages of convergence.

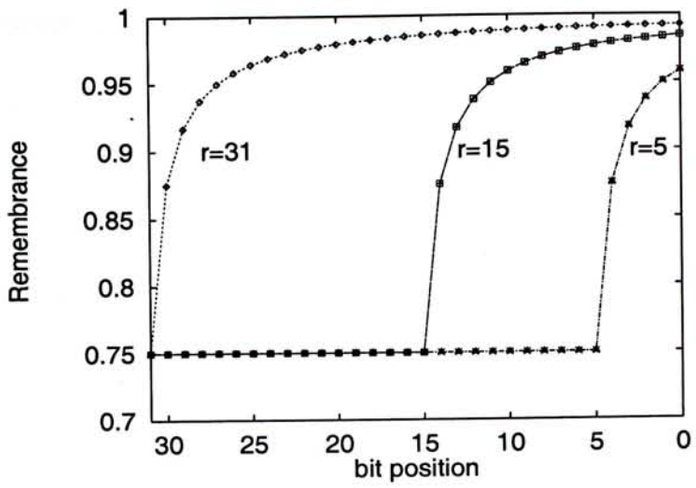


FIGURE 3.7: REMEMBRANCE FOR DIFFERENT BITS UNDER DIFFERENT STAGES OF CONVERGENCE

3.4.3 Cooperative refinement and feedback

It becomes apparent now that an information processing cycle is formed. The pseudo-code shown in Algorithm 3.1 summaries the basic algorithm comprising the two components: 1) a group of searching agents who search independently; and 2) a global environment. The algorithm starts out without any global information for the searching agents to rely on. The most promising searching strategy for the agents would then be random search. Gradually, global information emerges due to the interaction between the agents and the environment - the feedback from the environment to agents and the refinement of the environment by the agents.

Algorithm 3.1 THE INFORMATION PROCESSING CYCLE

```
Procedure InformationProcessingCycle
    global environment ← Empty
    While stopping criteria are not met
    Loop
        For each searching agent do
            search result ← Search( global environment )
        End For
        global environment ← Modify(collection of search result,
                                    global environment )
    End While
End Procedure
```

3.5 Enhancement features

3.5.1 Fitness scaling

The model should be able to handle functions with landscapes of various shapes and features at different scales. However, judging from the model described before, we noticed that the raw fitness used is not a good choice to achieve the goal just stated. It is because in order to better differentiate the excellence of the binary strings, their raw fitness should be different distinctively. When their raw fitness values are very similar or getting closer to each other, they cannot be distinguished easily. This leads to a problem of insufficient convergence power for flat landscape. Having a close minimum and maximum raw fitness values, flat landscape will mislead the algorithm that the population is converged to an optimum. The same problem exists in sharp landscape when the population is converged to a tiny point at the sharp peak. The more the population converged (but not yet reached) towards the peak, the more difficult to home in the optimal solution. We can see that convergence driven by raw fitness is not only landscape dependent, but also depends on the stage of convergence.

The same problem has already mentioned and tackled by [5, 6]. They analyzed a number of fitness re-mapping schemes such as fitness scaling, fitness windowing, and fitness ranking. However, their main concern is to find a good fitness re-mapping such that the fitness proportionate selection scheme in GAs would work well. They should then be very careful about the presence of a super-fit individual in a relatively poor population.

Our concern, as stated above, is to maintain a stable fitness range whatever the objective function is and wherever the current population is. Hence, we devised a very simple fitness measure capable of zooming at different scales (different resolution levels) to cope with both the prominent difference of fitness values in high gradient landscape and the nearly indistinguishable fitness values in plateau-like landscape. Even if the landscape is mixed with both features, the fitness measure can still provide equal and enough convergence power. The fitness measure used is defined as follows:

$$f_j = \frac{F(x_j) - F^{min}}{F^{max} - F^{min}} \tag{3.17}$$

where $F^{max} = \max_{0 \le i < N} F(x_i)$, and $F^{min} = \min_{0 \le i < N} F(x_i)$.

Effectively, raw fitness values equal to F^{max} will be scaled to 1.0, while those equal to F^{min} will be scaled to 0.0. No matter where the population goes and how close the population raw fitness values are, the relative fitness of each of the searching agents can best be revealed and hence enhancing the effect of information gathering. To achieve our goal stated at the beginning of this section, we modify the current component fitness

stated in Eq. (3.9) by the new scaled fitness f_j.

$$\begin{cases} u_{m,i} = f_j \text{ and } w_{m,i} = 0 & \text{if } b_{m,l-1-i} = 0, \\ u_{m,i} = 0 \text{ and } w_{m,i} = f_j & \text{if } b_{m,l-1-i} = 1. \end{cases} \tag{3.18}$$

3.5.2 Elitism

The generic model depends very much on the small number of searched samples. If the samples happen to be very poor, the global information gathered will be poor and produce poor and misleading guide to the future search. Successive dependence on the poor information would eventually cause the search to get stuck in the poor local optimum. Hence, we employ a well-known strategy in evolutionary computation to pull the deceived population away from the wrong guidance. The strategy used is *elitism* (see [30, 20, 19, 18] for the importance of this strategy.) which is a heuristic making use of the fittest individual found in the course of generations to guide the search. This heuristic is used with an assumption that the chance of finding a fitter elite is greater or equal at the region of the elite currently obtained than the region currently occupying. There are different variants of elitism. We implement ours as follows.

Throughout the searching process, we keep an elite $s^e \in S^n$ which is the best binary string vector found so far.

$$\begin{aligned} f^e &= F(x^e) \\ &= F((x^e), F_{max}) \\ &= 1. \end{aligned} \tag{3.19}$$

The elite is used as a reference to evaluate the population. In the presence of the adaptive fitness measure mentioned in section 3.5.1, the samples currently found will have fitness values relative to this elite: $F_{max} = F(x^e) = 1$. Regardless of what the raw fitness of the elite is, its scaled fitness is 1.0 and all other samples will have scaled fitness below 1.0. Effectively, this heuristic eliminates poor samples (in a relative sense) by giving them low relative fitness values. Poor samples will then share smaller amount of percentage in the component fitness $h_{m,k}$ than that revealed by their raw fitness values. Guidance provided by the the gathered global information will become more reliable. It should be noted that it is not a threshold type of elimination, but a smooth and gradual type. We have two reasons why this kind of elimination is a better choice than the threshold-type elimination:

1. *Poor samples are not necessarily bad.*

   They can be the source of new and better partitions. The presence of poor samples would stabilize the population and make the algorithm less greedy.

2. *Avoidance of an extra parameter*

   A clear-cut threshold type of elimination requires a parameter to determine where to do the elimination. In fact, we have no way to justify the use of any threshold value.

Apart from being a reference point for evaluation, the elite can take an active role in shaping the global information A as in [17]. We consider the elite an extra but standing searching agent of the population. In other words, the elite contributes to the component fitness $h_{m,k}$ as if a normal searching agent, i.e., depending on the bit values of the binary strings in s^e, $N/\mu \cdot f^{elite}$ is added to either one of the component fitness pair: $h_{m,k}$ and $h_{m,k+1}$. The quantity μ is the strength of elite. Factor N/μ keeps the strength of elite unchanged in different population size N. Without this factor, the effect of the elite will be overwhelmed by large population.

3.6 Illustration of the algorithm behavior

3.6.1 Test problem

In this section, trying to illustrate the basic properties, we present simulation results solving an artificial one-dimensional function AF1 (Figure 3.8). The artificial 1-D function is designed with the sub-optima and the exceptionally sharp global optimum far apart. Also, it has many local optimal points. This problem is designed for illustrating the algorithm behavior only. The question on the suitability of using the algorithm to solve this problem is out of the consideration. For $0.0 \leq x \leq 1.0$,

$$\begin{aligned} F_{AF1}(x) &= [1 - |x^2 - 1.8|]^{60} + \\ &\quad [0.3(1-x)\sin(30\pi(x - x^2 + 0.01))] + \\ &\quad [0.05\cos(66\pi(1-x^2) + 2.1))] + 0.35. \end{aligned} \tag{3.20}$$

The function is composed of three parts. The first part produces the exceptionally sharp peak at $x = 0.8999999999$. It is where the global maximum is located. The second and the third part are two amplitude-decreasing and frequency-increasing sine and cosine functions respectively. The latter one superimposes on the former one, creating local optima on the global landscape. Three optima that are of interest are listed in Table 3.1.

This experiment demonstrates the basic behavior of the algorithm by showing the global information gathering behavior. Since the purpose of this experiment is not going to determine the optimal parameter settings nor displaying completely the algorithm performance, we tried a set of parameters selected after carrying out the experiment described later in section 3.6.2: $N = \{10, 50, 90\}$, $\beta = \{0.00, 0.50, 0.95, 1.00\}$, $\mu = 1$, $\tau = 0.4$, $l = 32$, maximum allowed iterations T is 1500.

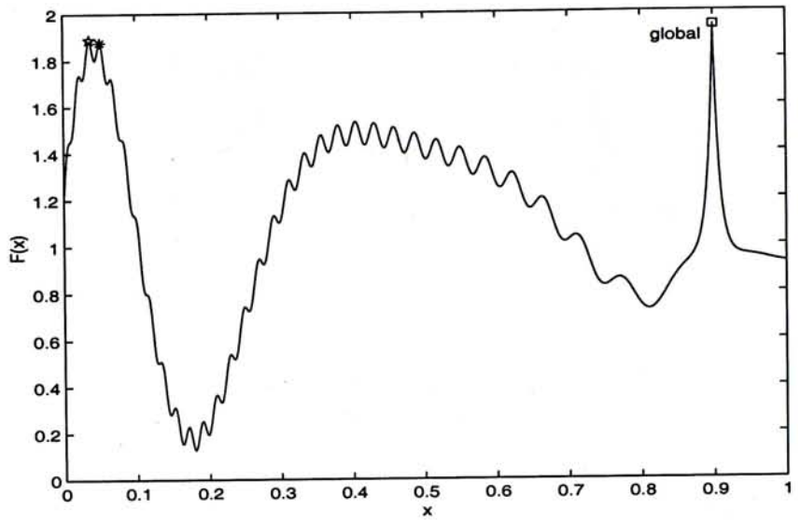


FIGURE 3.8: THE ARTIFICIAL ONE-DIMENSIONAL FUNCTION AF1

TABLE 3.1: OBJECTIVE FUNCTION VALUES AND LOCATIONS OF THE FIRST 3 OPTIMA

	f^+	x
Global optimum	1.9376390000600000	0.8999999999
First sub-optima	1.8861121654099478	0.0365754990, 0.0365754992, 0.0365754995
Second sub-optima	1.8770197336170074	0.0511608396, 0.0511608399

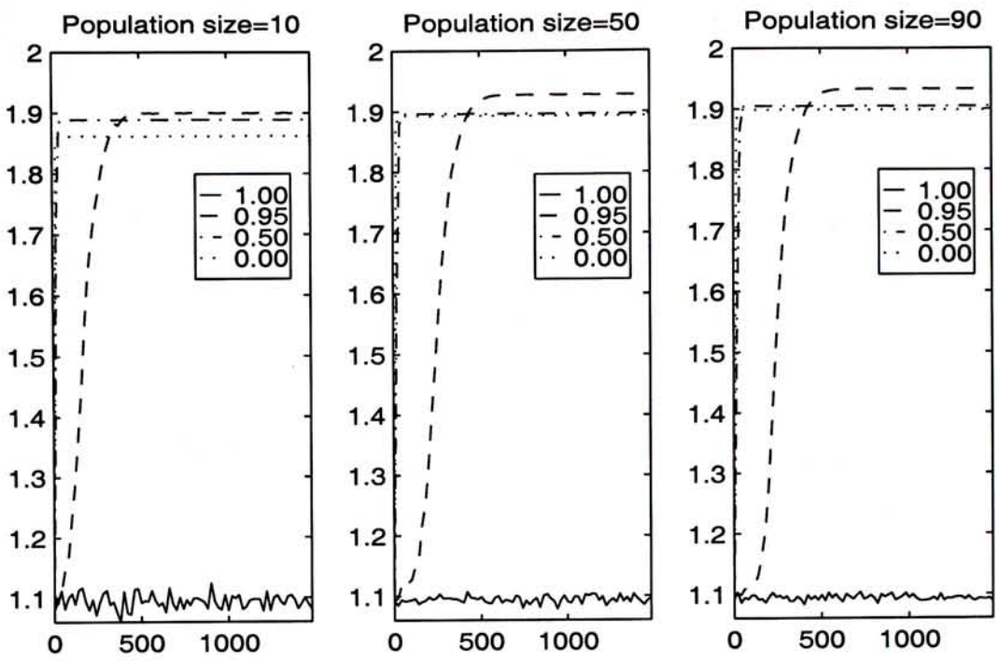


FIGURE 3.9: AVERAGE RAW FITNESS UNDER DIFFERENT REMEMBRANCES

Figure 3.9 shows the convergence under different remembrance. Remembrance equals to 1.0 behaves as if a random search that the average raw fitness shows strong jerkiness and stays at low values (≈ 1.1) without converging. With non-zero remembrance, all converge similarly reaching nearly the optimal values. Decreasing the remembrance from 0.95 to 0.0, the final converged average raw fitness drops, but the speed of convergence increases, reflecting the role of remembrance being a moderator between quality and speed.

Figure 3.10 compares the behavior of the algorithm under different population sizes. It is just another view of Figure 3.9. Another conclusion that can be drawn from Figure 3.10 is that using large population size, better result (on average) can be obtained.

Figure 3.11 shows the convergence profile of the best run in each of the parameter set. Besides supporting the arguments above, the profile demonstrates a resolution control behavior: convergence starts at the more-significant bit representing the landscape in lower resolution and finish at the least-significant bit representing the landscape in higher resolution. Proceeding from the 31-th bit to 0-th bit, the values of the global information $a_{m,k}$ approach to either one of the extremities steadily at the beginning and then quickly at the later stage.

3.6.2 Performance study

This experiment demonstrates the algorithm performance: 1) the effect of population size on the confidence of getting global optimum; and 2) the effect of remembrance

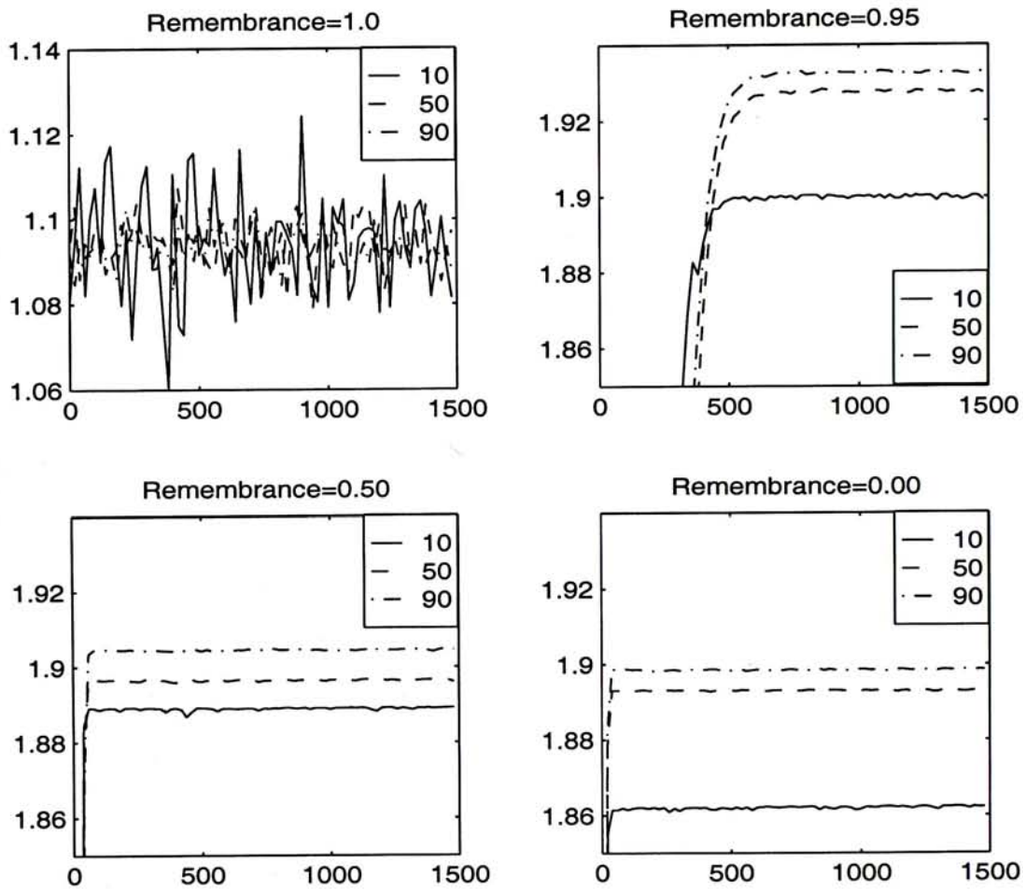


FIGURE 3.10: AVERAGE RAW FITNESS UNDER DIFFERENT POPULATION SIZE

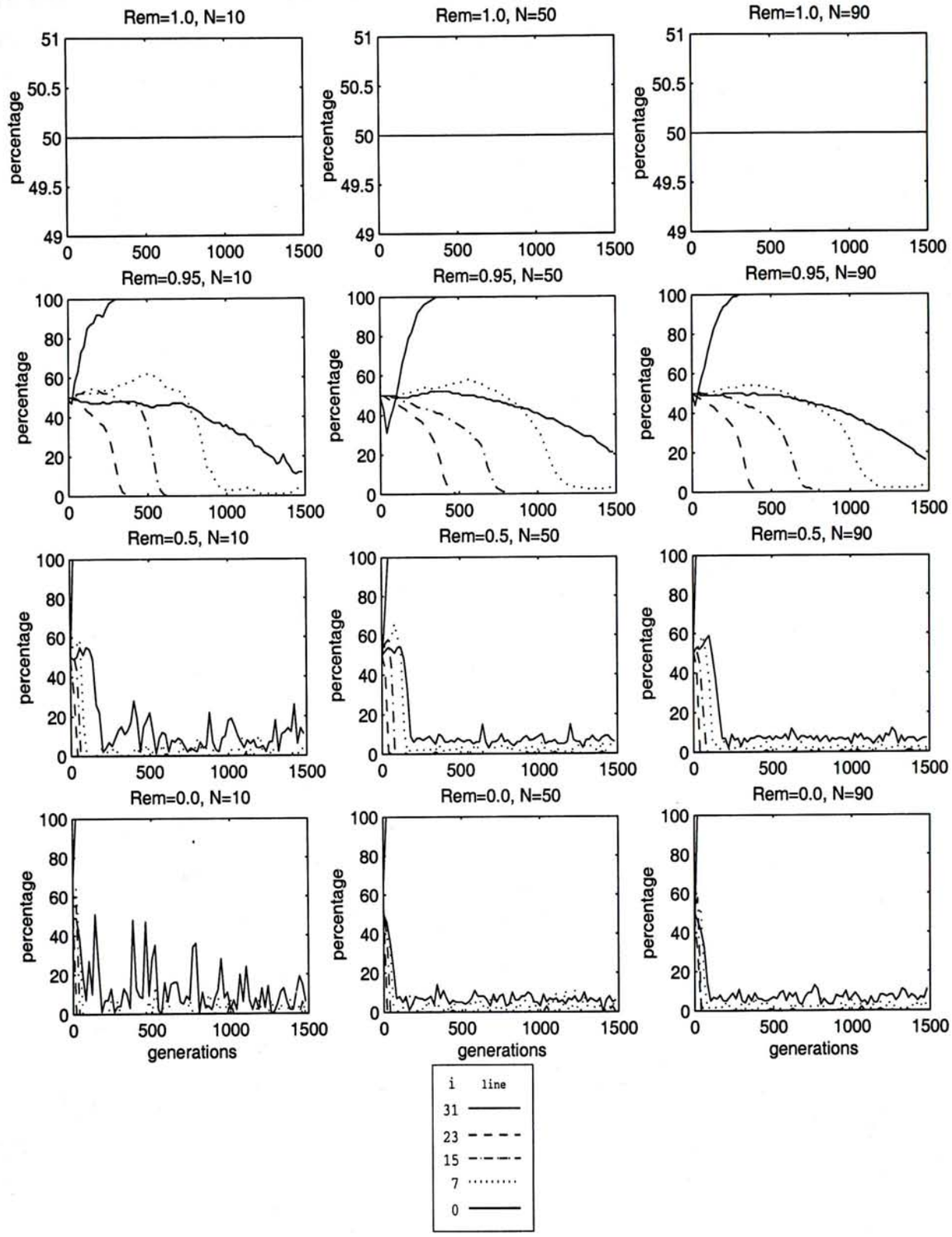


FIGURE 3.11: CONVERGENCE OF FIVE SELECTED a_{k+1}

on the confidence getting global optimum. The experimental settings are as follows: $N = \{10, 30, 50, 70, 90\}$, $\beta = \{0.00, 0.25, 0.50, 0.75, 1.00\}$, $\mu = 1$, $\tau = 0.4$, and $l = 32$. We tried each parameter set 100 times allowing maximally $T = 5000$ iterations. The same 1-D function is used. Tables 3.2 and 3.3 shows the percentage of runs out of 100 getting the global optimum and the first sub-optima. Throughout the experiment, we get none of the second sub-optima. In this experiment, getting the optimum means getting exactly the partition to where the optimum resides. In our case with bit-string of length 32, our goal is to get a solution with all 32 bits correct, i.e., search for one solution out of 2^{32}. Tables 3.4 and 3.5 shows the average number of iterations taken to get the optima.

TABLE 3.2: PERCENTAGE OF TRIALS GETTING THE GLOBAL OPTIMUM

β	N				
	10	**30**	**50**	**70**	**90**
1.00	0.00	0.00	0.00	0.00	0.00
0.75	13.00	20.00	29.00	45.00	58.00
0.50	7.00	21.00	22.00	24.00	34.00
0.25	9.00	13.00	16.00	14.00	18.00
0.00	9.00	13.00	10.00	11.00	22.00

TABLE 3.3: PERCENTAGE OF TRIALS GETTING THE FIRST SUB-OPTIMA

β	N				
	10	**30**	**50**	**70**	**90**
1.00	0.00	0.00	0.00	0.00	0.00
0.75	70.00	73.00	70.00	54.00	42.00
0.50	62.00	70.00	76.00	73.00	65.00
0.25	40.00	77.00	76.00	81.00	79.00
0.00	34.00	63.00	74.00	78.00	73.00

Effect of population size

Tables 3.2 and 3.4 show that by using a larger population size, the sharp and relatively difficult-to-find global optimum can be found with greater confidence while using less iterations. This implies that when large population size is used, each update of

TABLE 3.4: AVERAGE ITERATIONS REQUIRED TO REACH THE GLOBAL OPTIMUM

β	N				
	10	**30**	**50**	**70**	**90**
1.00	-	-	-	-	-
0.75	165.7	172.9	154.7	159.6	152.7
0.50	91.9	84.5	89.0	87.5	82.8
0.25	74.9	59.2	64.1	60.9	56.5
0.00	62.8	47.3	46.7	43.5	45.5

TABLE 3.5: AVERAGE ITERATIONS REQUIRED TO REACH THE FIRST SUB-OPTIMA

β	N				
	10	**30**	**50**	**70**	**90**
1.00	-	-	-	-	-
0.75	177.5	165.9	168.7	152.6	147.6
0.50	99.3	93.4	88.9	88.6	82.9
0.25	70.0	66.3	63.1	61.7	60.8
0.00	63.5	49.7	47.7	47.5	46.3

the global information is more effective in the sense of information quality. Increasing population size means increasing sample size per iteration and more information can then be gathered before every actual update, making the newly added information more reliable. However, greediness as well as convergence speed are reduced owing to the removal of bias produced by small population. In spite of this, the algorithm can still spend less iterations to produce the level of performance unattainable by small population size, reflecting the power of collective behavior. However more function evaluations are expected to pay for the low greediness.

Remembrance

The result we obtained shows the usefulness of the availability of global information on handling global optimization. In general, the performance on getting the sharp and narrow global optimum drops on decreasing β which can be explained as the decreasing dependence of global information. To illustrate this point, we can compare the results obtained for β equals 0.0 and the rest (except 1.0). With β equals 0.0, it is solely the current component fitness that is inherited to the next generation while losing completely those gathered previously. Having limited guidance provided by the current component fitness, the unsatisfactory performance of the algorithm is understandable. Despite the total loss of past global information except passing only the currently obtained information to the next generation, there is still a weak dependence of each generation to the past. This successive dependence explains the level of confidence achieved. Evolutionary algorithms (EAs) basically are the methods falling in this scenario.

TABLE 3.6: PERCENTAGE OF TRIALS GETTING THE GLOBAL OPTIMUM - LARGE REMEMBRANCE. (SHADED ENTRIES: > 95%))

β	N				
	10	**30**	**50**	**70**	**90**
0.99	74.00	99.00	100.00	100.00	100.00
0.98	55.00	90.00	97.00	100.00	100.00
0.97	39.00	82.00	89.00	99.00	100.00
0.96	38.00	69.00	87.00	95.00	98.00
0.95	32.00	63.00	86.00	96.00	97.00
0.90	21.00	42.00	59.00	70.00	80.00
0.85	19.00	27.00	51.00	55.00	73.00
0.80	11.00	20.00	43.00	49.00	54.00

Using large β, say 0.75, a significant portion (75%) of the gathered global information is preserved in each generation. In other words, most of the information obtained in many previous generations can be accumulated. The information becomes so rich that it gives a rather strong and correct guidance for the future search. Generally speaking, the larger the β value, the stronger the dependence of each generation to the past and the more reliable the gathered information will be. Hence the probability of selecting s^* stated in Eq. (3.3) can be increased.

Random search

The worst performance obtained with $\beta = 1.0$ is as expected (already illustrated in Figure 3.9). Remembrance equals exactly to 1.0 means none of the gathered information is lost, and at the same time, no information can be gathered. Since the algorithm starts out with equal a_k values for both states, the algorithm ends up selecting the states with equal chances, i.e., random search.

According to our argument on the advantage of using large remembrance, we look further into the performance with remembrances between 1.0 and 0.75. Keeping all the experimental settings unchanged, we tried another two sets of remembrance values: $\beta = \{0.80, 0.85, 0.90, 0.95\}$ and $\beta = \{0.96, 0.97, 0.98, 0.99\}$. The results shown in Tables 3.6, 3.7, 3.8 and 3.9 demonstrate the improved solution-finding capability of the algorithm, supporting our argument on using large remembrance values. Table 3.10 summaries the algorithm behavior that using large population and remembrance can improve the global solution finding performance of the algorithm.

TABLE 3.7: PERCENTAGE OF TRIALS GETTING THE FIRST SUB-OPTIMA - LARGE REMEMBRANCE.

β	N				
	10	**30**	**50**	**70**	**90**
0.99	26	1	0	0	0
0.98	45	10	3	0	0
0.97	61	18	11	1	0
0.96	62	31	13	5	2
0.95	68	37	14	4	3
0.90	75	58	41	30	20
0.85	72	73	49	45	27
0.80	72	79	56	50	46

TABLE 3.8: AVERAGE ITERATIONS REQUIRED TO REACH THE GLOBAL OPTIMUM - LARGE REMEMBRANCE

β	N				
	10	**30**	**50**	**70**	**90**
0.99	2,875.9	2,647.8	2,531.0	2,461.7	2,382.4
0.98	1,601.3	1,521.3	1,399.8	1,378.5	1,319.1
0.97	1,116.5	1,070.7	976.3	955.5	951.8
0.96	862.4	839.9	792.2	754.1	737.1
0.95	709.3	686.0	642.8	642.0	606.2
0.90	382.0	374.0	351.4	340.2	343.1
0.85	273.7	252.8	260.0	249.5	241.6
0.80	198.4	206.0	194.7	196.1	193.9

TABLE 3.9: AVERAGE ITERATIONS REQUIRED TO REACH THE FIRST SUB-OPTIMA - LARGE REMEMBRANCE

β	N				
	10	**30**	**50**	**70**	**90**
0.99	2,957.3	2,361.0	-	-	-
0.98	1,545.8	1,502.7	1,269.7	-	-
0.97	1,077.1	1,052.0	998.0	935.0	-
0.96	859.8	841.7	761.0	708.2	659.5
0.95	702.6	651.1	643.1	679.5	609.3
0.90	386.1	368.7	361.5	333.7	344.6
0.85	271.6	259.5	256.2	246.6	236.1
0.80	216.3	202.2	196.6	197.3	182.0

TABLE 3.10: PERCENTAGE OF TRIALS GETTING THE GLOBAL OPTIMUM AND THE FIRST SUB-OPTIMA - LUMP SUM

β	N					β	N				
	10	**30**	**50**	**70**	**90**		**10**	**30**	**50**	**70**	**90**
1.00	0	0	0	0	0	**0.85**	91	100	100	100	100
0.99	100	100	100	100	100	**0.80**	83	99	99	99	100
0.98	100	100	100	100	100	**0.75**	83	93	99	99	100
0.97	100	100	100	100	100	**0.50**	69	91	98	99	99
0.96	100	100	100	100	100	**0.25**	49	90	92	95	97
0.95	100	100	100	100	100	**0.00**	43	76	84	89	95
0.90	96	100	100	100	100	-	-	-	-	-	-

3.6.3 Benchmark tests

In this experiment, we tried another four problems listed in Table 3.11. They are commonly used in testing global optimization algorithms. We tried: $N = \{10, 50, 90\}$, $\beta = \{0.80, 0.85, 0.90, 0.95\}$, $\mu = 1$, $\tau = 0.4$, $l = 16$ for R2, GP2 and H3 problems. For S1, we tried: $N = \{10, 20, 30\}$, $\beta = \{0.93, 0.94, 0.95, 0.96\}$, $\mu = 1$, $\tau = 0.4$, $l = 16$.

Tables 3.12 and 3.13 show the percentage of trials and average iterations required to get the global optima of the respective functions. We obtained 100% success rate (reaching the prescribed f^+) on S1 and GP2 functions under majority of our experimental conditions. While for R2 and H3 functions, the low success rates for some test conditions can be explained by their rugged landscapes and the raise in dimensionality.

3.7 Discussion and analysis

3.7.1 Hierarchy of partitions

Solution precision Apart from the search space reduction, the hierarchical structuring of search space allows function optimization to increase precision without increasing the difficulty in the same pace. Any increase in precision by one single bit would double the search space size. As shown in Figure 3.11, the convergence is carried out in a more or less bit-by-bit fashion. Regardless of doubling of the search space, the convergence of a bit concerns only about the two regions separated immediate from the partition under its control. The increased precision is the subject matter of the added bit only. In fact, the difficulty is mainly determined by the ruggedness of the partition under control. So, if it is the partitions under the control of the additional bits that are highly rugged, the increased difficulty is attributed to these bits only and the convergence in the rest of the hierarchy (the part above the added bits) will not be adversely affected.

TABLE 3.11: BENCHMARK TEST FOR pBHS: TEST PROBLEMS

Problems	n	f^*	x^*	#eval†
S1 - Shekel	1	14.5926520	0.6858609	♭
R2 - Rastrigin	2	2.0000	[0 0]	♭
GP2 - Goldstein-price	2	-3.00001	[0 -1]	492 [57]
H3 - Hartman3	3	3.86	[0.4047 0.8828 0.8732]	1,014 [8]

†: Average number of function evaluations
♯: Number of iterations
♭: See the entry GA in the table below.

Algorithms	Function evaluations			
	S1	R2	GP2	H3
MS	-	1176	4400	2500
CRS	-	-	2500	2400
SA	-	-	563	1459
SAsde	-	-	5439	3416
HGA	-	-	146	191
ARS	-	-	492	-
NP	-	-	936	1014
PE	-	-	200♯	-
GA*	1185.9	3676	1644.6	972

♯: Number of iterations
⋆: Experiments carried by ourselves. See the following table for the experimental conditions.
-: No results reported.
MS - Multistart [52, 58]
CRS - Controlled random search [51]
SA - Simulated annealing [16]
SAsde - SA based on stochastic differential equations [3]
ASA - Adaptive simulated annealing [36]
HGA - Hybrid genetic algorithm [35]
APRS - Adaptive partitioned random search [57]
NP - New Price's algorithm [8]
PE - Perttunen's method [56]
GA - Genetic Algorithm

Experimental conditions for the GA test				
	S1	R2	GP2	H3
Population size	30	50	30	90
Mutation probability P_μ	$1/nl$, $l = 16$			
Crossover probability P_χ	1.0 (Two-point)			
Fitness	Fitness scaling			
Selection	2-Tournament			
Replacement	Proportional			
Success rate	100%			

TABLE 3.12: PERCENTAGE OF TRIALS GETTING THE GLOBAL OPTIMUM FOR S1, R2, GP2, AND H3

β	S1			β	R2		
	10	**20**	**30**		**10**	**50**	**90**
0.96	100	100	100	**0.95**	84	100	100
0.95	99	100	100	**0.90**	69	100	100
0.84	100	100	100	**0.85**	56	97	100
0.93	98	100	100	**0.80**	44	94	98
β	GP2			β	H3		
	10	**50**	**90**		**10**	**50**	**90**
0.95	100	100	100	**0.95**	100	98	100
0.90	100	100	100	**0.90**	95	96	98
0.85	99	100	100	**0.85**	83	92	98
0.80	99	100	100	**0.80**	74	90	95

TABLE 3.13: AVERAGE NUMBER OF FUNCTION EVALUATIONS REQUIRED TO REACH THE GLOBAL OPTIMUM FOR S1, R2, GP2, AND H3

β	S1			β	R2		
	10	**20**	**30**		**10**	**50**	**90**
0.96	1,236	1,908	2,510	**0.95**	1,260	4,565	6,993
0.95	1,001	1,600	2,174	**0.90**	704	2,695	4,023
0.94	915	1,444	2,039	**0.85**	514	1,925	3,006
0.93	825	1,331	1,800	**0.80**	398	1,530	2,412
β	GP2			β	H3		
	10	**50**	**90**		**10**	**50**	**90**
0.95	951	3,100	4,653	**0.95**	709	1,181	1,547
0.90	590	1,890	3,042	**0.90**	434	724	959
0.85	405	2,420	2,331	**0.85**	330	555	766
0.80	329	1,125	1,872	**0.80**	274	458	563

3.7.2 Availability of global information

The hierarchical structuring of search space is essential to our algorithm in global optimization. However, in the absence of global information, the structuring is almost useless in global problem solving. We know from the hierarchical structure that those most-significant bits controlling the main direction of the later search determines the possibility of finding the global solution in that run. That means the fate of that run is determined to a large extend at the very beginning. Unless in the presence of reliable global information, the result obtained cannot be explained satisfactorily.

3.7.3 Adaptation

Adaptive remembrance scheme The successfulness of applying our adaptive remembrance scheme as justified by the results obtained lies in the reasons behind the

design of the scheme. Taking a typical β setting: 0.95, we can see the initial fluctuation of the convergence curves (Figure 3.11) of those less significant bits. This reflects a fact that the information collected in these bits are not reliable enough initially. They will converge prematurely and incorrectly if we use small and constant β value for all bits throughout the generations. Using larger $\beta_{m,i}$ for less significant bits, however, can prevent the bits from converging too fast to mislead the global information gathering at the more significant bits. Only when the most significant bits have converged that the $\beta_{m,i}$ values for the remaining bits are allowed to increase. In fact, the maximum allowed $\beta_{m,i}$ for the less significant bits can be larger than the more significant bits, owing to the reduced sample space. However, presently we do not have a good scheme to decide how large the $\beta_{m,i}$ should be. Therefore, we employed a conservative way which sets the maximum allowed $\beta_{m,i}$ to β.

Fitness measure The ability to converge to the extend that the optimal binary string is generated should be attributed to our fitness measure. Representing numbers ranged from 0.0 to 1.0 using 32-bit binary string allows numbers be distinguished in a very precise way. Regardless of how powerful the algorithm is in locating the promising area where the global solution resides, pushing a nearly converged population, for instance, with several least significant bits diverged, to the exact optimum is extremely difficult. The sole reason is the weak convergence power generated by the nearly converged population. Our fitness measure is designed in such a way to cope with this kind of problem. The high percentage of trials reaching the exact optimal regions as shown in Table 3.2 and 3.6 clearly indicates the power of the fitness measure.

3.8 Summary

In this chapter, the basic pBHS algorithm is presented. The algorithm is based on a hierarchical view of sample space subdivision. Coupled with this partition hierarchy, the information processing cycle created by the collective contribution of samples and the global searching environment makes the crucial global information accessible. The experimental results proved this point that the computational expenses of pBHS on problems of low dimension are similar or even less than the existing advance techniques. However, the capability of this model on solving high-dimensional problems are quite limited as reported. We attribute this limitation as the inefficiency introduced by the one-to-one mapping of the samples among dimensions. In chapter 4, we introduce cooperation among the samples of each dimension to overcome this problem.

This basic model keeps one set of component fitness values as the global information. Such fitness values can only indicate the bit values of one optimal solution. That means the algorithm is incapable of locating more than one optimum simultane-

ously as required in multi-modal function optimization and deceptive problem-solving. A straightforward way to extend the algorithm to overcome this deficiency is to keep several sets of component fitness values. Each set should have a separate and independent group of searching agents responsible for information gathering. Introducing multiple groups of searching agents and component fitness values, we ought to have a way to prevent them from converging into the same optimal point (unless the function is unimodal). Having multiple number of groups, interactions among them should be encouraged to increase the diversity as in GAs. Chapter 5 is devoted to the discussion on the introduction of multiple number of groups.

Part III

Cooperation and Competition

CHAPTER 4

High-dimensionality

High-dimensionality poses a great challenge to all optimization algorithms, in particular searching algorithms. It is not only the exponential scale-up of the search space size, but also the presence of different degrees of dependency among the dimensions. This chapter describes how the pBHS can be extended to become pcBHS (*Probabilistic Cooperative Binary Hierarchical Search*) to handle high-dimensional problems by decoupling the dimensions to form subpopulations and by using a cooperation technique. Decoupling of dimensions allows individuals in different subpopulations to form solutions freely with each other in a cooperative manner. The employed cooperation technique is used to provide an appropriate fitness measurement so as to promote good cooperation. This extension is shown empirically to be useful to increase the efficiency over the basic pBHS model. The scaling property as well as the performance of the model are also studied. The results indicate that the extended model performs satisfactorily when compared with the existing advanced stochastic techniques.

4.1 Introduction

4.1.1 The challenge of high-dimensionality

The basic model presented in chapter 3 is rather limited and inefficient, though it can locate the global optimum accurately. The basic pBHS creates a population of N complete solutions to the problem. In the one-dimensional case, a complete solution is a vector consisting of one element only. While, in multi-dimensional problems, a single complete solution consists of a vector of multiple elements. The probability of generating n elements (a complete solution) simultaneously so as to optimize a single objective is much lower than optimizing them individually[1]. The probability of finding the optimum

vector $x^* = [\, x_0^* x_1^* \cdots x_{n-1}^* \,]$ for the two cases are shown respectively as follows:

$$Prob(x^*) = \begin{cases} \prod_{i=0}^{n} Prob(x_i^*) & \text{Optimize together} \\ \sum_{i=0}^{n} Prob(x_i^*) & \text{Optimize individually} \end{cases} \tag{4.1}$$

Optimization of the objective variables together is capable of overcoming the difficulty imposed by the presence of dependency among variables. The tradeoff is the efficiency of the algorithm. To optimize the objective variables separately, we can raise the probability dramatically even in the low dimensional cases. What is sacrificed is the solution quality when there is a strong dependency among various dimensions.

4.1.2 Cooperation - A solution to high-dimensionality

In a population of complete solutions, it is possible that some x_i in some solutions is x_i^* and some x_j in some other solutions is x_j^*, where both x_i^* and x_j^* are optimal solution of the i-th and j-th dimension. However, these solutions may have low fitness values owing to their recessive nature. Unless in the presence of some specific solution composition that their contributions and hence their fitness can be revealed. Since the complete solutions in which they reside have low fitness, they are subject to be lost. The coupling of the dimensions in the pBHS model then limits the chance of the good solutions to reveal their fitness. Extra computation has to be used to regenerate the lost solutions.

If we treat each of the n dimensions as a single subpopulation of size N instead of a population of N complete solutions, and each individual in a subpopulation does not tie with any individual in other subpopulations, they are said to be free and they are allowed to join any individual in other subpopulations to form complete solutions. By this way, it is easier to have good combinations. In the next section, a probabilistic cooperative binary hierarchical search (pcBHS) is presented based on this idea.

4.2 Probabilistic Cooperative Binary Hierarchical Search

4.2.1 Decoupling

In the basic pBHS model, a population is defined as a group of sample points consisting of samples from all n sub-spaces. In the following, sample point is referred to as a *complete solution* while a sample from a sub-space is referred to as a *solution fragment.* For instance, a complete solution for a three-dimensional function $F(x)$ is a vector $[x_0\ x_1\ x_2]$ which consists of three solution fragments: x_0, x_1 and x_2.

[1] What is mean by '*optimizing them individually*' does not mean optimizing them separately.

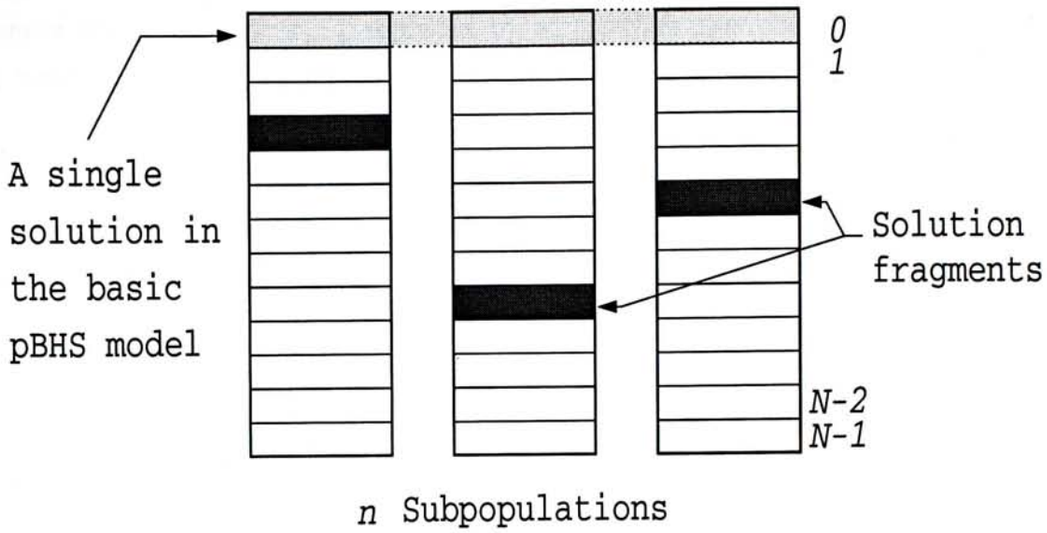


FIGURE 4.1: DECOUPLING

In pcBHS, decoupling is taken such that each sub-space exists as its own and a single population becomes n subpopulations. The size of each subpopulation is still kept at N in order to maintain the same varieties of solution fragments as in pBHS. The situation is illustrated in Figure 4.1. The shaded region enclosed by two dotted lines indicates a single complete solution in the basic pBHS model. In pcBHS, all solution fragments in a subpopulation are not tied with any solution fragment in other subpopulations. What we have are n sets of N solution fragments.

This decoupling allows a lot of flexibility on the formation of complete solutions. The basic pBHS model actually is a special case of pcBHS that the choice of solution fragments combination is restricted to a dedicated one fragment from each subpopulation. Enumerating all possibilities is another scheme to fully utilize the information in all subpopulations. However, the quality of the information extracted may not justify the computational expenses required. Hence, in order to exploit efficiently the advantage of the decoupling, complete solutions are generated by employing the scheme in [50].

4.2.2 Cooperative fitness

Before describing how to combine solution fragments, the issue on fitness measurement should be discussed first. The ordinary fitness measurement as used in pBHS becomes inappropriate in the cooperative model. Raw fitness is meaningful only when a single complete solution exists. After decoupling, fragments representing the problem sub-spaces are created. Their fitness values are undefined. *Cooperative fitness* as defined in [50] is employed to evaluate the solution fragments. Given n arbitrary solution fragments $\{x_0, x_1, \cdots, x_{n-1}\}$ from each subpopulation, each of their raw cooperative fitness equals $F(x)$ where $x = [x_0\ x_1\ \cdots\ x_{n-1}]$. Suppose that the same set of solution

fragments are given with x_{n-1} replaced by x'_{n-1}, their raw cooperative fitness become $F(x')$ where $x' = [x_0\ x_1\ \cdots\ x'_{n-1}]$.

4.2.3 The cooperative model

The cooperative pcBHS model differs from the basic pBHS model in three aspects: fitness evaluation, fitness scaling, and elitism. Owing to the change in the interpretation of the generated solution fragments, the use of cooperative fitness is introduced. Hence, we have to define how the fitness is going to be scaled and accumulated. We also have to define how elites can be formed and used. Algorithm 4.1 gives an overview on the differences between the two models. As shown in the algorithm, it is Steps 3–5 of the basic pBHS model (see Appendix A) that have to be modified.

Algorithm 4.1 PCBHS OVERVIEW

Step 1:	Initialization
Step 2:	Generation of a new population
Step 3:	**Evaluation** /* *modified in pcBHS* */ **See Algorithm 4.2**
Step 4:	**Fitness scaling** /* *modified in pcBHS* */ **See Equation 4.2**
Step 5:	**Information gathering** /* *modified in pcBHS* */
Step 6:	Information deposition
Step 7:	Adjustment of remembrance values
Step 8:	Goto Step 2 if (i) max. generation is not reached; and (ii) stopping criteria not met.

Fitness measurement and fitness scaling

As discussed in the previous sections, raw fitness is replaced by cooperative fitness owing to the decoupling of solution fragments. Suppose that there is a *global elite* $x^e = [\ x_0^e\ x_1^e\ \cdots\ x_{n-1}^e\ ]$, the cooperative fitness of each solution fragment $x_{m,j}$ in each subpopulation m is defined as $cF(x_{m,j}, x^e)$. Function cF is simply the objective function F applied to a complete solution formed by replacing the m-th element in x^e by $x_{m,j}$. Algorithm 4.2 shows how it is implemented. Under this scheme, there are $n \times N$ number of complete solutions centered around x^e formed. These raw cooperative fitness cF of each solution fragments are scaled within their subpopulations only. Denoting cf as the *scaled cooperative fitness*, the cf of the j-th individual in the m subpopulation is:

$$cf(x_{m,j}, x^e) = \frac{cF(x_{m,j}, x^e) - cF_m^{min}}{cF_m^{max} - cF_m^{min}} \tag{4.2}$$

where $cF_m^{max} = \max\{\ F(x^e),\ \max_{0 \le j \le N-1} cF(x_{m,j}, x^e)\}$
$cF_m^{min} = \min\{\ F(x^e),\ \min_{0 \le j \le N-1} cF(x_{m,j}, x^e)\}$

In Equation 3.18, it is f_j that is fed back into the system. We now use the scaled cooperative fitness cf. Given a binary string s_m of the m-th dimension, and $0 \leq i < l$, the component fitness for the $(l-1-i)$-th bit is determined as follows:

$$\begin{cases} u_{m,i} = cf(x_{m,j}, x^e) \text{ and } w_{m,i} = 0 & \text{if } b_{m,l-1-i} = 0, \\ u_{m,i} = 0 \text{ and } w_{m,i} = cf(x_{m,j}, x^e) & \text{if } b_{m,l-1-i} = 1. \end{cases} \tag{4.3}$$

Algorithm 4.2 pcBHS - Cooperative fitness assignment and *local* elites updating. This procedure assigns fitness (cooperative fitness) to all solution fragments. In each subpopulation, if the best fragment is better than the elite fragment x'^e_m, it becomes the new elite fragment. It should be noted that the current elite x^e is not changed in this procedure.

Procedure CoopEvaluation

  For each subpopulation P_m, $0 \leq m < n$

    $x'^e_m \leftarrow x^e_m$ /* x'^e_m: Elite fragment of the m-th subpopulation */

    For each solution fragment $x_{m,i}$ in P_m

      $x \leftarrow$ replace the m-th element of x^e by $x_{m,i}$

      $cf(x_{m,i}, x^e) \leftarrow F(x)$

      if $cf(x_{m,i}, x^e) > cf(x'^e_m, x^e)$ **then**

        $x'^e_m \leftarrow x_{m,i}$

      End if

    End for

  End for

End Procedure

Elitism

Under this model, the elitist strategy used in the basic pBHS model have to be modified. Since each subpopulation is individually responsible for a single unique dimension, elitism is applied separately to each subpopulation (see Algorithm 4.2) in each generation producing a set of new local elites $\{x'^e_0, x'^e_1, \cdots, x'^e_{n-1}\}$. The new global elite x''^e is selected from the following:

- **No-change**

  The existing global elite x^e with raw fitness $F(x^e)$.

- **Local**

  Any one of the local elite x'^e_m in cooperation with the existing global elite x^e.

- **Random**

  The best of n complete solutions formed be cooperating randomly picked solution fragment x_m^r with the global elite x^e. The cooperative fitness of a complete solution formed by cooperating a randomly selected solution fragment in the m-th subpopulation with the global elite is denoted as $cf(x_m^r, x^e)$.

- **Multiple**

  A complete solution formed by combining the existing global elite x^e and those $x_m'^e$ whose cf is greater than $F(x^e)$:

$$\forall m, 0 \leq m < n \bullet cf(x_m'^e, x^e) \geq F(x^e) \tag{4.4}$$

  Suppose that the set of local elites satisfies this criterion is $\psi = \{x_1'^e, x_3'^e\}$, the complete solution would be $\{x_0^e, \mathbf{x_1'^e}, x_2^e, \mathbf{x_3'^e}, \cdots, x_{n-1}^e\}$ and its cooperative fitness is denoted as $cf(\psi, x^e)$.

The global elite is replaced by any one of them with the maximum cooperative fitness:

$$F(x''^e) \geq \max\{F(x^e),\ cf(x_m'^e, x^e),\ cf(x_m'^e, x^r),\ cf(\psi, x^e)\} \tag{4.5}$$

The inclusion of the last two set of choices (*random* and *multiple*) is intended to lower the greediness of the simple *no-change+local* scheme of CCGA-1 as illustrated in [50]. Although the replacement scheme is still a winner-take-all strategy, the *random* scheme may introduce new solution fragments which may lead to a new and possibly optimal path, while the *multiple* scheme allows multiple-subspace movement in one single step.

4.3 Empirical performance study

4.3.1 pBHS versus pcBHS

In this section, the performance of the basic pBHS model and the cooperative pcBHS model are compared. Since the capability of the basic pBHS model on multi-dimensional problems is not satisfactory enough, the problems used for comparison are of low dimensions. They are listed in Table 4.1 with some key information about the problems. Details can be found in Appendix B.

Result

The results of this test are shown in Tables 4.2-4.5 and Figures 4.2-4.5. Judging from the result, we have the following conclusion. For all the tests, both the average

TABLE 4.1: pBHS VERSUS pcBHS: TEST PROBLEMS

Problems[♯]	n	f^+	Conditions[♭]
R2 - Rastrigin	2	1.99950	N = 10,50,100,150,200 β = 0.95,0.925,0.9,0.875,0.85,0.825,0.8
H3 - Hartman	3	3.86000	N = 30,50,100,150, β = 0.95,0.925,0.9,0.875,0.85,0.825
M5 - Michalweitz	5	4.68700	N = 50,100,150,200 β = 0.95,0.925,0.9,0.975
SP8 - Sphere	4	-0.0009	N = 10,30,50,70,100 β = 0.95,0.9,0.85,0.8,0.75,0.70,0.65

♯: see Appendix B for details
♭: 100 consecutive independent runs, l=16

TABLE 4.2: pBHS VS. pcBHS: SUCCESS RATE (%) FOR RASTRIGIN (N=2)

β	pBHS					pcBHS				
	10	**50**	**100**	**150**	**200**	**10**	**50**	**100**	**150**	**200**
0.950	61	97	100	100	100	89	100	100	100	100
0.925	50	95	99	100	100	80	99	100	100	100
0.900	38	91	99	100	100	71	99	99	100	100
0.875	28	76	95	99	100	66	100	98	99	99
0.850	27	81	94	95	100	60	95	100	96	99
0.825	19	66	86	96	99	51	92	96	98	95
0.800	22	75	85	92	95	43	93	93	95	95

number of iterations and the average number of function evaluations required to reach global optimum using pcBHS is several times lower than those using pBHS under the same experimental conditions. The difference for the number of function evaluations required is diminishing towards the smaller population size. This observation tells us two things:

1. pcBHS has higher potential of gaining speedup by parallelization.

2. pcBHS has better performance compared with pBHS on using large population size. That means for large problems that require larger sampling, pcBHS (using large population size) would be a better choice.

The second observation is that pcBHS is comparatively insensitive to the remembrance parameter. Decreasing the remembrance value carries the meaning of speeding up the convergence. It produces the effect on pBHS that the number of iterations and the function evaluations are both decreasing. However, it is not the case for pcBHS. Both the number of iterations and function evaluations fluctuate between a small range. It is especially true when using large population.

TABLE 4.3: pBHS vs. pcBHS: SUCCESS RATE (%) FOR HARTMAN (N=3)

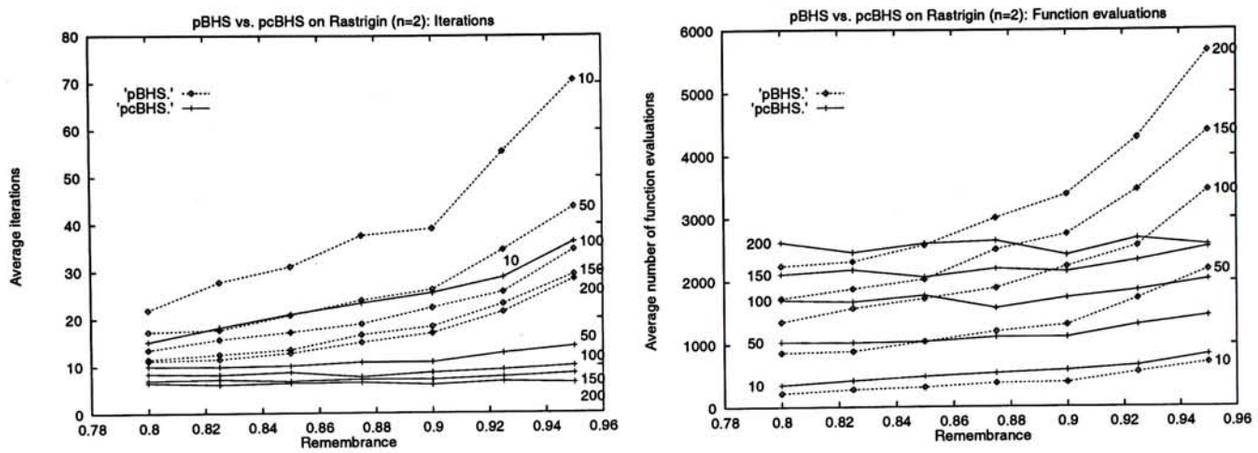
β	pBHS				pcBHS			
	30	**50**	**100**	**150**	**30**	**50**	**100**	**150**
0.950	100	100	100	100	98	96	97	98
0.925	100	100	100	100	99	97	97	98
0.900	99	100	100	100	98	93	96	94
0.875	98	100	100	100	93	89	92	94
0.850	97	100	100	100	91	86	88	94
0.825	96	99	100	100	92	90	92	94
0.800	92	97	99	100	90	88	89	93

TABLE 4.4: pBHS vs. pcBHS: SUCCESS RATE (%) FOR MICHALWEITZ (N=5)

β	pBHS				pcBHS			
	50	**100**	**150**	**200**	**50**	**100**	**150**	**200**
0.950	7	8	10	23	97	100	100	100
0.925	4	8	11	12	97	100	100	100
0.900	7	4	5	16	95	99	100	100
0.875	11	9	9	5	92	97	99	100

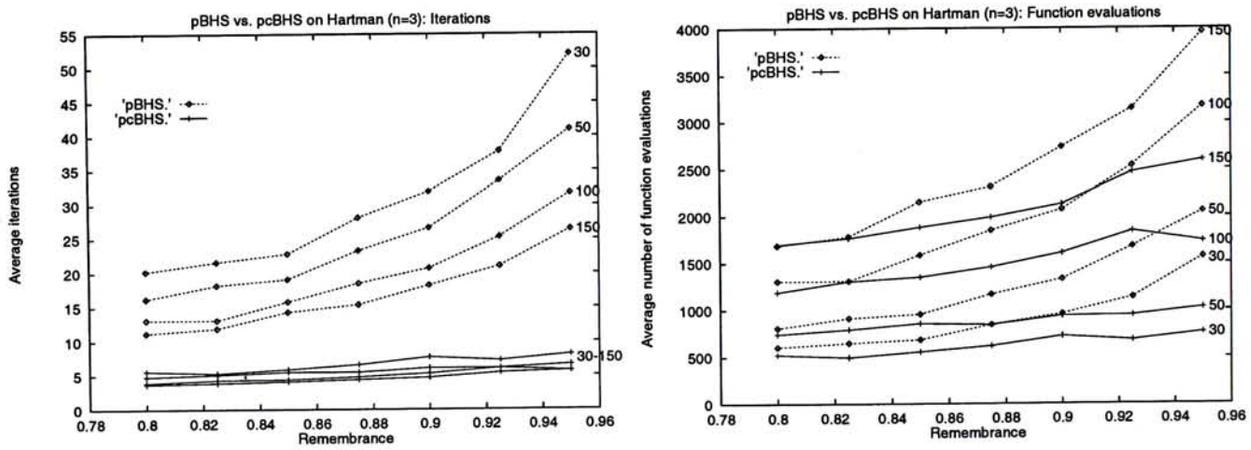
TABLE 4.5: pBHS vs. pcBHS: SUCCESS RATE (%) FOR SPHERE (N=8)

β	pBHS					pcBHS				
	10	**30**	**50**	**70**	**100**	**10**	**30**	**50**	**70**	**100**
0.95	46	80	92	92	98	94	99	100	100	100
0.90	85	99	100	100	100	95	100	100	100	100
0.85	54	97	100	99	100	95	100	100	100	100
0.80	24	94	98	100	100	97	99	100	100	100
0.75	12	91	99	99	100	93	99	100	100	100
0.70	9	70	95	99	100	88	98	100	100	100
0.65	2	64	86	96	99	85	100	100	100	100



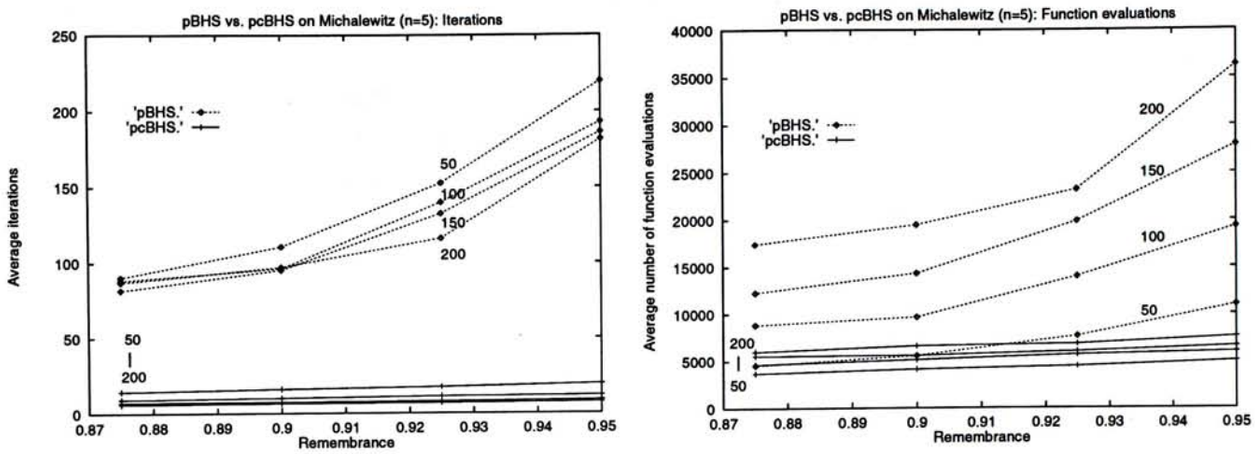
(a) (b)

FIGURE 4.2: pcBHS VS. pcBHS ON RASTRIGIN (N=2): (A) AVERAGE ITERATIONS AND AVERAGE (B) NUMBER OF FUNCTION EVALUATIONS



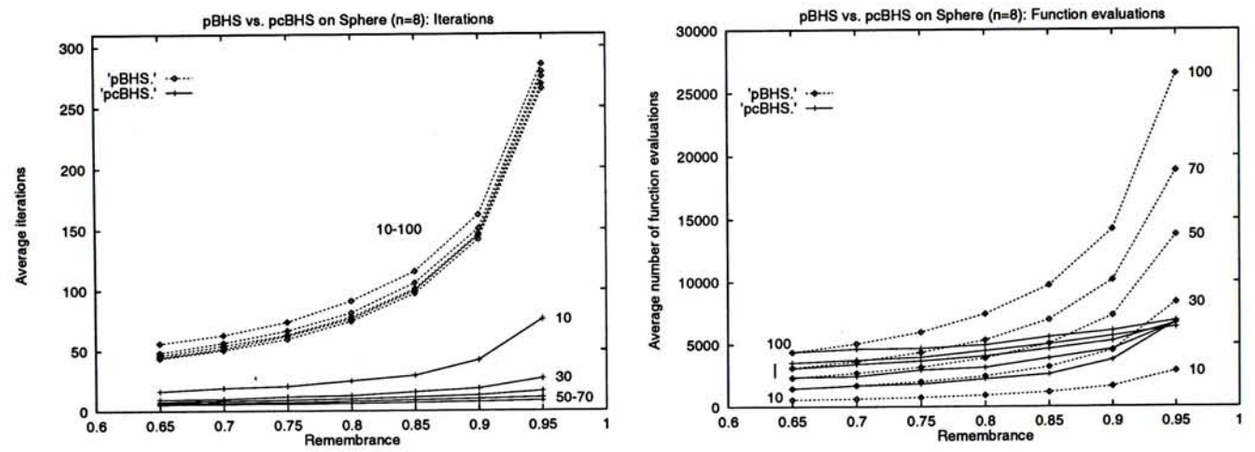
(a) (b)

FIGURE 4.3: pcBHS VS. pcBHS ON HARTMAN (N=3): (A) AVERAGE ITERATIONS AND AVERAGE (B) NUMBER OF FUNCTION EVALUATIONS



(a) (b)

FIGURE 4.4: pcBHS VS. pcBHS ON MICHALEWITZ (N=5): (A) AVERAGE ITERATIONS AND AVERAGE (B) NUMBER OF FUNCTION EVALUATIONS



(a) (b)

FIGURE 4.5: pcBHS VS. pcBHS ON SPHERE (N=8): (A) AVERAGE ITERATIONS AND AVERAGE (B) NUMBER OF FUNCTION EVALUATIONS

4.3.2 Scaling behavior of pcBHS

The purpose of this experiment is going to show how the cooperative pcBHS model scale with the problem size. Two areas that are of interest are: *mean iterations* and *mean function evaluations.* To serve this purpose, several test problems listed in Table 4.6 are used with different dimensionalities ranging from $n = 10$ to $n = 400$. The experimental conditions are stated in Table 4.7.

The results shown in Figures 4.6-4.8 illustrate clearly that the algorithm scales

TABLE 4.6: SCALING pcBHS: TEST PROBLEMS

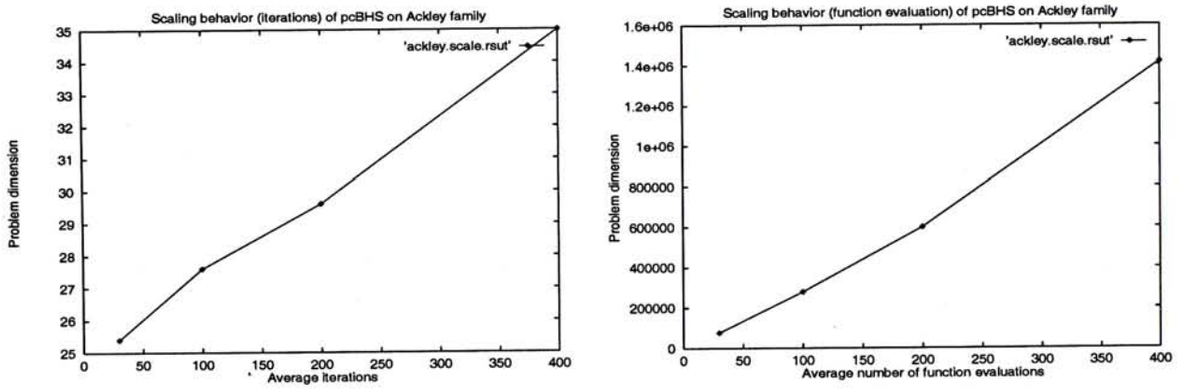
Problems♯	n	f^+
An - Ackley	30,100,200,400	-0.001843
SPn - Sphere	8,16,32,64,128	-0.0009
Rn - Rastrigin	20,50,100,200,400	-0.900

approximately linearly with the problem dimension of the three test problems.

TABLE 4.7: SCALING pcBHS: EXPERIMENTAL CONDITIONS AND RESULTS

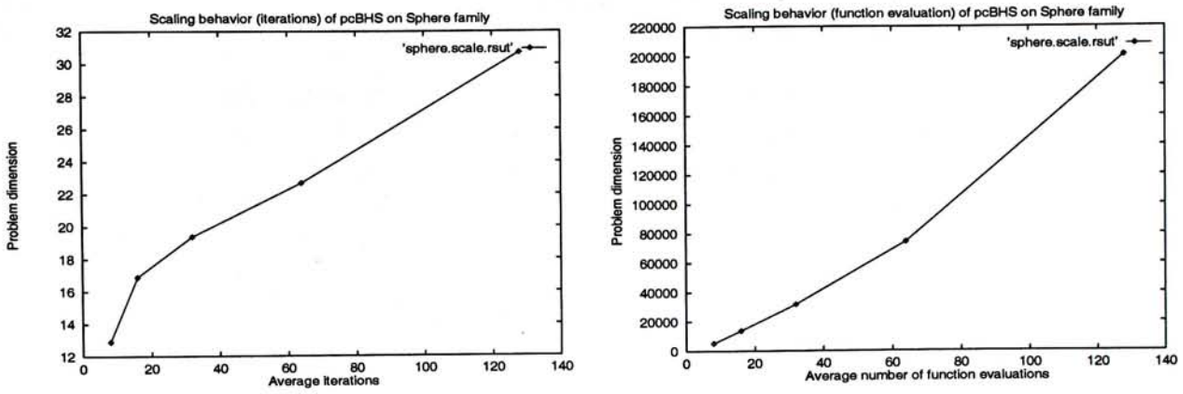
Conditions♯		Results	
Problems	β	Succ. rate	Figure
An - Ackley	0.70	89%-82%	4.6
SPn - Sphere	0.90	100%	4.7
Rn - Rastrigin	0.50	100%	4.8

♯: 100 consecutive independent runs, $l = 16$, $N = 100$, $\mu = 1$



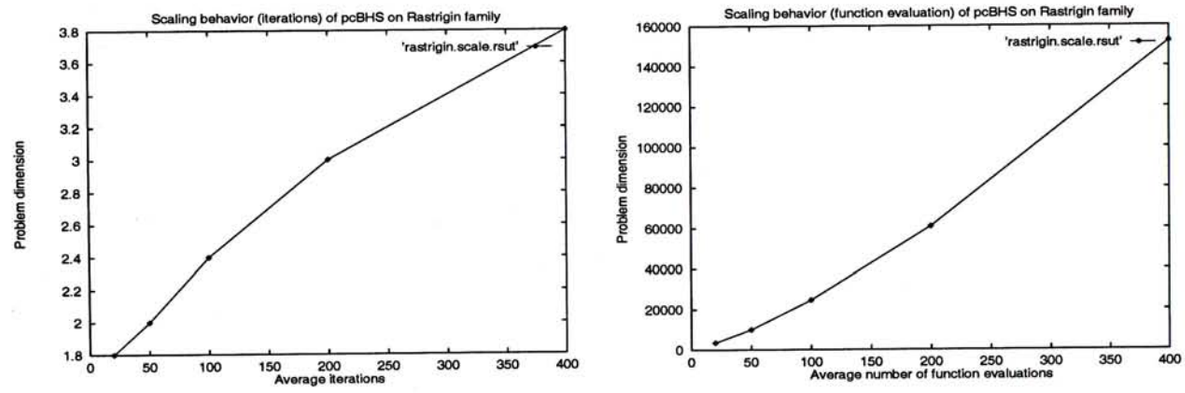
(a) (b)

FIGURE 4.6: SCALING BEHAVIOR OF pcBHS ON ACKLEY FAMILY



(a) (b)

FIGURE 4.7: SCALING BEHAVIOR OF PCBHS ON SPHERE FAMILY



(a) (b)

FIGURE 4.8: SCALING BEHAVIOR OF PCBHS ON RASTRIGIN FAMILY

4.3.3 Benchmark test

In this experiment, several problems with problem size up to 100 dimensions are tried. These problems are commonly used in testing global optimization algorithms. Each family of problems possesses characteristics quite different from each other. A brief summary of the test problems used are listed in Table 4.8. Detailed description of these problems can be found in Appendix B.

The results listed in Table 4.9 shows that the performance of our algorithm is comparable with the existing advanced techniques such as Breeder GA (BGA) [47] and Evolutionary Algorithm with Soft genetic operators (EASY) [59, 60]. Both variants of GAs are said to be highly effective yet a general model for evolutionary algorithms. However, the performance of the algorithm on the Shekel family is comparatively poor

TABLE 4.8: BENCHMARK: TEST PROBLEMS

Problems♯	n	f^+	#eval	Ref.
S5 - Shekel	4	9.9	5,403	⋆
S7 - Shekel	4	9.9	5,386	⋆
S10 - Shekel	4	9.9	5,862	⋆
H3 - Hartman	3	3.86	1,014	⋆
A30 - Ackley	30	0.001	13,997/19,420	†
A100 - Ackley	100	0.001	57,628/53,860	†
R20 - Rastrigin	20	0.9	6,098/3,608	†
R100 - Rastrigin	100	0.9	45,118/25,040	†

♯: see Appendix B for the description of the problems
⋆: A clustering technique: New Price's algorithm [8]
†: EA with soft genetic operators/Breeder GA [EASY/BGA] [59]
f^+: Function values at which the algorithms stop.
#eval: Number of function evaluations.

than those from the existing techniques We attribute this poor performance as their *golf-hole*-like landscapes. The figure shows that S5 has 5 prominent optima resting on a plateau, all of which has similar basin of attraction. Landscapes of this kind provides no useful information for guidance. The reason why the successful rate drops from S5 to S10 is due to the raise in the number of prominent optima.

TABLE 4.9: BENCHMARK TEST: RESULTS

Problems	f^+ attained	#eval	%‡	Conditions*
S5	10.004523	12,476.40	34%	β=0.97 N=50
S7	10.100106	13,168.60	29%	β=0.97 N=50
S10	10.126623	13,909.30	14%	β=0.97 N=50
H3	3.861696	755.80	100%	β=0.90 N=30
A30	-0.00078	18,679.68	100%	β=0.40 N=40
A100	-0.00074	58,216.17	90%	β=0.35 N=40
R20	-0.48987	5,413.22	100%	β=0.45 N=40
R100	-0.54718	45,194.86	100%	β=0.45 N=40

‡: Percentage of runs reaching f^+ stated in Table 4.8.
∗: 100 independent consecutive runs, $l = 16$.

4.4 Summary

High-dimensionality poses a great challenge to all kinds of optimization algorithms, including the basic pBHS model. In this chapter, we have presented an extended model–pcBHS–which is based on the idea of decoupling of dimensions. In principle, it allows the

solution fragments in each dimension to form complete solution with other dimensions freely. The main reason for the decoupling is to increase the chance for good solution fragments to come together. The basic pBHS is said to be a special case of the extended model, as it allows the formation of complete solution with one single unique solution fragment from each subpopulation. Owing to this major extension, three areas of the basic model have to be modified, namely the fitness evaluation, the fitness scaling and the elitism.

The improvement on the performance over pBHS have been studied which showed that pcBHS can reach the same accuracy with much computation saved. Since pcBHS is devised to cater for high-dimensional cases, we have studied its scaling property. The results showed that it scales approximately linear with problem dimension in all of the functions we have tested. Moreover, pcBHS is compared with the well-known advanced stochastic search-based techniques in solving optimization problems. It is comparable with (and even outperforms in some cases) those algorithms.

However, both the basic pBHS model and the cooperative pcBHS model are single-optimum-seeking models as if SA. The whole population cooperatively constructs a single piece of information—the global environment, which indicates where the global optimum is. Since both models rely on the information obtained on the course of searching to construct the global environment, it is easy for some deceptive problems to mislead the algorithm to construct an environment indicating the sub-optima only. To cater for this problem, we have to introduce redundancy of some kind. For instance, instead of constructing a global environment with one piece of information, we can build one with several pieces of information. In chapter 5, we will discuss how to introduce redundancy into the model.

CHAPTER 5

Deception

The basic design of both pBHS and pcBHS models are: (1) distribution of a population of individuals who search cooperatively for a single global optimum, and (2) assumption of no (or minimal) a-prior information about the problem to be solved. However, this design would make both algorithms easily be deceived, because (1) when the landscape of a problem has multiple number of similar basin of attractions, and (2) the landscape of a problem to be solved provide misleading information. In this chapter, we will present an extension of the pcBHS model to cater for these two cases by introducing *redundancy* and *competition.*

5.1 Introduction

5.1.1 The challenge of deceptiveness

Deception has been discussed rigorously in GA community [24, 15, 61, 62, 14, 48, 25, 53] in the last decade. Briefly, deceptive problems contain deceptive attractors which mislead the algorithm to search for sub-optima. Figure 5.1 shows a typical fully deceptive function for simple GAs on a maximization problem[1]. As shown in the figure, the global optimum and the suboptimum are located far apart with a big valley in-between. The basin of attraction favoring for the suboptimum is much larger than the one favoring for the global one, making the problem deceptive. Intensive analysis and the definition of deception in the context of GA can be found in [15, 61]. Qualitatively, the degree of deception varies according to the comparative size of the global attractor and the sub-optimal attractors.

Full deception occurs when the size of the global attractor approaches zero (one global optimum only) when comparing with that of the sub-optimal, while the marginal deception occurs when the size of the global attractor is comparable with the deceptive attractors. The sufficient conditions for the full deception to occur in the *folded-trap*

[1]The objective variable is the unitation of a chromosome, i.e. number of '1's bits.

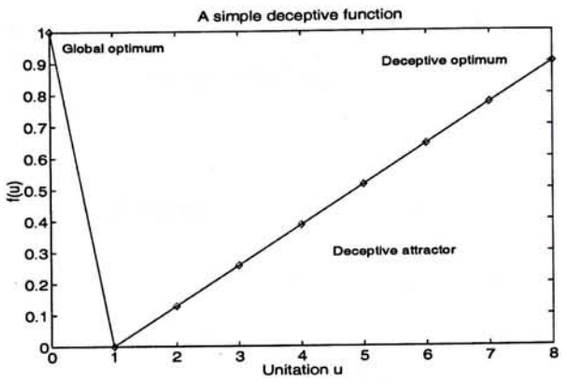


FIGURE 5.1: A SIMPLE TRAP FUNCTION

function $f(u)$, where $u = \{0, 1, \ldots, l\}$ is the unitation of binary strings of length l shown in Figure 5.1 is given in [15] as follows:

Condition 0: Primary optimality

$f(0) > f(l)$

Condition 1: Primary deception

$f(l) - f(0) > f(1) - f(l-1)$

Condition 2: Second deception

$f(i) \geq f(j)$ for $\lceil l/2 \rceil \leq i \leq l-1$ and $l - i \leq j < i$

These conditions state that the function favors for large unitation (conditions 1 and 2) for all unitations except 0 and l (condition 0), which are the global optimum and the deceptive optimum respectively.

Deception is, in fact, a relative term. Giving the same problem, some algorithms would be deceived while the others would not. All stochastic searching algorithms, except pure random search, applied to optimization are classified as heuristic search methods. What heuristic here means is not the domain specific knowledge, but the characteristics of the fitness landscape of the problems (see [7] for the simple classification based on landscape characteristic). In the study of the relationship of *operators* and *landscapes* by Jones [37], it is the operator(s) of the algorithms that define(s) the fitness landscape to be faced. A stochastic operator ϕ is defined in [37] as $\phi : \mathcal{M}(\mathcal{R}) \times \mathcal{M}(\mathcal{R}) \rightarrow [0 \cdots 1]$, meaning that a set of configurations of the search space is transformed into another set of configurations with certain probability by a single application of the procedures defined in that operator. In other words, a neighborhood is defined for every configuration of the search space. It is this neighborhood that determines the shape/appearance of the problem landscape (see Chapter 2 for more information on fitness landscape). Hence, a problem is deceptive to an algorithm because the landscape constructed by the operators of the algorithm is deceptive.

In the aforementioned discussion on deception, we can see that the effort to tackle deception should not be put on designing operators that can handle both normal and

deceptive cases, as it seems fruitless. On the contrary, operator design should be concentrated in handling the normal cases, leaving the deception problem tackled by other techniques.

5.1.2 Competition: A solution to deception

Niching [28, 13, 12, 43, 49, 45, 10] is a technique stemmed from the nature to tackle multi-modal optimization problems in GA community. Solving the problems having multiple number of 'peaks' in their landscapes using simple GAs would end up locating one of those peaks (not necessarily the global one). It is because of *genetic drift*, which can be thought of as stochastic fluctuation. Niching was then introduced to extend the capability of simple GAs in this kind of problems to locate as many peaks as possible. Besides the application to multimodal function optimization, niching is also used to cater for the deception problem in GAs. In general, problems that are deceptive to GAs comprises of deceptive attractors. By using niching, both the deceptive attractors and the true global attractor can be located and exploited simultaneously.

Existing niching techniques are: crowding [11], deterministic crowding [43], sharing [28, 44], and dynamic niching [45]. The central idea of all these niching techniques is *competition*. Individuals in a populations compete with each other for the occupancy of territories. Those who fail to occupy the territories will have smaller chance of survival, making the population more diverse. The side-effect of this mechanism is the dominance of individuals who can occupy territories, i.e. locating multiple number of peaks simultaneously.

In the following section, the idea of competition is employed in the cooperative pcBHS model to produce what we called *probabilistic cooperative-competitive binary hierarchical search (pccBHS)* model.

5.2 Probabilistic cooperative-competitive binary hierarchical search

In this section, an enhanced model named *pccBHS* incorporating redundancy and competition into the cooperative pcBHS model is presented. The competition model shares similarities with the existing sharing mechanism. The strength of the model over sharing mechanism is the avoidance of the requirement of specifying *niche radius*. Niche radius inherently limits the niching mechanisms to be applied to problems that requires locating niches at different resolution levels simultaneously. The drawback of the pccBHS model, at present, is that the number of niches to be occupied is bound by a prescribed number.

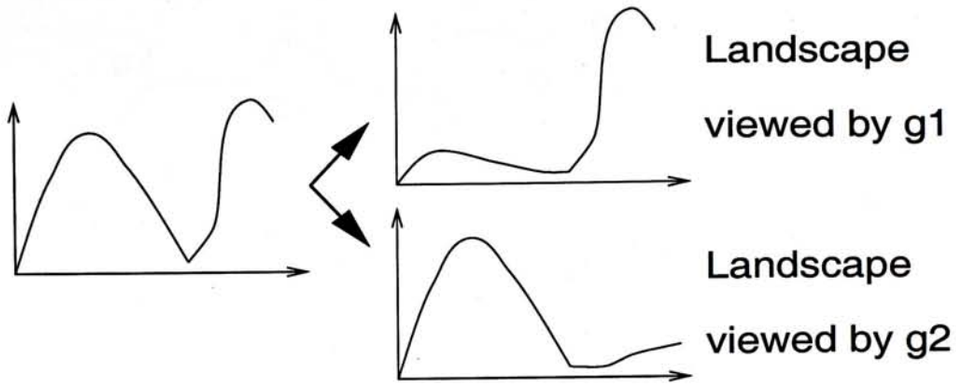


FIGURE 5.2: RE-MODELING OF FUNCTION LANDSCAPE

5.2.1 Overview

The main structural characteristic of the pccBHS model is the division of a whole population into a number of subpopulation groups (*subgroups*) to provide redundancy. They are allowed to gather their own set of global information. The one with the highest fitness is considered as the global solution.

In the course of searching, these subgroups are allowed to compete with each other for exclusive occupancy of territories. The aim of the competition is to force them to search different areas by separating them in the n-dimensional space. The competition is achieved by generating a repulsive force when two subgroups come together in the n-dimensional space. The closer the two subgroups, the greater the repulsive force. Once they are separated, the force disappears.

Effectively, pccBHS re-models the function landscape in such a way that the deceptive attractor is made hidden by another subgroup (see Figure 5.2). What each subgroup faces is a unimodal or non-deceptive landscape.

5.2.2 The cooperative-competitive model

Suppose there are G number of subgroups g_r, $0 \leq r < G$. Each of these subgroups is the same as a single population in pcBHS model. The only difference is the size of each subgroup which is equal to $\lfloor N/G \rfloor$. The cooperative pcBHS is a special case of the pccBHS model that $G = 1$.

For the sake of clarity, the model is presented using two subgroups only. Given two subgroups $g1$ and $g2$, we first check if all of their dimensions are overlapped, since two subgroups are said to be overlapped only when they are overlapped in *all* dimensions. Two metrics that are required to calculate the repulsive force are (i) *degree of overlapping* and (ii) *proximity*.

For each dimension m, we measure the distance F_m which is the maximum distance of all pairs of binary strings from the two subgroups in consideration (see Figure 5.3).

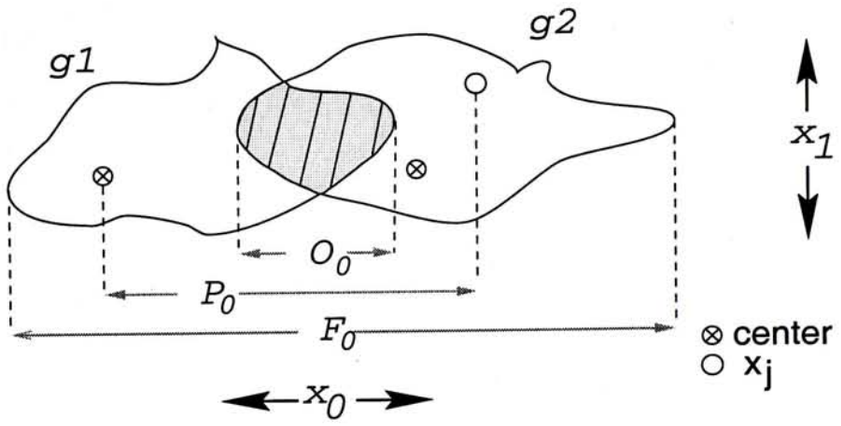


FIGURE 5.3: OVERLAPPING OF TWO SUBGROUPS

Denote $g1_m^{min}$ and $g1_m^{max}$ as the minimum and the maximum of the m-th dimension solution fragments of $g1$ respectively, $g2_m^{min}$ and $g2_m^{max}$ as the minimum and the maximum of m-th dimension solution fragments of $g2_m$ respectively. The distance F_m is:

$$F_m = \max\{g1_m^{max}, g2_m^{max}\} - \min\{g1_m^{min}, g2_m^{min}\}. \tag{5.1}$$

Minimum possible value of F_m is 0 when all of the m-th dimension solution fragments in $g1$ and $g2$ are identical, i.e. both $g1$ and $g2$ are merged together in the m-th dimension. Maximum possible value of F_m is $x_m^u - x_m^l$, i.e. the full range of the m-th dimension of x. We also measure the distance O_m of the region where they overlap (the shaded region in Figure 5.3). Overlapping distance O_m equals 0 when $g1_m^{max} < g2_m^{min}$ or $g2_m^{max} < g1_m^{min}$. Degree of overlapping $D_m(g1, g2)$ between the same dimension m of $g1$ and $g2$ is defined as:

$$D_m(g1, g2) = \frac{O_m}{F_m} \tag{5.2}$$

There are three distinct situations:

1. **Disjoint**

   $g1$ and $g2$ are totally separated.

$$D_m(g1, g2) = 0 \quad \text{when} \quad O_m = 0 \tag{5.3}$$

2. **Enclosure**

   $g1$ is totally enclosed by $g2$ or vice versa.

$$D_m(g1, g2) = 1 \quad \text{when} \quad \begin{cases} g1_m^{min} < g2_m^{min} \wedge g1_m^{max} > g2_m^{max}, & \text{or} \\ g2_m^{min} < g1_m^{min} \wedge g2_m^{max} > g1_m^{max} \end{cases} \tag{5.4}$$

3. **Overlapping**

$g1$ and $g2$ are overlapping.

$$0 < D_m(g1_m, g2_m) < 1 \quad \text{when} \quad \begin{cases} g1_m^{min} < g2_m^{min} < g1_m^{max} < g2_m^{max}, & \text{or} \\ g2_m^{min} < g1_m^{min} < g2_m^{max} < g1_m^{max} \end{cases} \tag{5.5}$$

The quantity D_m serves two purposes: (i) decides whether the repulsion exists; and (ii) determines the level of force required if repulsion exists. However, it does not reflect the fact that individuals farther away from the overlapping subgroup should receive less repulsive force. Thus, a *proximity value* is then introduced.

For the m-th dimension, proximity value $P_m(g1, x_{m,j})$ is defined over the j-th individual $x_{m,j}$ of $g2$ and its overlapping neighbor subgroup $g1$ as the normalized distance between $x_{m,j}$ and the center of $g1$ (see Figure 5.3):

$$P_m(g1, x_{m,j}) = \frac{|x_{m,j} - x_m^e|}{F_m} \tag{5.6}$$

where x_m^e is the m-th solution fragment of the elite in $g1$ and $P_m(g, x_{m,j}) \in [0, 1]$. Being the driving force within a subgroup, the subgroup elite is considered as the center.

Repulsive force $R_m(g1, x_{m,j}) \in [0, 1]$ experienced by the binary string $x_{m,j}$ of $g2$ due to the overlapping with $g1$ is defined as:

$$R_m(g1, x_{m,j}) = D_m(g1, g2) \times (1 - P_m(g1, x_{m,j})). \tag{5.7}$$

Finally, another quantity *interaction fitness* $I_m(x_{m,j}, x^e)$ is defined to indicate how well an individual performs in the competition: For the mth dimension,

$$I_m(x_{m,j}, x^e) = \frac{cf(x_{m,j}, x^e)}{R_m(g1, x_{m,j})} \tag{5.8}$$

where $cf(x_{m,j}, x^e)$ is the cooperative fitness of $x_{m,j}$ (see Eq. 4.3). Instead of feeding back cf into the system, I_m should be used:

$$\begin{cases} u_{m,i} = I_m(x_{m,j}, x^e) \text{ and } w_{m,i} = 0 & \text{if } b_{m,l-1-i} = 0, \\ u_{m,i} = 0 \text{ and } w_{m,i} = I_m(x_{m,j}, x^e) & \text{if } b_{m,l-1-i} = 1. \end{cases} \tag{5.9}$$

5.3 Empirical performance study

5.3.1 Goldberg's deceptive function

In the discussion of deception problem in GAs [27, 15], the sufficient condition for deception and the way to construct deceptive optimization function are presented. One of the functions that they have constructed is the *2l-bit bipolar deceptive function* ($2l$ is the length of the chromosome). This function is a fully deceptive function for GA.

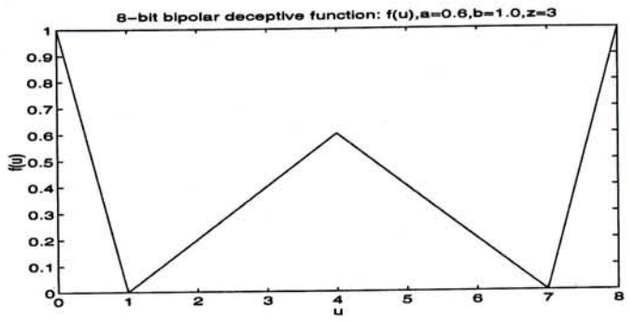


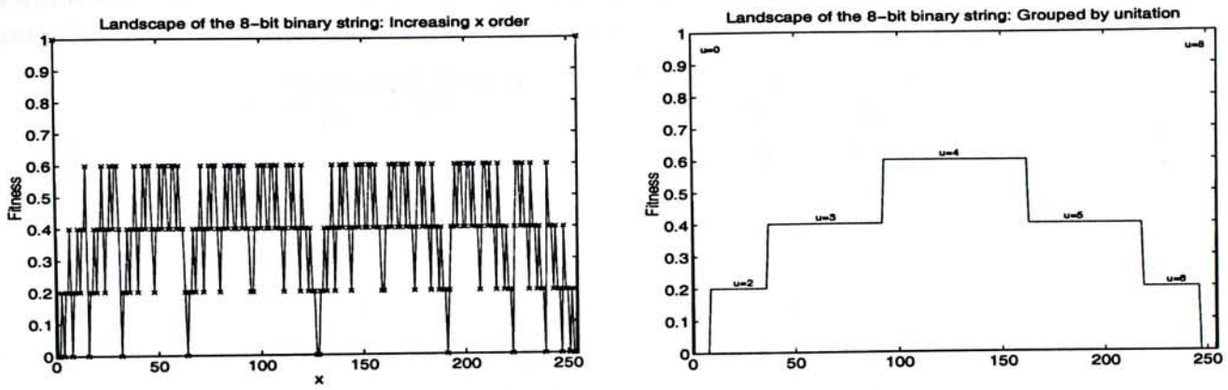
FIGURE 5.4: 8-BIT BIPOLAR DECEPTIVE FUNCTION (UNITATION VIEW): a=0.6, b=1.0, z=3

However, this function is probably not deceptive to our algorithm. As discussed in the following, the deceptive function is basically designed to deceive algorithms using bit-mutation operator. Since our algorithm does not have such operator, it will not be deceived as if GA. In spite of this, the $2l$-bit bipolar deceptive function is still a difficult function to our algorithm, because of the misleading information around the global optima and its multimodality (see Figure 5.5(a)). The bipolar deceptive function $f(u)$ for evaluating the binary strings is defined as follows (see Figure 5.4):

$$\begin{aligned} f(u) &= g(|u-l|), \\ g(e) &= \begin{cases} \frac{a}{z}(z-e), & \text{if } u \leq z \\ \frac{b}{l-z}(e-z), & \text{otherwise.} \end{cases} \end{aligned} \tag{5.10}$$

where l is half of the length of a binary string. In this experiment, l equals to 4 (i.e. binary string of 8 bits long). Unitation u of a binary string which is defined as the number of ones ('1') the string contains. For instance, the unitation of a string **00111101** is 5. Constants a, b, and z are used to define the shape of the function. The values used are $a = 0.6$, $b = 1.0$, and $z = 3$. The landscapes in the increasing x order and in the grouped unitation view are shown in Figure 5.5(a) and (b) respectively. Based on the argument in [37] that the landscape of a problem is tied to the operator used, the landscape of the function that GAs have to face should be the one shown in 5.5(b), provided that the mutation operator probability is $1/2l$, chromosome length $= 2l$ (i.e. Hamming distance $= 1$).

As mentioned, this bipolar function is probably not deceptive to our algorithm, we modified it to make it more difficult to our algorithm. The modification is to use the binary number equivalent instead of unitation as the function variable value. The function variable u in Eq. 5.10 is replaced by $u = \sum_{i=0}^{2l-1} 2^i * b_i$. Accordingly, z is changed to 32256 so as to keep the ratio of the size of both attractors (the deceptive one and the global one) the same as the $2l$-bit bipolar deceptive function. This modified function is shown in Figure 5.6.



(a) (b)

FIGURE 5.5: 8-BIT BIPOLAR DECEPTIVE FUNCTION. (A) THIS GRAPH SHOWS $f(u)$ IN AN INCREASING x ORDER SHOWING ITS MULTI-MODALITY. (B) THIS GRAPH GROUPS ALL x WITH SAME UNITATION TOGETHER AND ORDER THEM IN AN INCREASING UNITATION ORDER, I.E. THE UNITATION OF NEIGHBORING GROUPS DIFFER BY 1 ONLY. ANY NEIGHBOR CAN THEN BE REACHED BY ONE MUTATION. THE UNITATION u ARE LABELED CORRESPONDINGLY.

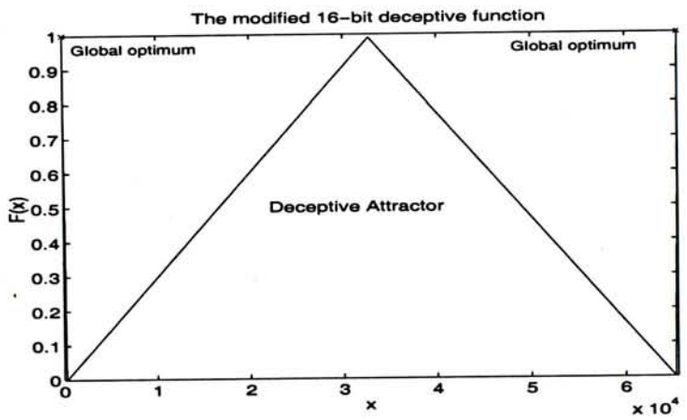


FIGURE 5.6: MODIFIED 16-BIT DECEPTIVE FUNCTION

The purpose of this experiment is to examine the global optimization capability of the pccBHS model on the modified bipolar function under different number of subgroups. Experiment on Goldberg's bipolar deceptive function is carried out to support our claim that it is not deceptive to our algorithm. Three criteria are used in evaluating this capability which are the success rate, the number of iterations and the number of function evaluations required on locating any one of the two true global optima. For both testings, the experimental conditions are set as: $N = \{10, 20, 30\}$, $G = \{1, 2, 3\}$, $\beta = 0.85$, $\mu = 1$, $\tau = 0.4$ and all are tried for 100 consecutive independent runs. The results are shown in Tables 5.1 and 5.2. Two conclusions can be drawn from the result: (1) Goldberg's

TABLE 5.1: RESULT ON THE MODIFIED 16-BIT DECEPTIVE FUNCTION AND THE GOLDBERG'S BIPOLAR DECEPTIVE FUNCTION: SUCCESS RATE (%)

G	Modified function			Goldberg's function		
	10	20	30	10	20	30
1	11	26	20	76	94	99
2	40	88	96	78	96	99
3	44	94	99	59	95	98

TABLE 5.2: RESULT ON THE MODIFIED 16-BIT DECEPTIVE FUNCTION AND THE GOLDBERG'S BIPOLAR DECEPTIVE FUNCTION: NUMBER OF FUNCTION EVALUATIONS

G	Modified function			Goldberg's function		
	10	20	30	10	20	30
1	581.5	1,074.6	1,432.0	109.6	147.7	180.0
2	657.6	950.5	1,227.7	142.5	302.0	348.8
3	629.7	1,061.4	1,266.8	160.7	254.3	347.4

bipolar deceptive function is not deceptive to our algorithm at all, and (2) it is useful to introduce multiple number of groups in our algorithm to handle deceptive problems. The high success rates obtained on solving Goldberg's function using one subgroup (first row of Table 5.1) implies that the function is not deceptive to our algorithm. This argument is further supported by the observation that the success rate is immune to the number of subgroups used. However, it is not the case for the modified function. The success rates obtained on using one subgroup is especially low, but they increase when multiple subgroups are used. We attribute the low success rates for the $N = 10$ case to the large stochastic error introduced when small population size is used, in addition to the deceptive nature of the function.

5.3.2 Shekel family – S5, S7, and S10

Besides handling GA deceptive problems, pccBHS improves the performance of pcBHS on problems having golf-hole-like landscape. Problems selected for testing belong to Shekel family—S5, S7 and S10. The landscapes of these three functions share a commonality: several sub-optima with similar basin of attractors sitting on a large flat plateau. The sizes of these attractors are so similar that no useful information can be

TABLE 5.3: SHEKEL FAMILY: PROBLEMS

Problems♯	n	pcBHS result		
		f^+	#eval	Succ. rate (%)
S5 - Shekel	4	10.004523	12,476.40	34%
S7 - Shekel	4	10.100106	13,168.60	29%
S10 - Shekel	4	10.126623	13,909.30	14%

TABLE 5.4: SHEKEL FAMILY: CONDITIONS

Problems	f^+	G	Conditions$^\sharp$
S5	9.9	1,2,3,4,5,6,7,8	$N = 100$
S7	9.9	1,2,3,4,5,6,7,8	$N = 100$
S10	9.9	1,2,4,6,8,10,12,14	$N = 140$

$\sharp$: 100 consecutive independent runs, $\mu = 1$, $\beta = 0.05$

obtained about where the global one is. This kind of landscape is normally referred as '*golf-hole*'-landscape.

A summary of these functions can be found in Table 5.3, while the experimental conditions are listed in Table 5.4. The purpose of this experiment is to illustrate the usefulness of redundancy with competition in improving the pcBHS model. Moreover, we examined the effect of different number of subgroups on the algorithm performance for these problems. The result is listed in Table 5.5. It shows clearly that by increasing the number of subgroup, the success rate can be increased, though using more computation.

5.4 Summary

In this chapter, we have discussed the challenges posed by deceptive problems and problems with golf-hole-like landscapes. The limitation of the basic pBHS and the cooperative pcBHS model in handling these two kinds of problems have been pointed out. This limitation is caused by the fact that only a single piece of information is carried by the global information. Both redundancy and competition are introduced in enhancing the model. By redundancy, we mean gathering of more than one piece of information of the global environment. By competition, we mean using of more than one population (subpopulation) to search for different areas of the landscape. Competition is introduced among them such that each subpopulation can search for a single sub-optimum with other sub-optima being masked out.

We have demonstrated the capability of this cooperative-competitive model in handling two deceptive problems and the Shekel family. One of the deceptive problems is Goldberg's bipolar deceptive function. The other one is the modification of it. We have shown that our algorithm can easily solve this GA-deceptive problem, owing to the fact that it is probably not deceptive to our algorithm. For the second deceptive function, which is especially designed to deceive our algorithm, we have to raise the number of subpopulations in order to increase the success rate, demonstrating the usefulness of this redundancy and competition model. Using the cooperative pcBHS model in handling the Shekel family, we got low success rate, but on using the enhanced model, they can be solved with high success rates.

Our competition model has a limitation. It requires a pre-set value indicating the number of sub-populations. It is disastrous that the model cannot be applied to problems that require the algorithm to locate all the optima. However, GAs suffer from the same difficulty.

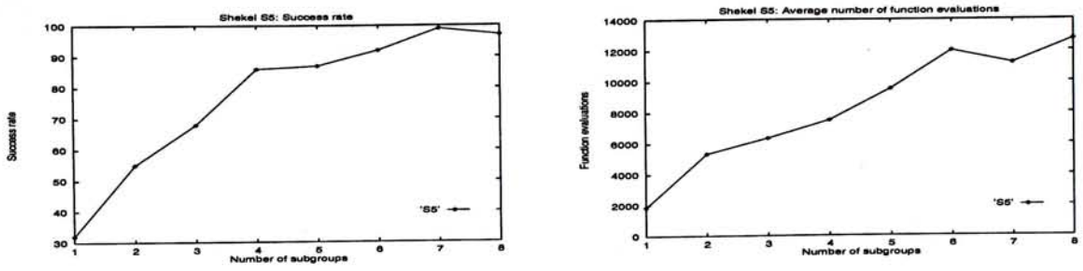
TABLE 5.5: SHEKEL FAMILY: RESULTS

S5			
G	Succ. rate (%)	f^+	#eval
1	32	10.030556	1,860.6
2	55	10.023210	5,323.7
3	68	10.012363	6,331.0
4	86	10.021969	7,487.6
5	87	10.011920	9,519.9
6	92	10.013254	11,991.6
7	**99**	**10.009583**	**11,196.6**
8	97	10.013743	12,717.8

S7			
G	Succ. rate (%)	f^+	#eval
1	31	10.171804	1,659.3
2	47	10.113239	4,885.9
3	66	10.121162	6,179.5
4	74	10.113394	6,841.3
5	89	10.145635	9,574.2
6	92	10.117482	9,579.1
7	95	10.098005	11,826.0
8	**99**	**10.100353**	**10,845.1**

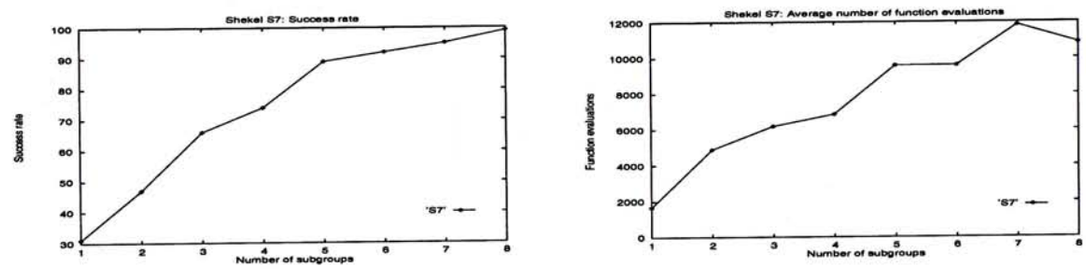
S10			
G	Succ. rate (%)	f^+	#eval
1	19	10.358868	2,170.7
2	31	10.189666	5,868.7
4	53	10.194831	7,526.4
6	74	10.135957	11,811.3
8	88	10.194929	13,040.2
10	97	10.169039	13,717.1
12	**99**	**10.164886**	**13,560.0**
14	99	10.173139	12,858.3

f^+: Average function value reached
#eval: Number of function evaluations



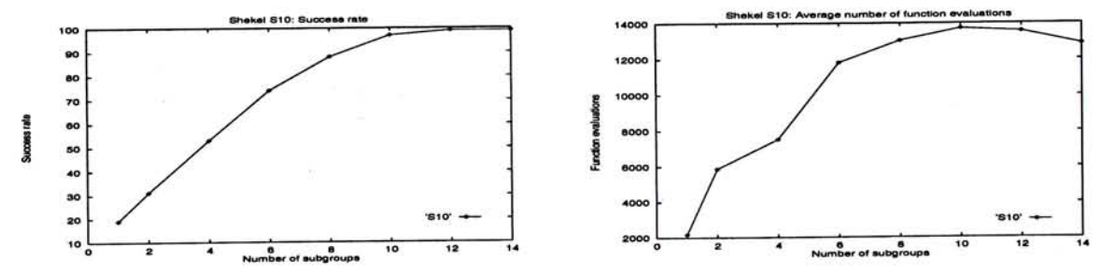
(a) (b)

FIGURE 5.7: SHEKEL FAMILY S5: (A) GRAPH SHOWING THE EFFECT OF DIFFERENT NUMBER OF SUBGROUPS ON THE PERCENTAGE OF RUNS GETTING THE GLOBAL OPTIMUM. (B) GRAPH SHOWING THE COMPUTATIONAL EXPENSES ON USING DIFFERENT NUMBER OF SUBGROUPS.



(a) (b)

FIGURE 5.8: SHEKEL FAMILY S7: (A) GRAPH SHOWING THE EFFECT OF DIFFERENT NUMBER OF SUBGROUPS ON THE PERCENTAGE OF RUNS GETTING THE GLOBAL OPTIMUM. (B) GRAPH SHOWING THE COMPUTATIONAL EXPENSES ON USING DIFFERENT NUMBER OF SUBGROUPS.



(a) (b)

FIGURE 5.9: SHEKEL FAMILY S10: (A) GRAPH SHOWING THE EFFECT OF DIFFERENT NUMBER OF SUBGROUPS ON THE PERCENTAGE OF RUNS GETTING THE GLOBAL OPTIMUM. (B) GRAPH SHOWING THE COMPUTATIONAL EXPENSES ON USING DIFFERENT NUMBER OF SUBGROUPS.

Part IV

Finale

CHAPTER 6

A new genetic operator

Being modeled as a population-based stochastic search using binary encoding, the basic pBHS is said to be easily integrated with canonical genetic algorithm. In this chapter, how GAs and our model are integrated is presented with experimental results to illustrate the benefit of the integration.

6.1 Introduction

In chapter 3, we pointed out that although GAs are effective algorithms, they rely on a large population size to keep the global information about the problem. By increasing the population size, the reliability of the global information can be increased. One of the advantages of our model is the use of memory. Generally, the longer the memory, the more reliable the global information. It can be seen that the approaches of both methods are complementary to each other. Consider the size of the population and the length of the convergence as two different dimensions: *width* and *depth*, GAs try to be as 'wide' as possible, while our model tries to be as deep as possible. Being complementary in nature, they are integrated so as to gain benefits from both.

Although GA and pBHS are two full-fledged algorithms, the integration is still easy owing to the fact that they are similar in several aspects. Firstly, both of them are classified as the same class of algorithm: *iterative probabilistic search.* Secondly, chromosome/binary string is the basic object to be manipulated. Thirdly, they are population-based approach.

In the following, we provided three variants of one integration model [42] which is based on the basic pBHS and canonical GA.

6.2 Variants of the integration

In this section, three variants of a GA-pBHS integration model are presented. Before describing the integration model, we list below their main characteristics:

- Merging of populations from both parties are taken. A control parameter called mix ratio γ is introduced to control the proportions of the chromosomes of GA and the binary strings of pBHS to be passed to the next generation.
- The operator pBHS is different in nature to the basic GA operators such as crossover and mutation. These basic GA operators can be said to be transformation functions mapping a population of chromosomes into another population of the same universal set. pBHS is different in that it generates a complete new set of binary strings instead of transforming the set from the last generation. Hence, in one aspect, the integration model has two cycles (GA cycle and pBHS cycle) running in parallel. In another aspect, pBHS can be viewed as an operator plugged into the GA cycle.

To aid in understanding, we show the integration model in Figure 6.1. On the left hand side of the figure, there shows a flowchart illustrating the standard components (selection, crossover, and mutation operators) of a typical canonical GA. In addition to these, we have pBHS at the end of the GA cycle. Since the basic pBHS model has memory effect, the ultimate function of this operator is to memorize/accumulate the searching experience gained in the genetic search. How to use the memory is subject to the design on specific usage. On the right hand side of the figure, the internal details of the pBHS operator is illustrated. The internal details of the operator is the same as that of the standalone basic pBHS algorithm except the followings:

- The introduction of a `Merging` component.
- It is the samples coming from both parties that are used for information gathering.
- The samples from both parties will become the next generation chromosomes of the GA, as opposed to the ordinary pBHS that all the samples are discarded.

This integration is flexible in that it is easy to be adapted to a pure GA, the pure pBHS or the integration of both. It is made possible by the mix ratio $\gamma \in [0.0, 1.0]$ mentioned before. This parameter determines the proportions of chromosomes of GA and binary strings of pBHS to be merged in the `Merging` component. Given a γ and a population size N, the merged population, i.e., the output of the `Merging` component, will have γN of chromosomes from GA and $(1-\gamma)N$ from pBHS. To turn the algorithm into a pure GA, one can switch off the pBHS operator totally by using $\gamma = 1.0$. The merged population will not have any binary strings generated from pBHS. Similarly, to

turn the algorithm into a pure pBHS, one can switch off GA totally by using $\gamma = 0.0$. With any other mix ratio, $0.0 < \gamma < 1.0$, the integrated model with varying fractions of GA chromosomes and pBHS binary strings will be resulted. The integration model is summarized in the Algorithm 6.1.

Algorithm 6.1 The integration of GA and pBHS. This algorithm is intended to show the logic of the integrated model, so the presence of multiple subpopulations and the competition are assumed.

Algorithm GA+pBHS

/* $P_{GA}(t)$ -- Population generated by GA at time t
* $P_{pBHS}(t)$ -- Population generated by pBHS at time t
*/

$t = 0$

Loop

$t = t + 1$

/* GA */

$P_{\chi}(t) \leftarrow \text{Crossover}(P(t-1))$

$P_{\mu}(t) \leftarrow \text{Mutation}(P_{\chi}(t))$

$F_{GA}(t) \leftarrow \text{Evaluation}(P_{\mu}(t))$

$P_{GA}(t) \leftarrow \text{Selection}(P_{\mu}(t), F_{GA}(t))$

/* pBHS */

$P_{pBHS}(t) \leftarrow \text{pBHS\_Generation}(a(t-1))$

$F_{pBHS}(t) \leftarrow \text{pBHS\_Evaluation}(P_{pBHS}(t))$

/* Meeting point */

$(P(t), F(t)) \leftarrow \text{Merging}(P_{GA}(t), P_{pBHS}(t), F_{GA}, F_{pBHS})$

$a(t) \leftarrow \text{Information\_gathering}(P(t), F(t))$

until stopping criteria are met.

end Algorithm

So far, we have not discussed the detail of the merging component besides the fraction of chromosomes/binary strings from both parties. In the following, we provided three merging methods: (1) Fixed-fraction-of-all; (2) Fixed-fraction-of-best; and (3) Best-from-both.

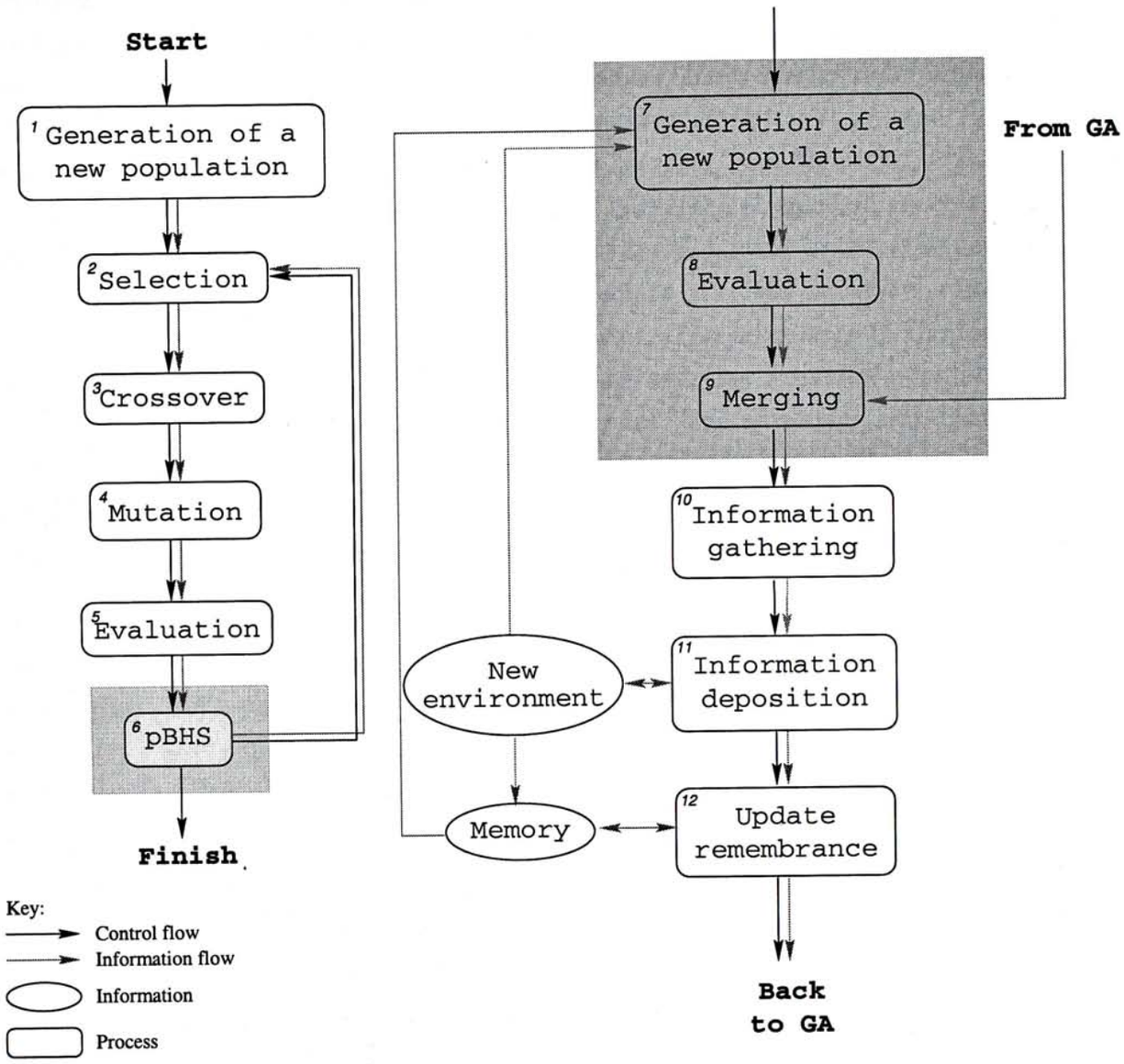


FIGURE 6.1: INTEGRATION OF GAS AND PBHS. LEFT HAND SIDE OF THE FIGURE IS AN OVERVIEW OF THE INTEGRATION MODEL. THE SHADED REGION AT THE BOTTOM IS THE NEWLY INTRODUCED OPERATOR **pBHS**. THIS IS THE FIRST MODULE OF THE INTEGRATION MODEL. THE ALGORITHM BECOMES A PLAIN GA WHEN THE OPERATOR IS REMOVED. THE DIAGRAM ON THE RIGHT HAND SIDE IS THE DETAIL OF THE **pBHS** OPERATOR.

6.2.1 Fixed-fraction-of-all

The idea behind this variant is to ensure a fixed proportion of contributions from both GA and pBHS. Since the samples generated are the main source of information in creating the searching environment, contribution is defined as the amount of samples of each party (GA and pBHS) injected into the next iteration. In this variant, both GA and pBHS will generate the amount of chromosomes/binary strings determined by mix ratio γ. The size of the intermediate populations such as $P_\chi(t)$, $P_\mu(t)$, and $P_{GA}(t)$ are γN, while that of $P_{pBHS}(t)$ is $(1-\gamma)N$. After merging (see Algorithm 6.2), the size of the resultant population $P(t)$ will become N. Since the population generated in the current generation will become the population of the GA cycle in the next generation, it is necessary to restore the large population back to γN. We can either restore it by the crossover operator or by the selection operator. The advantage of the former one is saving computation by mutating, evaluating and selecting a smaller population. The disadvantage is the loss of some possibly good genes. The advantage and disadvantage of the latter one are that the larger gene pool can stay longer in the population and the needs of a considerable amount of computation (the manipulation of γN versus N) respectively. In order to save computation, we chose the former method.

Algorithm 6.2 Merging: Fixed fractions of all from both

/* $P_{GA}(t-1)$ - `Population from GA at time` t-1
* $P_{pBHS}(t-1)$ - `Population from pBHS at time` t-1
* $P_{GA}(t)$ - `New population to GA at time` t
* $P_{GA}(i)(t)$ - `The` i-`th element of the new population to GA at time` t
*/

Function Merging($P_{GA}(t-1)$, $P_{pBHS}(t-1)$)

$P_{GA}(i)(t) \leftarrow P_{GA}(i)(t-1), \quad \forall\, i \in [0, \gamma N - 1]$

$P_{GA}(j+\gamma N)(t) \leftarrow P_{pBHS}(j)(t-1), \quad \forall j \in [0, (1-\gamma)N - 1]$

return $P_{GA}(t)$

end Function

6.2.2 Fixed-fraction-of-best

This merging scheme is similar to the Fixed-fraction-of-all scheme that the parameter mix ratio γ is required to control the relative proportions of GA chromosomes and pBHS binary strings. However, the γN chromosomes from GA are the best *selected* chromosomes from a size N GA population while the $\gamma(1-N)$ binary strings from pBHS are the best *selected chromosomes* from a size N pBHS population. To make this scheme possible, both GA and pBHS have to maintain a population of size N respectively. This

merging scheme is shown in Algorithm 6.3.

Algorithm 6.3 Merging: Fixed fractions of best from both

/* $P_{GA}(t-1)$ - Population from GA at time t-1
* $P_{pBHS}(t-1)$ - Population from pBHS at time t-1
* $P_{GA}(t)$ - New population to GA at time t
* $P_{GA}(i)(t)$ - The i-th element of the new population to GA at time t
*/

Function Merging($P_{GA}(t-1)$, $P_{pBHS}(t-1)$)

Sort $P_{GA}(t-1)$ in descending order (for maximization) according to raw fitness

Sort $P_{pBHS}(t-1)$ in descending order (for maximization) according to raw fitness

$P_{GA}(i)(t) \leftarrow P_{GA}(i)(t-1), \quad \forall\, i \in [0, \gamma N - 1]$

$P_{GA}(j+\gamma N)(t) \leftarrow P_{pBHS}(j)(t-1), \quad \forall j \in [0, (1-\gamma)N - 1]$

return $P_{GA}(t)$

end Function

6.2.3 Best-from-both

This merging method is similar to the Fixed-fraction-of-all scheme with the following two differences:

- Removal of the parameter — mix ratio γ.
- Each of the two parties (GA and pBHS) have to generate and evaluate N individuals in each iteration as if the Fixed-fraction-of-best scheme.

The purpose of the removal of the mix ratio parameter is to allow a dynamic mixing of chromosomes and binary strings. In the fixed fraction model, some individuals which have higher fitness than some individuals are not selected for merging owing to the quota imposed by the fraction parameter γ. In order to make use of all the best individuals found in both models, the fraction parameter is removed and the best N individuals from both populations are selected for the population of the next generation. It is illustrated in Algorithm 6.4.

6.3 Empricial performance study

The purpose of this study is to illustrate the performance of the integrated model. The three variants discussed in the previous sections were tested using a number of

Algorithm 6.4 Merging: Best from both

```
/*  P_pBHS(t) — Population of pBHS at time t
*   P_GA(t) — Population of GA at time t
*   P_GA(i)(t) - The i-th element of the new population to GA at time t
*/
Function Merging( P_GA(t − 1), P_pBHS(t − 1) )
    Sort P_GA(t − 1) in descending order (for maximization) according to raw fitness
    Sort P_pBHS(t − 1) in descending order (for maximization) according to raw fitness
    k = 0, j = 0, i = 0
    for i ← [0, N−1]
        if P_GA(j)(t − 1) < P_pBHS(k)(t − 1) then
            P_GA(i)(t) ← P_pBHS(k)(t − 1)
            k = k + 1
        else if P_GA(j)(t − 1) > P_pBHS(k)(t − 1) then
            P_GA(i)(t) ← P_GA(j)(t − 1)
            j = j + 1
        else if RAND# < 0.5 then
            P_GA(i)(t) ← P_pBHS(k)(t − 1)
            k = k + 1
        else
            P_GA(i)(t) ← P_GA(j)(t − 1)
            j = j + 1
        end if
    end for
    return P_GA(t)
end Function
```

TABLE 6.1: PERFORMANCE STUDY ON THE INTEGRATION MODEL: EXPERIMENTAL SETUP

Test functions		Conditions†			
		f^+	N	β‡	G
Rastrigin	(n=2)	1.9995	50	0.85/0.80/0.70	2
Hartman	(n=3)	-3.860	30	0.75/0.70/0.70	1
Shubert	(n=3)	186.73	60	0.90/0.85/0.85	3
Shekel	(n=4)	9.9	96	0.98/0.95/0.95	12

†: Common condition: $\gamma = \{0.00, 0.25, 0.50, 0.75, 1.00\}$

‡: $a/b/c$ means the β values for the fixed-fraction-of-all, the fixed-fraction-of-best and the best-from-both merging schemes are a, b, and c respectively.

TABLE 6.2: PERFORMANCE OF THE HYBRID MODEL - RASTRIGIN FUNCTION (N=2)

γ	Fixed-fraction-of-all		Fixed-fraction-of-best		Best-from-both	
	Succ. rate	#eval.	Succ. rate	#eval.	Succ. rate	#eval.
0.00	93	1838.1	93	2324.2		
0.25	99	1844.5	97	3658.8		
0.50	100	2018.2	99	3493.9	98	3168.4
0.75	100	2192.5	100	4120.0		
1.00	100	3676.0	100	3676.0		
Condition: $N = 50$, $\beta = 0.95$						
0.00	99	2364.3	99	2364.3		

commonly used low dimension functions: Rastrigin function (n=2), Hartman function (n=3), Shubert (n=3) and Shekel (n=4) functions. In order to illustrate the effect of different proportions of pBHS and GA mixture, we tried 5 different proportions $\gamma = \{0.00, 0.25, 0.50, 0.75, 1.00\}$. As mentioned before, γ equals 1.0 is a pure GA, while γ equals 0.0 is a pure pBHS. The experimental setup is shown in Table 6.1 and the results are shown in Tables 6.2-6.5. The five mix ratio γ used are listed on the leftmost column of each of the result tables. Subsequent columns show the success rates (percentage of consecutive trials out of 100 trials reached the prescribed f^+) and the average number of evaluations required for each of the merging schemes. The results of the two fixed-fraction schemes for $\gamma = 0.00$ and $\gamma = 1.00$ are the same (unless the β values used are different), since pure pBHS and pure canonical GA need no merging and hence should have no theoretically difference under both schemes. For the Best-from-both merging scheme, only one result is reported for each test function. It is because the mix ratio γ is not applicable. At the bottom of each result table, there is a row reporting a result for $\gamma = 0.00$ and the corresponding experimental condition. This result is reported especially for comparison. To be more specific, in Table 6.2, the result reported for $\gamma = 0.00$ is 1838.1 number of function evaluations on average to achieve 93% success rate. Since

TABLE 6.3: PERFORMANCE OF THE HYBRID MODEL - HARTMAN FUNCTION (N=3)

γ	Fixed-fraction-of-all		Fixed-fraction-of-best		Best-from-both	
	Succ. rate	#eval.	Succ. rate	#eval.	Succ. rate	#eval.
0.00	96	483.1	96	450.0		
0.25	97	489.6	93	794.8		
0.50	97	554.9	97	792.4	95	876.0
0.75	96	571.6	99	772.1		
1.00	88	1302.6	80	1302.6		
Condition: N =10, β =0.95						
0.00	100	709.9	100	709.9		

TABLE 6.4: PERFORMANCE OF THE HYBRID MODEL - SHUBERT FUNCTION (N=3)

γ	Fixed-fraction-of-all		Fixed-fraction-of-best		Best-from-both	
	Succ. rate	#eval.	Succ. rate	#eval.	Succ. rate	#eval.
0.00	94	4942.3	94	3360.0		
0.25	97	5585.6	96	7322.5		
0.50	93	5729.0	93	7127.8	95	7908.6
0.75	94	6608.3	96	6356.3		
1.00	95	9314.5	97	9314.5		

the algorithm using $\gamma = 0.00$ is essentially a pure pBHS algorithm, to compare the pure pBHS and the hybrid algorithms, we have to report the number of function evaluations to achieve the same level of performance as the hybrid algorithms. The bottom row of result table is dedicated for this purpose.

From the experimental results, there are three observations:

- The canonical GA performed poorly compared to the rest of the hybridized cases ($\gamma = \{ 0.25, 0.50, 0.75 \}$) in all of the functions tested.
- The hybridized cases ($\gamma = \{ 0.25, 0.50, 0.75 \}$) perform similarly to the pure pBHS model. For some tests, the hybridized cases outperformed pBHS. One implication of the result is that the performance of the canonical GA is not good enough to dominate the pBHS operator. It is suggested that a good balance of the performance of both parties is needed.

TABLE 6.5: PERFORMANCE OF THE HYBRID MODEL - SHEKEL FUNCTION (N=4)

γ	Fixed-fraction-of-all		Fixed-fraction-of-best		Best-from-both	
	Succ. rate	#eval.	Succ. rate	#eval.	Succ. rate	#eval.
0.00	84	38653.3	79	16445.2		
0.25	89	40518.4	89	32074.8		
0.50	95	44517.7	95	33181.6	94	30244.2
0.75	90	40129.1	85	27119.4		
1.00	46	65027.5	41	69265.2		

- Both the Fixed-fraction-of-best and the Best-from-both merging scheme showed no advantages over the Fixed-fraction-of-all scheme. It is because both the pBHS and the GA have to generate populations of size N. The number of function evaluations required in each iteration is doubled and hence the drop in performance. The result indicates that the extra computation used in both the Fixed-fraction-of-best and the Best-from-both schemes are not justified.

- The performance of the Fixed-fraction-of-best is generally outweigh that of the Best-from-both merging scheme. It is possibly due to the high greediness of the latter approach that extra computation has to be used to overcome local traps.

6.4 Summary

In this chapter, the model of incorporating pBHS into the canonical GA as an genetic operator has been presented. The role of the pBHS operator is firstly, the accumulation of searching experience of the past which includes those from the GA, and secondly injecting new and diverse binary strings into GA population. The result showed that the hybrid model outperformed canonical GA while having similar or slightly outperformed the pure pBHS algorithm. This hybridization is a preliminary trial and obviously requires substantial improvements. However, the increase in performance, though minimal, indicates that the hybridized model worth further investigation. For example, different merging techniques besides the fixed-fraction and best-from-both schemes are required.

Chapter 7

Conclusion and Future work

In this thesis, a new iterative stochastic searching algorithm called *probabilistic cooperative-competitive binary hierarchical search pccBHS* is proposed. It divides the n-dimensional search space into partitions and organizes the partitions into a binary hierarchy. As discussed, this organization provides a basis for resolution control and reduction of search space size. To bring these two features into reality, an algorithm has to be designed to take the advantages of the hierarchical organization. Hence, we have presented an algorithm which is an information processing cycle. In the cycle, a population of searching agents cooperatively search for the best possible result and construct a global environment in a collaborative manner. This global environment is the information representing the global nature of the problem to be solved. This global information is in turn used by the agents in the searching and the global environment construction. This cycle goes on continuously until the desired solution is obtained. The aim of the construction of global information is to provide reliable guidance to the agents in the future search. In fact, the reliable guidance is made possible by the effective pruning of sub-optimal areas of the search space and the smoothing of the rugged surface (the local optima are known to be traps of searching algorithms). By gathering information to construct the global information, sub-optimal search space can be eliminated from consideration. By organizing the search space hierarchically with resolution increasing towards the bottom of the hierarchy, the difficulty caused by the rugged surface can be lowered.

The positive feedback nature of the algorithm allows locating the solution very fast. However, it makes the algorithm trap inside the local optima easily. Competition has been introduced to level off the strong exploitation effect. By competition, we mean the use of multiple number of populations to search separately while keeping a repulsive force among them. By this way, each population can search in high speed to exploit the information gathered and at the same time maintain the solution quality by searching diversely. Besides this main purpose, this competition model has one advantage over

existing techniques is that it does not need a pre-defined *radius* which inherently limits the algorithm from finding optima in different resolutions simultaneously. Nevertheless, it has limitations. One of the limitations is that it favors for functions with few big optima. It does not mean that it is not applicable to massively multimodal functions. In fact, we have demonstrated the performance of the algorithm in solving a massively multimodal function. If the functions exhibit correlation among neighboring points, the algorithm is still capable of solving it. To remind, our algorithm is designed with resolution control in mind. Massively multimodality would not be a difficulty.

Dealing with high-dimensional functions, we have adopted a decoupling scheme. Simply, for a n-dimensional function, each of the n dimensions is handled by a single population (subpopulation). A n-dimensional solution is constructed by picking solution one from each subpopulation. The function value is no longer the sole and determining criterion in evaluating the n-dimensional solutions. Instead, cooperative fitness which favors for solutions that can produce better function value is used. The decoupling scheme improves the algorithm significantly on solving high-dimensional function.

Unless the algorithm is fully adaptive, algorithm parameter is not only unavoidable, but also vital to the versatility of the algorithm. Our algorithm provides several parameters, such as population size, remembrance, and the number of subpopulations, for controlling its properties.

Being categorized in the iterative stochastic search, our algorithm and GA share similarities. They have been integrated to form a hybrid algorithm. The hybrid algorithm is designed with GA being the backbone and our algorithm being a special genetic operator. Chromosomes injected from GA into the special genetic operator will be passed to the global information gathering component in which the information of the chromosomes are extracted. New chromosomes are generated by our algorithm. The GA chromosomes and those newly generated are mixed in different proportions. They are then passed back to the GA in the next generation. The preliminary result reported indicates that the hybrid algorithm can improve the performance of GA.

In this work, we exploited very minimal potential of the resolution control property of the hierarchy and the gathered global information. Hence, one of the future work would be the design of a better adaptive learning algorithm and a better searching mechanism to exploit the potential. Another future work would be the improvement of the GA+pBHS integration model such as providing new merging schemes and investigating the effect on using different GA algorithms.

APPENDIX A

The pBHS Algorithm

The following is the basic pBHS algorithm in detail. Algorithmic complexity of the algorithm is analyzed. For the ease of reading, the algorithm is presented in two levels of abstraction.

A.1 Overview

Step 1:	Initialization	$\mathcal{O}(nl)$
Step 2:	Generation of a new population	$\mathcal{O}(Nnl)$
Step 3:	Evaluation	$\mathcal{O}(N\,F(x))$
Step 4:	Fitness scaling	$\mathcal{O}(N)$
Step 5:	Information gathering	$\mathcal{O}(Nnl)$
Step 6:	Information deposition	$\mathcal{O}(nl)$
Step 7:	Adjustment of remembrance values	$\mathcal{O}(nl)$
Step 8:	Goto **Step 2** if (i) max.generations is not reached; and (ii) stopping criteria is not met	

FIGURE A.1: BASIC pBHS ALGORITHM OVERVIEW

As shown in above, steps 2 to 7 are enclosed in a loop running continuously for the maximum allowed generations specified or until the stopping criteria are met, so the maximum allowed generations T is taken into consideration on analyzing the complexity. On the whole, the algorithmic complexity of the algorithm is $O(T\,N\,n\,l)$ if the algorithmic complexity of the function to be optimized is less than or equal to $O(n\,l)$. Otherwise, it is equal to $O(T\,N\,F(x))$.

A.2 Details

Algorithm A.1 STEP 1 ALGORITHMIC COMPLEXITY: $O(n \cdot l)$

```
/* Step 1: Initialization */
for m = 0 to n-1
    for i = l–1 to 0
        a_{m,k} = 0.5
        a_{m,k+1} = 0.5
    end for
end for
```

Algorithm A.2 STEP 2 ALGORITHMIC COMPLEXITY: $O(N \cdot n \cdot l)$

```
/* Step 2: Generation of new population */
for j = 0 to N – 1
    /* Generate for all variables */
    for m = 0 to n – 1
        /* Generate a binary string */
        for i = l–1 to 0
            if RAND# < a_{m,k}
                b_{j,m,i} = 0
            else
                b_{j,m,i} = 1
            end if
        end for
    end for
end for
```

Algorithm A.3 STEP 3 ALGORITHMIC COMPLEXITY: $O(N \cdot F(x))$

```
/* Step 3: Evaluation */
for j = 0 to N – 1
    for m = 0 to n – 1
        x_{j,m} = s_{j,m}/V * (x_max_m – x_min_m) + x_min_m
    end for
    f_j = F(x_j)
end for
```

Algorithm A.4 STEP 4 ALGORITHMIC COMPLEXITY: $O(N)$ IF $N > m$ OR $O(m)$ OTHERWISE.

```
/* Step 4: Fitness scaling */
/* Find min. and max. fitness */
F_min = f_0; F_max = f_0
for j = 1 to N - 1
    if f_j < F_min then
        F_min = f_j
    else if f_j > F_max then
        F_max = f_j
        best_of_population = j
    end if
end for
if elitism is used then
    if f_best_of_population > F_elite then
        F_elite = f_best_of_population
        for m = 0 to n - 1
            s_elite,m = s_j,m
        end for
    else
        F_max = F_elite
    end if
end if

/* Scale fitness values */
for j = 0 to N - 1
    f_j = (f_j - F_min) / (F_max - F_min)
end for
```

Algorithm A.5 STEP 5 ALGORITHMIC COMPLEXITY: $O(N \cdot n \cdot l)$

```
/* Step 5: Information gathering */
for j = 0 to N − 1
    for m = 0 to n − 1
        for i = l–1 to 0
            if b_{j,m,i} == 0
                h_{m,k} += f_j
            else
                h_{m,k+1} += f_j
            end if
        end for
    end for
end for
if elitism is used
    for m = 0 to n − 1
        for i = l–1 to 0
            if b_{elite,m,i} == 0
                h_{m,k} += N/μ
            else
                h_{m,k+1} += N/μ
            end if
        end for
    end for
end if
```

Algorithm A.6 STEP 6 ALGORITHMIC COMPLEXITY: $O(n \cdot l)$

```
/* Step 6: Information deposition */
for m = 0 to n − 1
    for i = l–1 to 0
        /* Normalization */
        h_{m,k} = h_{m,k} / (h_{m,k} + h_{m,k+1})
        h_{m,k+1} = 1 - h_{m,k}
        /* Update */
        a_{m,k} = β_{m,i} a_{m,k} + (1 − β_{m,i}) h_{m,k}
    end for
end for
```

Algorithm A.7 STEP 7 ALGORITHMIC COMPLEXITY: $O(n \cdot l)$

```
/* Step 7: Adjustment of remembrance values */
for m = 0 to n − 1
    /* Search for the last converged point */
    r = l − 1
    for i = l–1 to 0
        if   | 0.5 − a_{m,2(l−1−i)} | > τ ∨
             | 0.5 − a_{m,2(l−i)} | < τ  then
             r = i
             break
        end if
    end for
    /* Update β_{m,i} */
    for i = l–1 to 0
        if i >= r then
            β_{m,i} = β
        else
            β_{m,i} = (r − i + β)/(r − i + 1)
        end if
    end for
end for
```

APPENDIX B

Test problems

Shekel family

For $0.0 \leq x_j \leq 10.0$, $1 \leq j \leq n$,

$$F_s(x) = \sum_{i=1}^{m} \frac{1}{(k_i(x - a_i))^2 + c_i}, \tag{B.1}$$

- S1, $n = 1$ [55, 58]

  The one-dimensional Shekel function has the global optimum $F_s(x) = 14.5926520$ located at $x = 0.6858609$. where $m = 10$

i	a_i	k_i	c_i	i	a_i	k_i	c_i
1	3.040	2.983	0.192	**6**	8.679	1.236	0.189
2	1.098	2.378	0.140	**7**	4.503	2.868	0.187
3	0.674	2.439	0.127	**8**	3.328	1.378	0.171
4	3.537	1.168	0.132	**9**	6.937	2.348	0.188
5	6.173	2.406	0.125	**10**	0.700	2.268	0.176

- Sm, $n = 4$, $m = \{5, 7, 10\}$ [58]

  There are three members of dimension equal to four in the Shekel family. All have global optimum approximately equals to $1/c_1$ at a_1.

i	a_i				c_i	i	a_i				c_i
1	4	4	4	4	0.1	**6**	2	9	2	9	0.6
2	1	1	1	1	0.2	**7**	5	5	3	3	0.3
3	8	8	8	8	0.2	**8**	8	1	8	1	0.7
4	6	6	6	6	0.4	**9**	6	2	6	2	0.5
5	3	7	3	7	0.4	**10**	7	3.6	7	3.6	0.5

Ackley family

Only one member (A30) of dimension 30 ($n = 30$) in the family was used. For $-30.0 \leq x_i \leq 30.0$,

$$F_{A30} = 20\exp(-20\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) + \exp(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)) - 20 - e. \quad \text{(B.2)}$$

Rastrigin family

- R2 $n = 2$ The two-dimensional Rastrigin function is a function having 50 optima with one global optimum $F_r(x) = 2.000000$ located at $x = (0, 0)$. For $-1.0 \leq x_i \leq 1.0$, $i = 1, 2$,

$$F_{R2}(x) = -x_1^2 - x_2^2 + \cos(18x_1) + \cos(18x_2). \quad \text{(B.3)}$$

- Rn, $n = \{20, 50, 100, 200, 400\}$ [59]

  For $-500.0 \leq x_i \leq 500$,

$$F_{Rn}(x) = 10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i)). \quad \text{(B.4)}$$

Goldstein-Price function

- GP2 $n = 2$

  For $-2.5 \leq x_i \leq 2.0$, $i = 1, 2$,

$$\begin{aligned} F_{GP2} \;=\; & -[1 + (x_1 + x_2 + 1)^2(10 - 14x_1 + 2x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \cdot \\ & [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)] \end{aligned} \quad \text{(B.5)}$$

  The global optimum is $f^* = -3.000$ at $x = [0\ -1]$.

Michalewitz family

- M5 $n = 5$

  For $0 \leq x_i \leq \pi$,

$$F_{M5} = \sum_{i=1}^{n} \sin(x_i)\sin((i+1)x_i^2/\pi)^{2m} \quad \text{(B.6)}$$

  The global optimum is $f^* = 4.687$.

Sphere family

Five members of this family are used: SP8, SP16, SP32, SP64, SP128. For $-5.12 \leq x_i \leq 5.12$,

$$F_{SPn}(x) = \sum_{i=1}^{n} (x_i - 1)^2. \tag{B.7}$$

Hartman family

Only one member of this family [58] is considered:

- H3, $n = 3$, For $0.0 \leq x_j \leq 1.0$, $1 \leq j \leq 3$,

$$F_{H3}(x) = -\sum_{i=1}^{4} c_i \exp(-\sum_{j=1}^{n} a_{i,j}(x_j - p_{i,j})^2). \tag{B.8}$$

$$a = \begin{bmatrix} 0.36890 & 0.11700 & 0.26730 \\ 0.46990 & 0.43870 & 0.74700 \\ 0.10910 & 0.87320 & 0.55470 \\ 0.03815 & 0.57430 & 0.88280 \end{bmatrix} \quad p = \begin{bmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{bmatrix} \quad c = \begin{bmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{bmatrix} \tag{B.9}$$

Goldberg's bipolar deceptive function

Binary strings of eight bits long is used. Given $2l = 8$, $a = 0.99$, $b = 1.00$, and $z = 3$,

$$\begin{aligned} F_{bpd}(u) &= g(|u - l|), \\ g(e) &= \begin{cases} \frac{a}{z}(z - e), & \text{if } u \leq z \\ \frac{b}{l-z}(e - z), & \text{otherwise.} \end{cases} \end{aligned}$$

u is the unitation of a binary string. It is defined as the number of ones in the string. For a binary string of length $2l$, $u = [0, 2l]$. Global optimum is 1.0.

Modified Goldberg's bipolar deceptive function

Binary strings of eight bits long is used. Given $2l = 16$, $a = 0.99$, $b = 1.00$, and $z = 32521$,

$$\begin{aligned} F_{bpd}(u) &= g(|u - l|), \\ g(e) &= \begin{cases} \frac{a}{z}(z - e), & \text{if } u \leq z \\ \frac{b}{l-z}(e - z), & \text{otherwise.} \end{cases} \end{aligned}$$

$u = [0, 65535]$ is the decimal equivalent of the binary number. Global optimum is 1.0.

BIBLIOGRAPHY

[1] A. Albrecht, S.K. Cheung, K.C. Hui, K.S. Leung, and C.K. Wong. Optimal placements of flexible objects. i. analytical results for the unbounded case. *IEEE Transactions on Computers*, (8):890–904, 8 1997.

[2] A. Albrecht, S.K. Cheung, K.C. Hui, K.S. Leung, and C.K. Wong. Optimal placements of flexible objects. ii. a simulated annealing approach for the bounded case. *IEEE Transactions on Computers*, (8):905–29, 8 1997.

[3] F. Aluffi-Pentini, V. Parisi, and F. Zirilli. Global optimization and stochastic differential equations. *Journal of Optimization Theory and Applications*, 47:1–16, 1985.

[4] T. Back. *Evolution strategies: an alternative evolutionary algorithm*, pages 3–20. Springer-Verlag; Berlin, Germany, 1996.

[5] J.E. Baker. Adaptive selection methods for genetic algorithms. In J.J. Grefenstette, editor, *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 101–111. Lawrence Erlbaum Associates, 1985.

[6] D. Beasley, D.R. Bull, and R.M. Ralph. An overview of genetic algorithms: Part 1, fundenmental. *University Computing*, 15(2):58–69, 1993.

[7] Kenneth Dean Boese. *Models for Iterative Global Optimization.* PhD dissertation, University of California, Los Angeles, 1996.

[8] P. Brachetti, M. De Felice Ciccoli, G. Di Pillo, and S. Lucidi. A new version of the price algorithm for global optimization. *J. Global Optimization*, 10, 1997.

[9] D.J. Cavicchio. *Adaptive search using simulated evolution.* PhD thesis, Ann Arbor, MI:University of Michigan, 1970.

[10] Beasley D., Bull D.R., and Martin R.R. A sequential niche technique for multimodal function optimization. *Evolutionary Computation*, 1(2):101–125, 1993.

[11] K.A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, The University of Michigan, 1975.

[12] K. Deb and D.E. Goldberg. An investigation of niche and species formation in genetic function optimization. In J.D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 42–50. San Mateo, CA: Morgan Kaufmann, 1989.

[13] K. Deb and D.E. Goldberg. An investigation of niche and species formation in genetic function optimization. In J.D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 42–50. San Mateo, CA; Morgan Kaufmann, 1989.

[14] K. Deb and D.E. Goldberg. Sufficient conditions for deceptive and easy binary functions. *Annals of Mathematics and Artificial Intelligence*, 10(4):385–408, 1994.

[15] K. Deb, J. Horn, and D.E. Goldberg. Multimodal deceptive function. *Complex Systems*, 7:131–153, 1993.

[16] A. Dekkers and E. Aarts. Global optimization and simulated annealing. *Mathematical programming*, 50:367–393, 1981.

[17] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics*, 26(1):29–41, 2 1996.

[18] P. Dutta and D.D. Majumder. Convergence of an evolutionary algorithm. In T. Yamakawa and G. Matsumoto, editors, *Proceedings of the 4th International Conference on Soft Computing (IIZUKA '96) - Methodologies for the Conception, Design, and Application of Intelligent Systems*, pages 515–518. World Scientific; Singapore, 1996.

[19] A.E. Eiben, E.H.L. Aarts, and K.M. Van Hee. Global convergence of genetic algorithms: A markov chain analysis. In H.P. Schwefel and R. Mănner, editors, *Parallel Problem Solving from Nature*, pages 4–12. Heidelberg,Berlin: Springer-Verlag, 1991.

[20] D.B. Fogel. Asymptotic convergence properties of genetic algorithms and evolutionary programming: Analysis and experiments. *Cybernetics and Systems*, 25(3):389–407, 1994.

[21] D.B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, 1 1994.

[22] Stephanie Forrest. *Emergent computation: self-organizing, collective, and cooperative phenomena in natural and artificial computing networks.* Special issue of Physica D. Cambridge, Mass; London: MIT Press, 1991.

[23] A.J. Gaul, E. Handschin, and W. Hoffmann. Optimal management of electrical loads using knowledge aware evolutionary strategies. *Engineering Intelligent Systems for Electrical Engineering and Communications*, 5(1):21–8, 1997.

[24] D.E. Goldberg. Simple genetic algorithms and the minimal, deceptive problem. In Lawerance Davis, editor, *Genetic algorithms and simulated annealing*, pages 74–88. London: Pitman, 1987.

[25] D.E. Goldberg. Genetic algorithms and walsh functions: Part ii. deception and its analysis. *Complex systems*, 3(2):153–171, 1989.

[26] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[27] D.E. Goldberg, K. Deb, and J Horn. Massive multimodality, deception, and genetic algorithm. In R. Männer and B Manderick, editors, *Parallel problem solving from nature, 2: Proceedings of the 2nd Conference on Parallel Problem Solving from Nature, Brussels, Belgium, 28-30 Sept 1992*, pages 37–46. Amsterdam, The Netherlands: Elsevier Science, 1992.

[28] D.E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In J.J. Grefenstette, editor, *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–9. Lawrence Erlbaum Associates; Hillsdale, NJ, USA, 1987.

[29] G.W. Greenwood, C. Lang, and S. Hurley. Scheduling tasks in real-time systems using evolutionary strategies. In *Proceedings of the Third Workshop on Parallel and Distributed Real-Time Systems*, pages 195–6. IEEE Comput. Soc. Press; Los Alamitos, CA, USA, 1995.

[30] J.J. Grefenstette. Optimization of control parameters for genetic algiorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128, 1986.

[31] A.K. Gupta and G.W. Greenwood. Applications of evolutionary strategies to fine-grained task scheduling. *Parallel Processing Letters*, 6(4):551–61, 1996.

[32] A.K. Gupta and G.W. Greenwood. Static task allocation using (μ, λ) evolutionary strategies. *Information Sciences*, 94(1–4):141–50, 1996.

[33] J.H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, 1975.

[34] R. Horst and P.M. Pardalos. *Handbook of Global Optimization*, page 832. Kluwer Academic Publishers, 1995.

[35] M.F. Hussain and K.S. Al-Sultan. A hybrid genetic algorithm for nonconvex function minimization. *Journal of Global Optimization*, 11:313–324, 1997.

[36] L. Ingber. Simulated annealing: Practice versus theory. *Journal of Mathematical and Computer Modeling*, 18(11):22–59, 1993.

[37] Terry Jones. *Evolutionary algorithms, fitness landscapes and search.* PhD thesis, The University of New Mexico, 1995.

[38] S. Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *J. Statistical Physics*, 34(5–6):976–986, 1986.

[39] S. Kirkpatrick, C.D. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[40] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, Mass : MIT Press, 1992.

[41] K.S. Leung, Terence Wong, and Irwin King. Probabilistic cooperative-competitive hierarchical modeling for global optimization. In *Proc. 6th Intl. Conf. Soft Computing (IIZUKA '98)*, 1998.

[42] K.S. Leung, Terence Wong, and Irwin King. Probabilistic cooperative-competitive hierarchical modeling as a genetic operator in global optimization. In *Proc. 1998 Intl. Conf. Systems, Man, and Cybernetics (SMC'98)*, 1998.

[43] S.W. Mahfoud. Crowding and preselection revisited. In R. Männer and B Manderick, editors, *Parallel problem solving from nature, 2: Proceedings of the 2nd Conference on Parallel Problem Solving from Nature, Brussels, Belgium, 28-30 Sept 1992*, pages 27–36. Amsterdam, The Netherlands: Elsevier Science, 1992.

[44] S.W. Mahfoud. *Niching methods for genetic algorithms.* PhD thesis, University of Illinois at urbana-Champaign, 1995.

[45] B.L. Miller and M.J. Shaw. Genetic algorithms with dynamic niche sharing for multimodal function optimization. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC'96).* IEEE; New York, NY, USA, 1996.

[46] Debasis Mitra, Fabio Romeo, and Alberto Sangivanni-vincentelli. Convergence and finite-time behavior of simulated annealing. *Advances in applied probability*, 18:747–771, 1986.

[47] H. Mühlenbein and D Schlierkamp-Vosen. Predictive models for the breeder genetic algorithm, i. continuous parameter optimization. *Evolutionary Computation*, 1(1):25–49, 1993.

[48] D.W. Page, S.E.; Richardson. Walsh functions, schema variance, and deception. *Complex Systems*, 6(2):125–135, 1992.

[49] A. Pétrowski. A clearing procedure as a niching method for genetic algorithms. In *Proceedings of the Third (1996) IEEE International Conference on Evolutionary Computation (ICEC'96)*, pages 798–803. IEEE; New York, NY, USA, 1996.

[50] M.A. Potter and K.A. De Jong. A cooperative coevolutionary approach to function optimization. In Y. Davidor, H-P Schwefel, and R. Manner, editors, *Parallel Problem Solving from Nature III*. Springer-Verlag; Berlin, Germany, 1994.

[51] W.L Price. *A controlled random search procedure for global optimization*, pages 71–84. North Holland, Amsterdam, 1978.

[52] A.H.G. Rinnooy Kan and G.T. Timmer. Stochastic methods for global optimization. *American Journal of Mathematical and Management Sciences*, 4(1), 1984.

[53] J. Rowe and I. East. Deception and ga-hardness. In Alander J.T., editor, *Proceedings of the First Nordic Workshop on Genetic Algorithms and Their Applications (1NWGA)*, pages 165–172. University of Vaasa; Vaasa, Finland, 1995.

[54] F. Schweitzer, W. Ebeling, H. Rose, and O. Weiss. Network optimization using evolutionary strategies. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN IV*, pages 940–9. Springer-Verlag; Berlin, Germany, 1996.

[55] J. Shekel. Test functions for multimodal search techniques. In *Proceedings of the 5th Princeton Conference on Infomration Science and Systems*, pages 354–359. Princeton University Press, Princeton, 1971.

[56] B.E. Stuckman and E.E. Easom. A comparison of bayesian/sampling global optimization techniques. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(5):1024–1032, 1992.

[57] Z.B. Tang. Adaptive partitioned random search to global optimization. *IEEE Transactions on Automatic Control*, 39(11):2235–2244, 1994.

[58] A. Törn and A. Žilinskas. *Global Optimization*. Lecture Notes in Computer Science. Springer-Varlag Berlin Heidelberg, 1989.

[59] Hans-Michael Voigt. Soft genetic operators in evolutionary algorithms. In W. Banzhaf and F.H. Eeckman, editors, *Evolution and Biocomputation.* Berlin; New York: Springer, 1995.

[60] Hans-Michael Voigt and Anheyer Thomas. Modal mutations in evolutionary algorithms. In *Proceedings of the First (1994) IEEE Conference on Evolutionary Computation (ICEC'94). IEEE World Congress on Computational Intelligence*, pages 88–92. IEEE; New York, NY, USA, 1994.

[61] D. Whitley. Fundamental principles of deception in genetic search. In G. Rawlins, editor, *Foundations of Genetic Algorithms.* San Mateo: Morgan Kaufmann, 1991.

[62] D. Whitley. Deception, dominance and implicit parallelism in genetic search. *Annals of Mathematics and Artificial Intelligence*, 5(1):49–78, 1992.