การจำแนกส่วนประกอบโดยเมทริกซ์ความพร้อมเพรียง

นาย ชัชวิทย์ อาภรณ์เทวัญ

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรดุษฎีบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2547

ISBN 974-17-4574-5

ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

BUILDING-BLOCK IDENTIFICATION BY SIMULTANEITY MATRIX

MR. CHATCHAWIT APORNTEWAN

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

A Dissertation Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Engineering in Computer Engineering

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2004

ISBN 974-17-4574-5

Thesis title	Building-block Identification by Simultaneity Matrix
By	Chatchawit Aporntewan
Field of Study	Computer Engineering
Thesis Advisor	Associate Professor Prabhas Chongstitvatana, Ph.D.

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Doctor's Degree

. . . . Dean of Faculty of Engineering
(Professor Direk Lavansiri, Ph.D.)

THESIS COMMITTEE

. . . . Chairman
(Professor Chidchanok Lursinsap, Ph.D.)

. . . . Thesis Advisor
(Associate Professor Prabhas Chongstitvatana, Ph.D.)

. . . . Member
(Associate Professor Somchai Prasitjutrakul, Ph.D.)

. . . . Member
(Assistant Professor Boonserm Kijsirikul, Ph.D.)

. . . . Member
(Assistant Professor Sanya Mitaim, Ph.D.)

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

ชัชวิทย์ อาภรณ์เทวัญ : การจำแนกส่วนประกอบโดยเมทริกซ์ความพร้อมเพรียง (Building-block Identification by Simultaneity Matrix). อาจารย์ที่ปรึกษา : รศ. ดร. ประภาศ จงสถิตย์วัฒนา, 77 หน้า. ISBN 974-17-4574-5

เมทริกซ์ความพร้อมเพรียงคือเมทริกซ์ของตัวเลขขนาด $\ell \times \ell$ เมทริกซ์ถูกสร้างขึ้นมาจากเซตของคำตอบที่มีความยาว ℓ บิท สมาชิกของเมทริกซ์ (m_{ij}) แสดงค่าความสัมพันธ์ระหว่างบิทที่ตำแหน่ง i และบิทที่ตำแหน่ง j เราแบ่งพาร์ทิชันของ $\{0, \ldots, \ell - 1\}$ โดยใส่ i และ j ไว้ในพาร์ทิชันสับเซ็ทเดียวกัน ถ้า m_{ij} มีค่ามาก พาร์ทิชันจะถูกใช้ในการผสมคำตอบ เพื่อที่ว่าบิทที่อยู่ในพาร์ทิชันสับเซ็ทเดียวกันจะติดไปด้วยกัน การใช้เมทริกซ์ความพร้อมเพรียงทำให้หาผลเฉลยที่ดีที่สุดของ Additively Decomposable Functions (ADFs) และ Hierarchically Decomposable Functions (HDFs) ได้โดยใช้จำนวนครั้งในการคำนวณฟังก์ชันเพิ่มขึ้นแบบพหุนามตามขนาดของปัญหา การเปรียบเทียบกับ hierarchical Bayesian Optimization Algorithm (hBOA) แสดงให้เห็นว่า hBOA คำนวณค่าฟังก์ชันเป็นจำนวนครั้งน้อยกว่า แต่การคำนวณเมทริกซ์เทียบกับการสร้างโครงข่ายของเบย์ ใช้เวลาน้อยกว่า 10 เท่า และใช้หน่วยความจำน้อยกว่า 10 เท่า

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

ภาควิชา	วิศวกรรมคอมพิวเตอร์	ลายมือชื่อนิสิต ....
สาขาวิชา	วิศวกรรมคอมพิวเตอร์	ลายมือชื่ออาจารย์ที่ปรึกษา ....
ปีการศึกษา	2547	ลายมือชื่ออาจารย์ที่ปรึกษาร่วม ....

##4371804321 : MAJOR COMPUTER SCIENCE

KEYWORDS : GENETIC ALGORITHMS, BUILDING-BLOCK IDENTIFICATION, LINKAGE LEARNING, SIMULTANEITY MATRIX

CHATCHAWIT APORNTEWAN : BUILDING-BLOCK IDENTIFICATION BY SIMULTANEITY MATRIX. THESIS ADVISOR : ASSOC. PROF. PRABHAS CHONGSTITVATANA, Ph.D., 77 pp. ISBN 974-17-4574-5

The simultaneity matrix is an $\ell \times \ell$ matrix of numbers. The matrix is constructed according to a set of ℓ-bit solutions. The matrix element m_{ij} is the degree of linkage between bit positions i and j. We partition $\{0, \ldots, \ell - 1\}$ by putting i and j in the same partition subset if m_{ij} is significantly high. The partition represents the bit positions of building blocks. The partition is exploited in solution recombination so that the bits governed by the same partition subset are passed together. It can be shown that identifying building blocks by the simultaneity matrix can solve the additively decomposable functions (ADFs) and hierarchically decomposable functions (HDFs) in a polynomial relationship between the number of function evaluations required to reach the optimum and the problem size. A comparison to the hierarchical Bayesian optimization algorithm (hBOA) is made. The hBOA uses less number of function evaluations than that of our algorithm. However, computing the matrix is 10 times faster and uses 10 times less memory than constructing Bayesian network.

Department	Computer Engineering	Student's signature ....
Field of study	Computer Engineering	Advisor's signature ....
Academic year	2004	Co-advisor's signature ....

Acknowledgements

I used to think about so many things that I plan to write here, but I forgot it. Somebody said “Ph.D. stands for Permanent Head Damage.” I begin to agree with him. In the first year, there are a lot of questions stuck in my mind. Why do I study in Ph.D. program? What do I study for? ....? ....? ....? Now I stop thinking about those questions. Do I grow up or give up to the destiny?

I would like to express my gratitude to my advisor, Assoc. Prof. Prabhas Chongstitvatana, for his valuable advice and continuous support. Many times that we have to stay over night. The funniest project is the IC design contest in 2000-2001. He endorsed our teams about fifty thousand baht. I worried about paying the money back to him. However, the teams can make a small profit. If we were able to forget the sufferings about qualification exam, finding a thesis topic, being unable to put the work forward, and rejected papers, Ph.D. life is not too bad.

I would like to thank many people, my grand mother, my parents, Assoc. Prof. David Ruffolo (for the name simultaneity matrix), Prof. Phillip Rogaway (for the partitioning algorithm), Dr. Athasit Surarerk (for the correctness proofs). I am being supported by Chulalongkorn university’s scholarship given in the occasion of the Sixth-Cycle (72$^{\mathrm{nd}}$) Birthday Anniversary of His Majesty King Bhumibol Adulyadej.

Chatchawit Aporntewan
March 18, 2004

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

Contents

Contents (cont.)

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

List of Tables

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

List of Figures

I shall strive to the end, but whether gain or loss is beyond my powers to foresee.

(Chuko Liang, Romance of the Three Kingdoms)

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER I

Motivation

1.1 Introduction to Genetic Algorithms

Genetic Algorithms (GAs), proposed by J. H. Holland in 1975, is an algorithm which simulates natural evolution [34]. Holland's motivation is to study the behavior of complex and adaptive systems. GAs are later popularized by D. E. Goldberg as it is a robust optimization algorithm [19, 45]. The robustness means that GAs can be applied to a wide range of optimization problems. The behavior of the simple GA on several functions (famously known as *De Jong's test suite*) is shown elsewhere [14]. For an introduction to the subject, "A Genetic Algorithm Tutorial" is recommended [78] (download at http://www.cs.colostate.edu/~genitor/MiscPubs/tutorial.ps.gz). The simple GA consists of a population of individuals [19]. An individual, $\mathcal{I}$, is defined as:

$$\mathcal{I} = b_0 \ldots b_{\ell-1},\ b_i \in \{0,1\} \tag{1.1}$$

A population at time t, $\mathcal{P}(t)$, is defined as:

$$\mathcal{P}^t = \{\mathcal{I}_0^t, \ldots, \mathcal{I}_{n-1}^t\} \tag{1.2}$$

Usually, n is an even number. A pair of individuals is the result of previous population.

$$\{\mathcal{I}_i^t, \mathcal{I}_{i+1}^t\} = \mathcal{M}(\mathcal{C}(\mathcal{S}(\mathcal{P}^{t-1}))),\ i = 0, 2, \ldots, n-2 \tag{1.3}$$

The operators, $\mathcal{S}$, $\mathcal{C}$, and $\mathcal{M}$, are called *selection*, *crossover*, and *mutation*, respectively. The three operators are defined as:

$$\mathcal{S}(\mathcal{P}^t) = \{\mathcal{I}_i^t, \mathcal{I}_j^t\} \text{ with probability } \frac{f(\mathcal{I}_i^t)}{\sum_{k=0}^{n-1} f(\mathcal{I}_k^t)} \times \frac{f(\mathcal{I}_j^t)}{\sum_{k=0}^{n-1} f(\mathcal{I}_k^t)} \tag{1.4}$$

In GAs literature, $f : \mathcal{I} \to R^+$ is called *fitness function*. The crossover is defined as:

$$\mathcal{C}(\{\mathcal{I}_1, \mathcal{I}_2\}) = \begin{cases} \{(\mathcal{I}_1 \ \&\ m_1) \otimes\ (\mathcal{I}_2 \ \&\ \sim m_1), (\mathcal{I}_1 \ \&\ \sim m_1) \otimes\ (\mathcal{I}_2 \ \&\ m_1)\} & \text{with probability } p_c \\ \{\mathcal{I}_1,\ \mathcal{I}_2\} & \text{with probability } 1 - p_c \end{cases} \tag{1.5}$$

where m_1 is an ℓ-bit binary number and m_1 is randomly chosen from $\{x \mid x = 2^k - 1,\ 1 \leq k \leq \ell - 1\}$. The symbols &, $\otimes$, and $\sim$ denotes the logical AND, logical EXCLUSIVE, and logical NOT, respectively. The mutation is defined as:

$$\mathcal{M}(\{\mathcal{I}_1, \mathcal{I}_2\}) = \{\mathcal{X}(\mathcal{I}_1),\ \mathcal{X}(\mathcal{I}_2)\} \tag{1.6}$$

$$\mathcal{X}(\mathcal{I}) = \mathcal{I} \otimes m_2 \tag{1.7}$$

where

$$m_2 = b_0 \ldots b_{\ell-1},\ b_i = \begin{cases} 0 & \text{with probability } 1 - p_m \\ 1 & \text{with probability } p_m \end{cases} \tag{1.8}$$

It should be noted that m_1 and m_2 are reproduced every time performing $\mathcal{C}$ and $\mathcal{X}$, but p_c and p_m are constant values. Let $\mathcal{P}^0$ be generated at random. Consequently, $\mathcal{P}^t, t \geq 1$ are obtained by the definition.

1.2 Schema Theorem

The schema theorem explains the behavior of the simple GA. A schema (a set of individuals), $\mathcal{H}$, is defined as:

$$\mathcal{H} = h_0 \ldots h_{l-1},\ h_i \in \{0, 1, *\} \tag{1.9}$$

Let an individual $\mathcal{I} = b_0 \ldots b_{l-1},\ b_i \in \{0, 1\}$. $\mathcal{I} \in \mathcal{H}$ if and only if for all $i,\ b_i = h_i$ or $h_i = *$. Let $m(\mathcal{H}, t)$ be the number of individuals in $\mathcal{P}^t$ which belong to $\mathcal{H}$. The order of a schema, $o(\mathcal{H})$, is defined as:

$$o(\mathcal{H}) = \sum_{i=0}^{l-1} c(h_i),\ c(h_i) = \begin{cases} 0 & ; \quad \text{if } h_i = * \\ 1 & ; \quad \text{otherwise} \end{cases} \tag{1.10}$$

The defining length of a schema, $\delta(\mathcal{H})$, is defined as:

$$\delta(\mathcal{H}) = \max_{i\, \in\, \mathcal{N}}(i) - \min_{i\, \in\, \mathcal{N}}(i),\ \mathcal{N} = \{j \mid h_j \neq *\} \tag{1.11}$$

The schema theorem is shown in Equation 1.12 [34, 19].

$$m(\mathcal{H}, t+1) \geq \frac{m(\mathcal{H}, t) f(\mathcal{H})}{f_{avg}} \left(1 - p_c \frac{\delta(\mathcal{H})}{l - 1} - p_m o(\mathcal{H})\right) \tag{1.12}$$

where $f(\mathcal{H}) = \sum_{\mathcal{I}\in\mathcal{H}} f(\mathcal{I}) \ / \ m(\mathcal{H},t)$ and $f_{avg} = \sum_{\mathcal{I}\in\mathcal{P}^t} f(\mathcal{I}) \ / \ n$ (n denotes the population size). Let $c = \frac{f(\mathcal{H})}{f_{avg}}\left(1 - p_c\frac{\delta(\mathcal{H})}{l-1} - p_m o(\mathcal{H})\right)$. Hence,

$$m(\mathcal{H},t) \geq m(\mathcal{H},0)c^t \tag{1.13}$$

The above-average, low-order, short-defining length schema results in $c > 1$. Therefore, $m(\mathcal{H},t)$ grows exponentially with t. This satisfies Holland's motivation, that is explaining the behavior of the simple GA [15]. However, the schema theorem shows the weakness if we consider GAs as an optimization algorithm. The simple GA performs badly when highly-fit individuals are not in the above-average, low-order, short-defining length schemata. Recent advancement of the schema theorem is shown elsewhere [3, 7, 73, 74].

1.3 Inductive Bias in Genetic Algorithms

No Free Lunch (NFL) theorem states that the performance of any optimization algorithms, averaged on all possible optimization functions, is identical [83, 84]. In other words, GAs are as good as a random search. The random search is defined as:

$$\mathcal{P}^t = \{\mathcal{I}^t\} \tag{1.14}$$

where $\mathcal{I}^t$ is drawn from a distribution, $\mathcal{D}^t$. In the random search, $\mathcal{D}^t$ is a uniform distribution for all t. If $\mathcal{D}^t$ is not a uniform distribution, it is said there is an *inductive bias*. Most optimization algorithms infer the distribution $\mathcal{D}^t$ from the past information, $f(\mathcal{I}^0), \ldots, f(\mathcal{I}^{t-1})$. The NFL theorem averages the performance of an optimization algorithm on all possible functions including the functions in which no optimization algorithm can gain the benefit of the past information. Such a function is rarely found in scientific and engineering applications.

Certainly, there is an inductive bias in the simple GA because the population is not random. An individual in the current population is a recombination of individuals in the previous population. The preceding individuals are selected proportionally to their fitness (see Equation 1.4). Informally speaking, the inductive bias is to explore an individual that is a recombination of highly-fit individuals. The inductive bias in GAs is referred to as *building-block hypothesis* – the highly-fit individuals are composed of building blocks.

Building block is what you infer from a set of highly-fit individuals [23, pp. 60–61]. Most people infer from Table 1.1 that the highly-fit individuals are composed of "00000" and "11111." The dependency between variables $b_i, b_{i+1}, b_{i+2}, b_{i+3}, b_{i+4}$ where $i = 0, 5, 10, 15, 20, 25, 30, 35, 40, 45$ is recognized by means of a statistical method, for example, Pearson's chi-square (pp. 23). The dependent variables are put in the same partition subset. We write only subscript letters. Hence,

$$\{\{0, 1, 2, 3, 4\}, \ldots, \{45, 46, 47, 48, 49\}\} \tag{1.15}$$

The partition shown in Equation 1.15 are bit positions of the building blocks. The bit values correspond to the population of highly-fit individuals. In the simple GA, the crossover operator plays an important role in mixing building blocks. The crossover operator recombines and also disrupts the building blocks because it does not consider the dependency between variables.

Table 1.1: A population of highly-fit individuals

$b_0 \ldots b_{49}$									
11111	11111	11111	11111	00000	11111	11111	11111	00000	00000
11111	00000	00000	11111	00000	00000	00000	11111	00000	11111
00000	00000	11111	00000	00000	11111	11111	11111	11111	00000
11111	11111	11111	00000	11111	11111	00000	11111	11111	11111
00000	11111	00000	00000	00000	00000	00000	11111	00000	11111
11111	00000	11111	00000	00000	00000	00000	00000	11111	00000
11111	00000	00000	11111	11111	00000	00000	00000	11111	00000

Michalski (2000) showed that the rule-based learning is able to learn the attributes of highly-fit individuals [44]. In Michalski's paper, it is clear what is an inductive bias because the rule-based learning is a learning algorithm in Machine Learning (ML) [46].

The GAs are robust because its inductive bias is general for many optimization problems. In contrast, a heuristic search is specifically designed to fit a problem. The heuristic search cannot be effectively used with another problem. However, with the current understanding of the simple GA, we can design a test function in which the simple GA performs badly. The test functions give the clear weakness of the simple GA. The test functions also provide an insight for improving the inductive bias of the simple GA. The next section introduces *deceptive functions* also called *trap functions* [1, 12].

1.4 Trap Functions

In this section, trap functions show the case in which the building blocks are more likely to be disrupted [1, 12]. A more general definition of trap functions will be shown in Section 2.1. The 5-bit trap function, F_5, is defined as:

$$F_5(b_0b_1b_2b_3b_4) = \begin{cases} 5 & ; \ u = 5 \\ 4 - u & ; \ \text{otherwise,} \end{cases} \qquad u = \sum_{i=0}^{4} b_i,\ b_i \in \{0, 1\} \tag{1.16}$$

An additively decomposable function (ADF), $F_{10\times5}$, is defined as:

$$F_{10\times5}(b_0 \dots b_{49}) = \sum_{i=0}^{9} F_5(b_{5i}b_{5i+1}b_{5i+2}b_{5i+3}b_{5i+4}) \tag{1.17}$$

The ADF fools the simple GA to favor substring “00000” while the highly-fit individuals are composed of “11111.”

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

The defining length of a schema varies with its ordering. For example, the same function, $F_{10\times5}$, is rewritten as:

$$F_{10\times5}(b_0 \dots b_{49}) = \sum_{i=0}^{9} F_5(b_ib_{i+10}b_{i+20}b_{i+30}b_{i+40}) \tag{1.18}$$

With the ordering in Equation 1.17, a schema $h_1 =$

“11111 *.”

With the ordering in Equation 1.18, h_1 denoted by $h_2 =$

“1 * * * * * * * * * 1 * * * * * * * * * 1 * * * * * * * * * 1 * * * * * * * * * 1 * * * * * * * * *.”

$\delta(h_1) = 4$ and $\delta(h_2) = 40$. The schema h_1 is identical to h_2, but $\delta(h_2) > \delta(h_1)$. Long defining length causes $c < 1$ in Equation 1.13. Consequently, the number of solutions that match schema h_2 *decreases* exponentially.

Thierens raised the scalability issue of the simple GA [71, 72]. He used the uniform crossover so that the building blocks are randomly mixed (the uniform crossover is defined by randomly choosing mask bits, m_1, from $\{x \mid 1 \leq x \leq 2^l - 2\}$ [70]). The fitness function is the $m{\times}5$-trap function. The analysis shows that either the computational time grows exponentially with the number of 5-bit trap functions or the population size must be exponentially increased. It is clear that scaling up the problem size requires information about the building blocks so that the solutions are efficiently mixed. In addition, the performance of simple GAs relies on the ordering of solution bits. The ordering may not pack the dependent bits close together. Such an ordering results in poor building-block mixing. Therefore the building blocks need to be identified to improve the scalability issue.

The building-block identification is sometimes referred to as *ordering problem* or *linkage learning*. The word “linkage” is borrowed from biology. In its biological meaning, linkage is the tendency for alleles of different genes to be passed together from one generation to the next [82]. A gene is analogous to a bit position. An allele is analogous to to a bit value. We prefer to avoid the word “linkage” because it may confuses biologists.

1.5 The Purpose of the Study

People are fascinated by GAs’ magic – a wide range of problems are applicable, only fitness function is required, and prior information is optional. Such an algorithm which satisfies these properties is referred to as *blackbox optimization algorithms*, for example, simulated annealing (SA) [42]. It is clear that an optimization algorithm cannot solve all problems effectively [83, 84]. While most blackbox optimization algorithms exploit gradients, building blocks distinguish GAs from the others. A goal of current research is to develop GAs to be an algorithm that is good at composing building blocks. The problems that should be effectively solved by GAs are narrowed to problems that can be solved by composing building blocks of order bounded by a constant.

The test functions described in Section 2.1 are accepted as a loosely agreed guideline for GA designers. The test functions resist gradient-based algorithms, but they can be solved by identifying and composing building blocks. The test functions serve as a candidate of functions that we are interested in. Choosing a test function is based on the distribution of functions that will be observed in application domain. Usually, functions that will be frequently observed are preferable. Using a test suite (or benchmark) is common in particular disciplines, for example, computer architecture [32] and data compression [66]. In computer architecture, benchmark is widely accepted in both academic and commercial institutes (see http://www.specbench.org).

The main reason for using benchmark is that a number of disciplines falls in No Free Lunch (NFL) theorem [83, 84]. Data compression is an obvious case for NFL theorem. You can safely say that no compression algorithm is able to compress all files of which the size is less than k bytes (all compressed files must be smaller than the originals). The proof is omitted because it is trivial. An approach for compression is to assume the input distribution, and therefore compression techniques are separated for text, image, voice, etc. Similarly, blackbox optimizations are separated in many categories: Evolutionary Programming (EP) [17, 18], Evolution Strategies (ES) [6, 81], Genetic Algorithms (GA) [19, 34], and Genetic Programming (GP) [43]. Each of them is efficient for a particular problem domain (finite-state machine, vector of real numbers, binary string, program tree). Certainly, an algorithm that is good at all problem domains is impossible due to NFL theorem. Thus, the test suite is indispensable and it must be predefined.

The test suite is a set of functions that can be solved by considering building blocks of order bounded by a constant k. The difficulty arises when k is large [24]. Current research studies the case $k < 10$. Without building-block identification, the complexity of the functions described in Chapter 2 grows exponentially. We aims to improve the performance of GAs so that the problems of bounded difficulty can be solved in a polynomial time. The core of our algorithm is the part that identifying building blocks. In brief, the building-block identification algorithm takes input that is a set of ℓ-bit binary strings. The output is a partition of $\{0, \ldots, \ell-1\}$ where ℓ is the number of solution bits. A comparison between the building-block identification algorithms will be made.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

Anyone who conducts an argument by appealing to authority

is not using his intelligence, he is just using his memory.

(Leonado da Vinci)

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER II

Literature Reviews

Many strategies in the literature use the bit-reordering approach to pack the dependent bits close together, for example, inversion operator [19], messy GAs [20], symbiotic evolution [54], recombination strategy adaptation [68], adaptive linkage crossover [65], and linkage learning GA [26, 27]. The bit-reordering approach does not explicitly identify building blocks, but it successfully delivers the optimal solution. Several works explicitly identify building blocks [37, 39, 53]. An approach is to find a partition of bit positions [29, 38, 41, 50]. For instance, Table 1.1 infers the partition:

$$\{\{0, 1, 2, 3, 4\}, \ldots, \{45, 46, 47, 48, 49\}\} \tag{2.1}$$

In the case of nonoverlapped building blocks, partition is a clear representation. Note that Kargupta [41] computes Walsh's coefficients which imply the partition. The bits governed by the same partition subset are passed together to prevent building block disruption.

Identifying building blocks is somewhat related to building a distribution of solutions [4, 13, 29, 48, 55, 56]. The basic concept of optimization by building a distribution is to start with a uniform distribution of solutions. Next, a number of solutions is drawn according to the distribution. Some good solutions (winners) are selected, and the distribution is adjusted toward the winners (the winners-like solutions will be drawn with higher probability in the next iteration). These steps are repeated until the optimal solution is found or reaching a termination condition. The works in this category are referred to as *probabilistic model-building genetic algorithms (PMBGAs)*. For a particular form of distribution used in the extended compact genetic algorithm (ECGA), building the distribution is identical to searching for a partition [29]. The Bayesian optimization algorithm (BOA) uses Bayesian network to represent a distribution [55]. Pelikan showed that if the problem is composed of k-bit trap functions, the network will be fully connected sets of k nodes [63, pp. 54]. In addition, Bayesian network is able to represent joint distributions in the case of overlapping building blocks. The hierarchical BOA (hBOA) is the BOA enhanced with decision tree/graph and a niching method called restricted tournament replacement [63]. The hBOA can solve the hierarchically decomposable functions (HDFs) in a scalable manner. Successful applications of building-block identification are financial applications [37], distributed data mining [40], cluster optimization [67], maximum satisfiability of logic formulas (MAXSAT) and Ising spin glass systems [64].

This chapter is organized as follows. The first section describes test functions that are commonly used. The second section presents the approaches that are used in the literature. The approaches are considerably different to each other. Some approaches have an underlying assumption which limits the practical use. Some approaches are able to identifying building blocks, but the resources they consume grow faster than the polynomial with the problem size. The third section presents probabilistic model-building genetic algorithms (PMBGAs) that are the main paradigm for building-block identification. The PMBGAs are scalable. More precisely, the computational time required to reach the optimal solution grows in a polynomial relationship with the problem size.

2.1 Test Functions

Several test functions have been developed [16, 30, 35, 36, 76, 79]. An important goal of designing a test function is to measure the effectiveness of building-block composition. Whitley gave the guidelines of a test function as follows [79]. The test function should be resistant to hill-climbing, nonlinear, nonseparable, nonsymmetric, scalable, and have a canonical form. Commonly used test functions are royal road, $n{\times}3$-trap, $n{\times}5$-trap, $n{\times}6$-bipolar, hierarchical if-and-only-if (HIFF), hierarchical trap 1 (HTrap1), and hierarchical trap 2 (HTrap2) [36, 63]. These functions are used by many researchers, hence it is possible to compare the results. At present, the number of test functions is small. The test functions used in the thesis are described in the following subsections.

2.1.1 n-trap functions

The n-trap function [1, 12] is defined as:

$$F_n : B \rightarrow R,\ B \in \{0,1\}^n,\ R \in [0, f_{high}] \tag{2.2}$$

$$F_n(B) = \begin{cases} f_{high} & ;\ \text{if } u = n \\ f_{low} - u\frac{f_{low}}{n-1} & ;\ \text{otherwise} \end{cases} \tag{2.3}$$

where u is the number of “1” bits in B. Usually, f_{high} is set at n and f_{low} is set at $n-1$. The f_{high} value is usually greater than f_{low} in order to deceive an optimization algorithm to favor “0,” but the optimal solution consists of all “1” bits. The 5-trap function is shown in Figure 2.1 ($f_{high} = 5$, $f_{low} = 4$).

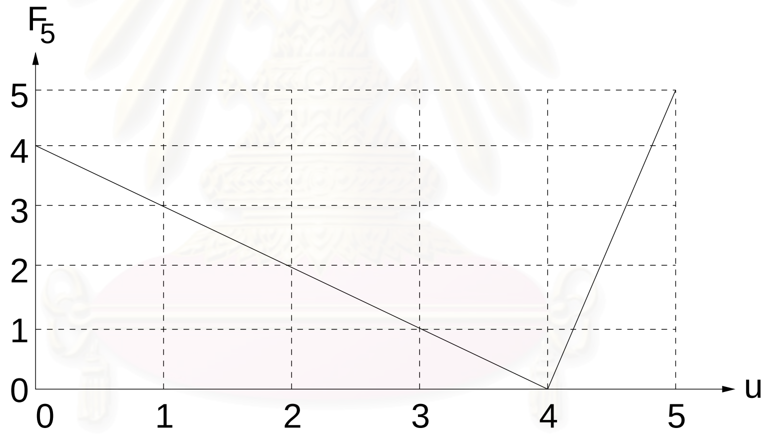


Figure 2.1: 5-trap function

2.1.2 $m{\times}k$-trap functions

The $m{\times}k$-trap function [55, 63] is defined as:

$$F_{m\times k} : B \rightarrow R,\ B \in B_0 \ldots B_{m-1},\ B_i \in \{0,1\}^k \tag{2.4}$$

$$F_{m\times k}(B_0 \ldots B_{m-1}) = \sum_{i=0}^{m-1} F_k(B_i) \tag{2.5}$$

The m and k are varied to produce a number of test functions. The $m{\times}k$-trap functions are often referred to as additively decomposable functions (ADFs). The optimal solution is composed of all “1” bits.

2.1.3 HIFF Functions

The HIFF, HTrap2, and HTrap1 functions are often referred to as hierarchically decomposable functions (HDFs). To compute the HIFF functions, a solution is interpreted as a binary tree. An example is shown in Figure 2.2. The solution is an 8-bit string, "00001101." It is placed at the leaf nodes of the binary tree. The leaf nodes are interpreted as the higher levels of the tree. A pair of zeroes and a pair of ones are interpreted as zero and one respectively. Otherwise the interpretation result is "-." The HIFF function returns the sum of values contributed from each node. The contribution of node i, c_i, shown at the upper right of the node, is defined as:

$$c_i = \begin{cases} 2^h & ; \text{ if node } i \text{ is "0" or "1"} \\ 0 & ; \text{ if node } i \text{ is "-"} \end{cases} \tag{2.6}$$

where h is the height of node i. In the example, the fitness of "00001101" is $\sum c_i = 18$. The HIFF functions do not bias an optimizer to favor zeroes rather than ones or vice versa. There are two optimal solutions, the string composed of all zeroes and the string composed of all ones.

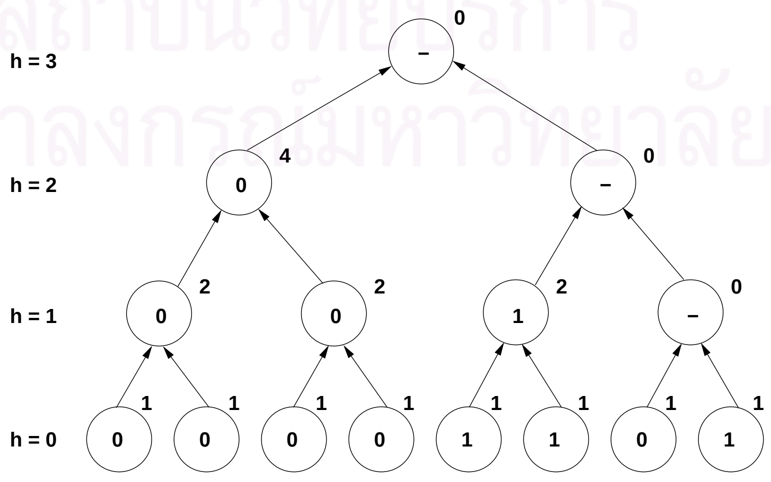


Figure 2.2: Interpreting the solution as a binary tree

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

2.1.4 HTrap1 Functions

The HTrap1 functions interpret a solution as a tree in which the number of branches is greater than two. An example is shown in Figure 2.3. The solution is a 9-bit string placed at the leaf nodes. The leaf nodes do not contribute to the function. The interpretation rule is similar to that of the HIFF functions. Triple zeroes are interpreted as zero and triple ones are interpreted as one. Otherwise the interpretation result is "-." The contribution of node i, c_i, is defined as:

$$c_i = \begin{cases} 3^h \cdot F_3(b_0 b_1 b_2) & ; \text{ if } b_j \neq \text{“-”} \text{ for all } 0 \leq j \leq 2 \\ 0 & ; \text{ otherwise} \end{cases} \tag{2.7}$$

where

h is the height of node i,

b_0, b_1, b_2 are the interpretations in the left, middle, right children of node i.

At the root node, the 3-trap function's parameters are f'_{high} = 1 and f'_{low} = 0.9. The other nodes use f_{high} = 1 and f_{low} = 1. In Figure 2.3, the HTrap1 function returns $\sum c_i$ = 13.05. The optimal solution is composed of all ones.

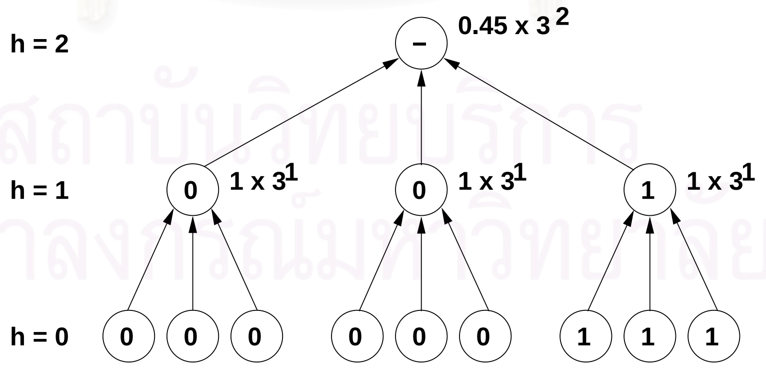


Figure 2.3: Interpreting the solution as a 3-branch tree

2.1.5 HTrap2 Functions

The HTrap2 functions are similar to the HTrap1 functions. The only difference is the 3-trap function's parameters. In the root node, $f'_{high} = 1$ and $f'_{low} = 0.9$. The other nodes use $f_{high} = 1$ and $f_{low} = 1 + \frac{0.1}{h}$ where h is the tree height. The optimal solution is composed of all ones if the following condition is true.

$$f'_{high} - f'_{low} > (h-1)(f_{low} - f_{high}) \tag{2.8}$$

The parameter setting $(f'_{high} = 1, f'_{low} = 0.9, f_{high} = 1, f_{low} = 1 + \frac{0.1}{h})$ satisfies the condition. The HTrap2 functions are more deceptive than the HTrap1 functions. Only root node guides an optimizer to favor ones while the other nodes fool the optimizer to favor zeroes by setting $f_{low} > f_{high}$.

2.2 Approaches for Identifying Building Blocks

The following subsections summarize the approaches for identifying building blocks. The approaches are presented chronologically.

2.2.1 Inversion operator

The first attempt to resolve the ordering problem may be the inversion operator [19, 21]. The inversion operator is similar to what happens in natural evolution. An attribute of a creature is defined by *a gene or multiple genes*. All genes are located on the long string called DNA strain. The value of a gene is called *allele*. The expression of genes are dependent to their alleles, but the expression are independent to the distance between two genes. The creature inherits genes from their parents. The gene locations can be changed by the inversion operator. By means of natural evolution, the interacting genes gradually become close together. This increases the probability that the attribute, the interaction of genes, will be inherited to the offspring.

The inversion operator is illustrated in Figure 2.4. First, a chunk of adjacent bits are randomly selected. Next, the chunk is inversed by left-to-right flipping. The bits are moved around the string, but the meaning (fitness) of the solution is not changed. Only the ordering of the structure is greatly affected. For example, the bits at positions 3 and 6 are passed together with a higher probability. The inversion operator alters the solution

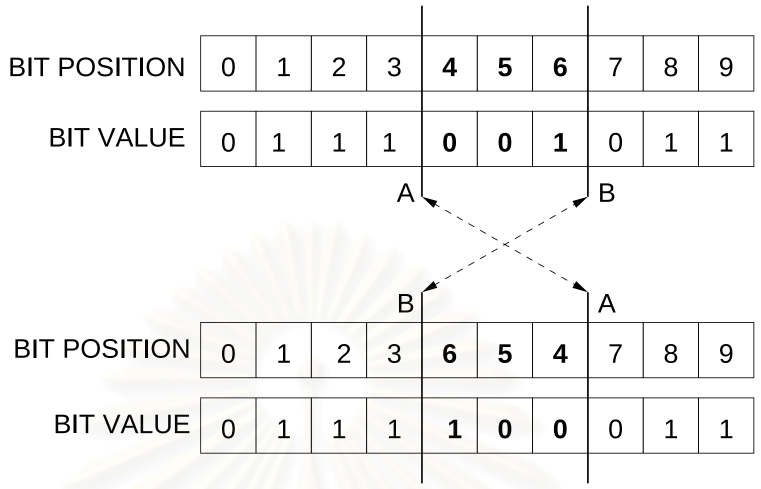


Figure 2.4: Inversion operator

structure at random. The tight structures (dependent bits being close together) are more likely to appear in the final generation. The simple GA enhanced with inversion operator is able to find the optimal solution for additively decomposable functions. In the worst case, the complexity grows exponentially with the problem size [71, 72].

2.2.2 Messy Genetic Algorithms

The messy GA encodes a solution bit to (p, v) where $p \in \{0, \ldots, \ell - 1\}$ is bit position and $v \in \{0, 1\}$ is bit value. For instance, "0111100011" is encoded as follows.

$$(0, 0) \quad (1, 1) \quad (2, 1) \quad (3, 1) \quad (4, 1) \quad (5, 0) \quad (6, 0) \quad (7, 0) \quad (8, 1) \quad (9, 1)$$

The bits are tagged with the position numbers so that they can be moved around without losing the meaning. When the solution is mixed, the mixed solution may be *over-specified* or *under-specified*. The over-specification is having more than one copy for a bit position. The under-specification is having no copy for a bit position. Several alternatives are proposed for interpreting over-specified and under-specified solutions. For example, the over-specification is resolved by majority voting or first-come, first-serve basis. The under-specification is resolved by means of the *competitive templates*.

The messy GA uses the operators called *cut* and *splice* to recombine solutions. The cut operator cuts the string with a probability that increases with string length. Long over-specified strings are more likely to be cut. The splice operator joins the head of a string to the tail of the others with a fixed probability. By cut and splice, the solution structure can be altered, and therefore forming a tight structure. The messy GA is later developed to fast messy genetic algorithms (FMGA) [22] and gene messy genetic algorithm (GEMGA) [38, 40]. Note that GEMGA is the early paper that can find the optimal solution for $m \times k$-trap functions in a polynomial time with the problem size.

2.2.3 Learning Linkage

The learning linkage genetic algorithm (LLGA) encodes ℓ-bit solutions to 2ℓ distinct pieces of (p, v) placed on a circular string where the bit position $p \in \{0, \ldots, \ell - 1\}$ and the bit value $v \in \{0, 1\}$. The 1-bit solution is encoded as it is shown in Figure 2.5 (left). Interpreting the solution is probabilistic. First, a starting point on the circular string is chosen. Second, walking clockwise and picking up (p, v) by first-come, first-serve basis. For instance, if $(0, 0)$ is encountered first, $(0, 1)$ will not be picked up. The 1-bit solution will be interpreted as $(0, 0)$ with probability $\frac{B}{A+B}$, but the interpretation will be $(1, 1)$ with probability $\frac{A}{A+B}$ where A and B are distances on the circular string. The interpretation result depends on the starting point and the distance between genes.

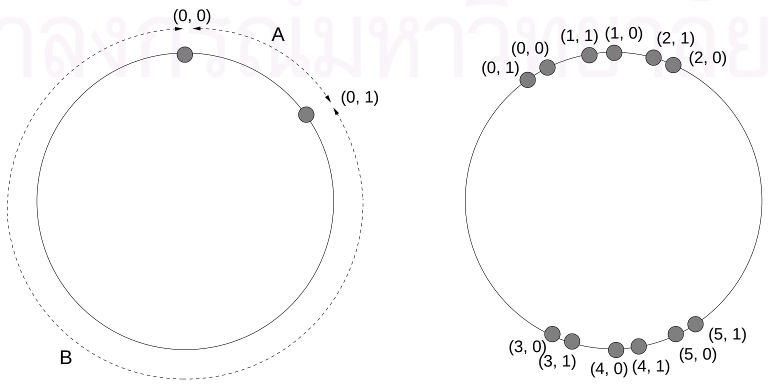


Figure 2.5: Learning linkage

Mixing solutions is done by cutting a continuous segment from the donor string. Then the segment is grafted onto the recipient. Over-specified (p, v) is removed when interpreting the mixed solution. Similar to the other approaches, the solution structure can be altered in order to form the tight structure. A final structure shown in Figure 2.5 (right) is interpreted as "111111" with a high probability. The bits at positions of $\{0, 1, 2\}$ and $\{3, 4, 5\}$ are very likely to be passed together. An analysis is shown in Harik's dissertation [27]. The analysis consists of two parts: *linkage skew* and *linkage shift*.

2.2.4 Non-monotonicity Detection

The definition of non-monotonicity are developed in a series of papers [49, 50, 51, 52]. The non-monotonicity is defined as:

if ($\triangle f_i(s) > 0$ and $\triangle f_j(s) > 0$) then ($\triangle f_{ij}(s) > \triangle f_i(s)$ and $\triangle f_{ij}(s) > \triangle f_j(s)$)

if ($\triangle f_i(s) < 0$ and $\triangle f_j(s) < 0$) then ($\triangle f_{ij}(s) < \triangle f_i(s)$ and $\triangle f_{ij}(s) < \triangle f_j(s)$)

where

$$\triangle f_i(s) = f(...\overline{s}_i...) - f(...s_i...) \tag{2.9}$$

$$\triangle f_{ij}(s) = f(..\overline{s}_i..\overline{s}_j..) - f(..s_i..s_j..) \tag{2.10}$$

f denotes fitness function. s denotes binary string $s_0 \ldots s_{\ell-1}$, $s_i \in \{0, 1\}$. $\overline{s}_i$ denotes $1 - s_i$. Algorithm linkage identification by non-monotonicity (LIMD) is outlined in Figure 2.6. The motivation is to find which bits should be passed together. That is described by the data structure called *linkage sets* denoted by $L_i,\ 0 \leq i \leq \ell - 1$. In the second step, $L_i = \{i\}$ means all bits are independent. If at least a randomized solution violates the non-monotonicity on bit positions i and j, the dependency between i and j is expected. Hence, $L_i \leftarrow L_i \cup \{j\}$ and $L_j \leftarrow L_j \cup \{i\}$.

There might be overlapping between the linkage sets, for example, $L_0 = \{0, 1\}$, $L_1 = \{1, 2\}$, $L_2 = \{2, 0\}$. It is unclear whether "1" should be passed with "0" or "2" or the "0, 1, 2" should be passed together. The author proposed *tightness detection*. The

```
1.  Randomize a set of solutions $S$;
2.  for $i = 0$ to $\ell - 1$ do
        $L_i \leftarrow \{i\}$;
    endfor
3.  for $i = 0$ to $\ell - 1$ do
        for $j = i + 1$ to $\ell - 1$ do
            if (for some $s \in S$, $non\text{-}monotonicity(s, i, j) = false$ then
                $L_i \leftarrow L_i \cup \{j\}$;
                $L_j \leftarrow L_j \cup \{i\}$;
            endif
        endfor
    endfor
```

Figure 2.6: Linkage identification by non-monotonicity detection (LIMD)

tightness between i and j is defined as:

$$tightness(i, j) = \frac{t_1}{t_1 + t_2} \tag{2.11}$$

where $t_1 = |\{L_k \mid \{i, j\} \subset L_k,\ 0 \leq k \leq \ell-1\}|$ and $t_2 = |\{L_k \mid \{i, j\} \not\subset L_k,\ 0 \leq k \leq \ell-1\}|$. If $tightness(i, j)$ is less than a predefined threshold, $L_i \leftarrow L_i \backslash \{j\}$ and $L_j \leftarrow L_j \backslash \{i\}$. As a result, the linkage sets are equivalent to the partition. For instance, the linkage sets $L_0 = \{1, 2\},\ L_1 = \{0, 2\},\ L_2 = \{0, 1\}, L_3 = \{4, 5\},\ L_4 = \{3, 5\},\ L_5 = \{3, 4\}$ (when writing L_i the member i is omitted) are equivalent to the partition $\{\{0, 1, 2\}, \{3, 4, 5\}\}$.

2.2.5 Walsh's coefficients

An approach for identifying the dependency between variables is to find the Walsh's coefficient [41]. Any function can be written in terms of Walsh's coefficients and Walsh's functions. For example, $f(x_1, x_2, x_3)$, can be written as:

$$\begin{aligned} f(x_1, x_2, x_3) &= w_0 + \\ & w_1\Psi_1(x_1) + \\ & w_2\Psi_2(x_2) + \\ & w_3\Psi_3(x_3) + \\ & w_4\Psi_4(x_1, x_2) + \\ & w_5\Psi_5(x_1, x_3) + \\ & w_6\Psi_6(x_2, x_3) + \\ & w_7\Psi_7(x_1, x_2, x_3) \end{aligned}$$

where

w_i is Walsh's coefficient ($w_i \in R$)

Ψ_i is Walsh's function ($f : R \times \ldots \times R \rightarrow \{-1, 1\}$).

The main algorithm is to find the Walsh's coefficients for a set of random solutions. If we use all 2^ℓ solutions, we can reconstruct the function that are being optimized. Using a small number of solutions is sufficient because we do not actually reconstruct the function. The non-zero Walsh's coefficient indicates the dependency between its associated variables. However, the number of Walsh's coefficients grows exponentially with variables. An underlying assumption is that the function has bounded variable interaction of order-k. Subsequently, the Walsh's coefficients can be calculated in a polynomial time.

2.3 Probabilistic Model-Building Genetic Algorithms

In the early stage, many researchers attacked the ordering problem by designing a crossover operator which well suits for a particular optimization problem [54, 68, 69, 70]. Such a research faces with scalability issue because its complexity grows exponentially with the problem size. A revolution brings GA research to a new paradigm – the probabilistic models [23]. The concept of probabilistic models is to represent a population with a *distribution* [8, 9, 56, 57]. The distribution could be as simple as a uniform distribution or it could be as complex as Bayesian network. The following subsections introduce the milestones in PMBGAs.

2.3.1 Population-based incremental learning (PBIL)

The population-based incremental learning (PBIL) uses a uniform distribution to represent a population [4, 33]. Similar papers that use the uniform distribution are the univariate marginal distribution algorithm [47], compact GA [28], and selfish-gene algorithm [11]. Only the compact GA will be presented because it will be used later in this thesis. The pseudocode of the compact GA is presented in Figure 2.7. The compact GA's parameters are population size (n) and string length (ℓ). A population is represented by an ℓ-dimensional probability vector $(\vec{p})$. The p_i, that is the i^{th}-element of the probability vector $\vec{p}$, is the probability that the i^{th}-position bit of an individual, randomly picked up from the population, will be one. First, $\vec{p}$ is initialized to $(0.5, \ldots, 0.5)$. Next, the individuals a and b are generated according to $\vec{p}$. The fitness values, f_a and f_b, are then assigned to a and b respectively. If $f_a \geq f_b$ then the probability vector will be updated towards the individual a. If $a_i = 1$ and $b_i = 0$ then p_i will be increased by $1/n$. If $a_i = 0$ and $b_i = 1$ then p_i will be decreased by $1/n$. The loop is repeated until each p_i becomes zero or one. Finally, $\vec{p}$ presents the final solution.

The compact GA has no implication more than an introduction to probabilistic models. Because the uniform distribution does not keep the dependency between solution bits, $\vec{p} = \{0.5, 0.5\}$ may be a population of {01, 01, 10, 10} or {00, 00, 11, 11} with an identical probability. The algorithm presented in the next subsection exploits a distribution that keeps the dependency between two variables.

2.3.2 Bivariate marginal distribution algorithm (BMDA)

The bivariate marginal distribution algorithm (BMDA) is more complex [58]. It can keep the dependency between two variables. Given a population, the probability of observing b_i and b_j of an individual $\mathcal{I} = b_0 \ldots b_{\ell-1}$, randomly drawn from the population,

```
for $i = 0$ to $\ell - 1$ do $p_i \leftarrow 0.5$;
repeat
    for $i = 0$ to $\ell - 1$ do
        $a_i \leftarrow \begin{cases} 1 & \text{with probability } p_i \\ 0 & \text{otherwise} \end{cases}$
        $b_i \leftarrow \begin{cases} 1 & \text{with probability } p_i \\ 0 & \text{otherwise} \end{cases}$
    endfor
    $f_a \leftarrow$ fitness($a$);
    $f_b \leftarrow$ fitness($b$);
    for $i = 0$ to $\ell - 1$ do
        if $f_a \geq f_b$ then
            if $a_i = 1$ and $b_i = 0$ then $p_i \leftarrow$ min(1, $p_i + \frac{1}{n}$);
            if $a_i = 0$ and $b_i = 1$ then $p_i \leftarrow$ max(0, $p_i - \frac{1}{n}$);
        else
            if $a_i = 1$ and $b_i = 0$ then $p_i \leftarrow$ max(0, $p_i - \frac{1}{n}$);
            if $a_i = 0$ and $b_i = 1$ then $p_i \leftarrow$ min(1, $p_i + \frac{1}{n}$);
        endif
    endfor
until each $p_i \in \{0, 1\}$
```

Figure 2.7: Pseudocode of the compact GA

is the conditional probability:

$$p(b_i \mid b_j) = \frac{p(b_i, b_j)}{p(b_j)} \tag{2.12}$$

The dependency between two random variables is identified by the Pearson's chi-square statistics which is defined as (n denotes the population size):

$$X^2 = \sum_{b_i, b_j} \frac{(n \cdot p(b_i, b_j) - n \cdot p(b_i) \cdot p(b_j))^2}{n \cdot p(b_i) \cdot p(b_j)} \tag{2.13}$$

With 95% confidence, b_i and b_j are independent if $X^2 < 3.84$. Then a dependency acyclic graph $G = (V, E, R)$ is constructed where V is a set of vertices, E is a set of edges, and R is a set of independent vertices. A vertex one-to-one corresponds to a bit position of an individual. The dependency-graph-construction algorithm is shown in Figure 2.8.

An initial population is random. Next, a dependency graph is constructed. A number of individuals is created by the algorithm shown in Figure 2.9. Some good individuals are preserved for the next generation. The process is iterated until for all $i,\ p(b_i)$ is closer to 0.0 or 1.0 more than $\epsilon,\ \epsilon > 0$. Similar papers that assume the pairwise dependency are the mutual-information-maximizing input clustering (MIMIC) algorithm [13] and dependency trees [5].

```
1.  $V \leftarrow \{0, 1, \ldots, l-1\}$;
    $A \leftarrow V;\ E \leftarrow \emptyset;\ R \leftarrow \emptyset$;
2.  $v \leftarrow$ any member of $A$;
    $R \leftarrow R \cup \{v\}$;
3.  $A \leftarrow A - \{v\}$;
4.  if $(A = \emptyset)$ finish;
5.  if (there is no dependency of $v$ and $v',\ v \in A,\ v' \in V - A$) goto 2;
6.  $v \leftarrow argmax(X^2_{v,v'}),\ v \in A,\ v' \in V - A$;
7.  $E \leftarrow E \cup \{(v, v')\}$;
8.  goto 3;
```

Figure 2.8: Construction of the dependency graph

```
1.  $K \leftarrow V$;
2.  for all $r \in R,\ b_r = a$ according to $p(a),\ a \in \{0, 1\}$;
    $K \leftarrow K - R$;
3.  if $(K = \emptyset)$ finish;
4.  choose $k$ such that $k \in K,\ k' \in V - K,\ (k, k') \in E$;
5.  $b_k \leftarrow a$ according to $p(a \mid b_{k'}),\ a \in \{0, 1\}$;
6.  $K \leftarrow K - \{k\}$;
7.  goto 3;
```

Figure 2.9: Creating a new individual

2.3.3 Extended compact genetic algorithm (ECGA)

The extended compact genetic algorithm (ECGA) was proposed by G. Harik [29]. Table 2.1 shows a nonuniform distribution which represents a population of 4-bit individuals. The {1,2}, value = “00”, and prob. = 0.5 mean that the first and the second bits of an individual, randomly picked up from the population, is “00” with probability 0.5. To build the distribution in Table 2.1, we must know the dependency between the first and the second bit. In the absence of prior information, an alternative distribution should obey the *Occam's Razor*, i.e., the number of bits used to store the distribution is minimal. In addition, the entropy of the alternative distribution should be minimal. The ECGA chooses the distribution which minimizes:

$$\log_2 n \sum_I 2^{S(I)} + n \sum_I E(M_I) \tag{2.14}$$

where n is the population size, I is the partition subset of all bit positions, $S(I)$ is the size of I, and $E(M_I)$ is the entropy of the marginal distribution over the subset I.

$$E(M_I) = \sum_p \begin{cases} -p \log p & ; \text{ if } p \neq 0 \\ 0 & ; \text{ otherwise,} \end{cases} \tag{2.15}$$

where p is the probability in the marginal distribution over subset I. The base of the logarithm in Equation 2.15 is $2^{S(I)}$. The ECGA employs a greedy search. Thus, the optimal distribution is not guaranteed.

Table 2.1: A nonuniform distribution

{1,2}		{3}		{4}	
value	prob.	value	prob.	value	prob.
“00”	0.5	“0”	0.5	“0”	0.6
“01”	0	“1”	0.5	“1”	0.4
“10”	0				
“11”	0.5				

The initial population is random. The ECGA iteratively

1) select some highly-fit individuals

2) approximate the distribution of the highly-fit individuals

3) draw the next population from the distribution.

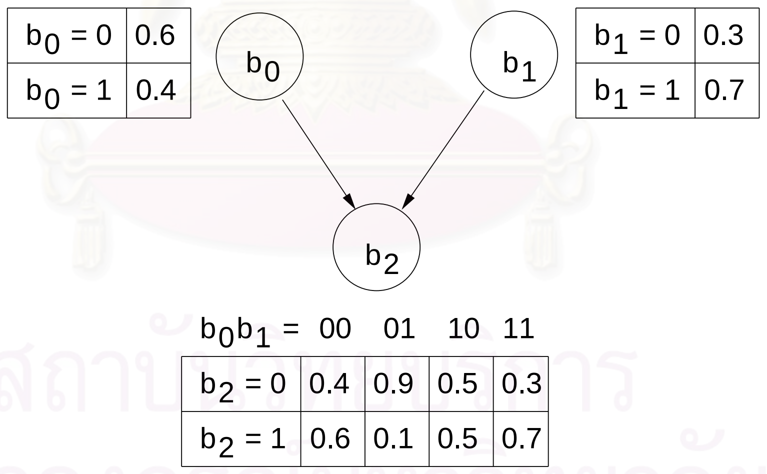
The test function is the 10×4-trap function. The ECGA converges to the partition $\{\{0, 1, 2, 3\}, \ldots, \{36, 37, 38, 39\}\}$. The optimal solution is obtained.

2.3.4 Bayesian optimization algorithm (BOA)

The Bayesian optimization algorithm, proposed by M. Pelikan [55, 60, 63], uses Bayesian network [31, 46] to model a population. Bayesian network is an acyclic directed graph that encodes joint probabilities:

$$p(b_0, \ldots, b_{l-1}) = \prod_{i=0}^{l-1} p(b_i \mid Parent(b_i)) \tag{2.16}$$

where b_i denotes the i^{th}-position bit of an individual, and $Parent(b_i)$ denotes the parent nodes of b_i. A Bayesian network is shown in Figure 2.10. The bit b_2 depends on b_0 and b_1. The tables attached with each node show the probability of being "0" or "1". Given a Bayesian network, we can compute the probability of observing an individual, $\mathcal{I} = b_0 b_1 b_2$. For instance, $p(b_0 b_1 b_2 = \text{“000”}) = p(b_0 = \text{“0”}) \cdot p(b_1 = \text{“0”}) \cdot p(b_2 = \text{“0”} \mid b_0 b_1 = \text{“00”}) = 0.6 \times 0.3 \times 0.4 = 0.072$.



$b_0 = 0$	0.6
$b_0 = 1$	0.4

$b_1 = 0$	0.3
$b_1 = 1$	0.7

$b_0b_1 =$	00	01	10	11
$b_2 = 0$	0.4	0.9	0.5	0.3
$b_2 = 1$	0.6	0.1	0.5	0.7

Figure 2.10: Bayesian network

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

The best network is the network that maximally corresponds to a population. Formally speaking, we search for the network structure that maximizes the scoring metric:

$$p(B|D) = \frac{p(B)}{p(D)} \int_{\theta} p(\theta|B) p(D|B,\theta) d\theta, \tag{2.17}$$

where B is the network structure, D is data set, and θ is the conditional probabilities in the network (i.e., the tables in Figure 2.10). Let the number of incoming edges of any node is limited at k. Only $k = 1$ there is a polynomial-time algorithm that guarantees the optimal network. If $k \geq 2$, the problem becomes NP-hard. Note that the complexity is measured in terms of individual length (l) and population size (n). In practice, a greedy search is used to construct the network. The greedy search begins with an empty network. The operators edge addition, edge removal, and edge reversal are performed by picking the operator which maximizes the scoring metric. The search terminates when no operation can increase the scoring metric. The number of incoming edges corresponds to the number of dependent variables. Pelikan showed that if the problem is composed of k-bit trap functions, the network will be fully connected sets of k nodes (see Figure 2.11).

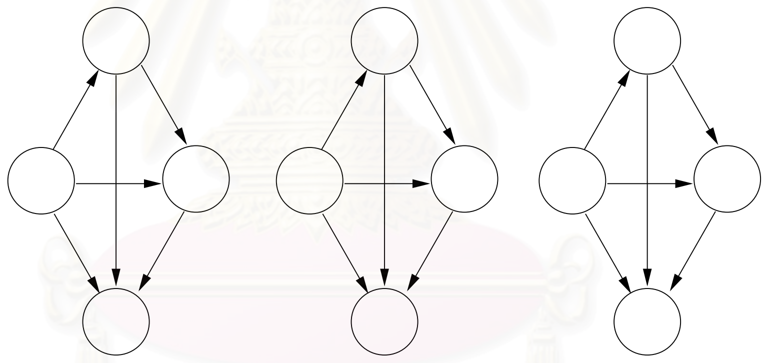


Figure 2.11: The network structure for $m \times 4$-trap functions

The initial population is random, the BOA repeatedly

1) select some highly-fit individuals

2) greedy search for a network that maximizing the scoring metric.

3) draw the next population from the network.

The BOA is more powerful than the ECGA because the Bayesian network can express joint distributions. However, the network complexity is limited by setting the maximum number of incoming edges. In practice, the maximum number of incoming edges is not known beforehand.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

2.3.5 Hierarchical BOA

In the later version of the BOA called hierarchical BOA (hBOA) [62, 63], the ℓ-vertex network is represented by ℓ decision trees/graphs. This is because the number of conditional probabilities in the network grows exponentially with the order of interactions (number of variables that are dependent). Decision trees/graphs are more compact. As a result, the hBOA is applicable for problems having high order of variable interactions, especially, the hierarchical problems presented in Chapter 2. The scoring metric often causes overly complex networks. Therefore the network complexity is limited by the maximum number of incoming edges. A major improvement of the hBOA is that the difficulty of predetermining the maximum number of incoming edges is resolved by adding a penalty term in the scoring metric. The penalty term is the description length of the network parameters. The decision trees/graphs are constructed in the similar fashion to that of the BOA. In summary, the hBOA borrows several well-known techniques to enhance the efficiency of the BOA in order to attack larger problem size. By using an up-to-date PC, the largest problem size that can be solved is approximately 1,000 bits. The problem size that is larger than a thousand of bits is limited due to unreasonable amount of execution time and memory usage.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

We have a habit in writing articles published in scientific journals
to make the work as finished as possible, to cover up all the tracks,
to not worry about the blind alleys
or describe how you had the wrong idea first, and so on.
So there isn't any place to publish, in a dignified manner,
what you actually did in order to get to do the work.
(Richard Feynman, Nobel Lecture, 1966)

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER III

An Observation of the Compact Genetic Algorithm

A relation between the simultaneously changed variables in the compact GA and the building blocks is observed. The pseudocode of the compact GA is outlined in Figure 2.7 [28]. Let $\vec{p} = (p_0, \ldots, p_{\ell-1})$ be an ℓ-dimensional probability vector in the compact GA where $p_i \in [0, 1]$. In the pseudocode, a_i denotes the i^{th}-bit of individual a. Let a *simultaneity matrix* be an $\ell \times \ell$ matrix of numbers. Let m_{ij} be the matrix element in row i and column j, $0 \leq i, j \leq \ell - 1$. Then m_{ij} is the number of times that p_i and p_j are simultaneously updated. The matrix is constructed by the following steps.

```
1. Initialize all matrix elements to zero.
2. loop $MaxIter$ do
   2.1 Randomize a probability vector $\vec{p} = (p_0, \ldots, p_{\ell-1})$.
   2.2 Execute the repeat-until loop in Figure 2.7 at most $MaxGen$ iterations. After
       each iteration, if $p_i$ and $p_j$ $(i \neq j)$ are updated, $m_{ij}$ and $m_{ji}$ will be incremented
       by one.
   endloop
```

The fitness function is set at 5×3-trap function. With $MaxIter = 100$, $MaxGen = 1000$, and $n = 100$, the resulting matrix is shown in Table 3.1. The positions of building blocks (positions of the 3-bit trap functions) are that of $\{\{0, 1, 2\}, \ldots, \{12, 13, 14\}\}$. It is seen that the vector elements governed by the same partition subset are simultaneously updated with higher probability (see the shadowed rectangles in Table 3.1). The partition $\{\{0, 1, 2\}, \ldots, \{12, 13, 14\}\}$ is *valid* for the matrix in Table 3.1. The validity is defined as follows. Let $M = (m_{ij})$ be an $\ell \times \ell$ matrix of numbers, each of whole rows are distinct numbers, $0 \leq i, j \leq \ell - 1$. Let P be a partition of $\{0, \ldots, \ell - 1\}$. Then P is valid for M if for all $B \in P$, for all $b \in B$, the largest $|B| - 1$ elements of M's row b are found in columns of $B \setminus \{b\}$.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

Table 3.1: Simultaneity matrix produced by repeating the compact GA

	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8	Col 9	Col10	Col11	Col12	Col13	Col14
Row 0	0	70220	70451	61129	61841	62405	63493	61560	63968	60455	61065	60472	62699	60534	60272
Row 1	70220	0	70130	61115	62569	61970	63075	62080	61943	61290	60002	61259	63515	60205	61223
Row 2	70451	70130	0	62233	63643	62571	64586	64432	64146	61489	61774	61260	63214	61133	62010
Row 3	61129	61115	62233	0	70999	70172	68228	68722	68782	61817	62222	62241	63219	62016	61715
Row 4	61841	62569	63643	70999	0	71543	68738	68474	68064	63443	63244	62739	65128	62765	62995
Row 5	62405	61970	62571	70172	71543	0	68715	68567	68727	62289	62683	62613	63685	62914	62791
Row 6	63493	63075	64586	68228	68738	68715	0	72764	73739	63571	63877	63976	65485	63230	62969
Row 7	61560	62080	64432	68722	68474	68567	72764	0	73045	63215	62996	63359	64957	62862	62538
Row 8	63968	61943	64146	68782	68064	68727	73739	73045	0	63289	63623	63590	66003	63272	63170
Row 9	60455	61290	61489	61817	63443	62289	63571	63215	63289	0	70259	70527	67390	67794	67619
Row 10	61065	60002	61774	62222	63244	62683	63877	62996	63623	70259	0	70457	67318	67258	67094
Row 11	60472	61259	61260	62241	62739	62613	63976	63359	63590	70527	70457	0	67025	67219	67465
Row 12	62699	63515	63214	63219	65128	63685	65485	64957	66003	67390	67318	67025	0	70316	71092
Row 13	60534	60205	61133	62016	62765	62914	63230	62862	63272	67794	67258	67219	70316	0	70832
Row 14	60272	61223	62010	61715	62995	62791	62969	62538	63170	67619	67094	67465	71092	70832	0

For example, $\{\{0,1,2\},\{3,4,5\},\{6,7,8\},\{9,10,11\},\{12,13,14\}\}$ is valid for Table 3.1. Consequently, the largest two elements in row 0 are found in columns of $\{1,2\}$ and the largest two elements in row 1 are found in columns of $\{0,2\}$ and the largest two elements in row 2 are found in columns of $\{0,1\}$ and so on. The three bits of each 3-bit trap function may not be packed close together. If each bit in the trap function is placed far away from the others, a valid partition could be $\{\{0,5,10\}$, $\{1,6,11\}$, $\{2,7,12\}$, $\{3,8,13\}$, $\{4,9,14\}\}$. The matrix might be hard to read. The next paragraph will explain why the vector elements governed by the same partition subset are simultaneously updated with higher probability.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

Every iteration of the compact GA, individual a competes with b. If $a_i \neq b_i$ and $a_j \neq b_j$, the matrix element a_{ij} will be incremented by one. All possible values of a_i, a_j, b_i, b_j that cause the increment are listed as follows.

$$a_ia_j = 00 \;\; b_ib_j = 11, \qquad a_ia_j = 11 \;\; b_ib_j = 00,$$

$$a_ia_j = 01 \;\; b_ib_j = 10, \qquad a_ia_j = 10 \;\; b_ib_j = 01.$$

The fitness of individuals a and b are improved every iteration. In the later iterations, the fitness of a and b is higher. Roughly speaking, the simultaneity matrix records any pair of 2-bits that are complement to each other between two highly-fit individuals drawn at random. The highly-fit individuals of the $m{\times}3$-trap functions are composed of triple zeroes and its complement, triple ones. Thus, we observe the simultaneous change between p_i and p_j if the bit positions i and j are governed by the same 3-bit trap function. All cases for mixing 2-bit building blocks are enumerated. Mixing 00 with 11 results in 01 and 10. Mixing 01 with 10 results in 00 and 11. Only mixing in these cases must be done carefully because the building blocks will be disrupted. Mixing building blocks in the other cases gives the same building blocks. As a result, it is reasonable to reward a pair of 2-bits that are complement to each other.

Though the matrix records the relation between two bits, it is possible to recognize multiple-bit building blocks. We shall begin with 2-bit building blocks. If the matrix element m_{ij} is significantly high, the bits at positions of $\{i, j\}$ should be passed together. The 3-bit building blocks are recognized by inserting k to $\{i, j\}$. If the matrix elements m_{ij}, m_{jk}, and m_{ik} are significantly high, i, j, k should be in the same partition subset. Longer building blocks can be recognized in the similar fashion. The observation of the compact GA is the starting point of the thesis. By explanation and generalization of what we observe, the building blocks can be identified by constructing the simultaneity matrix and finding a partition for the matrix.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

When we write programs that “learn,” it turns out we do and they don’t.

(Anonymous)

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER IV

The Algorithm

Our algorithm named building-block identification by simultaneity matrix (BISM) consists of two parts: simultaneity-matrix-construction (SMC) and partitioning (PAR) algorithms. The SMC input is a set of ℓ-bit binary string. The SMC outputs an $\ell \times \ell$ matrix of numbers. Next, PAR searches for a partition of $\{0, \ldots, \ell-1\}$ for the matrix.

4.1 Simultaneity-Matrix-Construction (SMC) Algorithm

SMC input is a set of ℓ-bit binary string denoted by:

$$S = \{s_0, \ldots, s_{n-1}\} \tag{4.1}$$

where s_i is the i^{th} string, $0 \leq i \leq n-1$. The s_i^j denotes the j^{th} bit of s_i, $0 \leq j \leq \ell - 1$. Algorithm SMC outputs an $\ell \times \ell$ symmetric matrix of numbers, denoted by $M = (m_{ij}),\ 0 \leq i, j \leq \ell - 1$. A closed form of m_{ij} is shown in Equation 4.2.

$$m_{ij} = \begin{cases} 0 & ;\ \text{if } i = j \\ \text{Count}_S^{00}(i,j) \times \text{Count}_S^{11}(i,j) + \text{Count}_S^{01}(i,j) \times \text{Count}_S^{10}(i,j) & ;\ \text{otherwise} \end{cases} \tag{4.2}$$

where $\text{Count}_S^{ab}(i,j) = |\{x \in \{0, \ldots, n-1\} : s_x^i = a \text{ and } s_x^j = b\}|$ for all $0 \leq i, j \leq \ell - 1,\ (a,b) \in \{0,1\}^2$.

Algorithm SMC is shown in Figure 4.1. Step 1 constructs only the upper triangle of the matrix using Equation 4.2. Step 2 copies the upper triangle $\{m_{ij} \mid i < j\}$ to the lower triangle $\{m_{ij} \mid i > j\}$. Step 3 returns the simultaneity matrix $M = (m_{ij})$. The time complexity of SMC is $O(\ell^2 n)$. In practice, the matrix is perturbed so that there are no identical elements. The matrix of which the elements are distinct is greatly helpful in partitioning. The perturbation techniques can be varied. For instance, we add a small real random number (ranging between 0 and 1) to the matrix elements. The perturbation does not totally change the matrix because the real part is small. The perturbation by adding an integer with a real number is practical for a random number generator with a sufficiently large period because it is hardly possible to produce identical random numbers.

```
Algorithm SMC($S$)
1.  for $i = 0$ to $\ell - 1$ do
        $m_{ii} \leftarrow 0$;
        for $j = i + 1$ to $\ell - 1$ do
            $m_{ij} \leftarrow \text{Count}_S^{00}(i, j) \times \text{Count}_S^{11}(i, j) + \text{Count}_S^{01}(i, j) \times \text{Count}_S^{10}(i, j)$;
2.  for $i = 0$ to $\ell - 1$ do
        for $j = i + 1$ to $\ell - 1$ do
            $m_{ji} \leftarrow m_{ij}$;
3.  return $M \leftarrow (m_{ij})$;
```

Figure 4.1: Simultaneity-Matrix-Construction (SMC) algorithm

A matrix element m_{ij} is proportional to the probability that 2-bit building blocks at bit positions i and j will be disrupted by the uniform crossover. All cases for mixing 2-bit building blocks are enumerated. Mixing "00" with "11" results in "01" and "10." Mixing "01" with "10" results in "00" and "11." Only mixing in the two cases must be done carefully because the processing building blocks will be lost. Mixing 2-bit building blocks in the other cases gives the same building blocks. Therefore SMC algorithm counts a pair of 2-bit building blocks that are complementary to each other. To exploit the matrix, the bits at positions i and j are passed together every time performing crossover if the matrix element m_{ij} is significantly high. The 3-bit building blocks are identified by inserting k to $\{i, j\}$. If the matrix elements m_{ij}, m_{jk}, and m_{ik} are significantly high, i, j, k should be in the same partition subset. Larger building blocks can be identified in a similar fashion.

The trap functions embedded in the HDFs bias the population to two aligned chunks of zeroes and ones, that are complementary to each other. Certainly, the dependency between every pair of bits in a chunk is stored in the matrix. The matrix is not limited to the cases where the two aligned chunks are complementary to each other. In the other cases, the matrix does not detect unnecessary dependency. For instance, the bits at positions of

$\{0, 1, 2, 3, 4\}$ are mostly "b_0b_1000" and "b_0b_1111" where $b_i \in \{0, 1\}$. The dependency among five bits is obvious, but passing the bits governed by $\{2, 3, 4\}$ together it is sufficient to guarantee that "b_0b_1000" and "b_0b_1111" will exist in the next generation with a high probability. In summary, the matrix records only dependency that is actually necessary for preserving building blocks.

4.2 Partitioning (PAR) Algorithm

The PAR input is an $\ell \times \ell$ simultaneity matrix. The PAR outputs the partition:

$$P = \{B_0, \ldots, B_{|P|-1}\}, \ \bigcup_{i=0}^{|P|-1} B_i = \{0, \ldots, \ell - 1\}, \ B_i \cap B_j = \emptyset \text{ for all } i \neq j \quad (4.3)$$

To exploit the simultaneity matrix presented in the previous chapter, we have to develop a partitioning algorithm that takes $\ell \times \ell$ matrix and gives a valid partition of $\{0, \ldots, \ell - 1\}$. The motivation is to put i and j into the same partition subset if m_{ij} is high. The number of ways to divide n objects into m non-empty subsets is called the *Sterling number of the second kind* [25]. The Sterling number of the second kind can be calculated from the recursion:

$$S(n, m) = mS(n - 1, m) + S(n - 1, m - 1) \quad (4.4)$$

$$S(n, m) = \frac{1}{m!} \sum_{k=0}^{m} \binom{m}{k} k^n (-1)^{m-k} \quad (4.5)$$

The number of possible partitions is referred to as *Bell number*. It is the sum of $S(n, m)$ over m. The Bell number grows exponentially. The brute-force technique is impossible even for small problem size.

Partitioning algorithms have been extensively used in VLSI design [2]. The VLSI partitioning problem consists of ℓ ICs and the number of wires connected between each pair of the components. The goal is to search for a partition of components that minimizes *mincut* – the number of wires between partition subsets. A pair of components that are connected together with many wires is likely to be in the same partition subsets (in order to minimize mincut). At the first glance, the VLSI partitioning algorithms may be applied to our problem. An $\ell \times \ell$ matrix can be converted to ℓ components where m_{ij} is the number of wires connected between components i and j. In the VLSI partitioning problems the number of partition subsets is given beforehand. In contrast, partitioning involves inference or induction. We turn to another approach for partitioning. The definition of the desired partition is developed. Next, we design the algorithm that search for the desired partition.

There are several definitions of the desired partition, for example, the definitions in the senses of non-monotonicity [50], GEMGA [38], Walsh coefficients [41], and entropy measurement [29]. We develop a definition in the sense of simultaneity matrix. Algorithm PAR searches for the partition P that satisfies the following conditions.

1. P is a partition.
   1.1 The members of P are disjoint set.
   1.2 The union of all members of P is $\{0, \ldots, \ell - 1\}$.
2. $P \neq \{\{0, \ldots, \ell - 1\}\}$.
3. For all $B \in P$ such that $|B| > 1$,
   3.1 for all $i \in B$, the largest $|B| - 1$ matrix elements in row i
      are founded in columns of $B \setminus \{i\}$.
4. For all $B \in P$ such that $|B| > 1$,
   4.1 $H_{max} - H_{min} < \alpha(H_{max} - L_{min})$ where $0 \leq \alpha \leq 1$,
      $H_{max} = max(\{m_{ij} \mid (i, j) \in B \times B,\ i \neq j\})$,
      $H_{min} = min(\{m_{ij} \mid (i, j) \in B \times B,\ i \neq j\})$, and
      $L_{min} = min(\{m_{ij} \mid i \in B,\ j \in \{0, \ldots, \ell - 1\} \setminus B\})$.
5. There are no partition P_x such that for some $B \in P$, for some $B_x \in P_x$,
   P and P_x satisfy the first, the second, the third, and the fourth conditions,
   $B \subset B_x$.

An example of the simultaneity matrix is shown in Figure 4.1. The perturbation is omitted because the values of $\{m_{ij} \mid i < j\}$ are distinct. The first condition is obvious. The second condition does not allow the coarsest partition because it is not useful in solution recombination. The third condition makes i and j, in which m_{ij} is significantly high, in the same partition subset. For instance, $P_1 = \{\{0,1,2\}$, $\{3,4,5\}$, $\{6,7,8\}$, $\{9,10,11\}$, $\{12,13,14\}\}$ satisfies the third condition because the largest two elements in row 0 are found in columns of $\{1,2\}$, the largest two elements in row 1 are found in columns of $\{0,2\}$, the largest two elements in row 2 are found in columns of $\{0,1\}$, and so on. However, there are many partitions that satisfy the third condition, for example, $P_2 = \{\{0,1,2\},\{3,4,5,6,7,8\},\{9,10,11\},\{12,13,14\}\}$. There is a dilemma between choosing the fine partition (P_1) and the coarse partition (P_2). Choosing the fine partition prevents the emergence of large building blocks, while the coarse partition results in poor mixing. To overcome the dilemma, the coarse partition will be acceptable if it satisfies the fourth condition. The fifth condition says choosing the coarsest partition that is consistent with the first, the second, the third, and the fourth conditions.

By condition 4.1, the partition subset $\{3,4,5\}$ is acceptable because the values of matrix elements governed by $\{3,4,5\}$ are close together (see Fi gure 4.1). Being close together is defined by $H_{max} - H_{min}$ where H_{max} and H_{min} is the maximum and the minimum of the nondiagonal matrix elements governed by a partition subset. The $H_{max}-H_{min}$ is a degree of irregularities of the matrix. The main idea is to limit $H_{max} - H_{min}$ to a threshold. The threshold, $\alpha(H_{max} - L_{min})$, is defined relatively to the matrix elements because the threshold cannot be fixed for a problem instance. The partition subset $\{3,4,5\}$ gives $H_{max} = 71543$, $H_{min} = 70172$, and $L_{min} = 61115$. L_{min} is the minimum of the nondiagonal matrix elements in rows of $\{3,4,5\}$. The fourth condition limits $H_{max} - H_{min}$ to $100 \times \alpha$ percent of the difference between H_{max} and L_{min}. An empirical study showed that α should be set at 0.75 for both ADFs and HDFs. Choosing $\{3,4,5,6,7,8\}$ yields ($H_{max} = 73739, H_{min} = 68064, L_{min} = 61115$) which does not violate condition 4.1. The fifth condition prefers a coarse partition $\{\{3,4,5,6,7,8\},\ldots\}$ to a fine partition $\{\{3,4,5\},\ldots\}$ so that the partition subsets can be grown to compose larger building blocks in higher levels.

elements governed by {3, 4, 5}

elements governed by {3, 4, 5, 6, 7, 8}

	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8	Col 9	Col10	Col11	Col12	Col13	Col14
Row 0	0	70220	70451	61129	61841	62405	63493	61560	63968	60455	61065	60472	62699	60534	60272
Row 1	70220	0	70130	61115	62569	61972	63075	62080	61943	61290	60002	61259	63515	60205	61223
Row 2	70451	70130	0	62233	63643	62571	64586	64432	64146	61489	61774	61260	63214	61133	62010
Row 3	61129	61115	62233	0	70999	70172	68228	68722	68782	61817	62222	62241	63219	62016	61715
Row 4	61841	62569	63643	70999	0	71543	68738	68474	68064	63443	63244	62739	65128	62765	62995
Row 5	62405	61972	62571	70172	71543	0	68715	68567	68727	62289	62683	62613	63685	62914	62791
Row 6	63493	63075	64586	68228	68738	68715	0	72764	73739	63571	63877	63976	65485	63230	62969
Row 7	61560	62080	64432	68722	68474	68567	72764	0	73045	63215	62996	63359	64957	62862	62538
Row 8	63968	61943	64146	68782	68064	68727	73739	73045	0	63289	63623	63590	66003	63272	63170
Row 9	60455	61290	61489	61817	63443	62289	63571	63215	63289	0	70259	70527	62390	62794	62619
Row 10	61065	60002	61774	62222	63244	62683	63877	62996	63623	70259	0	70457	61318	63258	61094
Row 11	60472	61259	61260	62241	62739	62613	63976	63359	63590	70527	70457	0	63025	61219	63465
Row 12	62699	63515	63214	63219	65128	63685	65485	64957	66003	62390	61318	63025	0	70316	71092
Row 13	60534	60205	61133	62016	62765	62914	63230	62862	63272	62794	63258	61219	70316	0	70832
Row 14	60272	61223	62010	61715	62995	62791	62969	62538	63170	62619	61094	63465	71092	70832	0

Table 4.1: Simultaneity matrix produced by SMC algorithm

Algorithm PAR is shown in Figure 4.2. A trace of the algorithm is shown in Table 4.2. The outer loop processes row 0 to $\ell - 1$. In the first step, the columns of the sorted values in row i are stored in $R_{i,0}$ to $R_{i,\ell-1}$. For $i = 0$, array $R_{i,0}$ to $R_{i,\ell-1} = \{2,1,8,6,12,5,4,7,3,10,13,11,9,14,0\}$. Next, the inner loop tries a number of partition subsets by enlarging A ($A \leftarrow A \cup \{R_{i,j}\}$). If A satisfies conditions 3.1 and 4.1, A will be saved to B. Finally, P is the partition that satisfies the five conditions. Checking conditions 3.1 and 4.1 is the most time-consuming section. It can be done in $O(\ell^2)$. The checking is done at most ℓ^2 times. Therefore the time complexity of PAR is $O(\ell^4)$.

4.3 Correctness Proofs

The following proofs show that the PAR algorithm always returns the partition that satisfies the five conditions. In addition, the partition that satisfies the five conditions is unique.

```
Note: $M = (m_{ij})$ denotes $\ell \times \ell$ simultaneity matrix, $0 \leq i, j \leq \ell - 1$.
$T_i$ and $R_{i,j}$ denote arrays of numbers indexed by $0 \leq i, j \leq \ell - 1$.
$A$ and $B$ are partition subsets. $P$ denotes a partition.
Algorithm PAR($M, \alpha$)
$P \leftarrow \emptyset$;
for $i = 0$ to $\ell - 1$ do    // outer loop processing row $i$
    if $i \notin B$ for all $B \in P$ then
        $T \leftarrow \{$matrix elements in row $i$ sorted in descending order$\}$;
        for $j = 0$ to $\ell - 1$ do $R_{i,j} \leftarrow x$ where $m_{ix} = T_j$;
        $A \leftarrow \{i\}$; $B \leftarrow \{i\}$;
        for $j = 0$ to $\ell - 3$ do    // inner loop enlarging $A$
            $A \leftarrow A \cup \{R_{i,j}\}$;
            if $A$ satisfies conditions 3.1 and 4.1 then
                $B \leftarrow A$;
        endfor
        $P \leftarrow P \cup \{B\}$;
    endif
endfor
return $P$;
```

Figure 4.2: The PAR algorithm

Lemma 1: For any two partition subsets B_1 and B_2 that satisfy condition 3.1 and $B_1 \cap B_2 \neq \emptyset$, one must be the subset of the other.

Proof: Let r be in $B_1 \cap B_2$. Since B_1 satisfies condition 3.1, B_1 completes all $n_1 - 1$ columns containing the maximum values in row r if n_1 is the number of elements of B_1. Similarly for B_2, B_2 completes all $n_2 - 1$ columns containing the maximum values in row r if n_2 is the number of elements of B_2. If $n_1 \leq n_2$ then $B_1 \subseteq B_2$, otherwise $B_2 \subset B_1$. This completes the proof. ■

Lemma 2: Let i, j be integers, $0 \leq i < j \leq n - 2$. In row r, $0 \leq r \leq \ell - 1$, if $R_{r,j} \in$ partition subset B containing r and B satisfies condition 3.1 then $R_{r,i} \in B$.

Table 4.2: A trace of the PAR algorithm

i	j	B_1	3^{rd} cond.	4^{th} cond.	B_2
0	0	{0, 2}	True	True	{0, 2}
0	1	{0, 2, 1}	True	True	{0, 1, 2}
0	2	{0, 2, 1, 8}	False	False	{0, 1, 2}
0	3	{0, 2, 1, 8, 6}	False	False	{0, 1, 2}
0	4	{0, 2, 1, 8, 6, 12}	False	False	{0, 1, 2}
0	5	{0, 2, 1, 8, 6, 12, 5}	False	False	{0, 1, 2}
0	6	{0, 2, 1, 8, 6, 12, 5, 4}	False	False	{0, 1, 2}
0	7	{0, 2, 1, 8, 6, 12, 5, 4, 7}	False	False	{0, 1, 2}
0	8	{0, 2, 1, 8, 6, 12, 5, 4, 7, 3}	False	False	{0, 1, 2}
0	9	{0, 2, 1, 8, 6, 12, 5, 4, 7, 3, 10}	False	False	{0, 1, 2}
0	10	{0, 2, 1, 8, 6, 12, 5, 4, 7, 3, 10, 13}	False	False	{0, 1, 2}
0	11	{0, 2, 1, 8, 6, 12, 5, 4, 7, 3, 10, 13, 11}	False	False	{0, 1, 2}
0	12	{0, 2, 1, 8, 6, 12, 5, 4, 7, 3, 10, 13, 11, 9}	False	False	{0, 1, 2}

Proof: Let $c_i = R_{r,i}$ and $c_j = R_{r,j}$. Since $i < j$, we have matrix elements $m_{rc_i} > m_{rc_j}$. Suppose that $c_j \in B$ containing r, and $c_i \notin B$. Then $B = \{r, c_j, x_0, \ldots, x_{k-1}\}$. According to condition 3.1, $k + 1$ maximum values in row r are in columns $c_j, x_0, \ldots, x_{k-1}$. But $m_{rc_i} > m_{rc_j}$, and $c_i \notin \{c_j, x_0, \ldots, x_{k-1}\}$. Thus B does not satisfy condition 3.1. This contradicts the hypothesis that B satisfies condition 3.1. This completes the proof. ■

Lemma 3: Let $M = (m_{ij})$ be an $\ell \times \ell$ simultaneity matrix satisfying the following conditions.

1) m_{ij} is a positive real number for all $0 \leq i, j \leq \ell - 1$.

2) m_{ii} = 0 for all $0 \leq i \leq \ell - 1$.

3) $m_{ij} \neq 0$, $m_{ij} = m_{ji}$ for all $0 \leq i < j \leq \ell - 1$.

4) m_{ij} are distinct for all $0 \leq i < j \leq \ell - 1$.

There exists a partition P which satisfies the five conditions.

Proof: It is obvious that $P = \{\{0\}, \ldots, \{\ell - 1\}\}$ satisfies the first, the second, the third, and the fourth conditions. If the partition P_x in the fifth condition does not exist, P satisfies the five conditions. Otherwise P_x satisfies the five conditions. This completes the proof. ■

Theorem 1: The problem of finding the partition P that satisfies the five conditions can be solved by the PAR algorithm.

Proof: Lemma 3 guarantees that P does exist. Next, it will be shown that PAR returns the partition P which satisfies the five conditions.

Condition 1.1) The members of P are disjoint sets. Let B_1 and B_2 be the members of P. The B_1 and B_2 are produced when PAR processes rows r_1 and r_2, respectively ($r_1 < r_2$). B_1 and B_2 can be written as follows.

$$B_1 = \{r_1, x_0, \ldots, x_{p-1}\},\ \text{for all } 0 \leq i \leq p-1,\ r_1 \neq x_i \tag{4.6}$$

$$B_2 = \{r_2, y_0, \ldots, y_{q-1}\},\ \text{for all } 0 \leq i \leq q-1,\ r_2 \neq y_i \tag{4.7}$$

The proof is separated in four cases.

Case 1: Suppose $r_1 = r_2$. Algorithm PAR processes row r_2 if $r_2 \notin B$ for all $B \in P$. Hence, $r_1 \neq r_2$.

Case 2: Suppose for some $0 \leq i \leq q-1$, $r_1 = y_i$. Then r_1 and r_2 become the members of the same partition subset B_2. If r_2 lies between $R_{r_1,0}$ and $R_{r_1,\ell-3}$, obviously r_1 and r_2 will be members of B_1. Subsequently, B_2 is not encountered because the row r_2 is not processed. In the case of $r_2 = R_{r_1,\ell-2}$, Lemma 2 said if r_2 is a member of partition subset B containing r_1, $R_{r_1,j} \in B$ for all $0 \leq j \leq \ell-3$ is a necessary condition to make B satisfying condition 3.1. As a result, B has ℓ elements. By the computation of PAR, any partition subset has at most $\ell - 1$ elements.

Case 3: Suppose for some $0 \leq i \leq p-1$, $x_i = r_2$. Algorithm PAR processes row

r_2 if $r_2 \notin B$ for all $B \in P$. Hence, $x_i \neq r_2$ for all $0 \leq i \leq p-1$.

Case 4: Suppose for some $0 \leq i \leq p-1$, for some $0 \leq j \leq q-1$, $x_i = y_j$. Let $i = j = 0$ and $r_C = x_0 = y_0$. According to the condition 3.1,

Case (i) the p maximum values in row r_C are in columns $r_1, x_1, \ldots, x_{p-1}$

Case (ii) the q maximum values in row r_C are in columns $r_2, y_1, \ldots, y_{q-1}$.

In all cases, (i) contradicts (ii).

Condition 1.2) The union of all members of P is $\{0, \ldots, \ell-1\}$. It is clear that, by the computation of PAR, P contains all rows.

Condition 2) $P \neq \{\{0, \ldots, \ell-1\}\}$. This is true since any partition subset has at most $\ell - 1$ elements.

Condition 3) and 4) Algorithm PAR puts only partition subsets that satisfy conditions 3.1 and 4.1 into P. As a result, P satisfies the third and the fourth conditions.

Condition 5) There are no partition P_x such that for some $B \in P$, for some $B_x \in P_x$, P and P_x satisfy the first, the second, the third, and the fourth conditions, $B \subset B_x$. Suppose $B \subset B_x$.

$$B = \{r, x_0, \ldots, x_{p-1}\} \tag{4.8}$$

$$B_x = \{r, x_0, \ldots, x_{p-1}, y_0, \ldots, y_{q-1}\} \tag{4.9}$$

Algorithm PAR produces B when processing row r. Since B_x satisfies condition 3.1, the first $p+q$ elements of $R_{r,0}$ to $R_{r,\ell-1}$ contains $x_0, \ldots, x_{p-1}, y_0, \ldots, y_{q-1}$. If $p+q \leq \ell-2$, obviously B_x will be encountered at the time processing row r. If $p+q = \ell-1$, B_x has ℓ elements, but by the computation of PAR any partition subset has at most $\ell-1$ elements. This completes the proof. ■

Theorem 2: The partition P that satisfies the five conditions is unique.

Proof: Suppose P and Q are partitions that satisfy the five conditions. Since $P \neq Q$, there must be a nonempty set $\phi = \{(B_P, B_Q) \mid B_P \in P,\ B_Q \in Q,\ B_P \neq B_Q,\ B_P \cap B_Q \neq \emptyset\}$. By Lemma 1, if $B_P \cap B_Q \neq \emptyset$ then $B_P \subset B_Q$ or $B_Q \subset B_P$. The proof can be separated in two cases.

Case 1: for some $(B_P, B_Q) \in \phi$, $B_P \subset B_Q$.

Therefore P does not satisfy the fifth condition.

Case 2: for some $(B_P, B_Q) \in \phi$, $B_Q \subset B_P$.

Therefore Q does not satisfy the fifth condition.

All cases contradict to the hypothesis. This completes the proof. ∎

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

A year spent in artificial intelligence is enough to make one believe in God.

(Anonymous)

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER V
Performance Comparisons

5.1 Methodology

Most papers report the performance in terms of function evaluations required to reach the optimum. Such a performance measurement is affected by selection method, solution recombination, and the other factors. At present, research community does not provide a formal framework for measuring the effectiveness of a building-block identification algorithm regardless of the other factors we have mentioned. Inevitably, we have to make a comparison in terms of function evaluations. We have presented the building-block identification by simultaneity matrix (BISM). An optimization algorithm that exploits the BISM is needed. We customize simple GAs as follows. Every generation, the simultaneity matrix is constructed. The PAR algorithm is executed to find a partition. Two parents are chosen by the roulette-wheel method. The solutions are reproduced by a restricted uniform crossover – bits governed by the same partition subset must be passed together. The mutation is turned off. The diversity is maintained by the rank-space method [80, pp. 520–523]. The population size is determined empirically by the bisection method [63, pp. 64]. The bisection method performs binary search for the minimal population size. There might be 10% different between the population size used in the experiments and the minimal population size that ensures the optimal solution in all independent 10 runs.

5.2 A Visualization of the Simultaneity Matrix

To illustrate how the matrix changes over time, a matrix element is represented by a square. The square intensity is proportional to the value of matrix element (see Figure 5.1). In the early generation (A), the matrix elements are nearly identical because the initial population is generated at random. After that (B), the matrix elements become more distinct. The solution recombination is more speculative. Multiple bits are passed together, and therefore forming larger building blocks. Finally (C), the building blocks are completely detected. The mixed trap function is additively composed of 5-bit onemax, 3-trap, 4-trap, 5-trap, 6-trap, and 7-trap functions. Note that the onemax function counts the number of “1” bits.

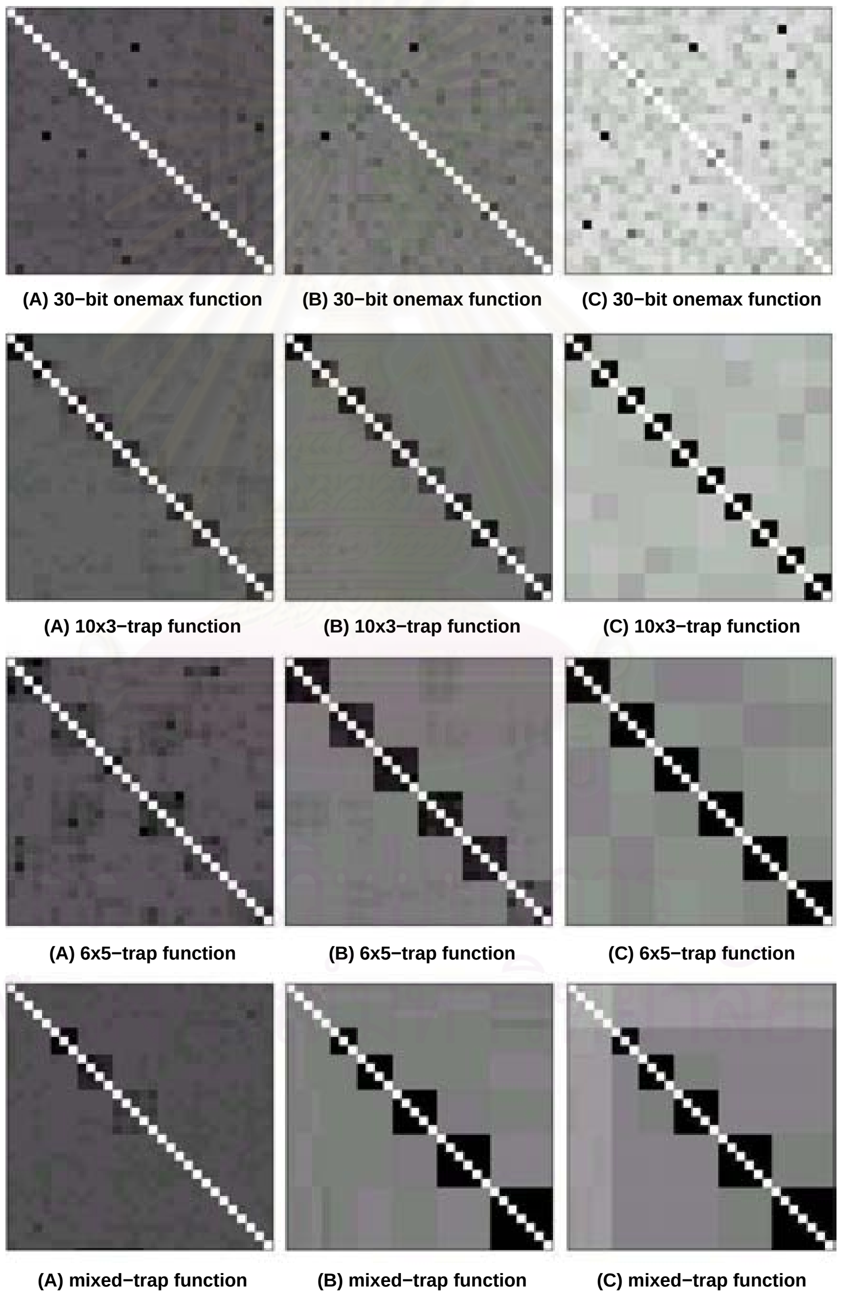


Figure 5.1: Matrix adaptation (additively decomposable functions)

For hierarchical decomposable functions, the matrix is shown in Figure 5.2. In the early generation (A), the matrix elements are nearly identical because the initial population is generated at random. After that (B), the matrix elements become more distinct. The building blocks in the lowest level are detected. The solution recombination is more speculative. Multiple bits are passed together, and therefore forming larger building blocks. A few generations later (C), the building blocks in higher levels are revealed. Finally (D), the population begins to lose diversity. The matrix elements are going to be identical. Note that the bits governed by the same building blocks do not need to be packed close together. It is done for the ease of presentation.

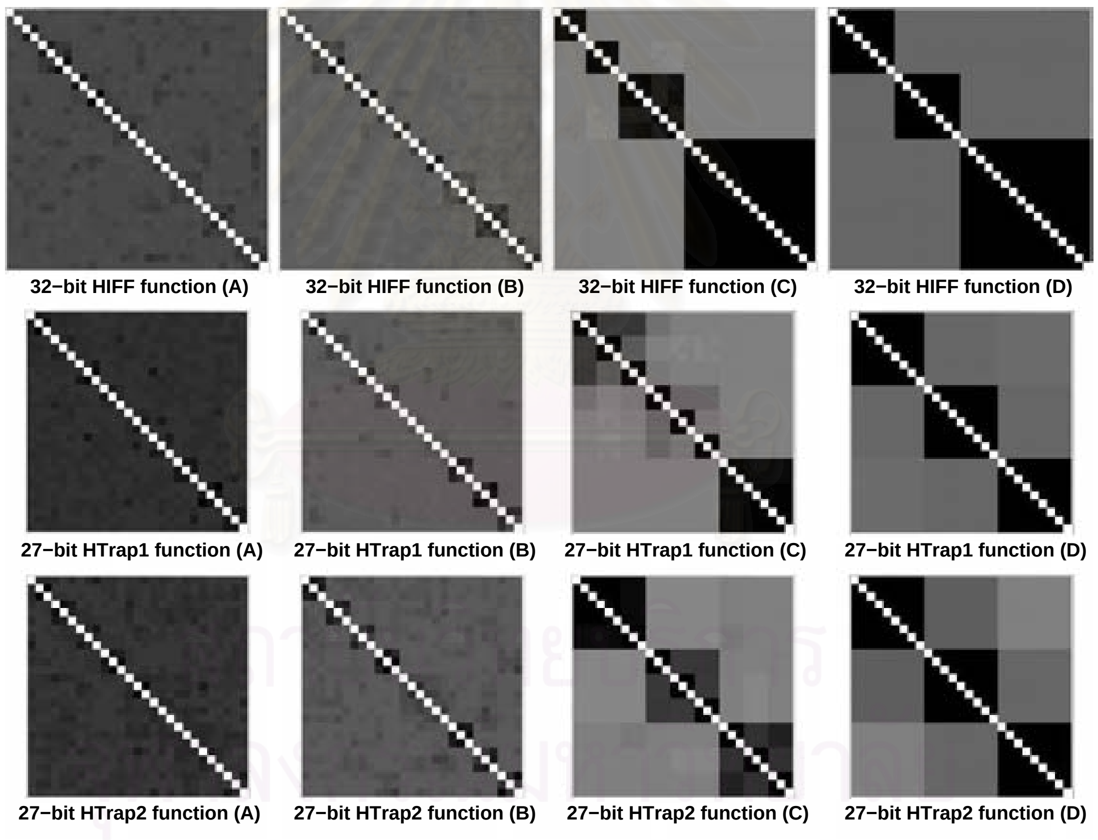


Figure 5.2: Matrix adaptation (hierarchically decomposable functions)

5.3 A Comparison to the BOA

Our algorithm is compared to the BOA [63, pp. 115–117]. Figure 5.3–5.5 shows the number of function evaluations required to reach the optimal solution. The linear regression in log-scale indicates a polynomial relationship between the number of function evaluations and the problem size. The degree of polynomial can be approximated by the slope of linear regression. It can be seen that the BOA and BISM can solve ADFs in a polynomial time. The BOA performs slightly better than BISM. However, the performance gap narrows as the problem becomes harder (onemax, $m{\times}3$-trap, and $m{\times}5$-trap, respectively).

We make another comparison in terms of elapsed time. The elapsed time is an execution time of a call on subroutine `constructTheNetwork` [59]. The hardware platform is HP NetServer E800, 1GHz Pentium-III, 2GB RAM, and RedHat 8.0 OS. The parameters of the BOA are set at default. The maximum number of incoming edges, a parameter of the BOA [59], limits the number of incoming edges for every vertices in the Bayesian network. The default setting is to set the number of incoming edges to $k-1$ for $m{\times}k$-trap functions. In the absence of prior information, k is not known beforehand. Figure 5.6 shows that the elapsed time required to construct the network increases with the maximum number of incoming edges, but the computational time of the matrix is fixed for a problem size. The difficulty of predetermining the maximum number of incoming edges is resolved in the later version of the BOA, called the hierarchical BOA [61, 63].

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

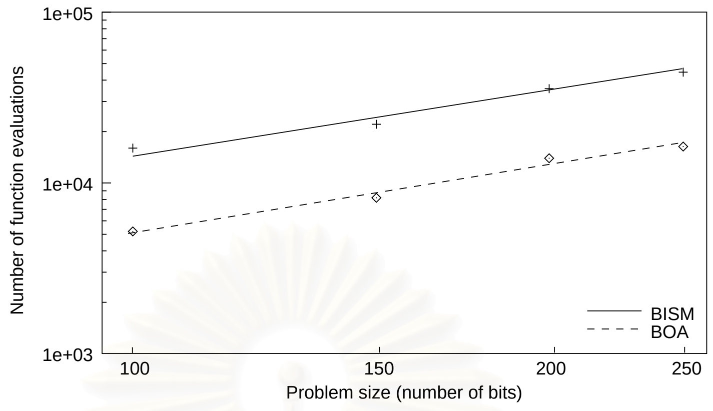


Figure 5.3: Performance comparison between the BOA and BISM (onemax functions)

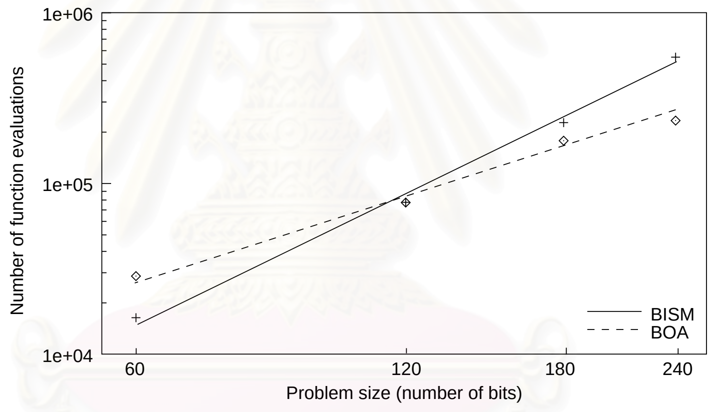


Figure 5.4: Performance comparison between the BOA and BISM ($m{\times}3$ functions)

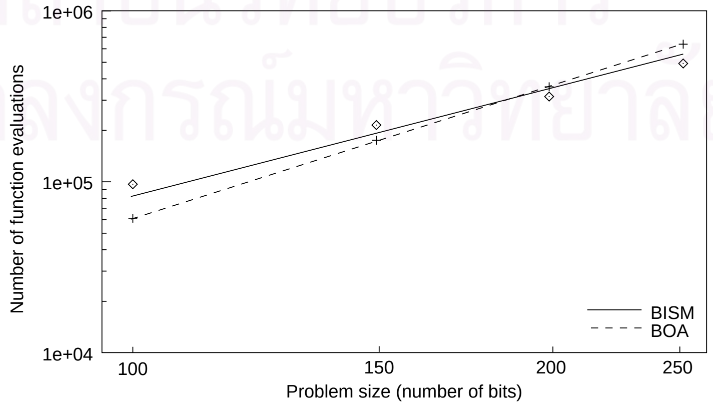


Figure 5.5: Performance comparison between the BOA and BISM ($m{\times}5$ functions)

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

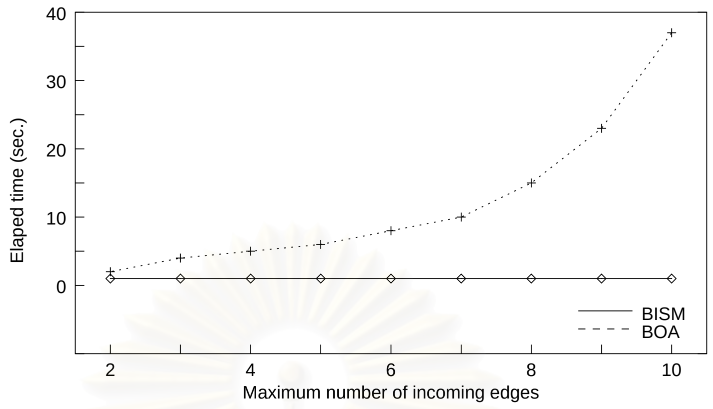


Figure 5.6: Elapsed time required to construct Bayesian network (in BOA) and matrix

5.4 A Comparison to the hBOA

Our algorithm is compared to the hBOA [63, pp. 164–165]. Figure 5.7–5.9 shows the number of function evaluations required to reach the optimal solution. The linear regression in log-scale indicates a polynomial relationship between the number of function evaluations and the problem size. The degree of polynomial can be approximated by the slope of linear regression. It can be seen that the hBOA and BISM can solve HDFs in a polynomial time. The hBOA performs slightly better than BISM. However, the performance gap narrows as the problem becomes harder (HIFF, HTrap1, and HTrap2, respectively).

We make another comparison in terms of elapsed time and memory usage. The elapsed time is an execution time of a call on subroutine `constructTheNetwork` [61]. The memory usage is the number of bytes dynamically allocated in the subroutine. The hardware platform is HP NetServer E800, 1GHz Pentium-III, 2GB RAM, and Windows XP. The memory usage in the hBOA is very large because of inefficient memory management in constructing Bayesian network. A memory-efficient implementation of Bayesian network is the WinMine Toolkit [10]. The WinMine is a set of tools that allow you to build statistical models from data. It constructs Bayesian network with decision tree that is similar to that of the hBOA. The WinMine's elapsed time and memory usage are measured by an execution of `dnet.exe` – a part of the WinMine that constructs the network. All experiments are done with the same biased population that is composed of aligned chunks of zeroes and ones. The parameters of the hBOA and WinMine Toolkit are set at default. The population size is set at three times greater than the problem size.

The elapsed time and memory usage are shown in Figure 5.10–5.11. Bayesian network is a powerful tool that builds a statistical model from data. However, constructing the network uses more time and more memory than computing the simultaneity matrix. This is because the network gathers all statistical dependency between bit variables. In contrast, the matrix records only dependency between two bits that are likely to be disrupted in the uniform crossover. Therefore the matrix computation is much faster. The empirical results show that BISM attains comparable number of function evaluations to that of the hBOA, but computing the matrix is 10 times faster and uses 10 times less memory than constructing Bayesian network.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

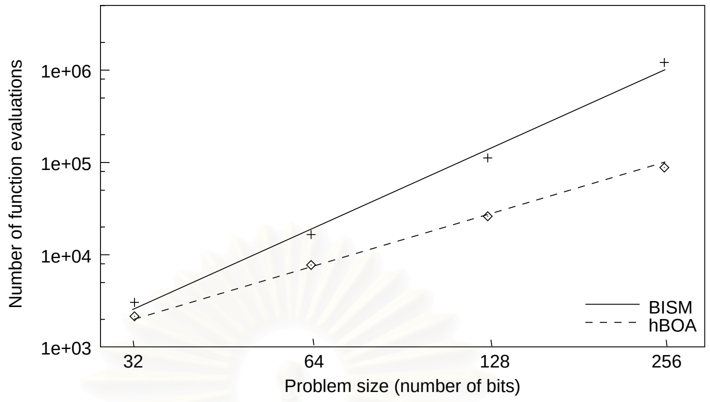


Figure 5.7: Performance comparison between the hBOA and BISM (HIFF functions)

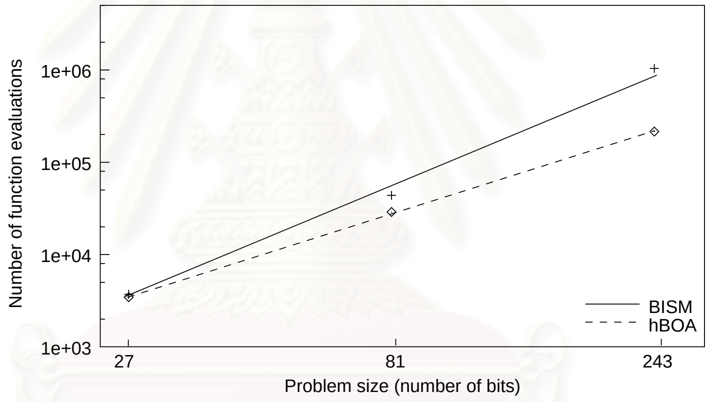


Figure 5.8: Performance comparison between the hBOA and BISM (HTrap1 functions)

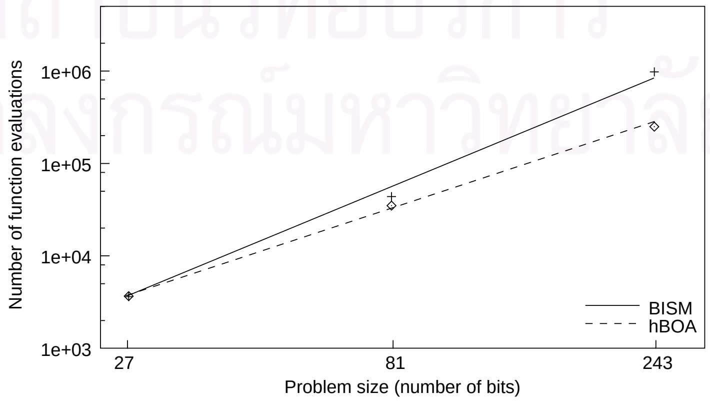


Figure 5.9: Performance comparison between the hBOA and BISM (HTrap2 functions)

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

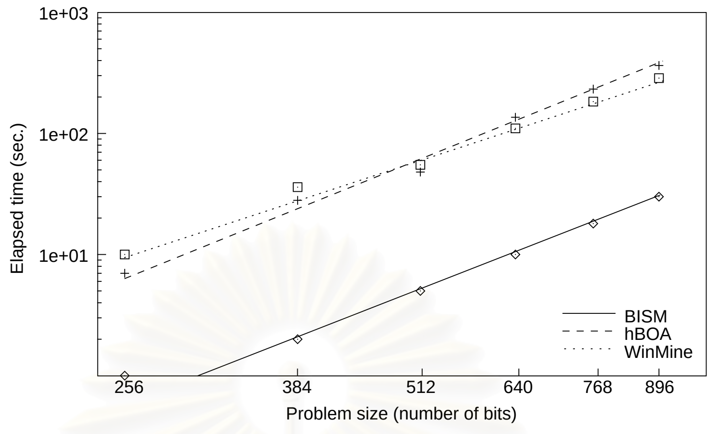


Figure 5.10: Elapsed time required to construct Bayesian network (in hBOA) and matrix

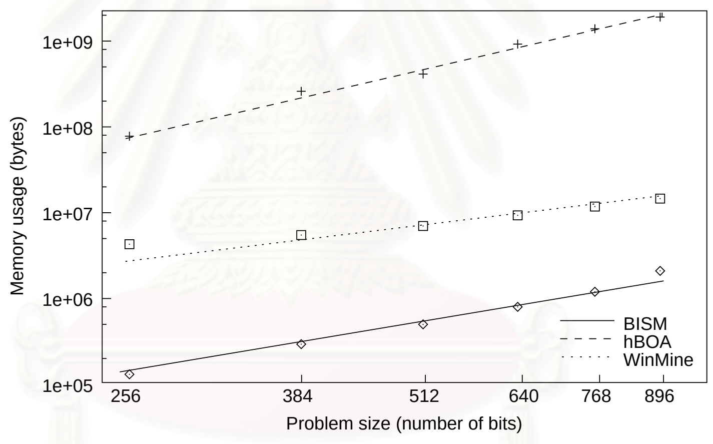


Figure 5.11: Memory usage required to construct Bayesian network (in hBOA) and matrix

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

The Road goes ever on and on

Down from the door where it began.

Now far ahead the Road has gone,

And I must follow, if I can,

Pursuing it with eager feet,

Until it joins some larger way

Where many paths and errands meet.

And whither then? I cannot say.

(J. R. R. Tolkien)

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER VI

Conclusions

The current building-block identification research relies on building a distribution of solutions. The Bayesian network is a powerful representation for the distribution, and therefore the network is able to identifying building blocks in most cases. Nevertheless, building the network is time-and-memory consuming. Finding the network that maximizes a scoring metric is NP-hard. A greedy algorithm is used to search for the network the maximizes the scoring metric, but the optimal network is not guaranteed. If the problem size is larger than a thousand of bits, building the network may not be practical. Eventually, the algorithm must be parallelized to overcome the memory bottleneck.

This thesis presents a distinctive approach for identifying building blocks. We do not build the distribution of solutions as it is in the current research. The building blocks are identified by means of the simultaneity matrix. The matrix element m_{ij} is the degree of linkage between bit positions i and j. The time complexity of computing the matrix is $O(n\ell^2)$ where n is the number of solutions and ℓ is the solution length. We have shown that the matrix computation is 10 times faster and uses 10 times less memory than building the Bayesian network.

To exploit the matrix, partitioning is an alternative. We write a definition of the desired partition. Consequently, we design an algorithm that searches for the desired partition. The time complexity of partitioning is $O(\ell^4)$ where ℓ is the solution length. The partitioning algorithm is deterministic. It can be proven that, given a matrix of which the elements are distinct, the desired partition is unique. The partition is used in the solution recombination so that the bits governed by the same partition subset are passed together.

We have shown that by exploiting the simultaneity matrix, the additively decomposable functions (ADFs) and the hierarchically decomposable functions (HDFs) can be solved in a scalable manner. Empirical results show that the number of function evaluations required to reach the optimum grows in a polynomial relationship with the problem size. The hierarchical Bayesian optimization algorithm (hBOA) uses less number of function evaluations than that of our algorithm. However, the matrix computation is much faster.

The thesis consists of two important parts: 1) simultaneity matrix construction and 2) partitioning. The second part is made in order to show an alternative for exploiting the matrix. This work could be extended by separately improving the first and the second parts. First, the matrix definition can be altered. Because the building blocks are inferred, you might have your own definition that fits your problem. For example, the matrix element m_{ij} could be the Pearson's chi-square – a statistical measurement that indicates the degree of dependency between bits at positions i and j. Second, the matrix can be exploited in another way rather than the partitioning. For example, the bits at positions i and j are passed together with a probability that is proportional to m_{ij}. Introducing the probability is necessary for the cases where the building blocks are overlapped because the deterministic partitioning always gives the same partition.

We have shown that the matrix is similar to an instance of netlist partitioning problem. The matrix element m_{ij} is the number of wires connected between components i and j. The goal of netlist partitioning is to minimize mincut. The number of partition subsets (subcircuits) is given beforehand, but identifying building blocks involves inference or induction. Showing a relevance between the netlist partitioning and the building-block identification is an interesting issue in future work.

The goal of genetic algorithms is to implicitly or explicitly infer building blocks, and therefore composing them. The success of genetic algorithms depends on the hypothesis that the better solutions are composed of the building blocks. When we are talking about building blocks, the hypothesis is assumed to be true. The existence of the building blocks in real-world applications is still a controversial issue. Most GA papers aim to solve the optimization problems and report the performance in terms of solution quality, but few of them tells how the solution is achieved. The visualization of the simultaneity matrix is an attempt to validate the building-block hypothesis. Nevertheless, the visualization is for human reading. Validating the building-block hypothesis should be done systematically and automatically so that we can validate the hypothesis over a wide range of real-world applications.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

References

[1] Ackley, D. H. (1987). A Connectionist Machine for Genetic Hillclimbing. Boston, MA: Kluwer Academic Publishers.

[2] Alpert, C. J., and Kahng, A. B. (1995). Recent directions in netlist partitioning. Integration, the VLSI Journal, 19(1-2): 1–81.

[3] Altenberg, L. (1997). The schema theorem and price's theorem. In Foundation of Genetic Algorithms 3, pp. 23–49, Sanmateo, CA: Morgan Kaufmann.

[4] Baluja, S. (1994). Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Pittsburgh, PA: Carnegie Mellon University.

[5] Baluja, S., and Davies, S. (1997). Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In Proceedings of the 14th International Conference on Machine Learning, pp. 30–38.

[6] Bäck, T., Hoffmeister, F., and Schwefel, H. (1991). A Survey of Evolution Strategies. In Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 1–5.

[7] Battle, D. L., and Vose, M. D. (1997). Isomorphisms of genetic algorithms. In Foundation of Genetic Algorithms, pp. 242–251, Sanmateo, CA: Morgan Kaufmann.

[8] Bosman, P., and Thierens, D. (1999). Linkage information processing in distribution estimation algorithms. In Proceedings of Genetic and Evolutionary Computation Conference, pp. 60–67.

[9] Bosman, P. A. N., and Thierens, D. (2000). Continuous iterated density estimation evolutionary algorithm within the IDEA framework. In Workshop Proceedings of Genetic and Evolutionary Computation Conference, pp. 197–200.

[10] Chickering, D. M. (2002). The WinMine Toolkit. Technical Report MSR-TR-2002-103, Microsoft Research, Redmond, WA.

[11] Corno, F., Reorda, M. S., and Squillero, G. (1998). A new evolutionary algorithm inspired by the selfish gene theory. In Proceedings of Congress on Evolutionary Computation, pp. 575–580.

[12] Deb, K., and Goldberg, D. E. (1993). Analyzing deception in trap functions. In Foundation of Genetic Algorithms 2, pp. 93–108, Sanmateo, CA: Morgan Kaufmann.

[13] De Bonet, J. S., Isbell, C. L., and Viola, P. (1997). MIMIC: Finding optima by estimating probability densities. Advances in Neural Information Processing Systems, 9: 424.

[14] De Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems. Doctoral dissertation, University of Michigan, Ann Arbor, Michigan.

[15] De Jong, K. A. (1997). Genetic algorithms are NOT function optimizers. In Foundation of Genetic Algorithms 2, pp. 5–17, Sanmateo, CA: Morgan Kaufmann.

[16] De Jong, K. A., Potter, M. A., and Spears, W. M. (1997). Using problem generators to explore the effects of epistasis. In Proceedings of the Seventh International Conference on Genetic Algorithms, pp. 338–345.

[17] Fogel, L. J., Angeline, P. J., and Fogel D. B. (1995). An evolutionary programming approach to self-adaptation on finite state machines. In Proceedings of Evolutionary Programming, pp. 355–365.

[18] Fogel, D. B. (2001). Evolutionary Computation : The Fossil Record. Piscataway, NJ: IEEE Press.

[19] Goldberg, D. E. (1989). Genetic Algorithms in Search Optimization and Machine Learning. Reading, MA: Addison Wesley.

[20] Goldberg, D. E., Korb, B., and Deb, K. (1989). Messy genetic algorithms: Motivation, analysis and first results. Complex Systems, 3(5): 493–530.

[21] Goldberg, D. E., and Bridges, C. L. (1990). An analysis of a reordering operator on

a GA-hard problem. Biological Cybernetics, 62: 397–405.

[22] Goldberg, D. E., Deb, K., Kargupta, H., and Harik, G. (1993). Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 56–64.

[23] Goldberg, D. E. (2002). The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Boston, Kluwer Academic Publishers.

[24] Goldberg, D. E., Sastry, K., and Ohsawa, Y. (2002b). Discovering deep building blocks for competent genetic algorithms using chance discovery via KeyGraphs. Technical Report 2002026, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Champaign, IL.

[25] Graham, R. L., Knuth, D. E., and Patashnik, O. (1989). Concrete Mathematics: A Foundation for Computer Science. Reading, MA: Addison-Wesley.

[26] Harik, G. R. (1997a). Learning linkage. In Foundation of Genetic Algorithms 4, pp. 247–262, Sanmateo, CA: Morgan Kaufmann.

[27] Harik, G. R. (1997b). Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms. Doctoral dissertation, University of Michigan, Ann Arbor, Michigan.

[28] Harik, G. R., Lobo, F. G., and Goldberg, D. E. (1999a). The compact genetic algorithm. IEEE Transaction on Evolutionary Computation, 3(4): 287–297.

[29] Harik, G. R. (1999b). Linkage learning via probabilistic modeling in the ECGA. Technical Report 99010, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.

[30] Heckendon, R. B., Rana, S., and Whitley, D. (1997). Test function generators as embedded landscapes. In Foundation of Genetic Algorithms 5, pp. 183–198, Sanmateo, CA: Morgan Kaufmann.

[31] Heckerman, D., Geiger, D., and Chickering, M. (1994). Learning Bayesian networks:

The combination of knowledge and statistical data. Technical Report MSR-TR-94-09, Redmond, MA: Microsoft Research.

[32] Hennessy, J. L., and Patterson, D. A. (1996). Computer Architecture: A Quantitative Approach. San Franciso, CA: Morgan Kaufmann.

[33] Höhfeld, M., and Rudolph, G. (1997). Towards a theory of population-based incremental learning. In Proceedings of the IEEE conference on Evolutionary Computation, pp. 1–5.

[34] Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. Ann Arbor, MI: University of Michigan Press.

[35] Holland, J. H. (2000). Building blocks, cohort genetic algorithms, and hyperplane-defined functions. Evolutionary Computation, 8(4): 373–391.

[36] Jones, T. (1994). A description of Holland's royal road function. Evolutionary Computation, 2(4): 409–415.

[37] Kargupta, H. (1995). SEARCH, polynomial complexity, and the fast messy genetic algorithm. Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, Illinois.

[38] Kargupta, H. (1996). The gene expression messy genetic algorithm. In Proceedings of Congress on Evolutionary Computation, pp. 814–819.

[39] Kargupta, H., and Goldberg, D. E. (1997). SEARCH, blackbox optimization, and sample Complexity. In Foundation of Genetic Algorithms 4, pp. 291–324, San-mateo, CA: Morgan Kaufmann.

[40] Kargupta, H. (1998). Revisiting the GEMGA: Scalable evolutionary optimization through linkage learning. In Proceedings of Congress on Evolutionary Computation, pp. 603–608.

[41] Kargupta, H., and Park, B. (2001). Gene expression and fast construction of distributed evolutionary representation. Evolutionary Computation, 9(1): 43–69.

[42] Kirkpatrick, S., Gelatt Jr., C. D., and Vecchi, M. P. (1983). Optimization by simu-

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

lated annealing. Science, 220: 671–680.

[43] Koza, J. (1992). Genetic Programming. Cambridge, MA: MIT Press.

[44] Michalski, R. S. (2000). Learnable evolution model: Evolutionary processes guided by machine learning. Machine Learning, 38: 9–40

[45] Mitchell, M. (1996). Introduction to genetic algorithms. Cambridge, MA: MIT Press.

[46] Mitchell, T. M. (1997). Machine Learning. Singapore: McGraw-Hill.

[47] Mühlenbein, H., and Paaß, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. In Proceedings of Parallel Problem Solving from Nature, pp. 178–187.

[48] Mühlenbein, H., and Mahnig, T. (1999). FDA – A scalable evolutionary algorithm for the optimization of additively decomposable functions. Evolutionary Computation, 7(4): 353–376.

[49] Munetomo, M., and Goldberg, D. E. (1998). Identifying linkage by nonlinearity check. Technical Report 98012, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.

[50] Munetomo, M., and Goldberg, D. E. (1999a). Linkage identification by nonmonotonicity detection for overlapping functions. Evolutionary Computation, 7(4): 377–398.

[51] Munetomo, M., and Goldberg, D. E. (1999b). A genetic algorithm using linkage identification by nonlinearity check. In Proceedings of Systems, Man, and Cybernetics, pp. 595–600.

[52] Munetomo, M. (2002). Linkage identification based on epistasis measures to realize efficient genetic algorithms. In Proceedings of Congress on Evolutionary Computation, pp. 1332–1337.

[53] Nicolau, M., and Ryan, C. (2002). LINKGAUGE: Tackling hard deceptive problems with a new linkage learning genetic algorithm. In Proceedings of Genetic

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

and Evolutionary Computation Conference, pp. 488–494.

[54] Paredis, J. (1995). The symbiotic evolution of solutions and their representations. In Proceedings of the Sixth International Conference on Genetic Algorithms, pp. 359–365.

[55] Pelikan, M., Goldberg, D. E., and Cantú-Paz, E. (1999a). BOA: The Bayesian optimization algorithm. In Proceedings of Genetic and Evolutionary Computation Conference, pp. 525–532.

[56] Pelikan, M., Goldberg, D. E., and Lobo, F. (1999b). A survey of optimization by building and using probabilistic models. Technical Report 99018, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.

[57] Pelikan, M., Goldberg, D. E., and Lobo, F. (1999b). A survey of optimization by building and using probabilistic models. Computational Optimization and Applications, 21(1): 5–20.

[58] Pelikan, M., and Mühlenbein, H. (1999c). The bivariate marginal distribution algorithm. In Advances in Soft Computing – Engineering Design and Manufacturing, pp. 521–535. London: Springer-Verlag.

[59] Pelikan, M. (1999). A simple implementation of the Bayesian optimization algorithm (BOA) in C++ (version 1.0). Technical Report 99011, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign,

[60] Pelikan, M., and Goldberg, D. E. (2000). Hierarchical problem solving by the Bayesian optimization algorithm. In Proceedings of Genetic and Evolutionary Computation Conference, pp. 267–274.

[61] Pelikan, M. (2000). A C++ implementation of the Bayesian optimization algorithm (BOA) with decision graph. Technical Report 2000025, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.

[62] Pelikan, M., and Goldberg, D. E. (2001). Escaping hierarchical traps with competent genetic algorithms. Technical Report 2001003, Illinois Genetic Algorithms Lab-

oratory, University of Illinois at Urbana-Champaign.

[63] Pelikan, M. (2002). Bayesian optimization algorithm: From single level to hierarchy. Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, Illinois

[64] Pelikan, M., and Goldberg, D. E. (2003). Hierarchical BOA solves Ising Spin Glasses and MAXSAT. In Proceedings of Genetic and Evolutionary Computation Conference, pp. 1271–1282.

[65] Salman, A. A., Mehrotra, K., and Mohan, C. K. (2000). Adaptive linkage crossover. Evolutionary Computation, 8(3): 341–370.

[66] Salomon, D. (2000). Data Compression: The Complete Reference. New York, NY: Springer-Verlag.

[67] Sastry, K., and Xiao, G. (2001). Cluster optimization using extended compact genetic algorithm. Technical Report 2001016, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.

[68] Smith, J., and Fogarty, T. (1996). Recombination strategy adaptation via evolution of gene linkage. In Proceedings of the IEEE International Conference on Evolutionary Computation, pp. 826–831.

[69] Spears, W. M., and De Jong, K. A. (1997). An analysis of multi-point crossover. In Foundation of Genetic Algorithms, pp. 301–315, Sanmateo, CA: Morgan Kaufmann.

[70] Syswerda, G. (1989). Uniform crossover in genetic algorithms. In Proceedings of the Third International Conference on Genetic Algorithms, pp. 2–9.

[71] Thierens, D., and Goldberg, D. E. (1993). Mixing in genetic algorithms. In Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 38–45.

[72] Thierens, D. (1999). Scalability problems of simple genetic algorithms. Evolutionary Computation, 7(4): 331–352.

[73] Vose, M. D. (1991). Generalizing the notion of schema in genetic algorithms. Artificial Intelligence, 50(3): 385–396.

[74] Vose, M. D. (1999). The simple genetic algorithm: Foundations and theory. Cambridge, MA: MIT Press.

[75] Watson, R. A., Hornby, G. S., and Pollack, J. B. (1998). Modeling building-block interdependency. In Proceedings of Parallel Problem Solving from Nature V, pp. 97–106.

[76] Watson, R. A., and Pollack, J. B. (1999a). Hierarchically consistent test problems for genetic algorithms. In Proceedings of Congress on Evolutionary Computation, pp. 1406–1413.

[77] Watson, R. A., and Pollack, J. B. (1999b). Incremental commitment in genetic algorithms. In Proceedings of Genetic and Evolutionary Computation Conference, pp. 710–717.

[78] Whitley, D. (1994). A Genetic Algorithm Tutorial. Statistics and Computing, 4: 65-85, http://www.cs.colostate.edu/~genitor/MiscPubs/tutorial.ps.gz.

[79] Whitley, D., Rana, S., Dzubera, J., and Mathias, K. E. (1996). Evaluating evolutionary algorithms. Artificial Intelligence, 85: 245–276.

[80] Winston, P. (1992). Artificial Intelligence. Reading, MA: Addison-Wesley.

[81] Winter, G., Periaux, J., and Cuesta, P. (1995). Genetic Algorithms in Engineering and Computer Science. John Wiley & Sons.

[82] Winter, P. C., Hickey, G. I., and Fletcher, H. L. (1998). Instant Notes in Genetics. New York: Springer-Verlag

[83] Wolpert, D. H., and Macready, W. G. (1995). No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe, NM: Santa Fe Institute.

[84] Wolpert, D. H., and Macready, W. G. (1997). No free lunch theorem for optimization. IEEE Transactions on Evolutionary Computation, 1(1): 67-82.

Biography

Chatchawit Aporntewan was born on the 19th of June 1977. My birthplace is Surin, Thailand. I had studied at my province till the primary school. Then, I had studied at Triam Udom Suksa School for two years. In 1994, I was a freshman in the Faculty of Engineering, Chulalongkorn University. Now I am a Ph.D. student in the Department of Computer Engineering, Chulalongkorn University.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย