

國立臺灣大學電機資訊學院電機工程學系

碩士論文

Department of Electrical Engineering
College of Electrical Engineering and Computer Science
National Taiwan University
Master Thesis

分佈估測演算法在基因成對獨立函數上之延展性探討
Scalability of Estimation of Distribution Algorithms
On Allelic Pairwise Independent Functions



陳思誠

Chen, Si-Cheng

指導教授：于天立 博士

Advisor: Yu, Tian-Li

中華民國 98 年 6 月

June, 2009

謝辭

首先要感謝我的指導教授于天立老師，老師在演化計算以及人工智慧領域中的學識淵博，在我研究的過程中無論遇見什麼樣的疑惑，老師總是能指引出明確的方向。並且老師給予我們很多的自由，訓練我們獨立思考與學習的能力，就算我們一路跌跌撞撞的出了許多大小問題，老師也能容忍並且幫助我們解決困難。甚至對於我們在異地求學研究生活中的麻煩，老師也會關心我們並且盡力協助，能夠完成這份論文，我必須誠摯的感謝老師為我們所做的一切。

接著要感謝演化智慧實驗室的成員們，兩位同學林偉楷以及楊皓鈞無論在課業上、研究上、乃至於生活上、甚至娛樂上都是最好的同伴以及戰友，兩位學弟游穎萱與連大鈞為了實驗室中的事務付出許多心力，也讓我們的研究生涯增添許多歡笑與活力。

最後我要感謝我的家人、親人以及所有朋友，感謝母親大人與姐姐的關心與支持、感謝父親以及爺爺在天之靈的守護，希望你們能看得見。感謝我最愛的也最愛我的女友林孜音，這些日子有妳陪伴的幸福，是上天賜與我最好的禮物。

中文摘要

本論文探討分佈估測演算法(Estimation of Distribution Algorithms)在基因成對獨立函數(allelic pairwise independent functions)上之延展性，以及此類函數例如奇偶性函數(parity function)、奇偶與陷阱函數(parity-with-trap function)、沃爾什編碼函數(Walsh-code function)對於鍊結學習(linkage learning)難度之影響。對於分佈估測演算法而言，奇偶性函數被認為是難解的函數，但是在我們的實驗中證實奇偶性函數可被精簡基因演算法(Compact Genetic Algorithms)在多項式時間之內解出，並且困難度更高之奇偶與陷阱函數也被延伸式精簡基因演算法(Extended Compact Genetic Algorithms)解出，縱然鍊結模型依然無法正確被認出。我們也計算出了精簡基因演算法在奇偶性函數上之收斂時間(convergence time)模型，並且此模型與前述之實驗結果吻合。此外，我們也討論了不同分佈估測演算法之性能出現歧異之根本原因。最後，本論文提出了另一個可欺騙大多數分佈估測演算法中鍊結學習機制之基因成對獨立函數，稱之為沃爾什編碼函數，然而此函數仍然能被精簡基因演算法解出。

關鍵字: 基因演算法, 鍊結學習, 分佈估測演算法, 奇偶性函數.

Abstract

This thesis investigates the difficulty of linkage learning, an essential core, in EDAs. Specifically, it examines allelic-pairwise independent functions including the parity, parity-with-trap, and Walsh-code functions. While the parity function was believed to be difficult for EDAs in previous work, our experiments indicate that it can be solved by CGA within a polynomial number of function evaluations to the problem size. Consequently, the apparently difficult parity-with-trap function can be easily solved by ECGA, even though the linkage model is incorrect. A convergence model for CGA on the parity function is also derived to verify and support the empirical findings. Then the root cause of the different performances between different EDAs is also discussed. Finally, this thesis proposes a so-called Walsh-code function, which is more difficult than the parity function. Although the proposed function does deceive the linkage-learning mechanism in most EDAs, EDAs are still able to solve it to some extent.

Keywords: Genetic Algorithms, Linkage Learning, Estimation of Distribution Algorithms, Parity Function.

Contents

謝辭	i
中文摘要	ii
Abstract	iii
Contents	iv
List of Figures	vi
List of Tables	viii
1 Introduction	1
1.1 Motivation and Objective	2
1.2 Road Map	3
2 Background	4
2.1 Simple Genetic Algorithm	4
2.2 Building Block Hypothesis and Deceptive Problems	5
2.3 Framework of EDAs	7
2.4 Compact GA and Extended Compact GA	8
2.4.1 Compact GA	8
2.4.2 Extended Compact GA	10
2.5 Previous Works on Parity Functions	12

3	Performance of CGA on CPF	14
3.1	Experimental Design	14
3.1.1	Bisection Method	15
3.2	Experimental Result and Discussions	16
4	Scalability Model of CGA	17
4.1	CPF with Single Building Block	17
4.2	Scalability with Multiple BBs	21
5	Performance of ECGA on CP/TF	23
5.1	Experimental Design	23
5.2	Experimental Result and Discussions	25
6	Effects of Different Probabilistic Models on ECGA	27
6.1	Experimental Design	27
6.2	Experimental Result and Discussions	28
7	Allelic Pairwise Independent Functions	32
7.1	Walsh Functions and Walsh Codes	33
7.2	Walsh-code functions and Experimental Design	34
7.3	Experimental Result and Discussion	35
8	Conclusions	36
8.1	Main Conclusion and future works	37
	Bibliography	38

List of Figures

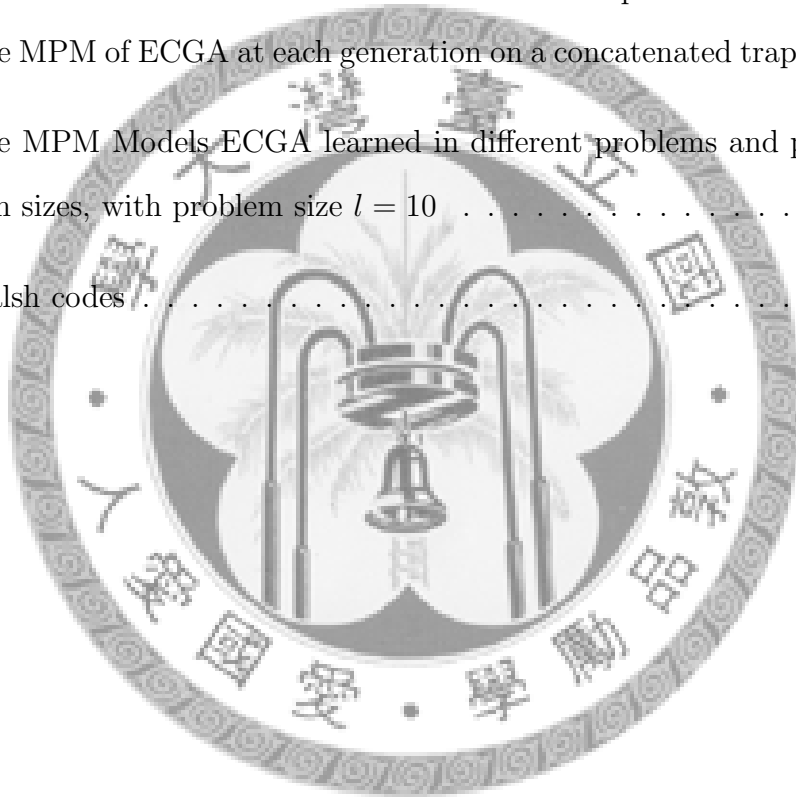
2.1	A 5-bit deceptive trap function.	6
2.2	The average fitness of each two samples of CGA on a one-max problem.	9
2.3	The proportion of BBs in the population of ECGA on trap Functions	11
3.1	Minimal population found by bisection method and the polynomial scalability of CGA on CPF (log-log scale) with a convergence time bound of $\Theta(m \log m)$, the slope of the trend line indicates the order of the scalability.	15
4.1	Verifications of Δ_k models, the ϵ values near zero has better accuracy because the model is a 1st-order estimation	20
5.1	Minimal population found by bisection method and the polynomial scalability of ECGA on CP/TF (log-log scale)	24
5.2	Minimal population found by bisection method and the polynomial scalability of ECGA on CPF (log-log scale)	24
6.1	Minimal population found by bisection method and the polynomial scalability of ECGA with random and fixed spurious-linkage models on CPF (log-log scale)	29
6.2	Minimal population found by bisection method and the exponential scalability of ECGA with RTR and random spurious-linkage models on CPF (log-log scale)	29

6.3	Minimal population found by bisection method and the exponential scalability of ECGA with RTR on CPF (log-log scale)	30
7.1	The example of Walsh functions with the string length $l = 2$	33
7.2	Minimal population found by bisection method and the polynomial scalability of CGA on Concatenated Walsh-code Functions (log-log scale)	34



List of Tables

2.1	The PV of CGA at different time on a one-max problem.	9
2.2	The MPM of ECGA at each generation on a concatenated trap problem.	11
5.1	The MPM Models ECGA learned in different problems and population sizes, with problem size $l = 10$	25
7.1	Walsh codes	33



Chapter 1

Introduction

In the field of genetic and evolutionary computation (GEC), which is inspired by the natural process of evolution, most of the studies focus on the metaheuristic optimization methods. For example, genetic algorithms (GAs) are robust approaches for black-box optimization problems, which do not provide any information about the structure of the problems for the algorithms. Although GAs can solve numerous problems from varied applications, they are still needed to be improved to handle more difficult problems. After many years of research, many variations of GAs have been invented and developed, and estimation of distribution algorithms (EDAs) are one of the most significant branch of GAs.

EDAs can be categorized into different classes based on their linkage-learning mechanisms. The most basic ones are univariate EDAs, *e.g.*, PBIL [2] and Compact GA [16], which do not have any linkage-learning mechanism and assume all the variables in the problem are independent. Bivariate EDAs, like MIMIC [3] and BMDA [24], can detect pairwise dependencies between the variables, but do not model the distribution among multiple variables. Finally, multivariate EDAs, such as ECGA [14], BOA [22], D^5 [25], EBNA [7] and DSMGA [27], which build interacting models to reveal dependencies among multiple variables, are more powerful for solving nearly decomposable problems.

1.1 Motivation and Objective

Linkage learning is an inevitable issue in bivariate and multivariate EDAs, which must exploit the linkage information to estimate joint probabilities and construct the models. The computational cost for determining the exact dependency model scales exponentially with problem size, and hence a proper mechanism is required for building an approximate model. Most of the multivariate EDAs construct the interacting models by starting with pairwise linkages between only two variables, since it is impractical to explore the whole possible search space of multi-variable linkage. It takes $O(l^k)$ to compute all dependencies among k variables in a problem of size l , so it makes more sense to start with only pairwise linkages with cost limited to $O(l^2)$. For example, ECGA [14] builds the marginal product models by merging the subsets of variables according to the minimum description length metric, and it needs to detect the pairwise dependency at the very beginning of model building. BOA [22] also exploits pairwise dependency to decide the addition or deletion of an edge between two variables in the Bayesian networks. However, a quick approach is generally deficient in precision, which may consequently delay the convergence time of linkage. As the linkage should be learned before the alleles converge in EDAs [12], reaching for balance between efficiency and precision is a critical issue in linkage learning.

The objective of this thesis is to realize how the difficulty of linkage learning impacts on the performance of EDAs. We identify those functions which are difficult for linkage learning first, and then analyze the performance of EDAs on these functions by observing their scalability. The influence of model accuracy in multivariate EDAs is investigated as well. Finally, we discuss the possibility of designing a function which is difficult for EDAs to solve by deceiving the linkage-learning mechanism.

1.2 Road Map

In the thesis we first review previous work on the parity function and investigate the performances of different EDAs on that problem. A convergence model is also derived to verify the scalability of EDA on the parity function. Then we investigate the reason of the bad performance of EDAs on the parity functions. We discuss about allelic-pairwise independent functions and experiment on a new problem we propose in the later chapter. Finally, we have some analyses and conclusions on the empirical results.



Chapter 2

Background

This chapter gives some relative knowledges of the algorithms which are used in this thesis. At the beginning of this chapter, a traditional genetic algorithm and its related theories are introduced. Then we introduce the general procedure of EDA, and give the details about EDAs used in this thesis, which are compact genetic algorithm (CGA) and extended compact genetic algorithm (ECGA). After that, we discuss about the parity functions, which are hard for EDAs to learn the linkage within, and review the previous works on it.

2.1 Simple Genetic Algorithm

A genetic algorithm (GA) requires an encoding of solutions, which are called chromosomes, and one or more objective functions which evaluate the chromosomes and decide its fitness. The types of chromosomes include binary strings, an array of real-number parameters, or even complex structures. After the solutions are encoded, initialized and evaluated, different operators are applied iteratively to the population, which is a set of individuals each containing a chromosome.

The traditional version of GA, which is called simple genetic algorithm (SGA), encodes the solutions into binary strings. Although there are a variety of operators, the procedure of SGA consists of the most common operators, which are selection, recombination and mutation. The general procedures of an SGA are:

1. **Initialization.** Initialize a random population of size N .
2. **Evaluation.** Calculate the fitness values of all individuals in the population.
3. **Selection.** Choose the better solutions from the population depending on their fitness values.
4. **Recombination.** Randomly choose two individuals as the parents, and generate new promising solutions by exchanging and perturbing their contents.
5. **Mutation.** Randomly alter some bits in some chromosomes, which is creating more alternative individuals.
6. **Termination.** Repeat from step 2 until the population converges to a optimized final solution or the predefined time constrain is exceeded.

Selection and recombination can be done in numerous ways. For instance, tournament selection choose s individuals to compete, then the winner gets s copies after selection, where s is called tournament size, or selection pressure in GA theory. Recombination is also called crossover, and the most typical recombination is one-point crossover, which randomly picks a point in the chromosomes and exchange the contents beyond this point. Using different operators in SGAs may results in different final outcomes.

2.2 Building Block Hypothesis and Deceptive Problems

In the research of GA theory, it has been discussed how GA can solve the optimization problems robustly. In the building block hypothesis [11], an optimal solution found by GA is considered as a combination of building blocks (BBs), which are minimal sequentially superior sub-solutions to the problem. The term 'minimal sequentially superior' means that a BB has to be short, low order and high average

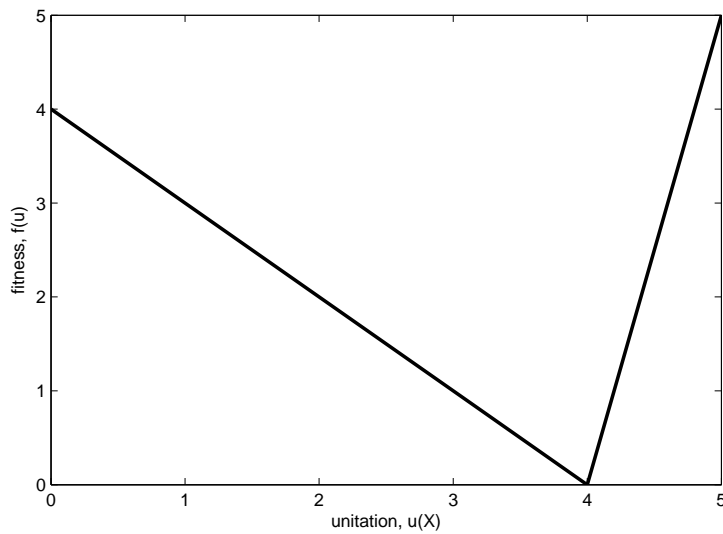


Figure 2.1: A 5-bit deceptive trap function.

fitness to become a part of the global optima. With this concept, Goldberg proposed a decomposition for the problem of designing competent GAs:

1. Know what GAs process – BBs.
2. Know thy BB challengers – BB-wise difficult problems.
3. Ensure an adequate supply of raw BBs.
4. Ensure increased market share for superior BBs.
5. Know BB takeover and convergence times.
6. Make decisions well among competing BBs.
7. Mix BBs well.

Most of the recent studies are focused on how competent GAs mix BBs well. One of these notable issues about BBs is that an effective BB should be as short as possible to avoid disruption by the recombination operators like one-point crossover. Since we want GAs to work efficiently and reliably, the capability of identifying BBs from current population is required, so that we can mix the BBs appropriately as if they are permuted sequentially in the chromosome.

Problems can be designed to deceive GA with its difficulty within a single BB, so GAs can not solve them successfully without identifying the BBs [9]. These deceptive problems mislead GAs to a local optimal when less than k bits are observed, while the k -bit global optima is in the opposite direction. For example, a k -bit trap function [1] can be designed by assigning the highest fitness to the string with all 1s, but other strings has a lower fitness if it has more bits of 1s. Generally, the trap subfunction is defined as

$$trap(X) = \begin{cases} k & \text{if } u(X) = k \\ k-1-u(X) & \text{otherwise} \end{cases} \quad (2.1)$$

Fig. 2.1 shows a 5-bit trap function, the unitation variable in the figure means the number of bit ones in a string X . A competent GA should be able to identify the k -bit BB by learning the linkage between the variables so that the trap functions can be solved. If BB is not identified, it may be disrupted by the recombination operators and the population will converge to the local optima.

2.3 Framework of EDAs

Estimation of Distribution Algorithms (EDAs) [18, 20], or probabilistic model-building genetic algorithms (PMBGAs) [23] are robust and scalable optimization methods which can solve various difficult problems. Instead of using conventional recombination operators as in GAs, EDAs use machine learning techniques like probabilistic model-building and solution sampling to generate candidate solutions, but the concepts of population evolving and selection are still employed in EDAs. Moreover, for nearly decomposable problems, the promising solutions to these problems can be decomposed to BBs. EDAs with linkage-learning can reconstruct the structure of the problem, which can be utilized to prevent the disruption of the BBs, and enhance the mixing of BBs as well [11].

The framework of EDAs is similar to the one of GAs, except the operators which

reproduce new individuals are replaced by building the probabilistic model of the population and sample new individuals from it:

1. **Initialization.** Generate the initial population randomly.
2. **Evaluation.** Calculate the fitness values of all individuals in the population.
3. **Selection.** Select the promising solutions depending on their fitness values.
4. **Probabilistic model-building.** Build the probabilistic model by calculating the distribution of current population.
5. **Sampling.** Use the model built by previous step to sample a new population.
6. **Replacement.** Incorporate the new population into the previous one.
7. **Termination.** Repeat from step 2 until the population converges to a optimized final solution or the predefined time constrain is exceeded.

The ability of EDAs to construct the structure of the problem is based on the assumption of that the linkages among the variables can be learned by estimating the joint distribution from promising individuals. If the problem structure is different to which EDAs built, good BBs in the current population may be disrupted after sampling the model, depending on whether the problem is deceptive or not.

2.4 Compact GA and Extended Compact GA

In this section, two EDAs used in this thesis are introduced, which are compact genetic algorithm (CGA) and extended compact genetic algorithm (ECGA). We describe the frameworks of these EDAs, and examples are also given.

2.4.1 Compact GA

Compact genetic algorithm (CGA) [16] is a simple and efficient univariate EDA. A CGA starts with a probability vector (PV) l probabilities and each of which is

Table 2.1: The PV of CGA at different time on a one-max problem.

Evaluation	$[p_0 \ p_1 \ p_2 \ p_3 \ p_4]$
0	[0.50 0.50 0.50 0.50 0.50]
600	[0.64 0.69 0.68 0.71 0.55]
1600	[0.98 0.93 0.91 0.82 0.88]
3000	[1.00 1.00 1.00 1.00 1.00]

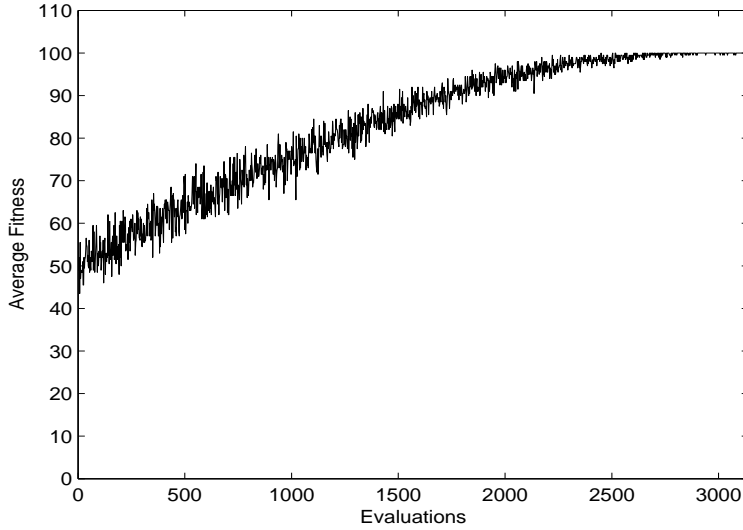


Figure 2.2: The average fitness of each two samples of CGA on a one-max problem. initialized to 0.5. Each probability p_i in the PV represents the probability that the i -th bit of the l -bit string X is 1. Then two individuals are generated for selection by the PV, the one with the higher fitness is the winner, and the other is the loser. After that, each probability in the PV is modified only if the corresponding bits in the winner and loser individuals are different. When the bit from the winner is 1, the probability is increased by $1/N$, on the contrary, it is decreased by the same amount when the bit from the winner is 0. So the PV at time t can be calculated by the following equation:

$$PV[i](t) = PV[i](t - 1) + \frac{1}{N}(winner[i] - loser[i]). \quad (2.2)$$

This procedure is repeated until the PV converges.

Figure 2.2 shows a sample run of CGA on a 100-bit one-max problem, which is

defined as

$$f(X) = \sum_{i=1}^{100} x_i. \quad (2.3)$$

Table 2.1 shows part of the PV in the same experiment. We set the population $N = 100$, so the differences of probabilities are always the multiple of $1/N = 0.01$. CGA is compact since it works as well as SGA without generating a population. Moreover, it is proven that CGA performs approximately the same as a SGA with population N and uniform crossover operator [16].

2.4.2 Extended Compact GA

Extended compact genetic algorithm (ECGA) is a multivariate EDA which is based on the same probabilistic modeling concept of CGA, but also includes a linkage-learning mechanism for adapting the problem structure. There are different ways to learn the linkages among the variables. ECGA builds marginal product models (MPMs) by calculating the joint probability distributions over more than one gene, and a greedy method is employed to find the most adaptive structure. In the beginning of model building, the initial model consists of l subsets, each containing only one variable, and two subsets are merged in each step of greedy search.

To build an ideal MPM, we need to choose a proper metric to decide which two subsets should be merged. Relying on Occam's Razor, which claims the simpler choice is preferred if all other things are equal, the minimum description length (MDL) metric is utilized to measure the MPMs. The description length, or combined complexity of MPM can be evaluated by calculating the sum of model complexity (MC) and compressed population complexity (CPC), which are

$$MC = \log_2(N + 1) \sum_i (2^{S_i} - 1) \quad (2.4)$$

and

$$CPC = N \sum_i Entropy(M_i), \quad (2.5)$$

where N is the population size, S_i is the size of i -th subset and M_i is the i -th subset

Table 2.2: The MPM of ECGA at each generation on a concatenated trap problem.

Generation	Marginal Product Models
0	[0 2 3 4][1][15 16 17 18 19][9][14][5 6 7 8][10 11 12 13]
1	[0 1 2 3 4][5 6 7 8 9][10 11 12 13 14][15 16 17 18 19]
2	[0 1 2 3 4][5 6 7 8 9][10 11 12 13 14][15 16 17 18 19]

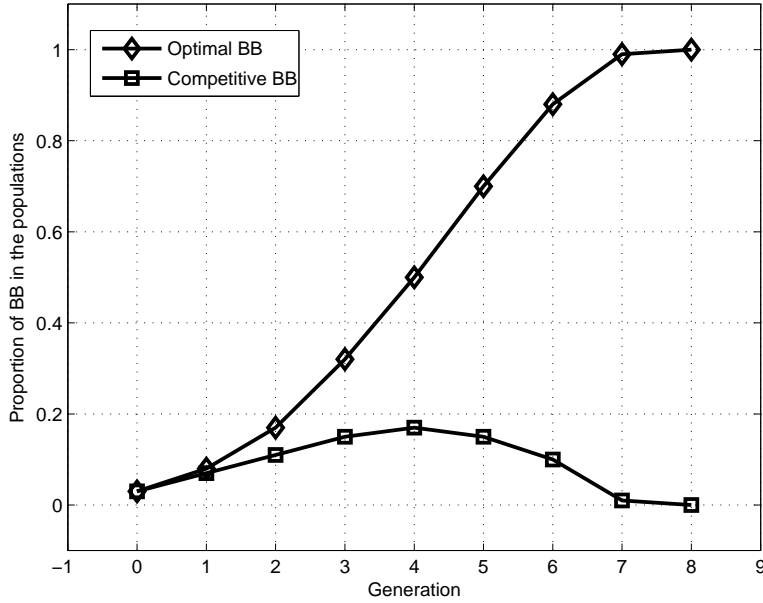


Figure 2.3: The proportion of BBs in the population of ECGA on trap Functions in MPM. The entropy of M_i can be calculated by $\sum -p_j \log_2 p_j$, where p_j is the joint probability of the outcome j to be generated by the i -th subset. The model with minimal combined complexity is chosen in each step of greedy search.

The procedure of ECGA is similar to simple GA, except the recombination operator is replaced by model building and sampling the population:

1. Initialize a random population of size N .
2. Apply tournament selection.
3. Build the MPM model using greedy search.
4. Generate a new population with the model.
5. Repeat from step 2 until the population converges, or time constrain exceeds.

Since the algorithm is able to build models among multiple variables, it can identify BBs and prevent the effective BBs from being destroyed. It has been verified that the problems with trap functions can be solved robustly by ECGA [14]. For instance, a concatenated trap function of $k = 5$ is defined as

$$f(X) = \sum_{i=0}^{m-1} \text{trap}(x_{5i}x_{5i+1}x_{5i+2}x_{5i+3}x_{5i+4}). \quad (2.6)$$

Table 2.2 shows the MPM built by ECGA at each generation, the population size is 1000 and the tournament size is 16. All BBs of the problem are identified at generation 1, so that the promising partial solutions are not disrupted after new populations are sampled and finally the optimal solution is found. Moreover, another sample, in the Fig. 2.3 shows that ECGA can identify the building blocks in the population of ECGA on trap functions with the problem size equal to 50. The population size is 5000 and the tournament size is 8. In this figure, a optimal BB means a BB with all bits are 1, while a competitive BB means a BB with all bits are 0, which is the local optimal solution. At the beginning, both type of BBs occupy 1/32 of the whole population, but finally ECGA can separate the BBs and make the population converge to the global optimal solution.

2.5 Previous Works on Parity Functions

We have introduced the deceptive trap functions which are difficult for SGAs to solve, but most of the multivariate EDAs are capable of tackling such problems successfully. However, there exist functions which are difficult for EDAs to learn the linkage within and build the correct model. For example, the parity function can deceive the linkage-learning mechanism of EDAs because variables in this function are pair-wise independent. The parity function is defined as:

$$\text{parity}(X) = \begin{cases} C_{\text{even}} & \text{if } u(X) \text{ is even} \\ C_{\text{odd}} & \text{otherwise} \end{cases}, \quad (2.7)$$

where $u(X)$ is the unitation or bit count of string X , and C_{even} and C_{odd} are constants. Apparently, the parity function is a generalization of the XOR function.

Coffin and Smith [4] described the concatenated parity function (CPF) and the concatenated parity/trap function (CP/TF) in their work. CPF is defined as a concatenation of parity functions:

$$CPF(X) = \sum_{i=0}^{m-1} \text{parity}(x_{ik} \dots x_{ik+(k-1)}). \quad (2.8)$$

Then CP/TF is defined as

$$CP/TF(X) = \sum_{i=0}^{m-1} \begin{cases} \text{parity}(x_{ik} \dots x_{ik+(k-1)}) & \text{if } i \text{ is even} \\ \text{trap}(x_{ik} \dots x_{ik+(k-1)}) & \text{otherwise,} \end{cases} \quad (2.9)$$

Coffin and Smith investigated the performance of hierarchical Bayesian optimization algorithm (hBOA) [21], the hierarchical version of Bayesian optimization algorithm (BOA), on these problems. BOA is a novel multivariate EDA which solves numerous hard problems reliably by utilizing Bayesian network to build the model of the problem structures. But surprisingly, hBOA scaled exponentially on both problems. It is believed that hBOA is robust and reliable on nearly decomposable problems, so they stated that CPF is hard for EDAs to solve and discussed about the reasons and possible solutions.

Furthermore, Echegoyen et al. [5] investigated the exact model building on Bayesian network based EDAs, and verified that the linkages in CPF can be learned by building an exact model. Emmendorfer and Pozo's work [6] also employed CPF to test their design of EDA.

Chapter 3

Performance of CGA on CPF

Coffin and Smith's work have shown that hBOA, one of the most powerful EDAs, scaled exponentially on the parity functions. To understand the reason that caused such unfavorable performance, we first test CGA on CPF, which is the simplest type of the parity functions. For a multivariate EDA like hBOA, it would degenerate to a univariate EDA if the linkages within the problem can not be learned, so we expect CGA may act like hBOA on CPF at the beginning. But surprisingly, the empirical result shows that CGA can solve CPF reliably in polynomial time.

3.1 Experimental Design

To verify whether a simpler GA or EDA might work on the parity functions, we investigate the scalability of CGA, which is a univariate EDA without any linkage learning, on CPF. Similar to the setup in [4], we choose the block size $k=5$, $C_{odd} = 5$ and $C_{even} = 0$ for CPF. Since we want to know if a CGA needs a polynomial time to solve parity functions or not. The number of fitness evaluations has to be counted in this experiment. To find the minimal number of fitness evaluations, the population size should be determined first.

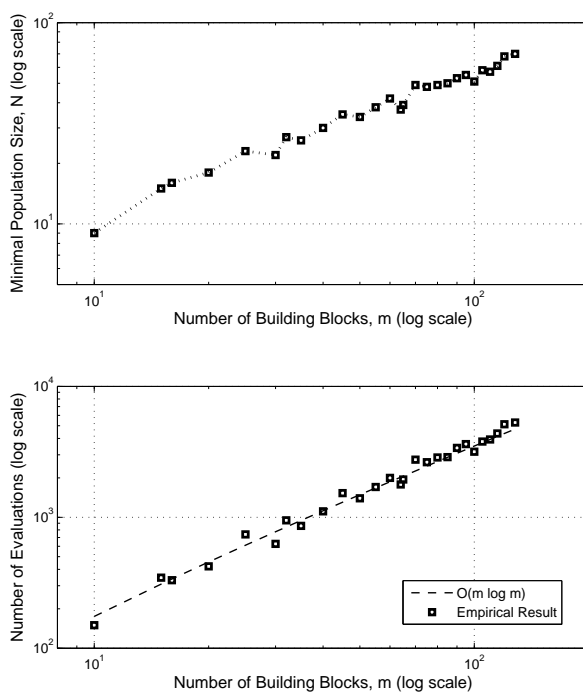


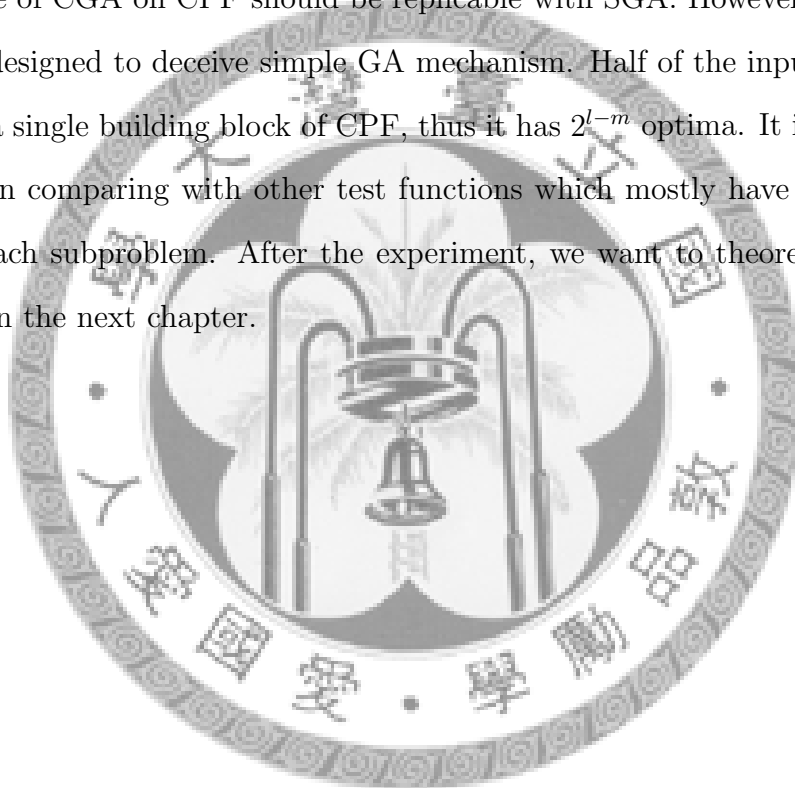
Figure 3.1: Minimal population found by bisection method and the polynomial scalability of CGA on CPF (log-log scale) with a convergence time bound of $\Theta(m \log m)$, the slope of the trend line indicates the order of the scalability.

3.1.1 Bisection Method

A bisection method is exploited here to determine the minimal population size required for different m value, where m denotes the number of BBs in CPF. There are two phases in the bisection method, the first phase is to determine the upperbound by doubling the population size until the algorithm solves the problem robustly. Here we define that the problem is solved robustly if the algorithm always find out the optimal solution in at least ten trials. Then the second phase is the bisection phase, which uses the upperbound found in previous phase and half of the upperbound as the lowerbound. Each step the middle point of the upperbound and lowerbound is chosen as the population size for a test, and the middle point becomes the upperbound of the next step if the test is passed, otherwise it becomes the lowerbound. The second phase repeats until it converges to a final result, which is employed to generate the average number of evaluations from 100 runs for each problem size.

3.2 Experimental Result and Discussions

As the result shown in Fig.3.1, CGA solves CPF reliably with obviously bounded polynomial times, which is about $\Theta(m^{1.32})$ and differs from previously assumption that CPF should be hard for EDAs. From this result, we can make sure that the parity functions can be solved by a simple univariate EDA like CGA, without building the correct model of the problem. Since the approximate equivalence between CGA and a simple GA (SGA) with uniform crossover has been verified in [16], the performance of CGA on CPF should be replicable with SGA. However, CPF is not a problem designed to deceive simple GA mechanism. Half of the input strings are optimal in a single building block of CPF, thus it has 2^{l-m} optima. It is a relatively large portion comparing with other test functions which mostly have only one optimum in each subproblem. After the experiment, we want to theoretically verify this result in the next chapter.



Chapter 4

Scalability Model of CGA

The result from the previous chapter revealed that CGA successfully solves CPF. To theoretically understand that why CGA performs differently from hBOA, we attempt to derive a model to theoretically prove the polynomial scalability of CGA on CPF. First we consider the parity function with only one BB, then we extend the result to the case of multiple BBs in the later section.

4.1 CPF with Single Building Block

Before the derivation, we extend the definition of CPF to simplify the calculation.

Let

$$C_{odd} = \begin{cases} 0 \\ k \end{cases} \quad C_{even} = \begin{cases} k & \text{if } k \text{ is even} \\ 0 & \text{otherwise,} \end{cases} \quad (4.1)$$

i.e., the individual X_k where $u(X_k) = k$ is always an optimum, k is the size of each subproblem.

In CGA, probability vector (PV) is adopted to generate two individuals X_a, X_b for every generation. When the PV is expected to be converged to an individual X_k , there must be an initial bias ε_0 , which may be positive or negative. Hence the initial PV become

$$\{p_i = \frac{1}{2} + \varepsilon_0 | i = 1 \dots k\} \quad (4.2)$$

Here we define $P_{mn} = p(u(X_a) = m, u(X_b) = n)$, so

$$P_{mn}(t=0) = \binom{k}{m} \binom{k}{n} \left(\frac{1}{2} + \varepsilon_0\right)^{m+n} \left(\frac{1}{2} - \varepsilon_0\right)^{2k-m-n} \quad (4.3)$$

Then we can define a_{mn} as the expected number of times that a single p_i in the PV is modified by $1/N$ when $u(X_a) = m$ and $u(X_b) = n$, where N is equivalent to the population size in SGA [16]. The PV will be updated only when X_a and X_b has different parities, so $a_{ij} = 0$ when i and j has the same parity. Note that $a_{mn} = a_{nm}$ since exchanging the winner and loser won't alter the result, therefore we can estimate the PV at generation t by the recursive equation

$$\begin{aligned} p_i(t) &= \frac{1}{2} + \varepsilon_t \\ &= \frac{1}{2} + \varepsilon_{t-1} + \frac{1}{N} \cdot 2 \sum_{\substack{m \in \text{even} \\ n \in \text{odd}}} a_{mn} P_{mn}(t-1). \end{aligned} \quad (4.4)$$

Although k may be greater than two in CPF, here we set $k = 2$ only for derivation. For $k = 2$, if $(X_a = \mathbf{11})$ and $(X_b = \mathbf{10}$ or $\mathbf{01})$ are generated, each p_i is expected to be updated by $1/2$ times, so we can get $a_{21} = 1/2$. Similarly when $(X_a = \mathbf{00})$ and $(X_b = \mathbf{10}$ or $\mathbf{01})$ are generated, we can calculate $a_{01} = -1/2$, the minus here means decreasing the PV. Using the equations above, we have

$$\begin{aligned} \varepsilon_{t+1} &= \varepsilon_t + \frac{2}{N} \left[\frac{1}{2} \left(2 \left(\frac{1}{2} + \varepsilon_t \right)^3 \left(\frac{1}{2} - \varepsilon_t \right) \right) - \frac{1}{2} \left(2 \left(\frac{1}{2} + \varepsilon_t \right) \left(\frac{1}{2} - \varepsilon_t \right)^3 \right) \right] \\ &\simeq \varepsilon_t + \frac{2}{N} \left[\frac{1}{4} \left(\frac{1}{4} + \varepsilon_t \right) - \frac{1}{4} \left(\frac{1}{4} - \varepsilon_t \right) \right] \\ &= \varepsilon_t + \frac{1}{N} \varepsilon_t. \end{aligned} \quad (4.5)$$

which is a 1st-order estimation. This recursive relationship can be extended to k -bit BB. Let $\Delta_k = \varepsilon_{t+1} - \varepsilon_t$, a k -bit BB can be normalized to that every generation CGA always generates two individuals, the winner is X_w and X_l is the loser, and every time each p_i is updated by Δ_k . Next, consider the $(k+1)$ -bit condition, each generation CGA generates two individuals, $\{X_w1$ or $X_w0\}$ and $\{X_l1$ or $X_l0\}$. Again we only consider the conditions when PV is modified, where the generated

individuals are $\{X_w1 \text{ and } X_l1\}$ and $\{X_w0 \text{ and } X_l0\}$. Note that concatenating a bit-1 does not effect the result of competition, while concatenating a bit-0 alters the result. Let $p_{k+1} = (1/2) + \varepsilon_t$, so the expected value of Δ_{k+1} can be calculated by

$$\begin{aligned}\Delta_{k+1} &= \left(\frac{1}{2} + \varepsilon_t\right)^2 \Delta_k - \left(\frac{1}{2} - \varepsilon_t\right)^2 \Delta_k \\ &= 2\varepsilon_t \Delta_k.\end{aligned}\tag{4.6}$$

From Eq.(4.5), we have $\Delta_2 \simeq (1/N)\varepsilon_t$, so

$$\Delta_k \simeq \frac{1}{N} 2^{k-2} \varepsilon_t^{k-1}.\tag{4.7}$$

Note that if $\varepsilon_0 < 0$, Δ_k is also less than zero only when k is even. It makes PV converge to all zeros, which is also an optimum with an even k . Otherwise, Δ_k is greater than zero when k is odd, although $\varepsilon_0 < 0$, so that PV will not decrease and always converge to all ones. Fig. 4.1 shows the verifications of the models of Δ_k with different k and ε values. Writing Eq.(4.7) in the differential form of ε_t , there is

$$\frac{\varepsilon_{t+1} - \varepsilon_t}{(t+1) - t} \simeq \frac{1}{N} 2^{k-2} \varepsilon_t^{k-1},\tag{4.8}$$

$$\frac{d\varepsilon_t}{dt} \simeq \frac{1}{N} 2^{k-2} \varepsilon_t^{k-1},\tag{4.9}$$

which becomes an integral equation

$$\int \frac{d\varepsilon_t}{\varepsilon_t^{k-1}} \simeq \frac{1}{N} \int 2^{k-2} dt.\tag{4.10}$$

Let $k > 3$, after solving the equation, there is

$$\frac{1}{2-k} \varepsilon_t^{2-k} \simeq \frac{1}{N} 2^{k-2} t + C_0.\tag{4.11}$$

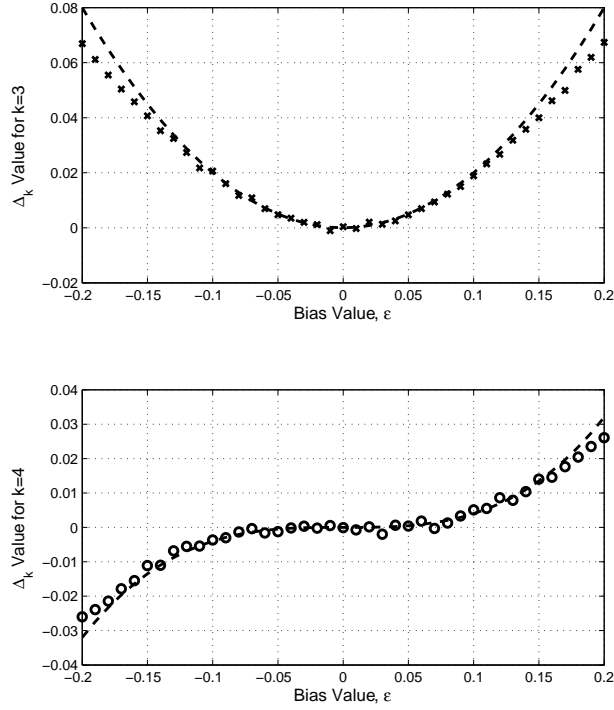


Figure 4.1: Verifications of Δ_k models, the ε values near zero has better accuracy because the model is a 1st-order estimation

With initial condition $\varepsilon = \varepsilon_0$ at $t = 0$,

$$C_0 = \frac{\varepsilon_0^{2-k}}{2-k}. \quad (4.12)$$

For estimating convergence time, $\varepsilon = 1/2$ when the PV converges. Finally we have

$$t \simeq \frac{N}{k-2} \left(\frac{1}{2\varepsilon_0^{k-2}} - 1 \right), \quad (4.13)$$

which is the time-to-convergence model for CPF of single BB. Note that the equation is only valid for $k > 3$, and the condition of $k = 2$ is not concerned because two variables in CPF with $k = 2$ are not independent.

4.2 Scalability with Multiple BBs

By the central limit theorem, the distribution of the fitness of a BB is approximately normally distributed. Using the decision-making argument [13], the probability of making the right decision between two competing individuals in a single trial of a problem with m independent equally-weighted BBs is:

$$p = \Phi\left(\frac{d}{\sqrt{2m'\sigma_{bb}}}\right), \quad (4.14)$$

where $m' = m - 1$, d is the difference of fitness between two competing BBs, σ_{bb}^2 is the variance of BBs, and Φ denotes the cumulative distribution function of a unit normal distribution. With the definition of CPF in this chapter, we have $d = k$ and

$$\sigma_{bb} = \sqrt{\frac{k^2}{2} - \left(\frac{k}{2}\right)^2} = \frac{k}{2}. \quad (4.15)$$

So the probability becomes

$$p = \Phi\left(\sqrt{\frac{2}{m'}}\right). \quad (4.16)$$

Being expressed in terms of the error function,

$$p = \frac{1}{2}\left[1 + \operatorname{erf}\left(\sqrt{\frac{1}{m'}}\right)\right]. \quad (4.17)$$

Using the first-order estimation of the error function with its Taylor expansion, which is accurate enough for $m > 10$, we have

$$p = \frac{1}{2} + \sqrt{\frac{1}{\pi m'}}. \quad (4.18)$$

We can incorporate this probability to estimate the scalability of CGA with multiple BBs. Since we might make wrong decision and lead the PV to the opposite side, the modified Δ_k value should be

$$\Delta'_k = p\Delta_k - (1 - p)\Delta_k. \quad (4.19)$$

With Eq.(4.7) and (4.18), it becomes that

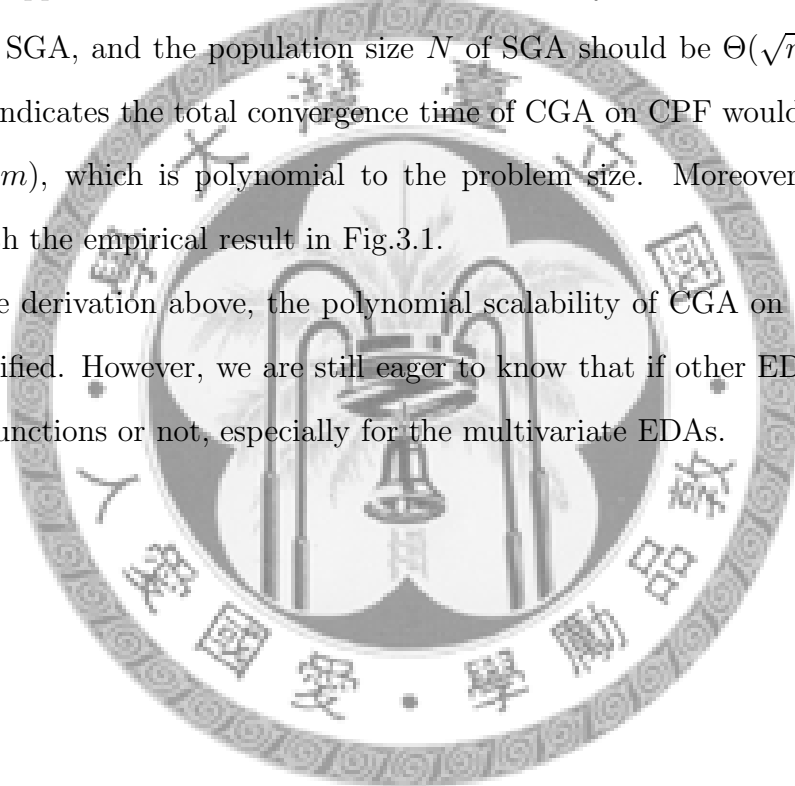
$$\Delta'_k \simeq \frac{2}{\sqrt{\pi m'}} \frac{1}{N} 2^{k-2} \varepsilon_t^{k-1}. \quad (4.20)$$

With the same integration procedure previously, the final convergence model become

$$t \simeq \frac{N\sqrt{\pi m'}}{2(k-2)} \left(\frac{1}{2\varepsilon_0^{k-2}} - 1 \right), \quad (4.21)$$

which is an approximated bound of the time scalability of CGA. Since CGA acts similarly to SGA, and the population size N of SGA should be $\Theta(\sqrt{m} \log m)$ [15], this model indicates the total convergence time of CGA on CPF would be bounded by $\Theta(m \log m)$, which is polynomial to the problem size. Moreover, this model matches with the empirical result in Fig.3.1.

With the derivation above, the polynomial scalability of CGA on CPF is theoretically verified. However, we are still eager to know that if other EDAs can solve the parity functions or not, especially for the multivariate EDAs.



Chapter 5

Performance of ECGA on CP/TF

Previously we have confirmed that CPF is solved by CGA in a polynomial time in both experimental and theoretical ways. But for a multivariate EDA, it would behave like a univariate EDA when the linkages in the BBs can not be learned. In this chapter, we choose ECGA for the experiment on two problems with parity functions, which are CPF and CP/TF. Finally the empirical results shows that ECGA can solve these problems reliably, which is different to hBOA.

5.1 Experimental Design

Two problems are involved in the experiments, CPF and CP/TF. First, to make sure that ECGA degenerates to CGA when the linkage model indicates that all genes are mutually independent, we try to solve CPF with ECGA. Then we experiment with ECGA on CP/TF to investigate the capability of EDAs on the problems which are designed to deceive linkage learning and hill climbing mechanisms simultaneously. Since the parity function is an obstacle to linkage learning, which is essential for solving trap functions, CP/TF is designed to deceive most of EDAs. Without any linkage-learning procedure, CGA is incapable of solving concatenated deceptive trap functions in CP/TF, On the contrary, ECGA is a multivariate EDA which can solve deceptive subproblems consistently [14], so we expect that ECGA may solve CP/TF.

The experimental design is similar to the one on CGA, but to solve the trap func-

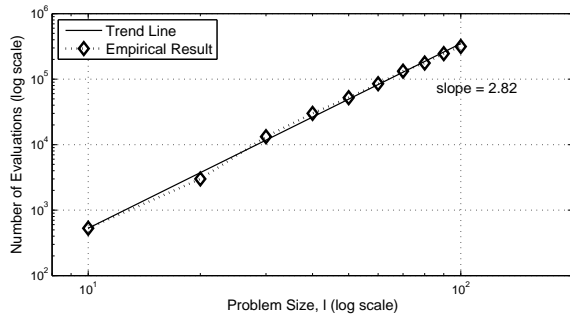
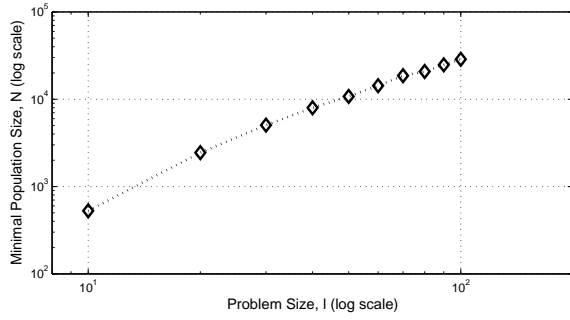


Figure 5.1: Minimal population found by bisection method and the polynomial scalability of ECGA on CP/TF (log-log scale)

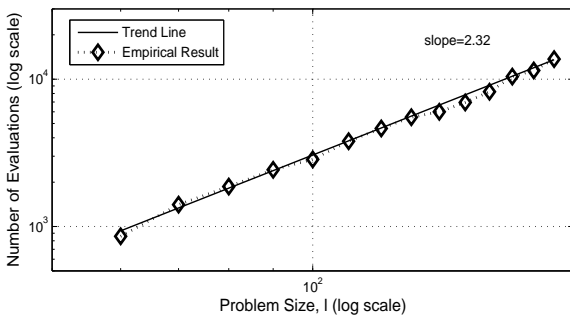
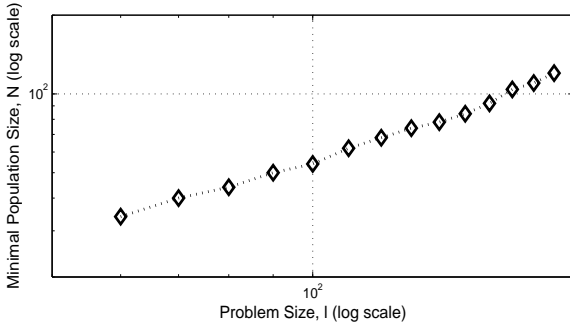


Figure 5.2: Minimal population found by bisection method and the polynomial scalability of ECGA on CPF (log-log scale)

Table 5.1: The MPM Models ECGA learned in different problems and population sizes, with problem size $l = 10$

		MPM Model
CPF	$N = 10$	[02-1-3-4-58-6-7-9]
	$N = 5000$	[0-1-2-3-4-5-6-7-8-9]
CP/TF	$N = 10$	[0-12-34-5-68-7-9]
	$N = 5000$	[0-1-2-3-4-56789]

tions effectively, a tournament size (s) of 8 is chosen when the problem is CP/TF. The block size k is still 5, and the bisection method is still employed to decide the population size.

5.2 Experimental Result and Discussions

As shown in Fig. 5.1, the number of evaluations that ECGA requires to solve CP/TF scales polynomially with the problem size. Besides, in Fig. 5.2 it can also be verified that ECGA can solve CPF in a comparable time as CGA does. So ECGA solve problems with parity functions reliably, although we thought these problems should be hard for EDAs.

How does ECGA solve CP/TF in polynomial time while hBOA does not? To answer the question, we first investigate their linkage models, table 5.1 shows the model built by ECGA only includes building blocks of trap functions when the population size is large enough, so ECGA is not able to learn the linkages in parity subproblems. But ECGA still can solve them with similar mechanism as CGA does, and learn the linkages in the trap subproblems at the same time. Since ECGA could solve these subproblems separately, no matter which subproblem converges slower, it always requires only polynomial time. However, it is evident that EDAs are unable to identify building blocks of parity functions correctly, so the linkages learned by hBOA are considered spurious possibly due to the selection bias. It has been confirmed that using tournament selection in BOA results in less accurate models than truncation selection [19]. Such spurious linkages may cause an overfitting model, which is different to the original problem structure. Furthermore, the overfitting

phenomenon observed in EDAs was correlated with the performance of EDAs [26], and might also caused the exponential scalability of hBOA on CPF. To figure out the reason that hBOA scales exponentially, we have to investigate the effect of different models on the performance of EDAs and find out the critical differences between ECGA and hBOA.



Chapter 6

Effects of Different Probabilistic Models on ECGA

After the performances of ECGA and hBOA are empirically verified to be totally distinct, we long for an explanation to this uncommon result. At first, we assume that a totally different model with spurious linkages leads to such a bad performance. To verify this assumption, we have to find out the effects on ECGA with different probabilistic models. Thus we create a random model with compulsorily added spurious linkages instead of original linkage-learning mechanisms in each generation of ECGA, and observe the influences. Moreover, one of the major difference between ECGA and hBOA is that hBOA employs restricted tournament replacement (RTR) [17] as its replacement scheme, so we also need to observe the models which ECGA builds when RTR is applied, and investigate the performance of ECGA with RTR on the parity functions.

6.1 Experimental Design

To make sure which reason causes the exponential scalability of hBOA, we test ECGA with three different settings, which are ECGA with spurious linkage models, with RTR, and with both conditions.

The procedure that we used to create a model with spurious linkages is random-

ized. Starting with l independent clusters each containing only one gene, we merge two randomly chosen clusters each step and repeat $l/2$ times for each model, , and the size of each cluster is limited to $k \leq 3$. Such a model can be applied in two ways, generating a new random model in each generation or using a fixed model which is generated at the beginning. Note that ECGA does not learn more linkages if this modification is applied.

RTR [17] is a replacement method which partially preserves local optima in the population. With RTR, after each new individual is generated, W individuals are chosen from the old population, where W is called the window size. Then the one which is the most similar to the new one is chosen from these individuals, and it is replaced if the new one has a higher fitness. RTR is utilized in hBOA, to preserve alternative candidate solutions and solve hierarchical problems [21].

The parameter settings are the same as previous experiments, the tournament size is still 2 since CPF does not contain trap functions, and the window size of RTR is equal to the population size n . We also apply bisection method to find the minimal populations in these experiment.

6.2 Experimental Result and Discussions

We experiment with different modified ECGAs on CPF. First, as the result shown in Fig. 6.1, ECGA scales polynomially on CPF with spurious-linkage models, no matter the models are fixed or random generated in each generation. It still can not explain the bad performance of hBOA on problems with parity functions. But a fixed model with spurious linkages obviously has a bad influence on the performance of ECGA, since the order of the number of evaluations is higher. Then we observe the results in Fig. 6.2 , we can notice that ECGA scales polynomially with both RTR and spurious-linkage models, but the number of evaluations is much greater than ECGA without RTR while the problem sizes are the same. Finally, Fig.6.3 shows that ECGA scales exponentially with RTR and the original model-building mechanism of ECGA. The numbers next to the segments in this figure indicate

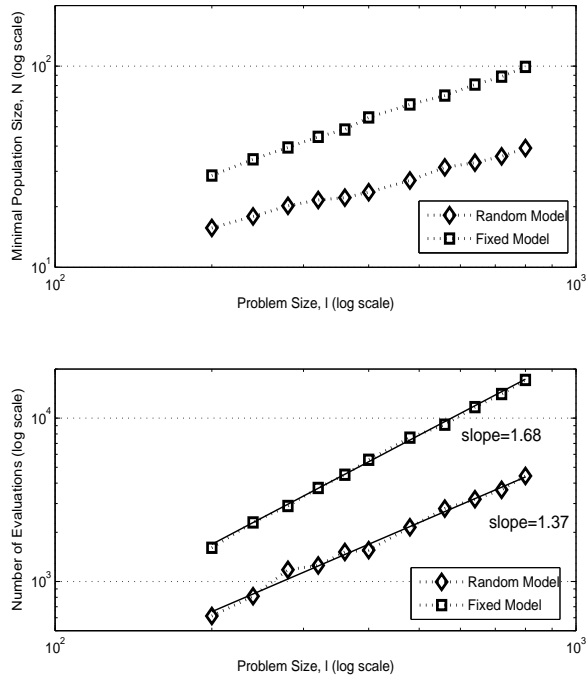


Figure 6.1: Minimal population found by bisection method and the polynomial scalability of ECGA with random and fixed spurious-linkage models on CPF (log-log scale)

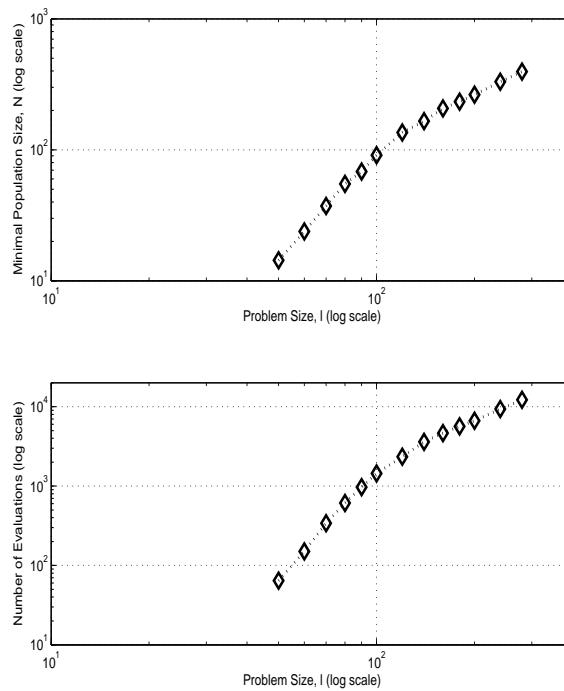


Figure 6.2: Minimal population found by bisection method and the exponential scalability of ECGA with RTR and random spurious-linkage models on CPF (log-log scale)

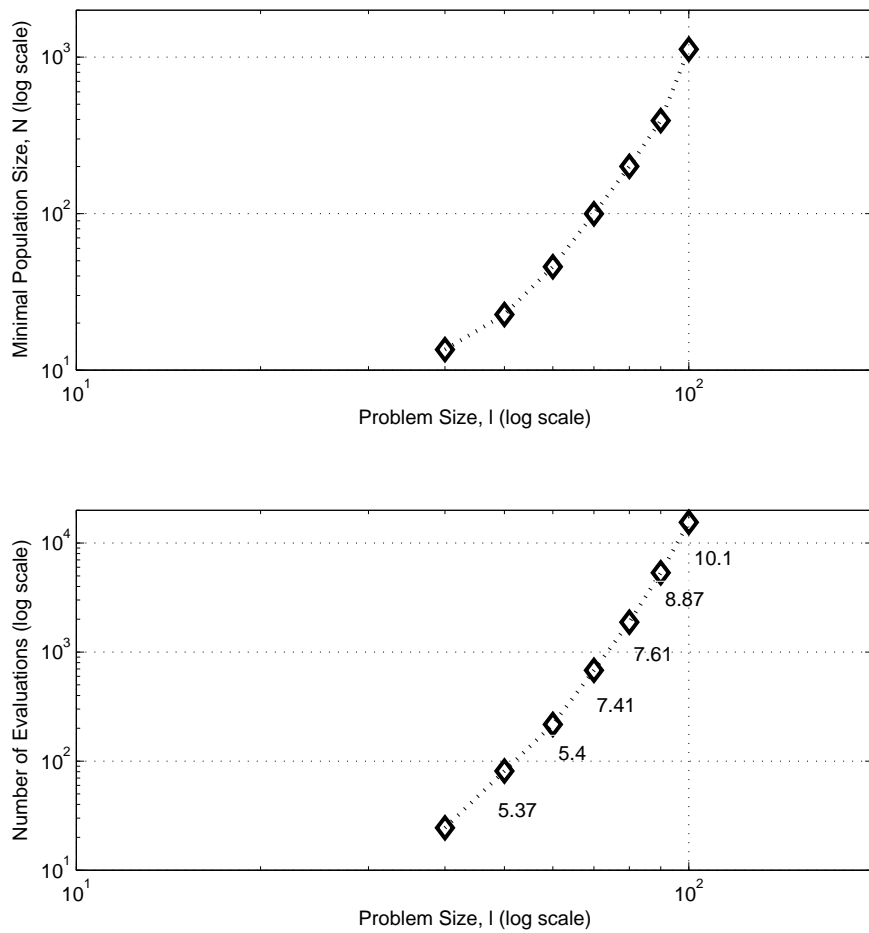
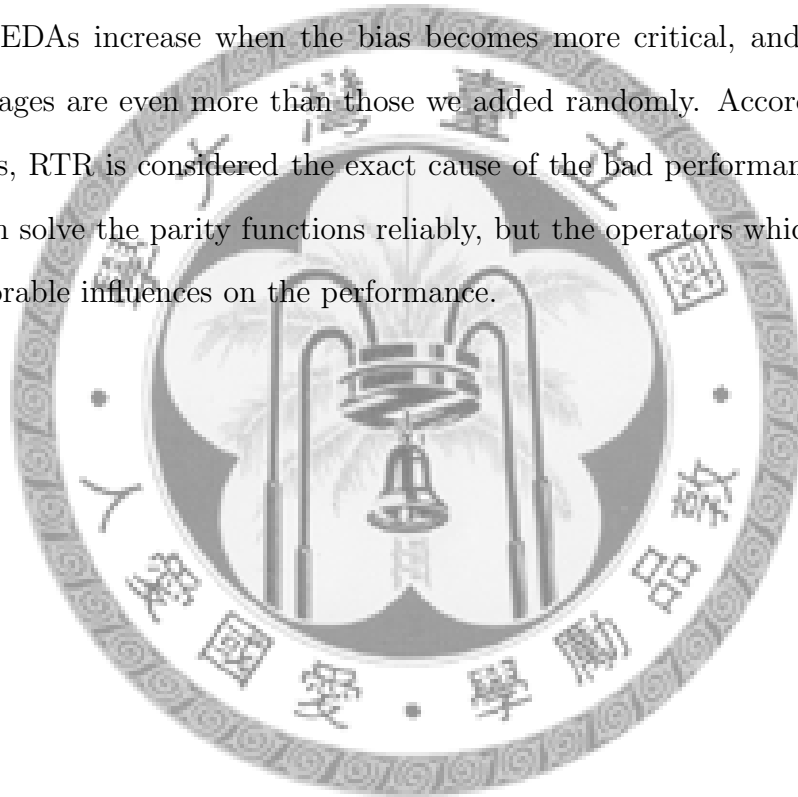


Figure 6.3: Minimal population found by bisection method and the exponential scalability of ECGA with RTR on CPF (log-log scale)


the slope of each segment, which increases with the problem size and shows the exponential scalability. Since ECGA also has a bad performance with RTR on CPF, RTR may be the main reason that hBOA scales exponentially.

Why does RTR make both ECGA and hBOA perform badly? With RTR, the old individuals is only replaced when the new one is better, so it takes much more evaluations for the population to converge. Furthermore, since RTR preserves local optima in the population, the models which EDAs built are effected by the individuals which contain these local optima. Then the spurious linkages which are learned by EDAs increase when the bias becomes more critical, and the number of such linkages are even more than those we added randomly. According to these observations, RTR is considered the exact cause of the bad performance of hBOA. So EDA can solve the parity functions reliably, but the operators which it use may have unfavorable influences on the performance.



Chapter 7

Allelic Pairwise Independent Functions



The discussion here can be extended to a more general and critical issue: Is it possible to design a problem that deceives linkage learning and consequently allelic convergence as well? If such a problem exists, EDAs would be unable to solve it. Because we have confirmed that the parity functions is solvable by EDAs, we want to find more functions which are considered hard for EDAs to learn the linkages within. This chapter focuses on one type of such functions, which are called allelic-pairwise independent functions. These functions appear to be independent when observing only two variables, but have strong dependencies among multiple variables. In these functions, the joint distribution and the product of marginal distribution between any two variables are identical. To satisfy this requirement, we need to assign the same fitness to a group of individuals which are also allelic pairwise independent to make each of them occupies the same portion of the population. The parity function is one of these functions, since the group of all the strings with the same parity is also pairwise independent. However, we propose another type of functions which has less global optimal than parity function, thus we can expect that they are more difficult than the parity functions.

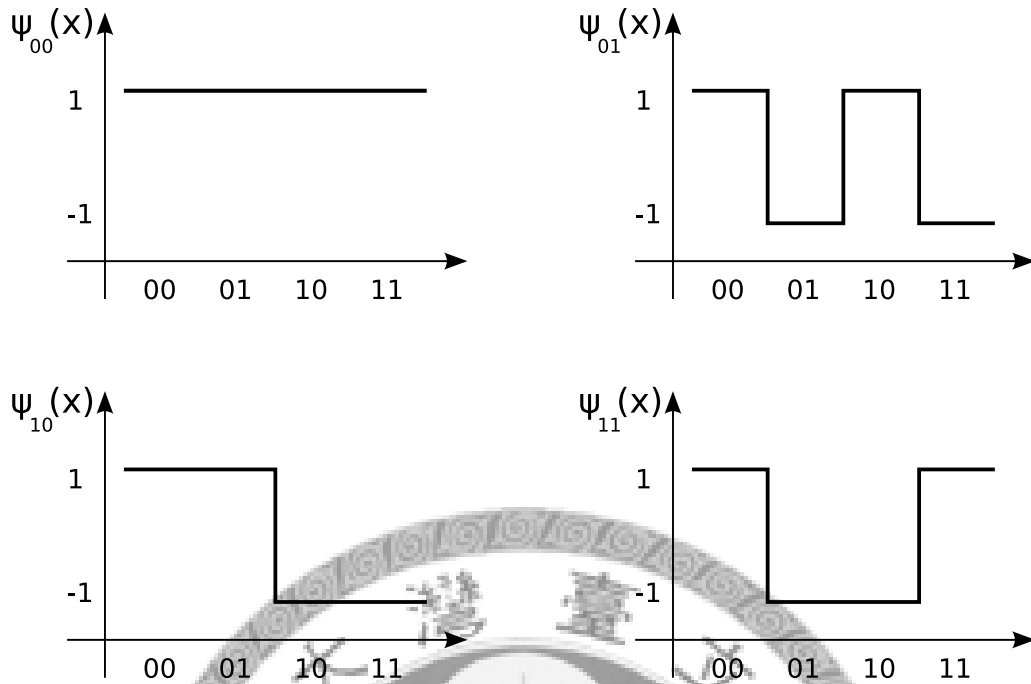


Figure 7.1: The example of Walsh functions with the string length $l = 2$

Table 7.1: Walsh codes

j	$l = 2$	$l = 3$
0	1111	11111111
1	1010	10101010
2	1100	11001100
3	1001	10011001
4		11110000
5		10100101
6		11000011
7		10010110

7.1 Walsh Functions and Walsh Codes

To design such a problem, we utilize Walsh functions [8, 10], which are defined as

$$\psi_j(x) = \begin{cases} 1 & \text{if } u(x \wedge j) \text{ is even} \\ -1 & \text{otherwise,} \end{cases} \quad (7.1)$$

where x and j are both strings of length l , and ‘ \wedge ’ denotes bit-wise logical AND operation. Figure 7.1 is the example of Walsh functions with $l = 2$. Then we define

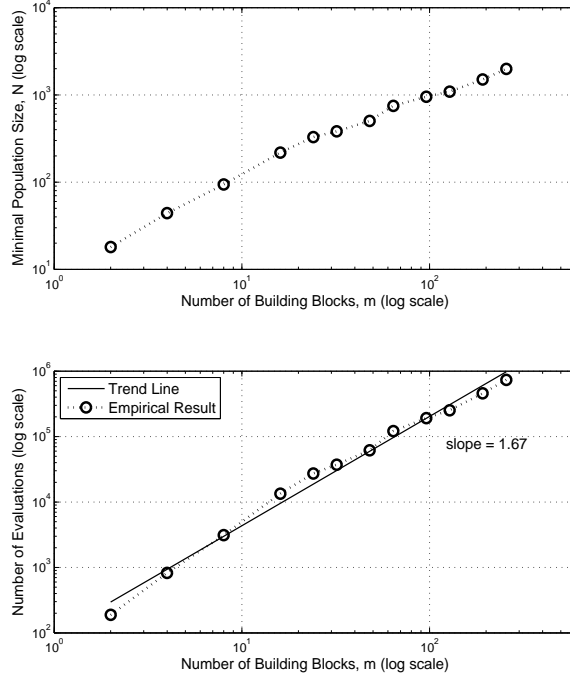


Figure 7.2: Minimal population found by bisection method and the polynomial scalability of CGA on Concatenated Walsh-code Functions (log-log scale)

Walsh codes W_j of length 2^l as

$$\text{the } i\text{-th bit of } W_j = \frac{1}{2}(\psi_j(i-1) + 1) \quad (7.2)$$

which converts analog -1 in Walsh functions to digital bit-0. The definition here is similar to the digital Walsh codes which are also used in applications of wireless telecommunication. Table 7.1 shows the Walsh codes of $l = 2$ and 3. Note that the Walsh codes are allelic-pairwise independent except the first bit, and so are their complements.

7.2 Walsh-code functions and Experimental Design

Thus for block size $3 < k < 2^l$, we can randomly choose k bit from each W_j to form a group of allelic-pairwise independent strings, which includes $2^{(l-k)}$ proportion to the

search space of all 2^k strings. Therefore, we can design a function which is believed more difficult for EDAs than the parity function, for example, the generalized Walsh-code functions is defined as:

$$w(X) = \begin{cases} C_1 & \text{if } X \in W_j \\ C_2 & \text{if } X \in W'_j \\ C_3 & \text{otherwise,} \end{cases} \quad (7.3)$$

where W'_j are one's complements of W_j , which are designed to be the most competing building blocks in this problem.

Since EDAs with linkage learning should be unable to learn the linkages between variables in the Walsh-code function, the remaining question is whether it can be solved by univariate EDAs or not. Thus we experiment with CGA on the concatenated Walsh-code function (CWCF), where $k = 5$, $C_1 = 5$, $C_2 = 4$ and $C_3 = 0$.

7.3 Experimental Result and Discussion

As the empirical result shown in Fig. 7.2, although the order of evaluations on CWCF is slightly greater than CPF, which indicates that CWCF is more difficult than CPF, it is still solved reliably by CGA. But with a larger subproblem size k , CWCF becomes more difficult since the number of optima is always 2^l when $k < 2^l$ while CPF has 2^{k-1} optima, which are proportional to 2^k . However, we still can not make CWCF unable to solve by EDAs at this moment.

Chapter 8

Conclusions

In this thesis, we discussed the difficulty of linkage learning in EDAs, which mostly starts from learning pairwise dependency. So we investigated the performance of EDAs on problems with the allelic-pairwise independent functions, which has been believed to be hard for EDAs.

Our empirical results showed that parity function can be solved in polynomial time by CGA, which is $\Theta(m \log m)$ according to our derivation. We also tested ECGA on the parity functions, and ECGA solved them in polynomial time, but hBOA scaled exponential on it in the previous works by Coffin and Smith [4]. In our discussion, it was believed that hBOA did not perform well due to the spurious linkages [19] it had learned with RTR [17], which has been the default replacing method of hBOA, and such linkages caused the overfitting issue [26]. To verify the exact reason of the different performance between hBOA and ECGA, we also investigated the performance of ECGA with random spurious linkages and RTR, and only ECGAs with RTR scaled exponentially on CPF like hBOA did.

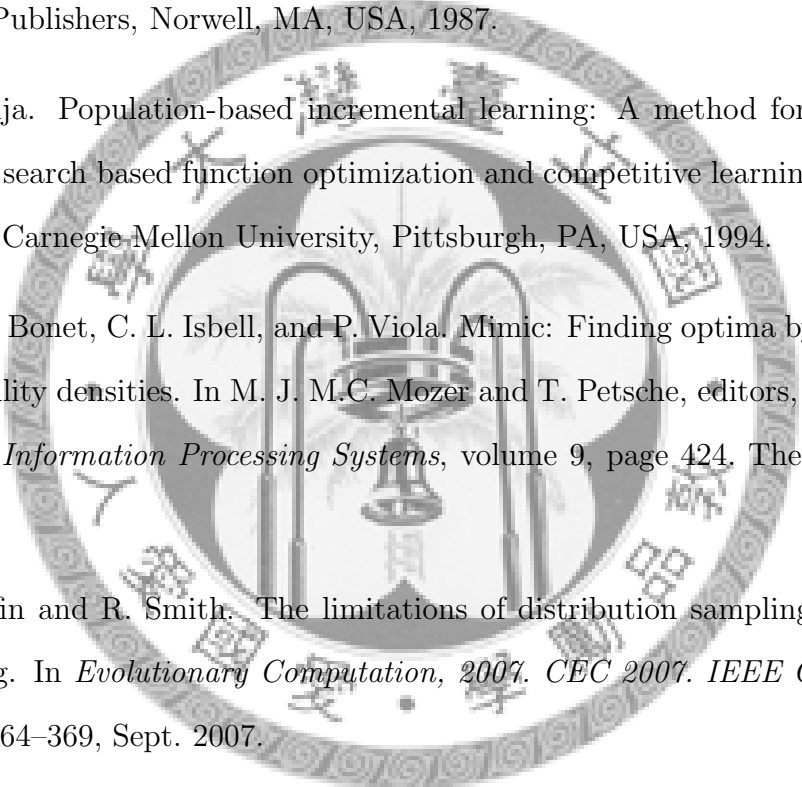
Moreover, we proposed the Walsh-code function, which was another function which could deceive the linkage-learning mechanisms in EDAs successfully. With such a function, the concatenated Walsh-code function is a more difficult problem for EDAs than CPF, even though not significantly.

8.1 Main Conclusion and future works

Although EDAs are unable to build exact models for the allelic-pairwise independent functions, they still can solve these problems without learning any correct linkages according to our experiments and derivational models of convergence time, which is a polynomial time model of $\Theta(m \log m)$. But for multivariate EDAs, they may perform worse than univariate EDAs if too many spurious linkages are learned or added, so linkage learning may become unwanted when it is handling allelic-pairwise independent functions. However, since linkage learning is one of the most important parts of multivariate EDAs, we can also alter some operators to prevent the pre-convergence in the population for solving these problems. Besides, a k -bit-wise model-building method can also solve these problems reliably with a limited order k , since the problems with higher order are too difficult to be considerable.

Because multivariate EDAs are robust and effective methods to solve most of nearly decomposable problems reliably, it is desirable to discover another problem which can not be solved reliably by multivariate EDAs. This problem should be hard for EDAs just because multivariate EDAs can not construct the exact linkage model of this problem correctly before all the alleles converge, and subsequently end up with the local optima. If such a problem actually exists, multivariate EDAs have to search with $O(l^k)$ time to build an exact distribution model. Otherwise, if we can prove the problem does not exist, it can be considered that EDAs may not have an inevitable weakness on linkage learning.

Bibliography

- 
- [1] D. H. Ackley. *A connectionist machine for genetic hillclimbing*. Kluwer Academic Publishers, Norwell, MA, USA, 1987.
- [2] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.
- [3] J. S. D. Bonet, C. L. Isbell, and P. Viola. Mimic: Finding optima by estimating probability densities. In M. J. M.C. Mozer and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, page 424. The MIT Press, 1997.
- [4] D. Coffin and R. Smith. The limitations of distribution sampling for linkage learning. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 364–369, Sept. 2007.
- [5] C. Echevoyen, R. Santana, J. A. Lozano, and P. Larrañaga. The impact of exact probabilistic learning algorithms in edas based on bayesian networks. *Linkage in Evolutionary Computation*, pages 109–139, 2008.
- [6] L. Emmendorfer and A. Pozo. A clustering-based approach for linkage learning applied to multimodal optimization. *Linkage in Evolutionary Computation*, pages 225–248, 2008.

- [7] R. Etxeberria and P. Larrañaga. Global optimization using bayesian networks. In *Second Symposium on Artificial Intelligence (CIMA-99)*, pages 332–339, 1999.
- [8] S. Forrest and M. Mitchell. What makes a problem hard for a genetic algorithm? some anomalous results and their explanation. In *Machine Learning*, pages 285–319, 1993.
- [9] D. E. Goldberg. Simple genetic algorithms and the minimal, deceptive problem. In *Genetic algorithms and simulated annealing*, pages 74–88, 1987.
- [10] D. E. Goldberg. Genetic algorithms and walsh functions: Part i, a gentle introduction. In *Complex Systems*, volume 3, pages 129 – 152, 1989.
- [11] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.
- [12] D. E. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*, chapter 12, pages 187–216. Kluwer Academic Publishers, Norwell, MA, USA, 2002.
- [13] D. E. Goldberg, K. Deb, and J. H. Clark. Genetic algorithms, noise, and the sizing of populations. In *Complex Systems*, volume 6, pages 333 – 362, 1992.
- [14] G. Harik. Linkage learning via probabilistic modeling in the ecga. Technical report, IlliGAL, University of Illinois at Urbana-Champaign, 1999.
- [15] G. Harik, E. Cantú-Paz, D. E. Goldberg, and B. L. Miller. The gambler’s ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, 7(3):231–253, 1999.
- [16] G. Harik, F. Lobo, and D. Goldberg. The compact genetic algorithm. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 523–528, May 1998.

- [17] G. R. Harik. Finding multimodal solutions using restricted tournament selection. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 24–31, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.
- [18] P. Larrañaga and J. A. Lozano, editors. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2002.
- [19] C. Lima, M. Pelikan, D. Goldberg, F. Lobo, K. Sastry, and M. Hauschild. Influence of selection and replacement strategies on linkage learning in boa. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 1083–1090, Sept. 2007.
- [20] H. Mühlenbein and G. Paass. From recombination of genes to the estimation of distributions i. binary parameters. In *PPSN IV: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pages 178–187, London, UK, 1996. Springer-Verlag.
- [21] M. Pelikan and D. E. Goldberg. Escaping hierarchical traps with competent genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2001)*, pages 511–518. Morgan Kaufmann, 2001.
- [22] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. Boa: The bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, pages 525–532, 1999.
- [23] M. Pelikan, D. E. Goldberg, and F. G. Lobo. A survey of optimization by building and using probabilistic models. *Comput. Optim. Appl.*, 21(1):5–20, 2002.
- [24] M. Pelikan and H. Mühlenbein. The bivariate marginal distribution algorithm. In R. Roy, T. Furuhashi, and P. Chawdhry, editors, *Advances in Soft Computing - Engineering Design and Manufacturing*, pages 521–535, London, 1999.

- [25] M. Tsuji, M. Munetomo, and K. Akama. Modeling dependencies of loci with string classification according to fitness differences. In *Genetic and Evolutionary Computation (GECCO 2004)*, pages 246–257, 2004.
- [26] H. Wu and J. L. Shapiro. Does overfitting affect performance in estimation of distribution algorithms. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 433–434, New York, NY, USA, 2006. ACM.
- [27] T.-L. Yu, D. E. Goldberg, A. Yassine, and Y.-P. Chen. A genetic algorithm design inspired by organization theory: a pilot study of a dependency structure matrix driven genetic algorithm. Technical report, IllGAL, University of Illinois at Urbana-Champaign, Urbana, IL, 2003.

