การควบคุมพารามิเตอร์แบบปรับตัวในขั้นตอนวิธีเชิงพันธุกรรม

นาย ชิษณุ ทองฉิม

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรดุษฎีบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2547

ISBN 974-17-5899-5

ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

ADAPTIVE PARAMETER CONTROL IN GENETIC ALGORITHMS

Mr. Shisanu Tongchim

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

A Dissertation Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy in Computer Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2004
ISBN 974-17-5899-5

Thesis Title	Adaptive Parameter Control in Genetic Algorithms
By	Shisanu Tongchim
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 the Faculty of Engineering
(Professor Direk Lavansiri, Ph.D.)

THESIS COMMITTEE

. . . . Chairman
(Assistant Professor Boonserm Kijsirikul, D.Eng.)

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

. . . . Member
(Assistant Professor Krung Sinapiromsaran, Ph.D.)

. . . . Member
(Veera Muangsin, Ph.D.)

. . . . Member
(Assistant Professor Nachol Chaiyaratana, Ph.D.)

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

ชิษณุ ทองฉิม : การควบคุมพารามิเตอร์แบบปรับตัวในขั้นตอนวิธีเชิงพันธุกรรม. (Adaptive Parameter Control in Genetic Algorithms) อ. ที่ปรึกษา : รศ.ดร. ประภาส จงสถิตย์วัฒนา, 107 หน้า. ISBN 974-17-5899-5.

วิทยานิพนธ์นี้เสนอวิธีการที่ช่วยแก้ปัญหาเกี่ยวกับการปรับพารามิเตอร์ของขั้นตอนวิธีเชิงพันธุกรรม วิธีนี้เรียกว่า ‘ขั้นตอนวิธีการควบคุมพารามิเตอร์แบบปรับตัว’ หรือ ‘Adaptive Parameter Control Algorithm’ (APCA) หลักการทำงานของ APCA อยู่บนพื้นฐานของขั้นตอนวิธีเชิงพันธุกรรมสองชั้น โดยขั้นตอนวิธีเชิงพันธุกรรมชั้นล่างแก้ปัญหาที่กำหนดให้ ส่วนขั้นตอนวิธีเชิงพันธุกรรมชั้นบนจะปรับพารามิเตอร์ของระดับล่างให้เหมาะ ทั้งสองชั้นจะทำงานไปพร้อมๆ กัน ทั้งนี้สมาชิกแต่ละตัวในประชากรของขั้นตอนวิธีเชิงพันธุกรรมชั้นบนคือชุดพารามิเตอร์ของขั้นตอนวิธีเชิงพันธุกรรมชั้นล่าง การประเมินคุณภาพของสมาชิกแต่ละตัวในประชากรชั้นบนทำโดยการกำหนดให้สมาชิกนั้นเป็นพารามิเตอร์ควบคุมของขั้นตอนวิธีเชิงพันธุกรรมชั้นล่าง ประสิทธิภาพของขั้นตอนวิธีเชิงพันธุกรรมชั้นล่างจะถูกใช้เป็นคะแนนคุณภาพของสมาชิกนั้นๆ ขั้นตอนวิธีเชิงพันธุกรรมชั้นล่างแบบที่มีประชากรย่อยหลายๆ กลุ่มถูกใช้เพื่อให้การประเมินคุณภาพของประชากรพารามิเตอร์เป็นไปแบบขนาน ซึ่งวิธีการนี้เหมาะสมเป็นอย่างดีกับการประมวลผลแบบขนานของขั้นตอนวิธีเชิงพันธุกรรมแบบหน่วยหยาบ ผลการทดลองแสดงให้เห็นว่า APCA ไม่เพียงแต่หาคำตอบได้เร็วกว่าขั้นตอนวิธีแบบอื่นๆ แต่ APCA ยังสามารถหาผลเฉลยได้แน่นอนกว่า

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

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

##4371807221 : MAJOR COMPUTER ENGINEERING
KEY WORD: GENETIC ALGORITHMS/ADAPTIVE PARAMETER CONTROL

SHISANU TONGCHIM : ADAPTIVE PARAMETER CONTROL IN GENETIC ALGORITHMS. THESIS ADVISOR : ASSOC. PROF. PRABHAS CHONGSTITVATANA, Ph.D., 107 pp. ISBN 974-17-5899-5.

This thesis proposes a method to overcome the parameter setting problem of genetic algorithms. This method is denoted as 'Adaptive Parameter Control Algorithm' (APCA). The concept of APCA is based on two levels of genetic algorithms. The task level genetic algorithm (lower level genetic algorithm) solves the original problem, while the meta-level genetic algorithm (upper level genetic algorithm) optimizes the parameters of the task level. Both levels operate concurrently. Each individual in the population of the meta-level genetic algorithm is a parameter set for the task level genetic algorithm. The evaluation of each individual in the meta-level population is carried out by assigning it as the parameter set of the task level genetic algorithm, the performance of the task level genetic algorithm is then used as the fitness. The task level genetic algorithm with multiple subpopulations is used to parallelize the evaluation of the meta-level population. This fits well with a coarse-grained model parallel genetic algorithm. The empirical results indicate that APCA is not only faster than other algorithms, but APCA also more reliably finds optimal solutions.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

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

Acknowledgements

I would like to express my gratitude to my supervisor, Assoc. Prof. Prabhas Chongstitvatana, for his instructive and invaluable advice. He always encourages me to find my own way. Without his guidance, this research would not have been possible. I would also like to thank Prof. Xin Yao for his help and support during my visit in the University of Birmingham. I am very fortunate in having the opportunity to work with him.

I would like to thank members of Intelligent System Laboratory who have provided help and friendship during my Ph.D. study. I thank Chatchawit Aporntewan for his useful LaTeX style file that was used in preparing this thesis.

I thank Pariyarat Kiatchoosakul for her support, patience and encouragement that helped me to keep going during the hardest time. Finally, I thank my parents and my sister.

My research was supported by the Royal Golden Jubilee Ph.D. Program of the Thailand Research Fund.

Shisanu Tongchim
May 18, 2004

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

Contents

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

Contents (cont.)

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

List of Tables

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

List of Figures

Page

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

List of Figures (cont.)

CHAPTER I
Introduction

1.1 Background

Before using Genetic Algorithms (GAs) to solve given problems, the decision on which is the best parameter set from various possible parameter combinations has to be made. The control parameters largely determine the success and efficiency of GAs in solving problems. Unfortunately, finding a good parameter set is far from a trivial task since these parameters interact with each other in a complicated way. Although practitioners, who are familiar with GAs, may have conventional intuition and experience in the parameter setting, GAs may not always work as expectation. A study by De Jong *et al.* [1] showed that even a slightly change in one component can make empirical results differ from what we expect.

Many practitioners find a promising parameter set for a specific problem by trying various combinations of the control parameters. However, this method obviously requires a lot of computation due to an explosion in the number of trials required for testing all combinations of parameters. A prominent example of exhaustively testing several combinations of parameters was shown in the study done by Schaffer *et al.* [2]. That study carefully examined the performance of a GA using various combinations of the control parameters. The experiments that involved several test functions and parameter combinations took approximately 1.5 CPU years. Indeed, the results may not generalize beyond the test problems used since the best parameter set for one GA or one problem may not be optimal in many other situations.

Some researchers have investigated the theoretical studies of the control parameters [3, 4, 5, 6, 7]. However, these studies have been made by using simplified algorithms and problems. To date there are no rules to select parameters that can generalize to all cases. Some researchers have investigated proper parameter settings from the theoretical findings. For instance, Lobo [8] proposed a parameter-less GA. The results of the theoretical studies were used to derive a proper parameter set that would perform well in many circumstances. The author claimed that the important parameters were already determined. However, this framework was limited to the binary representation of solutions and the search operator was restricted to crossover only.

Alternatively, some researchers have explored adaptive and non-adaptive mechanisms to overcome the parameter selection problem in GAs. These techniques controlled the parameter values during the evolutionary search. The research in this area has been appeared almost continuously. Some techniques have been accepted as standard practices in some branches of evolutionary computation (e.g., setting mutation step sizes in evolution strategies (ESs)).

1.2 Goal of This Research

This thesis will develop an adaptive mechanism for controlling parameters in GAs. This technique will have the following features:

- *On-line mechanism*: The proposed method adjusts parameters while the search is ongoing. This method is distinct from some methods which perform the parameter selection before solving a given problem and the obtained parameter set is then used for solving that problem (e.g., [9]).
- *Adaptive parameter control*: The parameter adjustment relies on the performance of the evolutionary search of GAs, as opposed to the use of deterministic rules in some studies. Since this method uses only the observed performance of GAs in order to adjust parameters, *a priori* knowledge about the underlying implementation of GAs or the information of the given problem is not necessary.
- *Multiple parameters*: Several parameters are involved in the parameter selection process. Note that many proposed techniques only dealt with one

parameter (e.g., [10, 11]).

- *Parallel implementation*: Our proposed method can be viewed as an extension to a conventional coarse-grained model for parallelization. Thus, this method can obtain the advantages of this parallel model.

The general idea is to use a meta-level algorithm to optimize parameters of GAs. We use a GA variant for meta-level optimization. A number of parameter sets of the lower level GA, called the *task level GA*, are evaluated and optimized by the meta-level GA. The use of the meta-level GA causes a new problem associated with the computational cost of evaluating a number of parameter sets. To overcome this problem, we adopt the concept of coarse-grained parallelization. The population is divided into a few large subpopulations. These subpopulations evolve independently and concurrently on different processors. The parameter sets optimized by the meta-level GA are distributed among these subpopulations. Each subpopulation is controlled by the received parameter sets. The observed performance of subpopulations will be used to determine how to optimize parameter sets by the meta-level GA.

1.3 Organization

The contents of the thesis are divided into seven chapters. Note that chapter 2 may be skipped if the reader is already familiar with GAs. The details of all chapters are as follows:

- Chapter 1 provides a general introduction to the research. In the beginning, the chapter points out to general practices for setting the parameters of GAs. Then, the chapter presents the goal of this research, and outlines the contents of the remaining chapters.
- Chapter 2 presents a brief introduction to GAs. The chapter also compares GAs with other search algorithms. The second part of the chapter provides a concept of parallel GAs, and the models of parallelization. It focuses on a coarse-grained model since our study is based on this model.

- Chapter 3 reviews relevant and related literature associated with techniques for controlling and setting the parameters of GAs. The chapter begins with some classifications that have been proposed by several researchers. Thereafter, the chapter concentrates on two selected areas: subpopulation-level methods and meta-level optimization.
- Chapter 4 describes the concept of an adaptive parameter control algorithm (APCA). The chapter starts by describing the details of the parameters that are controlled by APCA. Then, the chapter explains the development of APCA and its details.
- Chapter 5 provides the experimental results of APCA. The first part compares APCA against five algorithms on five test problems. The second part uses a multimodal problem generator to examine how well APCA adapts itself to new problem instances. The last part considers the evolved parameter settings from the meta-level algorithm.
- Chapter 6 applies APCA to an examination timetabling problem. The results provide some insights about the application of APCA on a more complicated and realistic problem.
- Chapter 7 gives conclusions and suggestions for further research.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER II

Overview of Genetic Algorithms

2.1 Introduction

The advent of algorithms inspired by the natural evolution and adaptation has brought a new research area called *Evolutionary Computation (EC)*. The history of EC can be traced back to some ground-breaking studies of the 1950s. However, three primary algorithms, namely *evolutionary programming*, *evolution strategies* and *genetic algorithms*, that have been used up until the present time were established by the mid-1960s. At that time three evolutionary algorithms were developed independently by three research groups.

Evolutionary programming (EP) was introduced by Fogel *et al.* [12, 13]. It was first used to evolve finite state machines. Evolution strategies (ESs) were investigated by Rechenberg and Schwefel at the Technical University of Berlin. In early developments, ESs were used to optimize the body shape which gave the smallest drag [14]. A comprehensive introduction to ESs can be found from the article by Beyer and Schwefel [15]. The third variant is genetic algorithms (GAs) that were developed by Holland [16, 17]. GAs are also often attributed to the book by Goldberg [18].

In the beginning, three main evolutionary algorithms were quite different. As the development of evolutionary algorithms has advanced, however, the gap between these algorithms seems to be narrowed. EP and ESs are nearly identical when are used for numerical optimization [19]. In addition, some techniques have been shared among evolutionary algorithms. For example, the concept of self-adaptation has been used extensively in ESs and EP. It was later applied to GAs by Bäck [11].

Evolutionary computation is an attractive field of study. In the past decade, there are a growing number of successful applications of evolutionary algorithms.

They are sometimes better than existing techniques. One of the advantages is that EAs are more robust to inaccurate, noisy data. EAs can even tackle a problem where its objective function cannot be formulated mathematically.

In this study, we focus our attention on GAs. The next section presents a framework of a conventional GA. Then, the final section presents the concept of parallel GAs which is fundamental to the proposed method in the later chapter.

2.2 Genetic Algorithms

A GA starts with a set, or population, of candidate solutions to a problem. The first task in applying a GA to a given problem is to determine the representation of candidate solutions. The representation changes the given problem from one space to another space. Basically, each solution is normally encoded as a fixed-length array of values. A traditional GA uses binary representation. That is, each potential solution is represented by a number of binary bits. Each encoded solution is called a *chromosome*.

Typically, a population of solutions is randomly generated at the beginning. Assume $\mathcal{P}(t) = (x_1^t, ..., x_n^t)$ is a population of n individuals for iteration t. Then, the process of a GA usually proceeds through the following steps:

1. *Evaluation*: Each member x_i^t in the population $\mathcal{P}(t)$ is assigned the fitness value according to how good this solution to the problem it is. A user-defined fitness function is used in the quality assessment. A good fitness function should be able to compare two potential solutions and determine the better one.
2. *Selection*: Some individuals in the population $\mathcal{P}(t)$ are selected. The probability that a member will be selected is proportional to its relative fitness compared with the fitness of the other competing members in the population.

   Consider *proportional selection* which is a simple and well-known selection method. Assume a fitness function $f : x \rightarrow \Re^+$ assigns non-negative scores to candidate solutions and the objective is to maximize the fitness values. Then, the selection probability for individual x_i^t is calculated as follows:

$$p = \frac{f(x_i^t)}{\sum_{j=1}^{n} f(x_j^t)} \tag{2.1}$$

3. *Breeding*: In this step, some modifications are applied to the selected individuals from the previous step in order to create a new population $\mathcal{P}(t+1)$. The population for the next iteration is created by means of genetic operators. There are two primary types of genetic operators.

   - *Crossover*: A new individual is produced by recombining features of two, or more individuals. A simple crossover operator is *one-point crossover*. Given two selected individuals of length n, one crossover point between 1 and $n-1$ is selected uniformly. The values up to the crossover point are copied to the alternate child, while the remaining values are copied to the respective child. This operator will produce two offspring while they are the complement of each other. Assume two 10-bit parents and the crossover point is 3. The result of one-point crossover is shown in Figure 2.1.

```
                  ↓
parent1 : 0 0 0  0 0 0 0 0 0 0
parent2 : 1 1 1  1 1 1 1 1 1 1
                  ↑

child1  : 0 0 0  1 1 1 1 1 1 1
child2  : 1 1 1  0 0 0 0 0 0 0
```

Figure 2.1: One-point crossover

   - *Mutation*: A new individual is generated by slightly changing a selected individual. Mutation can be implemented in several ways. We will use simple bit-flip mutation for the illustration purpose. Assume the mutation rate is p_m. Normally, the mutation rate is a small value

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

(e.g. 0.01). Every bit has an equal probability to undergo mutation. For each bit within a selected individual, generate a random number between 0 and 1. If the number is less than p_m, then mutate this bit by changing from 0 to 1 or vice versa. Next, the same procedure is repeated for the other bits.

4. *Update*: There are several strategies to update an existing population. The newly generated individuals may replace some or all of the individuals in the current generation. Another strategy attempts to replace the infeasible solutions.
   A traditional GA replaces a whole of a population with a new population. This strategy is referred to as generational reproduction. If a few least fit individuals are replaced in each iteration, this strategy is called steady state reproduction [20].

After finishing the fourth step, the algorithm repeats all steps again until some termination criteria are met, i.e., the number of generations reaches a certain value, or until the fitness of the best solution is better than a specific value. To make the framework of GAs clearer to see, a schematic of GAs is shown in Figure 2.2.

The process of a GA has several parameters. The main parameters that are typically found in most GAs are as follows:

- population size
- selection strategy
- update strategy
- rate for applying crossover
- crossover type
- rate for applying mutation
- mutation type

Some parameters are continuous values (e.g., crossover rate, mutation rate),

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

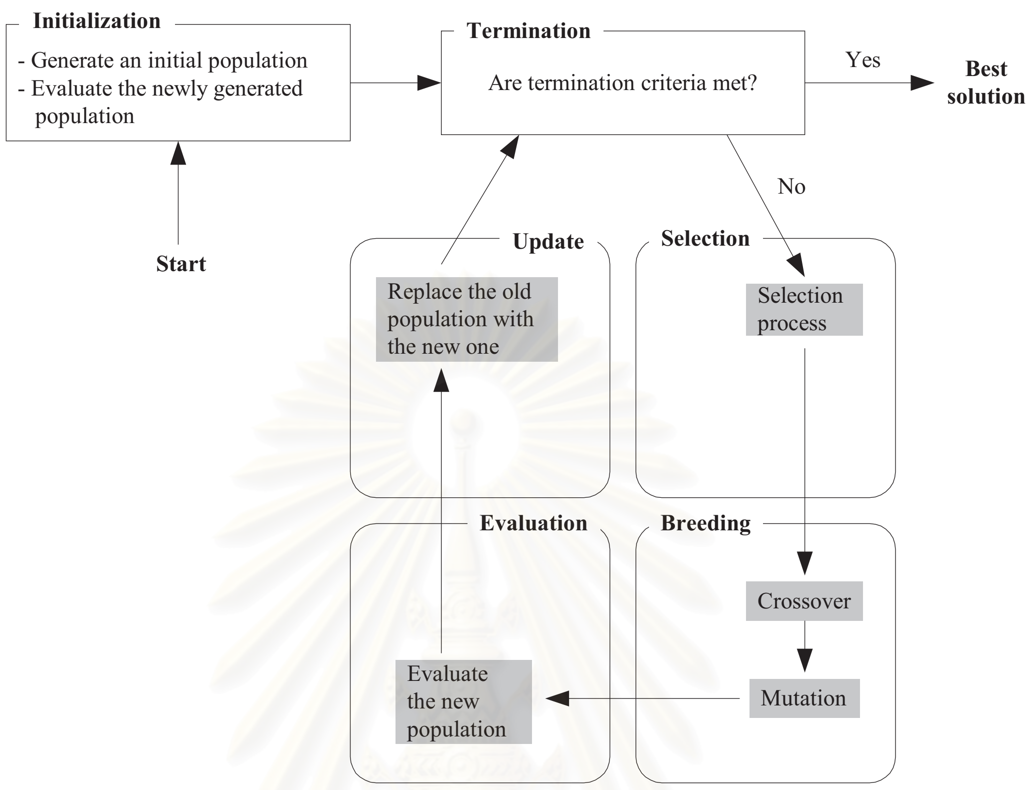
Initialization
- Generate an initial population
- Evaluate the newly generated population
Start
Termination
Are termination criteria met?
Yes
Best solution
No
Update
Replace the old population with the new one
Selection
Selection process
Evaluation
Evaluate the new population
Breeding
Crossover
Mutation

Figure 2.2: A schematic of GAs

whereas some parameters are discrete values (e.g., crossover type, mutation type). In this work, we focus on four main parameters: crossover rate, crossover type, mutation rate and mutation type. Moreover, the concrete parameters (i.e., crossover rate, mutation rate) will be discretized.

Yao [19, 21] pointed out that EAs can be understood by using a framework of *generate-and-test* search. The advantage of formulating EAs as a case of generate-and-test search is that the relationships between EAs and other search algorithms, such as hill-climbing algorithms, simulated annealing (SA), tabu search (TS), and others, can be made clearer. A framework of general generate-and-test search is illustrated in Figure 2.3.

Most EAs can be viewed as a population-based version of generate-and-test search. For GAs, the search operators like crossover and mutation play a role in

```
1. Generate the initial solution at random and accept it as the current solution
2. **Generate** the next solution from the current one by *perturbation*
3. **Test** to see whether the generated solution is *acceptable*
        3.1 Accept it as the current solution if yes
        3.2 Keep the current solution otherwise
4. If the current solution is not satisfactory, goto step 2.
```

Figure 2.3: A framework of general generate-and-test search

perturbing the current solutions in order to generate the next solutions, whereas the selection process determines whether or not the newly generated solutions are acceptable. Many search algorithms can be analyzed by the same manner. By using the same framework, we can easily compare GAs with other search algorithms. For example, hill-climbing algorithms require the next solution to be better than the current one. Conversely, GAs, variants of EAs and some search algorithms, such as simulated annealing have probabilistic methods to test whether the generated solutions are acceptable or not. It is not necessary that the new solutions have to be better than the current ones. However, highly fit solutions have better chance to be accepted.

2.3 Parallel Genetic Algorithms

Parallelization has been applied to many evolutionary algorithms. The general concept is to divide the task of serial algorithms; then distribute on different processors. The divide-and-conquer approach can be applied in numerous ways and this leads to different models. Genetic algorithms are the most popular search algorithms that have been used in the studies of parallel EAs. A review of the studies devoted to parallel GAs can be found in the article by Cantú-Paz [22], and Alba and Troya [23]. A recent article by Alba and Tomassini [24] also presented a review of the studies in this area by using the term of parallel EAs.

Although the vast majority of studies in this area are based on GAs, several techniques used in these studies are not limited to parallel GAs. Many researchers also applied the same concept of parallelization to other classes of EAs, such as genetic programming (GP) [25, 26, 27, 28], evolutionary programming (EP) [29, 30]. This confirms a fundamental fact about EAs that they are naturally capable of being parallelized.

Basically, parallel GAs can be divided into three classes: 1) master/slave parallel GAs, 2) coarse-grained parallel GAs and 3) fine-grained parallel GAs.

2.3.1 Master-slave Parallel GAs

This model uses a single global population. In general, one processor (master processor) maintains the population while the additional processors (slave processors) perform the time-consuming operations and send the results back to the master. There are two main operations that are usually done in parallel: the fitness evaluation and the application of genetic operators. The master processor sends a portion of population to each slave processor. Each slave processor is responsible for evaluating the received individuals or applying the genetic operators to these individuals. Then, the results from slave processors are sent back to the master processor. The communication overhead occurs when individuals are sent to slave processors and the results are returned to the master. A schematic picture of a master-slave parallel GA is shown in Figure 2.4. From the picture, the master assigns portions of the population to the slaves (or the workers). After that, the workers send the fitness values of the received individuals back to the master.

The selection and mating are still performed over the whole population by the master processor. That is, any individual may mate and compete with any other. Accordingly, this model does not change the nature of the algorithm. Apart from the reduction of processing time, the results of the parallel algorithm are nearly identical to those of the serial counterpart. This can be considered as the advantage of this model since the behavior of the algorithm is not altered.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

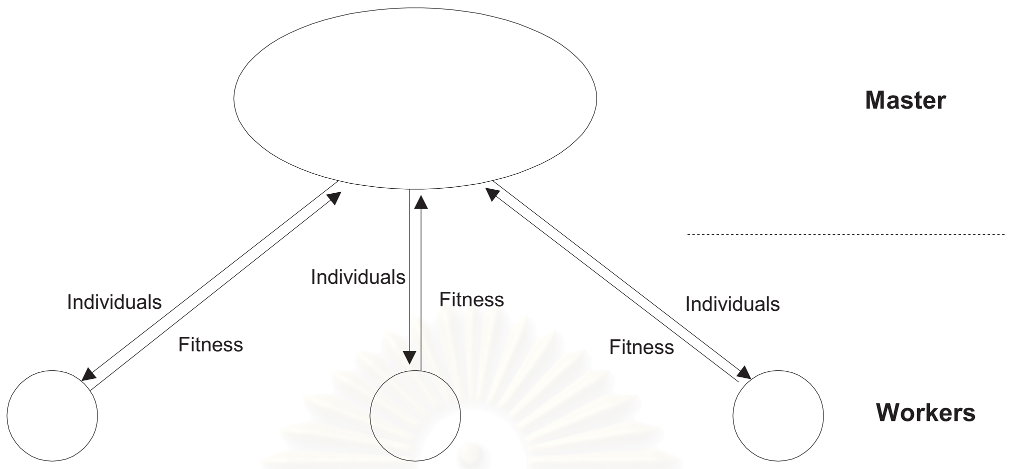

Figure 2.4: A schematic of a master-slave parallel GA

Specifically, the theory and the experience for the serial GAs can apply to the parallel version directly.

2.3.2 Coarse-grained Parallel GAs

The concept of a coarse-grained model is to divide a population into a few large subpopulations which are maintained by different processors (see Figure 2.5). At the beginning, all processors create their own random subpopulations. Then, each processor maintains the evolution of its population independently. While the evolution is ongoing, some selected individuals are periodically exchanged via a migration operator. Generally speaking, each processor executes a serial GA with the smaller population and some selected individuals occasionally migrate among the processors. In general, the migration process is usually infrequent and only few individuals are exchanged during each migration. Thus, the communication overhead of this model is quite small.

Unlike the master-slave model, this model introduces a new algorithm since the behavior of the parallel algorithm is different from that of its serial counterpart. The behavior change is due to the spatial distribution of subpopulations that limits the selection and mating on the level of subpopulations. Many studies have reported that the use of multiple subpopulations and the restricted mating leads to

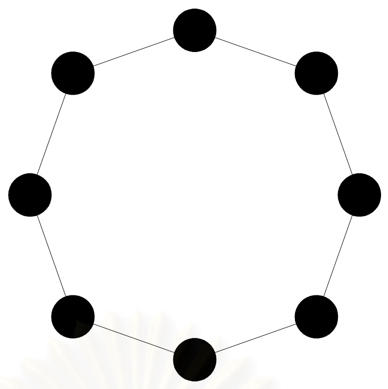

Figure 2.5: A schematic of a coarse-grained parallel GA in a ring topology (the black dots represent the subpopulations and the lines show the communication links

better numerical performance. That is, the parallel algorithm has better efficiency in finding the optimal solution.

The reduction of processing time due to the distribution of the population among processing nodes and the improvement of numerical performance sometimes make the parallel algorithm achieve superlinear speedup. Superlinear speedup means that the speedup factor is greater than the number of processing nodes. Examples of such results can be found from [31, 32, 33, 34, 35]. However, the claim of such superlinear speedup in parallel EAs is often controversial. In fact, the comparison between the serial algorithm and its parallel counterpart seems to be unfair since both algorithms perform different amounts of work.

The advantage of this model is that it requires relatively little effort to convert a serial GA into a coarse-grained parallel GA. Apart from the migration process, most of the program still remains the same. Another advantage is that this model has quite small communication overhead. Only few individuals are sent during the migration, not the entire population as done in the master-slave model. The last advantage is that this model improves the numerical performance of GAs. The parallel algorithm has better performance in finding the solution, even the multiple subpopulations are simulated on a serial machine. From these

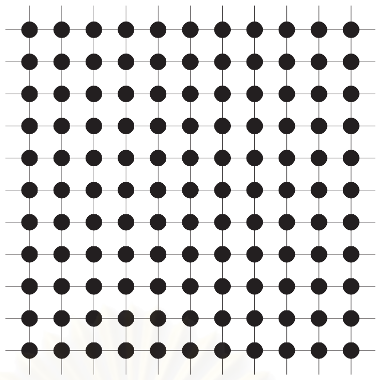

Figure 2.6: A schematic of a fine-grained parallel GA in a 2-D grid (the black dots represent the subpopulations and the lines show the communication links

advantages, it is not surprise that this model is so popular. Our study in the later chapters is also based on this model.

2.3.3 Fine-grained Parallel GAs

The population is separated into a large number of very small subpopulations, which are maintained by different processors. The subpopulation may be only one individual and the interaction between an individual to other individuals is limited to its neighbors. In particular, any individual only competes and mates with its neighbors.

This model is suitable for massively parallel architecture that consist of a huge number of basic processors. These processors are connected with a specific high speed topology. The widely used topology of the processor connection in many massively parallel computers is a 2-Dimensional grid [22].

A schematic picture of a fine-grained parallel GA in a 2-Dimensional grid is shown in Figure 2.6. The structure of neighbors can be configured in several ways. Figure 2.7 shows two examples of the neighbor configuration.

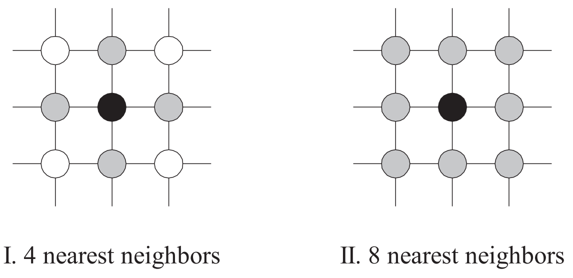

Figure 2.7: Two examples of the neighbor configuration in a fine-grained parallel GA

2.4 Summary

This chapter presents a quick introduction to GAs. GAs can be viewed as a population-based version of generate-and-test search. The search operators like crossover and mutation perturb the current solutions in order to generate new solutions. The selection process is responsible for testing the newly generated solutions and considering whether to accept these solutions. In the second part of this chapter, a classification of parallel GAs is presented. The concept of coarse-grained parallel GAs is fundamental to the study in the later chapters.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER III

Literature Review

3.1 Introduction

The studies of the effects of the control parameters have a long history. One of the earliest studies was done by De Jong [36]. He examined several combinations of parameters on five test functions which were later used as a standard test suite by many researchers. The findings from his empirical study showed that the following parameters yielded an optimal performance: population size 50–100, one-point crossover probability of 0.6 and bit mutation probability of 0.001. These parameter values have been widely used by many researchers. The study done by De Jong was later extended by Schaffer *et al.* [2]. To date, many researchers are still working on this area, both theoretical and empirical. The findings of these kinds of studies can help in selecting the proper parameter sets.

As an alternative to study the effects of the control parameters, some researchers have investigated some techniques for setting parameters. The research in this area has been started almost from the beginning of the evolutionary computation. One of the earliest experiments in adapting parameters was conducted by Reed *et al.* [37]. The control parameters were encoded in the individuals themselves. These parameters were optimized by the evolutionary search along with the solutions to the problem. According to the classification of Eiben *et al.* [38], this scheme is known as *self-adaptive parameter control.* The ability to self-adapt the search is one of the advantages of evolutionary computation [39].

Currently, there are several works devoted to study the techniques for setting or controlling parameters. These studies are differentiated from each other in their control mechanisms and level of adaptation. Some classifications have been proposed in the literature. In the next section, these classifications will be reviewed and discussed.

3.2 Classification Schemes

Many studies devoted to investigate the techniques for setting or controlling parameters have been presented in recent years. Unfortunately, the terms used in these studies are defined differently. Some researchers have been attempted to revise the terminology, survey various techniques used in the literature and provide some classifications of the whole research in this area. In this section, we will discuss about these classifications. We then draw some conclusions about these classifications.

To our knowledge, the earliest remarkable classification was proposed by Angeline [40]. The division criterion is based on the type of update rules and levels of adaptation. The update rules specify how the control parameters are changed over time. In particular, there are two types of update rules: *absolute* update rules and *empirical* update rules. Absolute update rules are predetermined. These rules define how the change in parameters will be made. An example of absolute update rules is the 1/5 success rule. This rule is used to control the mutation strength for a (1+1)ES. Specifically, the mutation strength is increased after a certain number of generations if the success probability of mutation is greater than 1/5, otherwise the mutation strength is decreased.

Empirical update rules modify parameters by using the selection and variation process of an evolutionary algorithm. For instance, the self-adaptation method lets the adaptive parameters co-evolve with the potential solutions to the given problem. The values of the adaptive parameters are encoded into chromosomes. If the parameter values are appropriate to offspring development, they tend to survive in the later generations. If the parameter values are infeasible, the individuals with these parameter values will eventually be replaced by those with more fit parameters.

For the division by the levels of adaptation, three levels are defined:

- *Population-level*: The global values of parameters are adjusted. The modifications effect the entire population.
- *Individual-level*: The parameters of a particular individual within the population are modified.
- *Component-level*: This level of adaptation dynamically changes how the distinct subindividual structures of each individual will be manipulated independently from each other.

Smith and Fogarty [41] provided the classification based on three division principles:

- *What is being adapted*: By using this principle, the studies in this area can be divided into two classes. The first class adapts the probability of application of operators. The second class changes the action of the operators.
- *The scope of the adaptation*: The authors adopted the classification by Angeline [40] to define three distinct levels of adaptation.
- *The basis of change*: The authors drawn a further distinction according to these principles: 1) the algorithms or rules that execute the change, 2) self-adaptation.

Eiben *et al.* [38] provided the most comprehensive review of the literature in this area. This paper attempts to revise the terminology used in this area, provide a classification and survey various proposed techniques. They classified the *parameter setting* techniques into two categories: *parameter tuning* and *parameter control*. Parameter tuning finds parameter values before running the algorithm. The obtained parameter set is used as a static parameter set during the run of the algorithm. In contrast to parameter tuning, parameter control dynamically changes the parameter values while the algorithm is still running. Parameter control is divided into the following three methods:

- *Deterministic parameter control*: The parameters are adapted by using some heuristic rules. Feedback from the status of the algorithm is not used.
- *Adaptive parameter control*: Some feedback from the observed status of the algorithm is used to decide the change in parameters.
- *Self-adaptive parameter control*: The control parameters are encoded into the chromosome. Each parameter set is applied to each chromosome separately. The idea of this method is that good parameter sets will produce good offspring, which will have a better chance to survive and create a new population.

Another classification was proposed by Harik and Lobo [42]. They divided the research in this area into three classes: *empirical studies*, *facetwise theoretical studies* and *parameter adaptation*. Empirical studies are conducted by doing experiments with various combinations of parameters. Facetwise theoretical studies use theoretical analysis to study the effect of parameters. Parameter adaptation changes parameters during the run of the algorithm.

A general observation that is drawn from these classifications is that the studies can be classified by using various division criteria, resulting in different classification schemes. Each class of research can be further divided into several classes. This reflects the diversity of research in this area. Due to a huge amount of work, we will limit our literature review for some related areas. The proposed work relates to two main areas:

1. *Subpopulation-level Methods*: We will provide a review of research works that utilize subpopulation-level methods for setting parameters. Note that some researchers (e.g., Smith and Fogarty [41]) classified these studies to population-level methods. Most studies in this area applied a different parameter set to each subpopulation. Some studies used meta-level algorithms to adjust parameter sets according to the observed performance, while some studies used simple rules to adjust parameter sets. Apart from

dynamically adjusting parameters, some studies reported the use of various static parameter sets in multiple subpopulations.

2. *Meta-level optimization*: Some studies incorporated meta-level algorithms to optimize parameters of underlying algorithms, but these studies were not based on distributed subpopulations. We will also provide a discussion about these studies.

3.3 Subpopulation-level Methods

3.3.1 Using Rules/Algorithms for Adaptation

This section considers several proposed techniques based on the level of subpopulations. All techniques maintained several subpopulations, each with a different parameter set. Some sorts of rules or algorithms were used to adjust parameter sets, or promote promising parameter sets. These were done in several ways. Some techniques promoted promising parameter sets by adapting the subpopulation sizes. Specifically, subpopulations with promising parameter sets increased their sizes, whereas ones with unsuccessful parameter sets reduced their sizes. A technique that conformed to this concept was done by Schlierkamp-Voosen and Mühlenbein [43]. Several subpopulations with different parameter sets were maintained. The performance criteria based on the best fitness values was used to modify the population sizes. The size of the best group was increased, while all other groups were decreased. However, the total number of individuals from all subpopulations was fixed. The test functions were composed of unimodal and multimodal functions. A small number of subpopulations (four groups) were used in the experiment. Only one parameter in various subpopulations was distinct from each other. The findings showed that the proposed method focused its attention on the proper parameter set according to the current state of evolution. For example, the group with largest mutation steps dominated at the beginning of the evolution. Then, the second largest mutation steps took over in the second phase of the evolution. Finally, the smallest mutation steps were used to locate the optimum with high precision. However, the presented method is a first step. Only one parameter is varied over all subpopulations. Clearly, the introduction of other adaptive parameters is limited by the number of subpopulations. That is, the required number of subpopulations is proportional to the number of parameter combinations. Moreover, the algorithm relies on the adaptation of population sizes. Thus, it tends to face the uneven workload problem if it is implemented as a parallel algorithm. The study was later extended by the article of the same authors [44]. The total number of individuals from all subpopulations in the extended model was not fixed, but adapted during the run. However, the extended model was tested by using only two subpopulations. One subpopulation was devoted to crossover, whereas another was based on mutation. Hence, the available parameter combinations are still limited.

Another study based on the modification of subpopulation sizes was proposed by Hinterding *et al.* [45]. They used three subpopulations with sizes of P1, P2 and P3 (initial values of 50, 100 and 200). Some heuristic rules were used to adjust the proper sizes of subpopulations. These rules were designed to maximize the performance of *middle subpopulation* (P2). In addition, they also used an individual-level adaptive method to control mutation strength. This method can be regarded as a self-adaptation method, which the randomly generated values of mutation strength were embedded into chromosomes (one value per each chromosome). This value controlled the standard deviation of the Gaussian mutation in each chromosome independently. By using this method, mutation strength were evolved along with chromosomes. The concept of the self-adaptation method is not new. This technique has been accepted as a standard practice in the field of EP and ESs. In this study, several types of numeric functions included nonlinear, non-separable and scalable functions were used as test functions. The experimental results showed the advantages of using two levels of adaptation. All subpopulations were able to adapt their population sizes to smaller values on continuous unimodal functions. Thus, the exploitation capabilities were increased in such functions. For functions which needed more exploration, the algorithm was able to adapt the population sizes to maximize both exploitation and exploration. However, this technique seems not to be suitable for implementing as a parallel algorithm. The population size adjustment easily leads to the uneven workload among processing nodes, especially in a coarse-grained model. In addition, the proposed method is limited to adapt only the mutation strength and the proper population size.

The last example of using the adaptation of subpopulation sizes for choosing promising parameters was presented by Eiben *et al.* [46]. The proposed technique attempted to select the best crossover operator. Each subpopulation used a different crossover operator. The sizes of successful subpopulations were increased, whereas the sizes of inferior subpopulations were decreased. The modification of population sizes was based on the migration operator. The proposed algorithm was tested on artificial problems. Unfortunately, the experimental results showed that the proposed adaptive method was not able to detect and reward better crossover operators. Namely, the population sizes of better crossover operators did not have a significant distinction from other crossover operators. However, a GA with the adaptive method outperformed a standard GA using 1-point crossover and had comparable performance to a GA with the non-adaptive method using the best crossover. Like the studies [43, 45], the change of population sizes makes the algorithm be impractical for a parallel implementation. Moreover, the algorithm has only one adaptive parameter.

Some techniques did not change subpopulation sizes. The promising parameter sets were promoted by obtaining additional processing time. An example is the study by Pham [47]. Several subpopulations, each with a different parameter set, were evolved on the same computer. Then, the performance of each population was recorded. Two populations that conformed with the selection criteria received additional processing time to evolve further. The first was the population that had the fittest individual. The second was the population which had the fastest improvement. On the whole, this method obviously requires several subpopulations if there are several adaptive parameters in parameter selection. Hence, the algorithm tends to waste the computational time until it can detect the right parameter combinations.

Some studies directly adjusted parameter sets based on the performance of subpopulations. The first example that controlled only the mutation rate was proposed by Lis [10]. This method was based on a master/slave model. After creating an initial population, the master processor sent portions of the population to the slave processors. These subpopulations evolved separately on different slave processors by using various mutation rates. After a predetermined interval, the slave processors sent their best individuals to the master processor. The master processor gathered these individuals and sent the best individuals back to the slave processors. The mechanism of adaptation is that the mutation probabilities of processors were shifted by one level if the best result was acquired from the processor with the highest mutation rate. The mutation rates were also reduced by one level if the best result was obtained from the processor with the lowest mutation rate. Overall, the parameter adaptation in this study is limited to the mutation rate only.

Another study that directly adapted parameter sets was done by Schnecke and Vornberger [48]. They used some rules for adjusting the parameter sets based on the rank of performance. The proposed method applied a different parameter set to each subpopulation. After a specific interval, all parameter sets were ranked based on the performance of subpopulations. Then, each parameter set was adapted to the next better one. Moreover, if the progress on a subpopulation got stuck, the parameter set of this subpopulation was reset. The experimental results showed that the proposed method outperformed a parallel GA using one fixed parameter set and a parallel GA using different parameter sets without adaptation. However, some implementation details are omitted. For instance, it is not clear how the algorithm adapts each parameter set to the values of the next better one.

Wang *et al.* [49, 50] used another GA to directly adjust parameter sets. This method was denoted as DAGA2. DAGA2 consisted of two levels of GAs. The lower level GA contained multiple subpopulations of a traditional GA, in which each individual represented a solution to a particular problem. The upper level GA maintained the evolution of control parameters of the lower level GA. In particular, the individuals in the upper level GA represented the control parameters of the lower level GA. The DAGA2 system was based on a coarse-grained parallel GA. The test problems were composed of several artificial problems. The performance of the DAGA2 system was better than the non-adaptive algorithms from other studies. Moreover, the robustness of the DAGA2 system was also investigated. That is, the performance of the DAGA2 system did not heavily rely on the parameters of the upper level (user-specified). However, the DAGA2 system was not able to compete with other algorithms for easy problems since the startup cost for adapting itself to given problems was relatively high on easy problems. An investigation of the DAGA2 system on a real world problem — 2-dimensional layout problem — was also presented in [51]. The concept of DAGA2 is, to some degree, close to our proposed method. However, the framework of DAGA2 presented in the literature is quite abstract. Some details of the algorithm and the experiments are omitted, i.e., the replacement of individuals in the meta-level algorithm, the rate of parameter adaptation and migration. Moreover, the research also focused on comparing with some static parameter settings rather than other adaptive methods.

3.3.2 Using Static Parameters

As an alternative to the dynamic methods mentioned earlier, some researchers proposed simpler methods. The studies in this section take the advantage of multiple subpopulations to utilize several parameter sets simultaneously. These parameter sets are defined at the beginning of the run and are used statically through the whole process. All studies indicated that the techniques based on this concept were more robust than the use of identical parameter sets on each subpopulation.

An earlier work by Tanese [52] introduced the application of different parameter sets for each subpopulation. Some or all subpopulations used different mutation and crossover probabilities. The findings showed that the use of different parameter settings was more robust than the use of identical parameter settings. Adamidis and Petridis [53] also proposed a method that was close to the idea by Tanese. By using the concept of a coarse-grained parallel GA, several subpopulations with different parameter sets were evolved independently. The proposed algorithm was tested on the problem of training a Recurrent Artificial Neural Network (RANN). Eight subpopulations with different search operators and their parameters were used in the experiment. The algorithm was compared with a parallel GA using the same parameter set in all subpopulations. The proposed algorithm outperformed the algorithm using the single parameter set in all performance metrics. However, it is not clear how the authors select the type of search operators on each subpopulation.

Miki *et al.* [54] proposed a similar concept for setting parameters of subpopulations on a coarse-grained parallel GA. They noticed that the best crossover rate should be well matched with the mutation rate, the population size, the number of subpopulations, the problems to be solved. They also noticed that the optimal crossover rate depended on the state of evolution. To solve these problems, the authors proposed the use of a different parameter set in each subpopulation, called *distributed environment scheme*. The method controlled only the mutation rates and the crossover rates. The mutation rates were discretized into 3 possible values. The crossover rates were discretized into 3 possible values as well. Thus, there were total 9 combinations of parameter setting. Each parameter combination was assigned to a different subpopulation. The effectiveness of distributed environment scheme was compared with a single population GA and a multiple population GA using various parameter combinations. The experimental results showed that the proposed method achieved the best performance compared with a single population GA using the optimal parameter set, and the relatively high performance compared with a multiple population GA using the optimal parameter set. Obviously, the possible parameter combinations of this method are limited by the number of subpopulations. If a new parameter is introduced to the algorithm, it is necessary to increase the number of subpopulations as well.

Another study that used multiple subpopulations with different configurations was presented in the study by Herrera and Lozano [55]. The distributed subpopulations were differentiated from each other by using various crossover operators with different degrees of exploration and exploitation. By using a hypercubic topology (see Figure 3.1), the subpopulations of the front side (namely, A1 A2 A3 A4) were devoted to exploration, while the subpopulations of the rear side (namely, a1 a2 a3 a4) were devoted to exploitation. In each side, the exploration or exploitation degrees were gradually changed. By using a sophisticated migration operator, the authors claimed that the migration process induced the refinement (e.g., from an exploratory subpopulation to an exploitative one) or the expansion (e.g., from an exploitative subpopulation to an exploratory one). From the experimental results, the proposed method delivered both reliability and accuracy in solving the test problems. Since the prior knowledge about the characteristic of search operators used in this article is readily available, it is possible to design the algorithm to conform to this idea. In many problem domains, however, such knowledge is not always known in advance.

3.4 Meta-level Optimization

The research in this category considers the problem of parameter setting as an optimization problem. Some algorithms, including variants of evolutionary algorithms, have been applied to the problem. This results in two levels of optimization. The upper level or the meta-level algorithm optimizes the parameter setting, while the lower level or the task level algorithm solves the given problem.

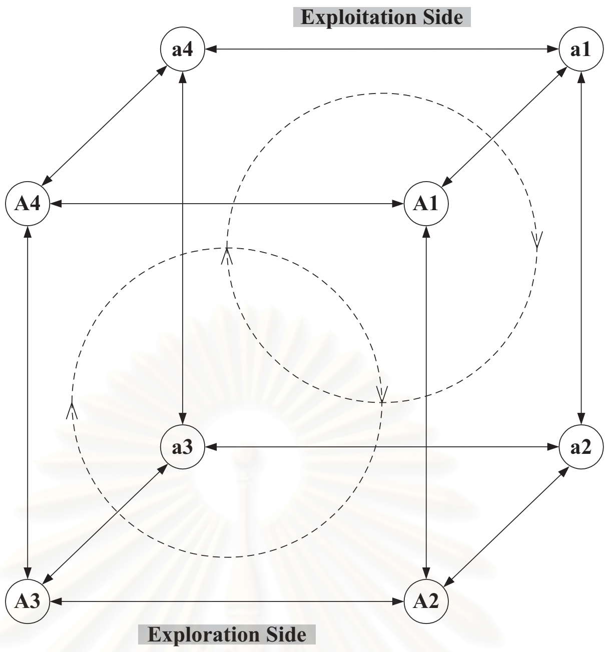

Figure 3.1: A schematic of gradual distributed real-code GAs

Unlike the studies in section 3.3.1, it is not necessary that the optimization of both levels occur concurrently. In addition, the algorithms in this section do not rely on the use of multiple subpopulations.

The article by Freisleben [56] provided a formal description of meta-evolutionary approaches. The term meta-evolutionary approach is used to denote the use of an evolutionary algorithm to optimize the parameter setting of the task level algorithm, which results in two levels of evolutionary algorithms. The article also reviewed some studies in this area.

One of the early remarkable meta-evolutionary approaches was conducted by Grefenstette [57]. This method optimized parameters of a GA by employing an additional level GA. The higher level GA, or meta-level GA, optimized the parameter values of the underlying GA for two different performance measurements: *on-line* performance and *off-line* performance. The higher level GA maintained a population of parameter sets. To evaluate the fitness of each parameter set, the lower level GA using this parameter set was evaluated under five test functions from the study by De Jong [36]. In particular, each lower level GA was subjected to perform each test function for 5000 function evaluations. One thousand lower level GAs were evaluated under the process of parameter optimization. Therefore, the substantial processing time was required to evaluate a population of the meta-level GA. The author also examined the generalization of parameter sets obtained from the meta-level GA on an image registration task. The results showed that the parameter set optimized for on-line performance did slightly better than the optimal parameter set found by De Jong, while the parameter set optimized for off-line performance achieved no significant different from the setting of De Jong.

The concept by Grefenstette has been adopted and extended by several researchers. Shahookar and Mazumder [58] used this concept to optimize a GA for standard cell placement. The parameters optimized by a meta-level GA were the rates for applying three search operators, namely, the crossover rate, the mutation rate and the inversion rate. The types of search operators were not involved in the meta-level optimization. The final results of the meta-level GA provided some insights into the acceptable range of parameter values.

Another study by Mernik *et al.* [59] used a meta-level GA to determine the best combination of crossover operators for a traveling salesman problem. Each position of an individual in the meta-level population represented the type of crossover operators used in one generation of the task level GA. Thus, the length of an individual in the meta-level population was proportional to the number of generations in the task level GA. The rates of applying search operators and other parameters were fixed. The results showed that the best combination found by the meta-level GA was better than that of the random search algorithm.

The method by Grefenstette [57] handled all parameters as discrete values. However, it is possible to design a meta-level algorithm that can optimize both continuous and discrete parameters. An example is the study by B̈ack [60]. B̈ack optimized parameters of a GA by using using a hybrid of GAs and ESs as the meta-level algorithm. By using this scheme, the meta-level algorithm was able to optimize both continuous and discrete parameters simultaneously. Moreover, B̈ack used a class of parallel GAs, called master/slave parallel GAs, for speeding the evaluation of a population of the meta-level algorithm. In order to evaluate the meta-level population, each individual was assigned to the task level GA, which performed on one of the slave processors, and then delivered the final result to the master processor. The problem used in the experiments was a simple optimization problem, the sphere model. The results showed that the obtained parameters from the meta-level optimization were significant better than that of a standard GA.

The meta-evolutionary approaches required several runs of the task level algorithms in order to evaluate parameter sets in the meta-level population. The number of runs depends on the size of the meta-level population and the number of generations of the meta-level algorithm. Thus, these approaches are obviously computational intensive. Some researchers proposed the use of estimation, instead of evaluating several parameter sets in order to reduce the computational cost. Cicirello and Smith [9] introduced the use of a neural network for estimating the fitness values of parameter sets. This work also used two levels of GAs. In the beginning, a neural network was trained to learn the relation between the parameter sets and their fitness values. This neural network was later used for predicting the fitness values of parameter sets in the process of parameter optimization. The author used the largest common subgraph problem in the study. The results showed that the performance of parameter sets from the meta-level algorithm outperformed the hand-tuned parameter set.

The meta-level algorithms of the studies [57, 58, 59, 60] obviously requires several runs of the task level algorithms since the evaluation of each individual requires at least one run of the task level algorithm. Although a technique for reducing the computational cost associated with the fitness evaluation was proposed by Cicirello and Smith [9], this technique still requires a number of runs of the task level algorithm. Clearly, these techniques are computational intensive. They seem to be impractical to some applications. Indeed, it is not clear that the best parameter setting found by the meta-level algorithms can be generalized beyond the problems used in the evaluation of parameter sets.

Apart from using meta-level algorithms to adjust parameter values or select operators, Teller [61] provided a framework of using meta-level GP to automatically design genetic operators. In particular, the meta-level GP evolved a population of crossover operators for the lower level GP. Both levels used a graph-based GP. Unlike the studies mentioned before, a population of the meta-level GP co-evolved with a population of the lower level GP. This is possible since the quality assessment of each crossover operator is calculated from the relative fitness of parent and children. Thus, it is not necessary to perform the entire run of the task level algorithm in order to evaluate merits of each operator. However, only some types of operators are possible to be evaluated in this way. From the experimental results, the evolved crossover outperformed the random one. Kantschik *et al.* [62, 63] extended this concept and performed expanded experiments. Edmonds [64] also investigated a similar concept on a traditional GP.

3.5 Summary

This chapter provided a survey of the important studies devoted to investigate the methods for setting and controlling the parameters of GAs. The review starts with the classifications presented in the literature. Among these classifications, the terms were defined differently. We select two important research areas that are fundamental to our research, and conduct a literature review.

The first area relates to the subpopulation-level methods. Most GAs in this area make use of multiple subpopulations. That is, the main population is divided into few large subpopulations. A different parameter setting or search strategy is assigned to each subpopulation. Some techniques simply exploit the use of multiple parameter settings on different subpopulations. Some techniques employ more sophisticated algorithms or rules to adapt or modify these parameter settings during the run. The decision to adapt the parameter settings is based on the performance of subpopulations.

The second area is the meta-level optimization. The algorithms in this category use some meta-level algorithms to optimize the parameters of the task level algorithm. Many proposed techniques use some kinds of evolutionary algorithms as meta-level algorithms. The evaluation of each individual in the meta-level population is done by assigning this individual as the parameters of the task level algorithm, and the performance of the task level algorithm is then used as the fitness of this individual.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER IV

Adaptive Parameter Control Algorithm

Finding a feasible parameter set of a GA for a given task is a time-consuming job. Many users rely on trying a number of parameter combinations in several experiments. One parameter combination requires at least one run of a GA in order to determine its efficiency. In fact, it had better use several runs to evaluate a parameter setting. The feedback is then used to adjust the parameters. In a large parameter space, users sometimes spend a lot of time in tuning the GA parameters rather than solving a problem. If the time is concerned, searching through various parameter combinations during the run of a GA is more promising. A number of parameter sets are evaluated, and adjusted in the same run. This requires a special algorithm, or some heuristic rules in order to adjust the parameters effectively. This chapter investigates an algorithm that conforms to this idea. The algorithm adjusts the GA parameters while the search is ongoing. This algorithm is denoted as 'an Adaptive Parameter Control Algorithm' (APCA). The concept of APCA is largely based on our articles [65, 66].

This chapter starts by describing the parameters that are found in most GAs. Then, we will discuss about the parameters that are used in the parameter adaptation. Thereafter, the concept of the algorithm is presented.

4.1 Parameters in Genetic Algorithms

There are some decisions that have to be made before applying a GA to a particular problem. The following decisions are found in most GAs.

1. *Choose the encoding*: The encoding defines how to represent a solution to the problem by using a structure containing decision variables. A particular solution can be represented by an assignment of values to the decision variables. Traditionally, solutions are represented by binary strings. However, there is continuous development of the encoding schemes. Several

applications of GAs have used other representations, e.g. graphs, real-valued vectors, integer permutation. The choice of representation should match with the nature of the problem. Moreover, any possible solutions can be represented by the encoding.

2. *Design the fitness function*: The fitness function assigns a score to any possible solution. Given two individuals, the fitness function should be able to distinguish the better one from another.
3. *Choose the operators*: There are two main types of operators. The first is the selection method. There are several proposed selection mechanisms. The general concept is to replicate the highly fit individuals, and remove the infeasible individuals. The second is the genetic operators. The genetic operators are used to generate new offspring from the selected individuals. There are two major operators in GAs : crossover and mutation.
4. *Choose the parameters*: After selecting the operators, the operator probabilities have to be determined by users. The rate at which the operator is applied is controlled by the operator probability. Another parameter is the population size.

The parameters that are controlled and adjusted by the proposed algorithm are referred to as '*adaptive parameters*'. In addition, a set of *adaptive parameters* is called an '*adaptive set*'. From the list of decisions mentioned before, the first and the second decisions are largely based on the nature of the problem. Therefore, both decisions are not included in the adaptive set. Most of this research will be based on the binary string representation. However, the concept can be applied to other representations which will be illustrated in an examination timetabling problem (Chapter 6).

Four parameters are included in the adaptive set. These parameters have been well recognized that their settings play an important role in determining the success and efficiency of GAs. The detail of these adaptive parameters and their options are as follows:

```
0.  subroutine onepoint($\mathcal{A}, \mathcal{B}$)
1.  sample $\mu \in U(1, l-1)$
2.  for $i = 1$ to $l$ do
3.      if $i > \mu$
4.          $a'_i = b_i$
5.          $b'_i = a_i$
6.      else
7.          $a'_i = a_i$
8.          $b'_i = b_i$
9.      end
10. end
11. return($\mathcal{A}', \mathcal{B}'$)
```

Figure 4.1: One-point crossover

1. *Crossover operator*: There are five crossover operators used in this study.
   (a) One-point crossover : This is the simplest one. A single crossover point is randomly selected. The bits up to the crossover point are copied to the alternate child, while the remaining bits are copied to the respective child. Let $\mathcal{A}$ and $\mathcal{B}$ are l-bit binary strings.
   $\mathcal{A} = (a_1, ..., a_l) \in I = \{0,1\}^l$, $\mathcal{B} = (b_1, ..., b_l) \in I = \{0,1\}^l$
   Pseudo-code of the one-point crossover is shown in Figure 4.1.
   (b) Two-point crossover : Two crossover points are randomly selected. Pseudo-code of the two-point crossover is presented in Figure 4.2.
   (c) Uniform crossover with a probability of 0.5 : This operator is introduced by Ackley [67]. However, it is often attributed to Syswerda [68]. Pseudo-code of the uniform crossover is shown in Figure 4.3. The value P_x is set to 0.5. This has been accepted as the standard setting for the uniform crossover.
   (d) Uniform crossover with a probability of 0.1 : The value $P_x = 0.1$ is used.
   (e) Uniform crossover with a probability of 0.2 : The value $P_x = 0.2$ is used. The uniform crossover with probabilities other than 0.5 is inspired by the studies [69, 70].
2. *Crossover rate* (P_c): Normally, this value is a continuous variable ranging

```
0.  subroutine $twopoint(\mathcal{A}, \mathcal{B})$
1.  $sample\ \mu_1, \mu_2 \in U(1, l-1), \mu_1 < \mu_2$
2.  for $i = 1$ to $l$ do
3.      if $i > \mu_1$ and $i \leq \mu_2$
4.          $a'_i = b_i$
5.          $b'_i = a_i$
6.      else
7.          $a'_i = a_i$
8.          $b'_i = b_i$
9.      end
10. end
11. $return(\mathcal{A}', \mathcal{B}')$
```

Figure 4.2: Two-point crossover

```
0.  subroutine $uniform(\mathcal{A}, \mathcal{B})$
1.  for $i = 1$ to $l$ do
2.      $sample\ \mu \in U(0, 1)$
3.      if $\mu \leq P_x$
4.          $a'_i = b_i$
5.          $b'_i = a_i$
6.      else
7.          $a'_i = a_i$
8.          $b'_i = b_i$
9.      end
10. end
11. $return(\mathcal{A}', \mathcal{B}')$
```

Figure 4.3: Uniform crossover

from 0 to 1. In order to reduce the search space of parameter combinations, this value is discretized. Five possible values for the crossover rate are used ranging from 0.2 to 1 in increments of 0.2. The zero rate is omitted in order to guarantee that at least one search operator is working (if the mutation is disabled).

3. *Mutation operator*: Each bit has a chance to be modified with the probability controlled by the mutation rate. Thus, mutation may occur several times on one individual. There are five mutation operators used in the experiments.
    (a) Invert a bit : The bit at the mutation point is flipped to the opposite value. Pseudo-code is shown in Figure 4.4.

```
0. **subroutine** $invert(\mathcal{A})$
1. **for** $i = 1$ to $l$ **do**
2.     $sample\ \mu \in U(0,1)$
3.     **if** $\mu < mutation\_rate$
4.         $a_i = \neg a_i$
5.     **end**
6. **end**
7. $return(\mathcal{A})$
```

Figure 4.4: Inverting mutation

(b) Swap two values : Another point is randomly selected. Then, the positions of two bits are swapped. Pseudo-code is presented in Figure 4.5.

(c) Random bit value : The bit at the mutation point is set to the new value. Pseudo-code is shown in Figure 4.6. The value P_x is set to 0.5. Therefore, the probabilities of being '0' or '1' are equal.

(d) Random bit value with a bias toward '0' : The new value at the random mutation position is biased toward zero. The probability of being zero is 0.9. In pseudo-code (see Figure 4.6), the P_x is set to 0.9.

(e) Random bit value with a bias toward '1' : The new value at the random mutation position is biased toward one. The probability of being one is 0.9. In pseudo-code (see Figure 4.6), the P_x is set to 0.1.

4. *Mutation rate* (P_m): Like the crossover rate, this value is also discretized. Six mutation rates are allowed varying from 0 to 0.1 in increments of 0.02. It is interesting to note that the crossover rate and the mutation rate are discretized in order to simplify the application of search operators in the meta-level algorithm. It is unnecessary to have two separate sets of search operators in order to handle the concrete parameters and the discrete parameters. An example of the work that requires two sets of search operators is shown in the study by B̈ack [60].

```
0.  **subroutine** $swap(\mathcal{A})$
1.  **for** $i = 1$ to $l$ **do**
2.      $sample\ \mu \in U(0,1)$
3.      **if** $\mu < mutation\_rate$
4.          $sample\ j \in U(1,l), j \neq i$
5.          $temp = a_i$
6.          $a_i = a_j$
7.          $a_j = temp$
8.      **end**
9.  **end**
10. $return(\mathcal{A})$
```

Figure 4.5: Swap mutation

```
0.  **subroutine** $random(\mathcal{A})$
1.  **for** $i = 1$ to $l$ **do**
2.      $sample\ \mu \in U(0,1)$
3.      **if** $\mu < mutation\_rate$
4.          $sample\ \epsilon \in U(0,1)$
5.          **if** $\epsilon \leq P_x$
6.              $a_i = 0$
7.          **else**
8.              $a_i = 1$
9.          **end**
10.     **end**
11. **end**
12. $return(\mathcal{A})$
```

Figure 4.6: Random mutation

4.2 Development of an Adaptive Parameter Control Algorithm (APCA)

The proposed algorithm is called as 'an Adaptive Parameter Control Algorithm' (APCA). The idea is to consider the problem of parameter adjustment as an optimization problem. A GA is used to solve this problem. Thus, there are two levels of GAs. The meta-level GA adjusts the parameters, while the base-level GA (task level GA) solves the given problem. The meta-level GA operates on a population of parameter settings of the base-level GA. Each individual in the population represents one configuration of the base-level GA. At the bottom level, the base-level GA operates on a population of individuals which represent possible solutions to the problem. The parameter settings of this level GA are adopted from the population of the meta-level GA. Then, the performance is used as the fitness values for the meta-level algorithm.

The parameter adjustment by the meta-level GA is done in a single run of the base-level GA. That is, a population of the meta-level GA co-evolves with a main population of the base-level GA. Inevitably, a number of individuals in a population of the meta-level GA have to be evaluated in just one run of the base-level GA. This is impractical for a single population GA. In order to realize this, a coarse-grained model of parallelization is used to evaluate a population of parameter settings simultaneously. By using a coarse-grained model, several subpopulations evolve simultaneously on different processors. Each processing node can be assigned with a different parameter setting. Accordingly, the evaluation of parameter sets can be parallelized.

The proposed method differs from some studies [57, 60, 9, 59, 58] which try to find an optimal or near optimal parameter set for a particular problem. Those studies are relied on a substantial number of GA trials in order to evaluate a population of meta-level algorithms. Obviously, the time used to find an optimal or near optimal parameter set is much larger than the time used for solving a given problem by a GA itself. Moreover, it is in doubt whether the obtained parameter sets can be generalized beyond the problem instances used in the evaluation of parameter sets. To precisely evaluate each parameter set, the assessment needs an adequately number of runs on the representative problem instances. In many situations, choosing an adequately number of runs and the problem instances is not trivial. Moreover, using too much in either value will result in a waste of computation resources. This issue was also discussed in [71]. In contrast to those methods, our method tries to find some feasible parameter sets between the problem solving is ongoing. Although the evolved parameter sets from the meta-level GA may not be optimal, these parameter sets are expected to work well. The advantage is that small computational cost is added to a conventional GA.

The meta-level GA starts with candidate parameter sets. The chromosome representation of each parameter set is a vector of integer numbers. Let us assume that $\mathcal{S} = (s_1, \ldots, s_l)$ $(s_i \in [a_i, b_i] \subset I^+, i = 1, \ldots, l)$ is an integer chromosome. The value of a particular gene (s_i) denotes the value of the i^{th} parameter. A population of parameter sets at time t is $\mathcal{P}^t = \{\mathcal{S}_1^t, \mathcal{S}_2^t, \ldots, \mathcal{S}_n^t\}$.

Like a traditional GA, the meta-level GA iterates through the following steps:

1. *Evaluation*: The evaluation of each parameter set is based on the execution of the task level GA. Since several parameter sets must be evaluated simultaneously in each generation, the population of the task level GA is divided into multiple subpopulations. Then, evaluate parameter sets on these subpopulations.
   We adopt a coarse-grained parallel GA for improving the evaluation of parameter sets. In a coarse-grained parallel GA, the subpopulations are spatially distributed among the processing nodes. These subpopulations evolve independently and concurrently. The evaluation of parameter sets is realized by assigning these subpopulations with different parameter sets. In the beginning, the number of parameter sets is equal to the number of subpopulations. Let $f(\mathcal{S}_j^t)$ be the fitness of the parameter set $\mathcal{S}_j^t$ measured on the subpopulation j. In this research, the fitness $f(\mathcal{S}_j^t)$ is the average fitness value of the task level GA of the subpopulation j.
2. *Selection and Breeding*: Since there is no a global processor to handle the selection and produce new parameter sets, we use some local selection rules for selecting some parameter sets for breeding. The basic idea is to eliminate the inferior parameter sets and try to create better parameter sets. Each subpopulation compares its best parameter set with the neighboring subpopulations. The subpopulations with inferior parameter sets attempt to produce new parameter sets by using genetic operators.

Specifically, each subpopulation sends a pair of a parameter set and its fitness value to other subpopulations for comparison. In this study, all subpopulations are connected by using the one-way ring topology. Let $p(j)$ be the adjacent subpopulation of the subpopulation j. The subpopulation j will receive a pair of parameter set and fitness value, $< \mathcal{S}^t_{p(j)}, f(\mathcal{S}^t_{p(j)}) >$, from the subpopulation $p(j)$.

In the case of maximizing the fitness function, two new parameter sets are produced as follows:

$$\{\mathcal{S}^{t+1}_{j1}, \mathcal{S}^{t+1}_{j2}\} = \begin{cases} \mathcal{M}(\mathcal{C}(\mathcal{S}^t_j, \mathcal{S}^t_{p(j)})), & \text{if } f(\mathcal{S}^t_{p(j)}) > f(\mathcal{S}^t_j); \\ \{\mathcal{S}^t_j, \mathcal{S}^t_j\}, & \text{otherwise.} \end{cases}$$

The operator $\mathcal{M}$ is mutation, while the operator $\mathcal{C}$ is crossover. From the rule, if the received parameter set is better than the local parameter set, two distinct parameter sets are produced. This raises a new question of how to select the best parameter set from both parameter sets. We have found empirically that the strategy of evaluating both parameter sets and selecting the best one is better than randomly choosing one parameter set. Namely, each parameter set is evaluated on each half of the subpopulation. When the time for parameter exchange is reached, a comparison between two parameter sets is made; then the best parameter set is selected.

$$\mathcal{S}^{t+1}_j = \begin{cases} \mathcal{S}^{t+1}_{j1}, & \text{if } f(\mathcal{S}^{t+1}_{j1}) > f(\mathcal{S}^{t+1}_{j2}); \\ \mathcal{S}^{t+1}_{j2}, & \text{otherwise} \end{cases}$$

Pseudo-code of APCA for each node is shown in Figure 4.7. In order to make a comparison, pseudo-code of a conventional coarse-grained model is shown in Figure 4.8. In the beginning of APCA, each node creates one parameter set as a member of the population of the meta-level GA. Therefore, the number of individuals in the meta-level GA is equal to the number of nodes.

After the selection and breeding process of the meta-level GA, two new offspring are produced. In order to evaluate both offspring, each offspring is applied to each half of the population. The average fitness values of both halves are compared. Then, the best parameter is selected.

4.3 Summary

This chapter introduces the concept of an adaptive parameter control algorithm. The idea is to consider the parameter adjustment as an optimization problem, and use another GA to solve it. Four parameters are controlled by the proposed algorithm, namely crossover operator, mutation operator, crossover rate and mutation rate. The progress of the meta-level GA occurs in parallel with the base-level GA. The optimization by the meta-level algorithm is done in a single GA trial. Indeed, APCA is much different from other proposed meta-level optimization techniques that rely on several GA trials.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

```
0.  initialize the population, $\theta$                /* population of the task level GA */
1.  initialize the parameter set, $\mathcal{S}$
2.  while generation < max_generation
3.      evaluate $\theta$
4.      if one parameter set exists
5.          apply genetic operators determined by $\mathcal{S}$
                to create the new population, $\acute{\theta}$
6.      else                                           /* two parameter sets */
7.          apply genetic operators determined by $\mathcal{S}_1$
                to create the fist half of the new population, $\acute{\theta}_1$
8.          apply genetic operators determined by $\mathcal{S}_2$
                to create the second half of the new population, $\acute{\theta}_2$
9.          merge $\acute{\theta}_1$ and $\acute{\theta}_2$ to $\acute{\theta}$
10.     end
11.     replace $\theta$ with $\acute{\theta}$
12.     if an interval of $\mathcal{K}$ generations is reached
13.         Migration
14.         Parameter_adaptation
15.     end
16.     generation = generation + 1
17. end
18.
19. subroutine Migration
20.     send and receive migrants
21.     add migrants to $\theta$
22. end
23.
24. subroutine Parameter_adaptation
25.     if two parameter sets exist
26.         select $\mathcal{S}$ from $\mathcal{S}_1$ and $\mathcal{S}_2$
27.     end
27.     send $< \mathcal{S}, f(\mathcal{S}) >$ to the adjacent node
28.     receive $< \acute{\mathcal{S}}, f(\acute{\mathcal{S}}) >$
29.     if $\acute{\mathcal{S}}$ is better
30.         $\{\mathcal{S}_1, \mathcal{S}_2\} = \mathcal{M}(\mathcal{C}(\mathcal{S}, \acute{\mathcal{S}}))$
31.     end
32. end
```

Figure 4.7: Pseudo-code of APCA in each node

```
0.  initialize the population, $\theta$
1.  while generation < max_generation
2.      evaluate $\theta$
3.      apply genetic operators to create the new population, $\acute{\theta}$
4.      replace $\theta$ with $\acute{\theta}$
5.      if an interval of $\mathcal{K}$ generations is reached
6.          Migration
7.      end
8.      generation = generation + 1
9.  end
10.
11. subroutine Migration
12.     send and receive migrants
13.     add migrants to $\theta$
14. end
```

Figure 4.8: Pseudo-code of a conventional coarse-grained model in each node

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER V

Experimental Results

In the previous chapter, the concept of APCA is presented. The algorithm is designed to help GA practitioners from the problem of parameter setting. In this chapter, the framework of APCA is seen in action. Like many empirical studies, this chapter starts by defining the test suite used in our experiments. We also clarify the performance measurements along with other algorithms that are used to make a comparison. We then present the results on test functions.

Following that, some additional experiments are carried out to gain some insights into the proposed algorithm. We examine the sensitivity of the parameters in the meta-level GA. The intention is to investigate how the change of meta-level parameters affects the performance of the algorithm. We also use a problem generator to test the algorithm. The problem generator is capable of producing a number of problem instances with different characteristics. The results would suggest whether the algorithm can adapt itself to different problem instances. Finally, some results are presented to validate a hypothesis about the evolution of parameters. An experiment is designed to show that the evolutionary process of the meta-level algorithm is capable of improving the initial parameter sets to the better ones.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

5.1 Performance on Test Problems

5.1.1 Test Problems

The first part of this chapter provides some experimental results on test problems. There are five test problems used in this study.

1. *Onemax problem*: The fitness function simply returns the number of '1' bits in an individual. The objective is to maximize the fitness function. Thus, the optimal solution is an individual with all '1' bits. The individual length

Table 5.1: A two-bit subfunction

Binary Code	Function Value
00	2
01	1
10	0
11	3

is 300 bits.

2. *Contiguous bits problem*: This problem was used by Syswerda [68]. The fitness function returns the number of '1' bits that at least one adjacent bit is '1'. Like the onemax problem, the optimal solution is an individual with all '1' bits. The individual length is also 300 bits.
3. *Minimal deceptive problem*: The function is the concatenation of 50 copies of a two-bit subfunction (see Table 5.1). The detail can be found from [18].
4. *Zero/one multiple knapsack problem*: We use the 'sento1-60' problem which was introduced in the article by Senyu and Toyoda [72]. The number of knapsacks is 30, while the number of objects is 60. The known optimal solution is 7772. The problem instance is available from OR-Library [73]. The fitness function proposed in [74] is used.
5. *Royal road problem*: This problem was introduced by Holland [75]. It was designed as a function that would be simple for a GA, but difficult for a hillclimber. A description of the problem was presented in the article by Jones [76]. By using Holland's default settings (see Table 5.2), the optimal solution is 12.8. The individual length is 240 bits.

5.1.2 Algorithms and Performance Measurements

APCA is compared against five algorithms. These algorithms are based on a coarse-grained model for parallelization. The difference between these algorithms is how to determine or adjust the four adaptive parameters (see the previous chapter for their details). The other parameters (e.g. migration rate, topology)

Table 5.2: Holland's default settings

Variable	Value
b	8
g	7
k	4
m^*	4
u	0.3
u^*	1.0
v	0.02

are identical.

The details of these algorithms are as follows:

1. *Uniform random algorithm*: A single parameter set is randomly generated at the beginning of the algorithm. All subpopulations use this parameter set.
2. *Diverse random algorithm*: At the beginning, each subpopulation randomly creates its own parameter set. These initial parameter sets are used throughout the run without any modifications. This algorithm is comparable to APCA without the parameter adaptation. The similar techniques of using different static parameter sets on the multiple subpopulations were also presented in some studies [53, 55]. However, their parameters in each subpopulation were intuitively selected rather than randomly set.
3. *Static algorithm*: This algorithm uses a static parameter set from the systematic study by De Jong [36]. This parameter setting is sometime accepted as a standard parameter setting. It has been adopted by many researchers (e.g. [77, 78, 79, 80, 81]).
4. *Strategy adaptation*: The algorithm is adopted from the study by Schnecke and Vornberger [48]. The parameter adaptation is done on the subpopulation level. Each subpopulation employs a different parameter set. After a fixed interval, theses parameter sets are ranked. Each parameter set is then

adapted to the values of the next best strategy. Unfortunately, the article by Schnecke and Vornberger [48] did not exactly report how to adjust the values of a parameter set to the next best one. We will assume that the parameter sets are randomly generated at the beginning, like APCA. After ranking these parameter sets, each parameter set inherits one randomly selected value from the next best parameter set. The frequency of parameter adaptation is set to the same as APCA.

5. *Adaptive genetic algorithm (AGA)*: This algorithm was proposed by Srinivas and Patnaik [82]. The proposed algorithm controlled only the crossover rate (P_c) and the mutation rate (P_m). Some derived heuristic rules were used to adapt the operator rates.

$$P_c = k_1(f_{max} - f')/(f_{max} - \overline{f}), f' \geq \overline{f} \quad (5.1)$$

$$P_c = k_3, f' < \overline{f} \quad (5.2)$$

$$P_m = k_2(f_{max} - f)/(f_{max} - \overline{f}), f \geq \overline{f} \quad (5.3)$$

$$P_m = k_4, f < \overline{f} \quad (5.4)$$

where $k_1, k_2, k_3, k_4 \leq 1.0$, f_{max} and $\overline{f}$ are the maximum fitness value and the average fitness value of the population respectively, f' is the average fitness value of parents and f is the fitness value of an offspring needed to be mutated.

The operator rates are determined separately for each individual. The crossover operator is the one-point crossover, while the mutation operator is the inverting mutation. The default constants recommended in the article are as follows: $k_1 = 1.0, k_2 = 0.5, k_3 = 1.0$ and $k_4 = 0.5$. However, the use of the recommend values cannot solve the test problems. We found empirically that the following values worked better ($k_1 = 0.6, k_2 = 0.05, k_3 = 0.6, k_4 = 0.05$). Nevertheless, the use of new values can solve only one from five test problems. Thus, we will show the results of AGA in the problem that can be solved.

We adopt two measurements from the study by Deb and Agrawal [83]:

Performance and *Unuse Factor*. The performance is the ratio of the number of runs yielding the optimal solution to the total number of runs, except that the number of runs reaching 1% from the optimal is used in the knapsack problem. The unuse factor (U) is calculated as follows:

$$U = 1 - \frac{g}{g_{max}} \tag{5.5}$$

where g is the number of generations required to solve the problem, g_{max} is the maximum number of generations. The larger the unuse factor, the faster the algorithm is. Note that the unuse factor is originally calculated by using the number of function evaluations. Nevertheless, we use the number of generations instead since it is easier to compute in the parallel environment.

5.1.3 Experimental Design and Implementation

All algorithms mentioned in the previous section are implemented using the same model for parallelization. A coarse-grained model is applied to all algorithms. We implement the parallel algorithms on a cluster of PC workstations with 1GHz Pentium III processors, each with 256 MB of RAM, and running Linux as an operating system. All are linked by a fast ethernet switch (100 Mbps). The number of processing nodes used in the experiments is 8. The program is based on a modified version of LibGA software package [84]. MPICH, a message passing interface standard, is used for providing basic communication functions. This library also provides the mechanism for starting and controlling several processes on multiple processing nodes.

The migration topology is a one-way ring topology (see Figure 5.1). All algorithms use the same parallel parameters. The details about these parallel parameters are as follows:

1. *Migration interval*: The migration interval defines the interval between two

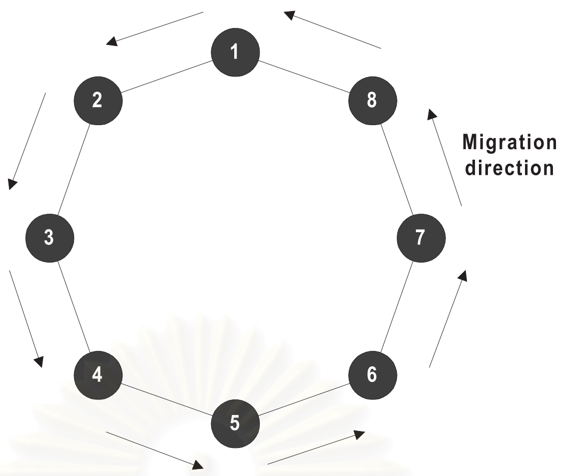

Figure 5.1: The one-way ring topology

migrations in terms of the number of generations. This parameter is set to 5 generations.

2. *Selection policy*: The selection policy specifies how to select individuals for each migration. The migrants may be selected from the best individuals in a subpopulation, or randomly chosen from all available individuals. In the experiments, the selection policy that we used lies between these two extreme cases. The migrants are selected by using the selection function that normally used in the process of GA.
3. *Replacement policy*: The received migrants may replace the worst individuals in the receiving subpopulation, or they may randomly replace any individuals. In the experiments, the migrants are appended to the receiving subpopulation. The rest is then generated by the genetic operators.
4. *Migration rate*: This parameter determines the number of migrants in each migration. The parameter is set to 6 individuals.
5. *Synchronization*: We use the synchronous migration in our experiments. Specifically, every processor will wait until all processors are ready for the migration.

The population size plays an important role in determining the success and the computational cost in finding a solution to a particular problem. Thus, all experiments are conducted over a range of population sizes in order to reduce bias associated with the setting of population size. All reported results are averaged over 20 runs with different random seeds. The maximum number of generations for all experiments is 500. The selection operator is the roulette-wheel selection. Migrants are also selected by this operator.

5.1.4 Results on Test Functions

Figures 5.2 and 5.4 show the performance on the onemax problem and the contiguous bits problem respectively. As mentioned earlier, the results are averaged from 20 runs. Each algorithm is tested on 12 population sizes. Accordingly, the result of each algorithm on a particular problem is based on 240 trials. The results on the onemax problem and the contiguous bits problem are nearly identical. APCA can find the optimal solution in all runs over the range of population sizes. The performance of the diverse random algorithm nearly resembles the performance of the strategy adaptation algorithm. The uniform random algorithm achieves the moderate performance. The static algorithm attains the lowest performance.

Figures 5.3 and 5.5 illustrate the unuse factor on the onemax problem and the contiguous bits problem respectively. The unuse factor graphs indicate that the proposed method has the highest remaining generation number. This means that the proposed method uses the shortest period in finding the optimal solution. The static algorithm has the lowest convergence rate. When increasing the population size, the unuse factor reduces to zero. This means that the static algorithm is unable to find the optimal solution in the given time. Overall, the unuse factor slightly drops as the population size increases.

The performance on the minimal deceptive problem is depicted in Figure 5.6. APCA finds the optimal solution in all runs when the population size increases to 40. The diverse random algorithm requires the population size at least 220 to find the optimal solution in all runs. This algorithm performs slightly better than the

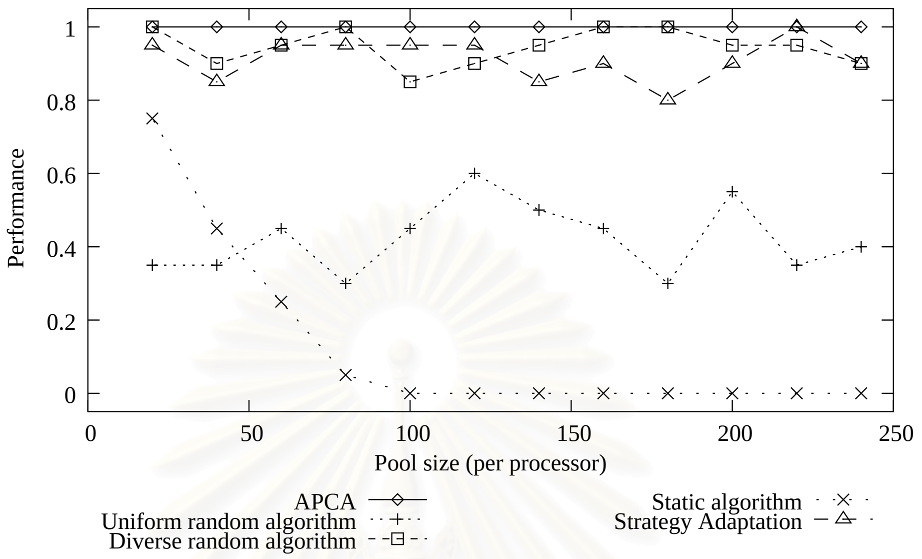

Figure 5.2: Performance for the onemax problem

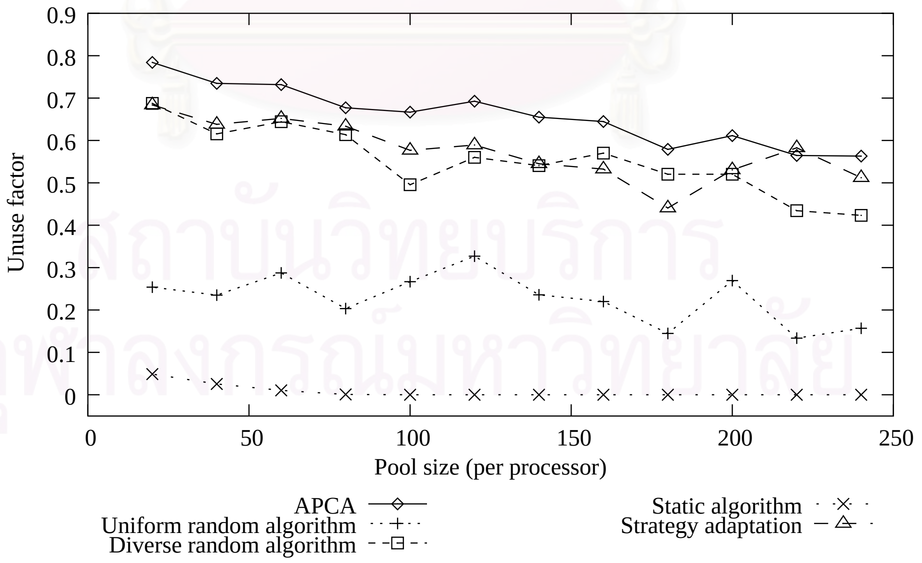

Figure 5.3: Unuse factor for the onemax problem

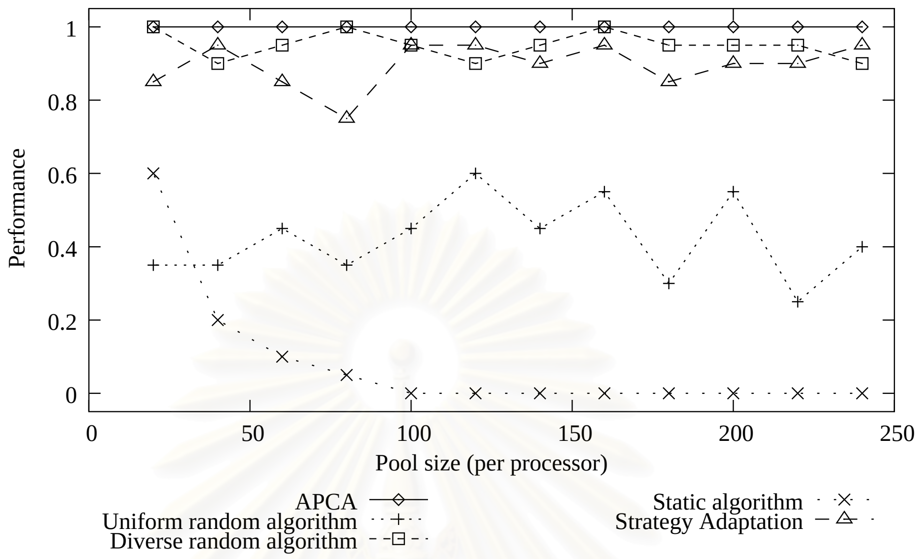

Figure 5.4: Performance for the contiguous bits problem

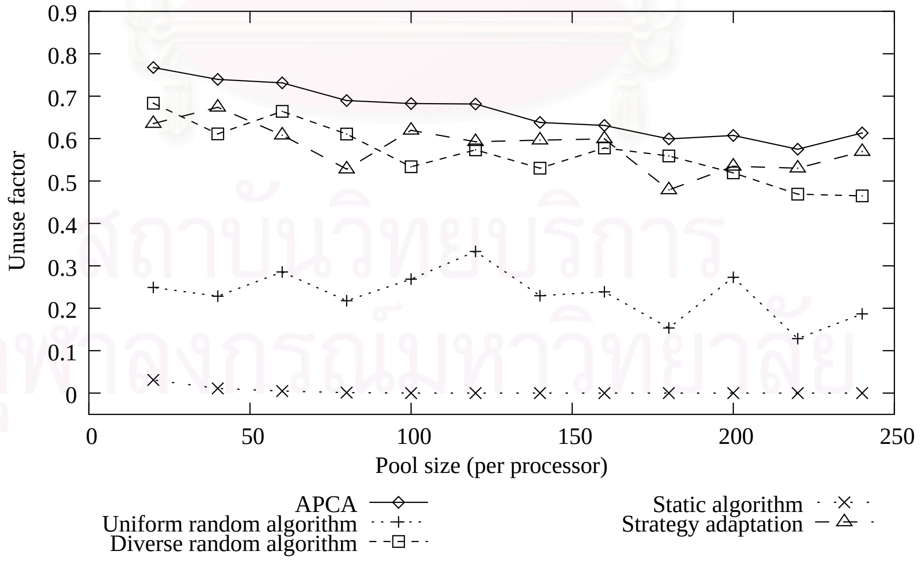

Figure 5.5: Unuse factor for the contiguous problem

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

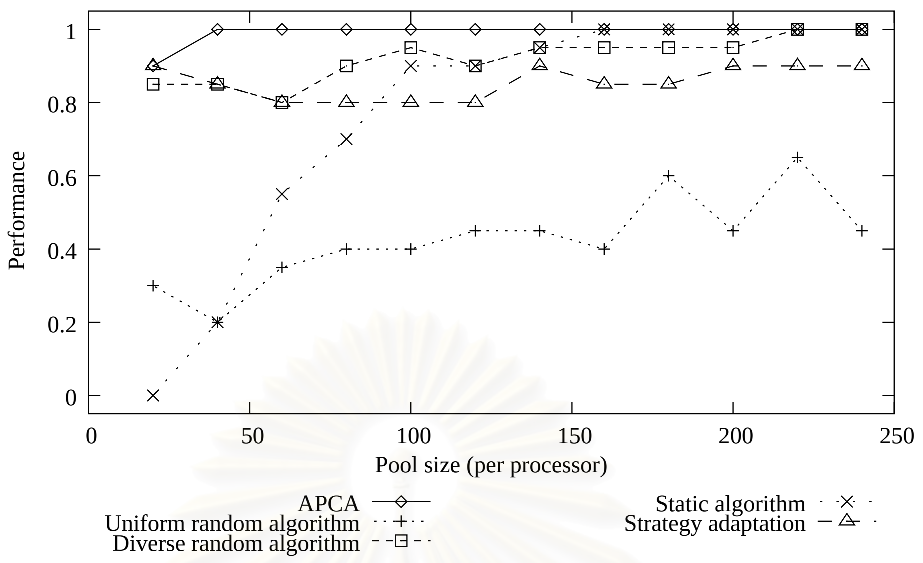

Figure 5.6: Performance for the minimal deceptive problem

strategy adaptation algorithm. For the static algorithm, the performance improves considerably as the population size increases. Moreover, the static algorithm finds the optimal solution in all runs by using the population size at least 160. The uniform random algorithm achieves its best at about 0.65. Again, the unuse factor (See Figure 5.7) indicates that APCA has the highest remaining generation number. Regarding the population size, the proposed method uses the shortest duration in finding the optimal solution.

Figure 5.8 shows the performance on the knapsack problem. APCA achieves significant better performance in finding the optimal solution than the other competing methods. Apart from the results in previous problems, the strategy adaptation algorithm outperforms the diverse random algorithm. Figure 5.9 presents the unuse factor on the knapsack problem. Obviously, the proposed algorithm has the highest remaining generation number.

The performance on the royal road problem is shown in Figure 5.10. The ratios of runs reaching the optimal solution of APCA and the diverse random algorithm are nearly equivalent. From the unuse factor in Figure 5.11, the remaining generation numbers of both algorithms are also approximately

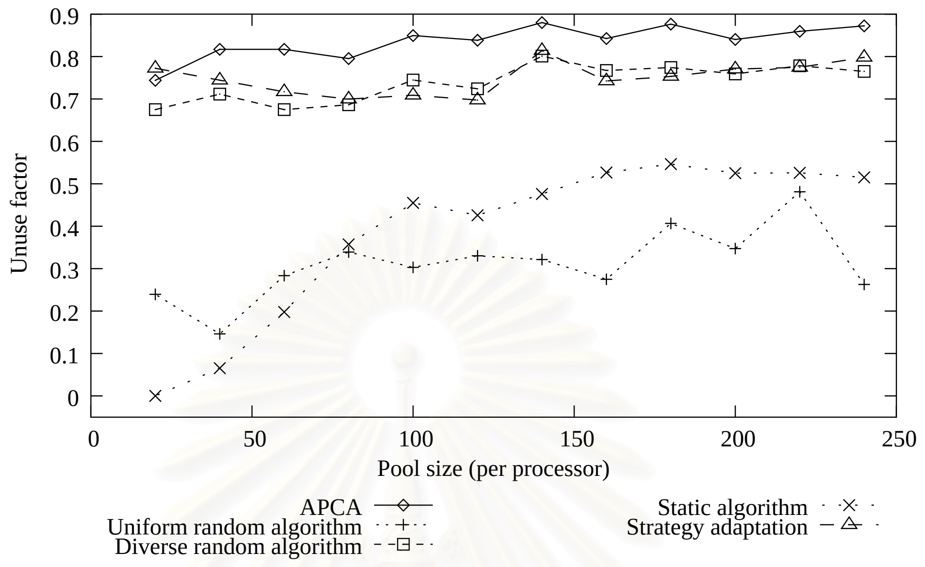

Figure 5.7: Unuse factor for the minimal deceptive problem

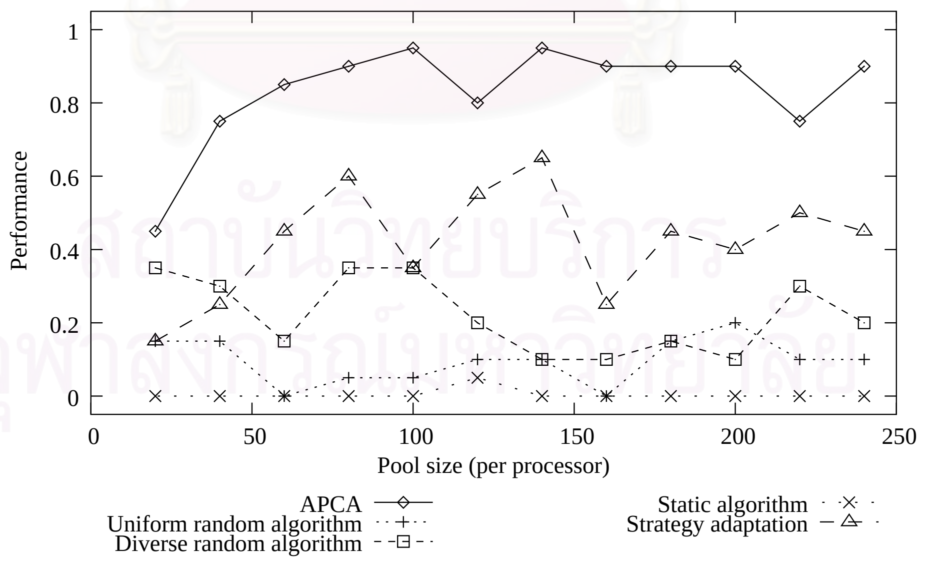

Figure 5.8: Performance for the knapsack problem

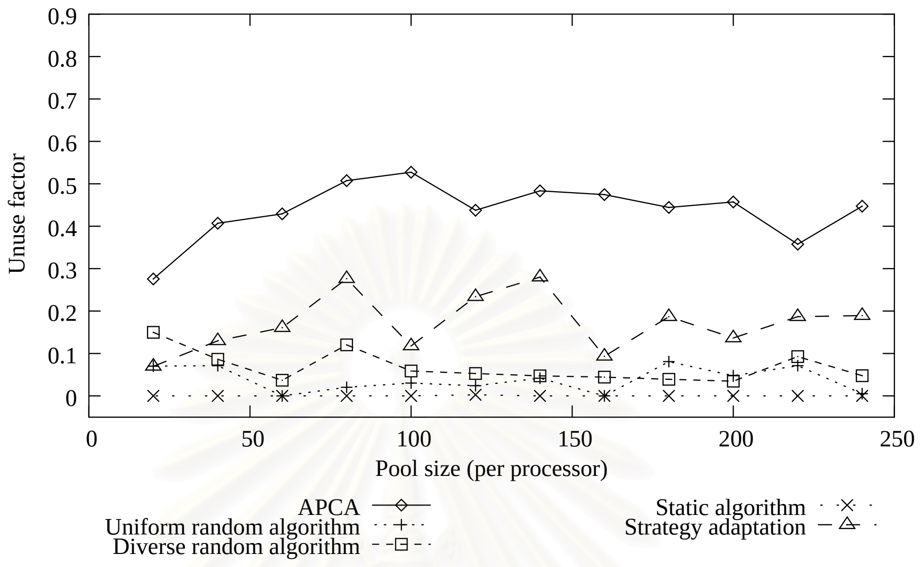

Figure 5.9: Unuse factor for the knapsack problem

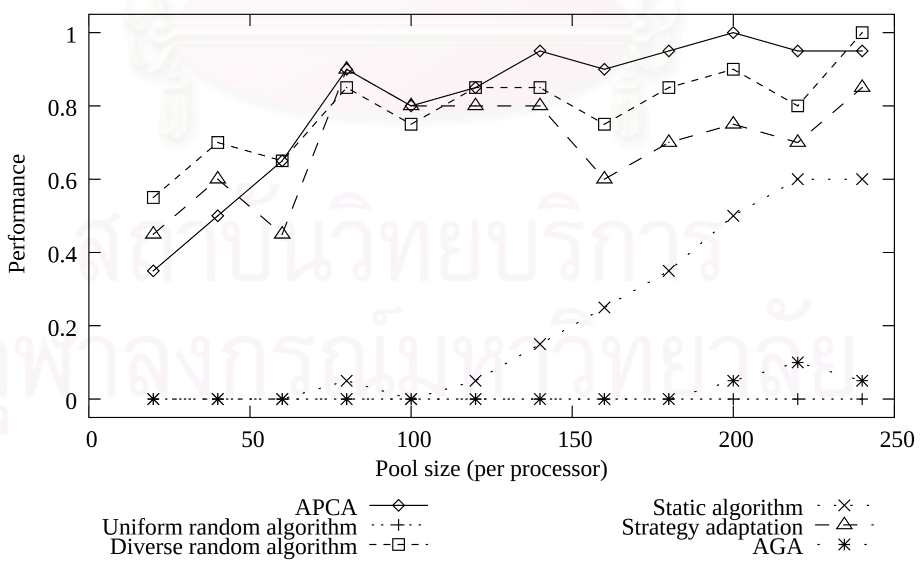

Figure 5.10: Performance for the royal road problem

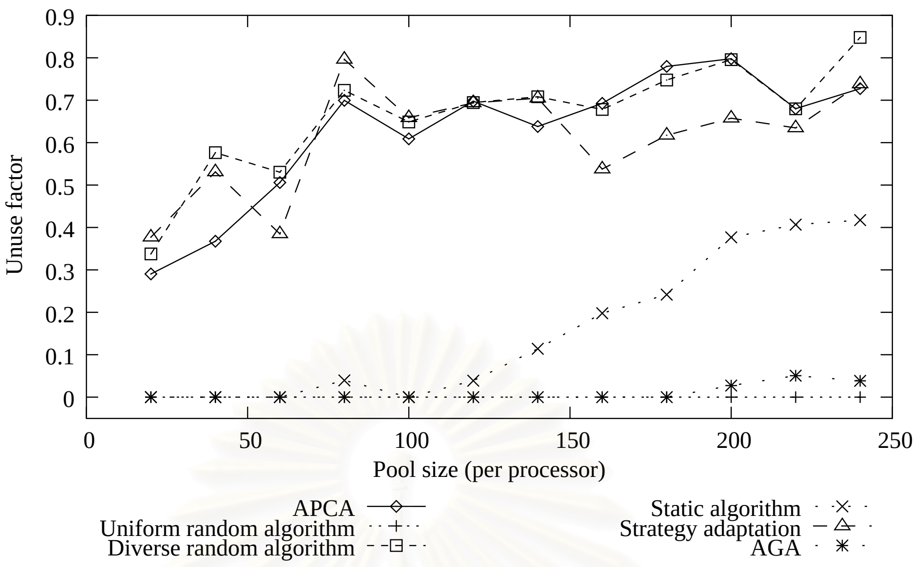

Figure 5.11: Unuse factor for the royal road problem

equivalent. The strategy adaptation is also comparable to both algorithms when the population sizes are small. However, the performance seems not to increase as much as both algorithms when the population sizes increase. Note that the uniform random algorithm cannot find the optimal solution in the given period. The static algorithm is able to find the optimal solution in some runs when the population size is sufficient. AGA can find the optimal solution on this problem, though the result of this algorithm is quite poor. However, this does not mean that AGA cannot solve these test problems. In fact, AGA is designed for multimodal problems. It may converge too slow and cannot find the optimal solutions in the given period.

5.1.5 Compared with a Random Walk Algorithm

APCA starts with a group of randomly generated parameter sets. Then, the process of parameter adaptation is applied to these parameter sets every 5 generations. If the parameter adaptation is disabled, APCA and the diverse random algorithm are exactly the same. In fact, the results from APCA obviously differ from those of the diverse random algorithm. The dynamically change of parameter sets plays an important role in the success of APCA.

In this section, we argue that the change of parameter sets in a systematic way is an important key to the success of APCA. In order to verify this, we compare APCA with a random walk algorithm. The random walk algorithm changes the parameter sets every 5 generations as is done in APCA. However, the change is not directed by any rules, just completely random. If the random walk algorithm has comparable efficiency to that of APCA, the meta-level GA is obviously useless.

The results on five test problems are shown in Figures 5.12, 5.13, 5.14, 5.15 and 5.16. Overall, APCA outperforms the random walk algorithm, except the last problem that both algorithms have comparable efficiency. The results of the random walk algorithm are close to those of the diverse random algorithm rather than APCA.

In fact, the random walk algorithm explores the parameter space more thoroughly than the diverse random algorithm. Approximately 800 modifications of parameters are done by the random walk algorithm in a single run, while the diverse random algorithm utilizes 8 different parameter sets in each run. When comparing with the number of possible parameter combinations (i.e., 750), in some sense, the random walk algorithm would be able to find some feasible parameter sets. However, the improvement of the random walk algorithm from the diverse random algorithm is marginal. We argue that the reason is the lack of a control algorithm in the random walk algorithm. Thus, the modifications of parameters cannot keep feasible solutions. Unlike APCA that also modifies parameters with the same frequency, it has a mechanism for controlling the modifications of parameters. This helps APCA to perform more reliably.

5.1.6 Parameters of the Meta-level GA

This section considers the sensitivity of the parameters of the meta-level GA. Initially, APCA uses both the crossover operator and the mutation operator as its search operators. In this section, three variants of the meta-level GA are examined: one using only crossover (APCA-c), one using only mutation (APCA-

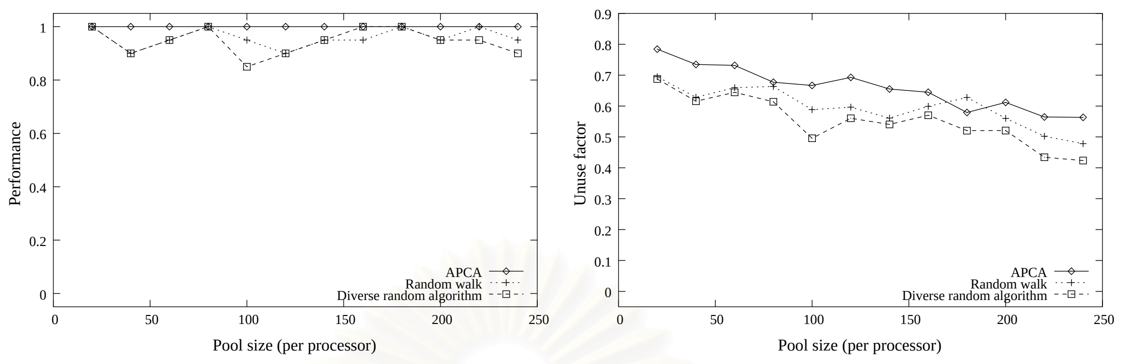

Figure 5.12: A comparison with the random walk algorithm on the onemax problem

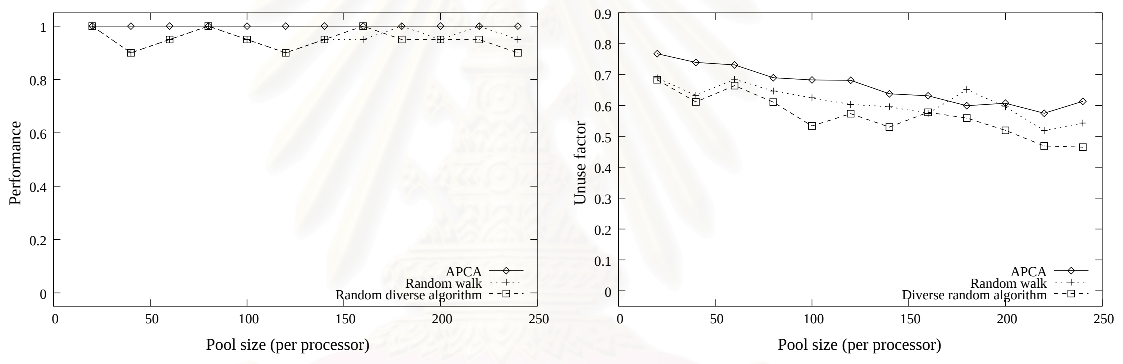

Figure 5.13: A comparison with the random walk algorithm on the contiguous bits problem

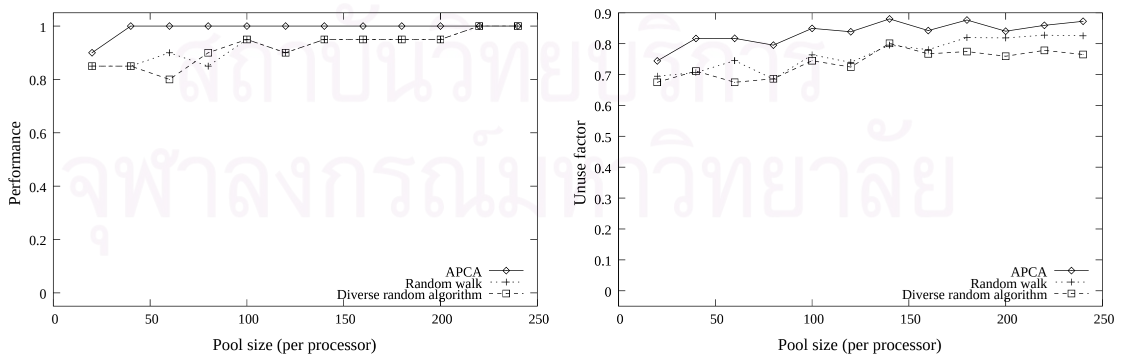

Figure 5.14: A comparison with the random walk algorithm on the minimal deceptive problem

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

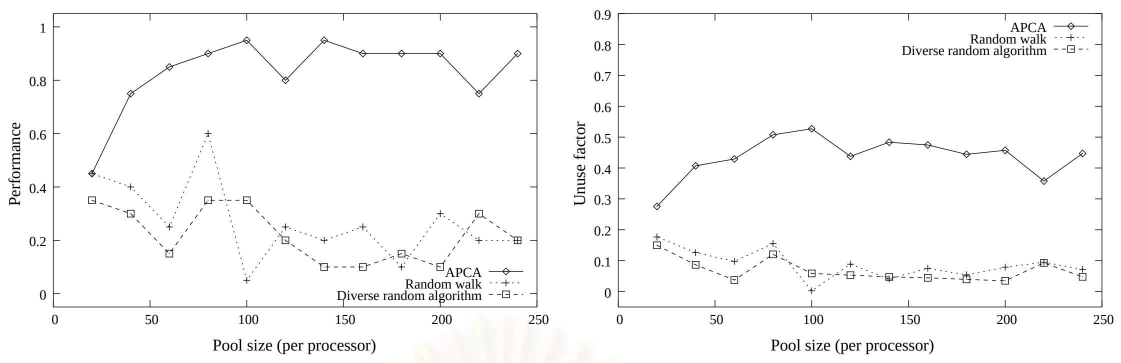

Figure 5.15: A comparison with the random walk algorithm on the knapsack problem

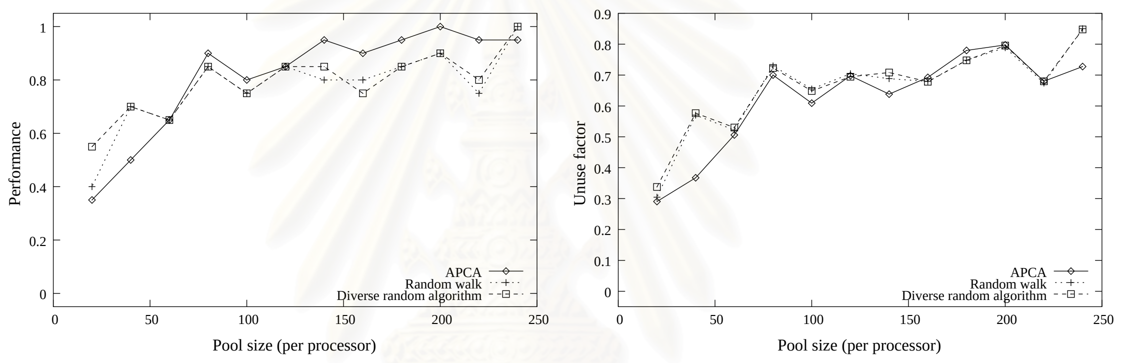

Figure 5.16: A comparison with the random walk algorithm on the royal road problem

m), and the normal setting using both operators (APCA). The experiments in this section intend to investigate how the change of the meta-level parameters affects the overall performance.

The results of three variants of the meta-level GA are presented in Figures 5.17, 5.18, 5.19, 5.20 and 5.21. The results suggest that mutation seems to be more important than crossover. The disabling of crossover in APCA-m does not make any significant difference. Conversely, the disabling of mutation in APCA-c has a detrimental effect on performance. In general, the use of both operators is quite safe. Thus, the normal setting seems to be a promising choice whether crossover is necessary or not.

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

There is one possible reason why the use of crossover alone obtains unsatisfactory performance. When the meta-level population is converged to a particular parameter setting, it is unable to evolve further. Specifically, the application of crossover between two identical individuals still results in the same. If the population is stuck in the local optima, the use of crossover alone cannot escape from the local optima. In contrast, the use of mutation allows the population to evolve further. Accordingly, mutation may help to avoid premature convergence and escape from the local optima. In addition, crossover alone tends to lack adaptability. The task level GA may require a different parameter setting for each stage of evolution. Crossover alone may not adequately response to the stage of evolution. Conversely, the use of mutation enables the meta-level population to evolve further and response to the performance and stage of the task level GA.

The reason mentioned earlier can also explain why APCA performs better than the strategy adaptation algorithm. The strategy adaptation algorithm is, to some degree, close to APCA-c. Each parameter set is adapted to the values of the next best strategy. This can be viewed as each parameter set performs a form of crossover with the next best strategy. No form of mutation is used in the strategy adaptation algorithm. Thus, the algorithm may face with the same problem that happens to APCA-c.

5.2 Generalization on a Test Problem Generator

This section studies the behavior of algorithms when the nature of a problem is changed. The experiments demonstrate how the adaptive algorithms perform when the characteristic of a problem varies. We also consider the results of a static parameter set.

The experiments are realized by using a problem generator. We use the multimodal generator [1, 4, 85] in our experiments. The problem generator is capable of producing a number of random test problems that vary the

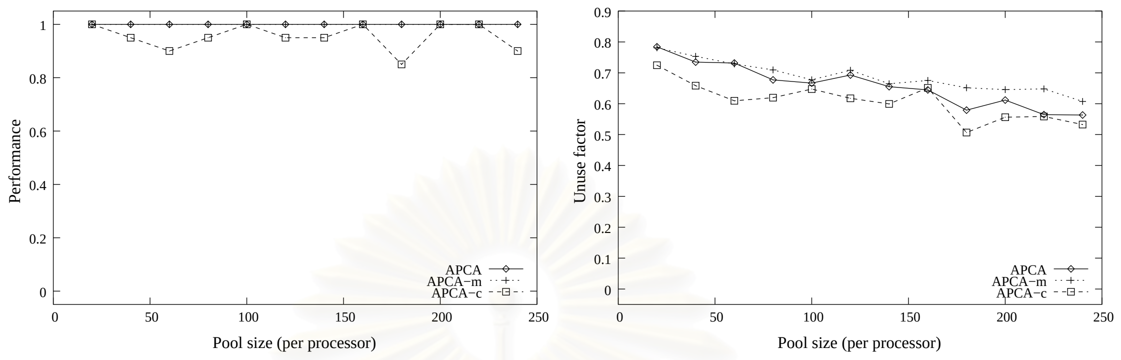

Figure 5.17: Variants of the meta-level GA on the onemax problem

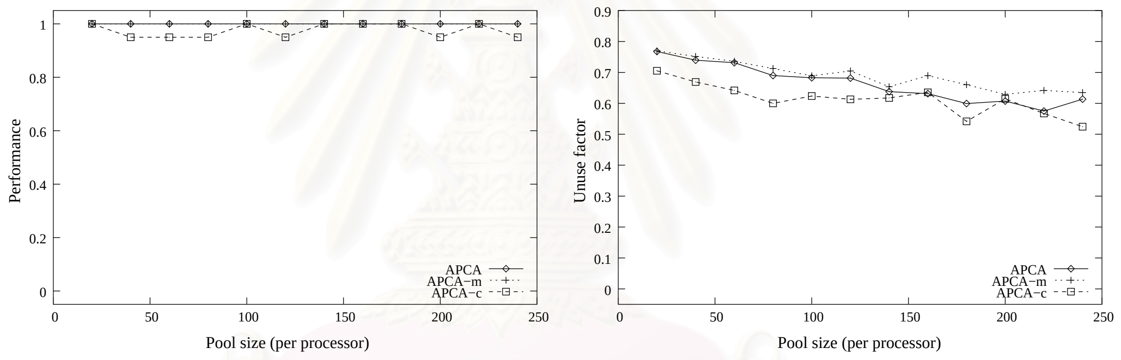

Figure 5.18: Variants of the meta-level GA on the contiguous bits problem

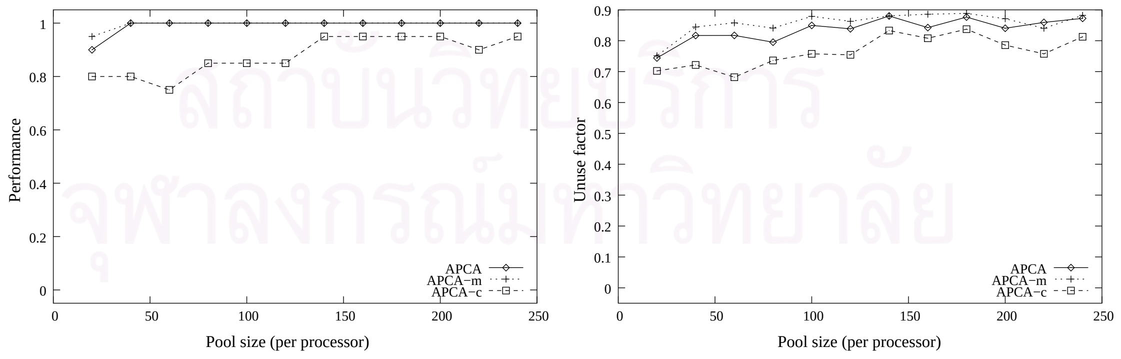

Figure 5.19: Variants of the meta-level GA on the minimal deceptive problem

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

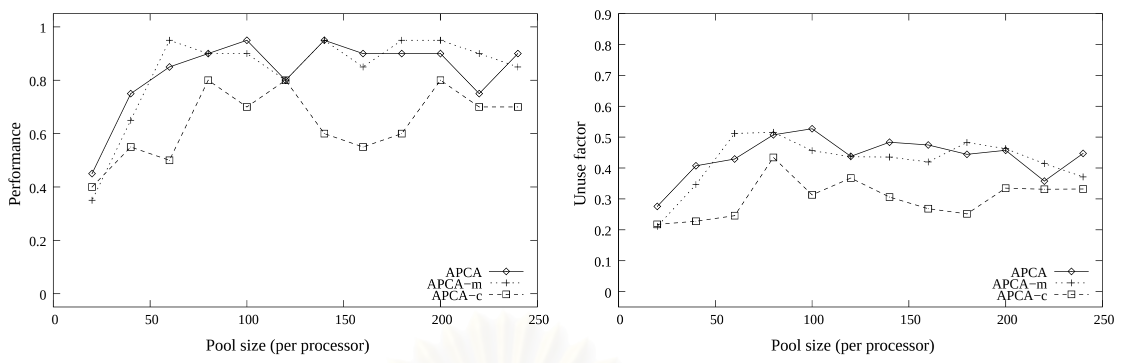

Figure 5.20: Variants of the meta-level GA on the knapsack problem

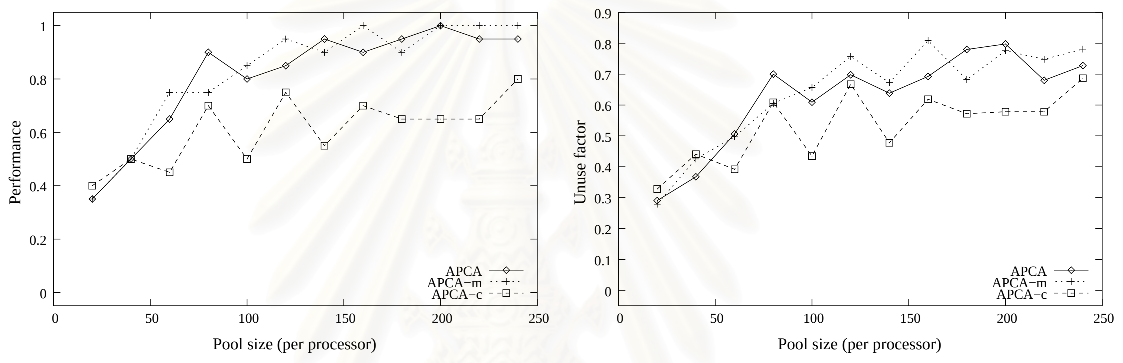

Figure 5.21: Variants of the meta-level GA on the royal road problem

multimodality — the number of peaks. One of the interesting features of this problem generator is that the change of the generator characteristic does not affect the difficulty of the generated instances.

The problem generator creates a set of $\mathcal{P}$ random N-bit strings. These random strings represent the location of the $\mathcal{P}$ peaks (the degree of multimodality) in the search space. The fitness is the number of bits an individual has in common with the nearest peak, divided by N.

$$f(\mathcal{A}) = \frac{1}{N} \max_{i=1}^{P} \{N - Hamming(\mathcal{A}, Peak_i)\} \tag{5.6}$$

Consider a simple 10-bit problem, with two optimal peaks at '00. . . 00' and '11. . . 11'. An example of individuals and their fitness values is shown in Table 5.3.

Table 5.3: Individuals and their fitness values

Individual	Fitness Value
0 111 111 111	0.9
1 000 000 000	0.9
0 000 000 111	0.7
1 111 111 000	0.7
1 111 100 000	0.5
0 000 011 111	0.5

The individuals with 50% 1's and 0's have the lowest fitness, while individuals with mostly 1's or mostly 0's have high fitness. When applying mutation to a highly fit individual located on any peak, the offspring tends to move up or down the peak to a small degree. However, crossover has a different effect. If both highly fit individuals are from different peaks, the offspring are likely to be in the low fitness region.

Spears [4] pointed out that crossover may introduce a detrimental effect on performance on high multimodality problems, while mutation seems to be more promising on such problems. Conversely, crossover helps in improving the GA performance when the multimodality is low (e.g. 1-peak problems). The results that confirm this hypothesis are shown in the studies [1, 4].

The experiments are conducted on two different problem sets. The first set is low multimodality problems (or, to be precise, 1-peak problems). The second set is high multimodality problems (or, to be precise, 500-peak problems). Each set contains 20 randomly generated instances. The problem length is 200 bits. The first experiment investigates the generalization of a static parameter. We start by searching for a promising parameter for the 1-peak problems. Then, we will test this parameter set on the 500-peak problems. The selection of a promising parameter set is carried out by testing 40 randomly generated parameter sets. The performance of each parameter set is averaged on 20 trials, each on one random problem instance. Thus, the total number of runs required for finding a promising

Table 5.4: Results of a static parameter set

Problem	Performance	Unuse factor
1-peak problems	1.0	0.65
500-peak problems	0.0	0.0

Table 5.5: Results of APCA

Problem	Performance	Unuse factor
1-peak problems	1.0	0.44
500-peak problems	0.8	0.12

parameter set is 800. Due to this large number of runs, the population size in the experiments is limited to 100.

From the results, only 4 parameter sets from 40 random parameter sets can reach the optimal solutions. Indeed, only one parameter set can find all optimal solutions. We then test this parameter set on the 500-peak problems. Like the 1-peak problems, there are 20 instances. The results are presented in Table 5.4. It is clear that the use of a static parameter set cannot reach any optimal solutions on the 500-peak problems. The results indicate that the selected parameter cannot generalize to other instances.

We also investigate how well the adaptive algorithms perform on two different sets of problems. By running these algorithms on the 1-peak problems and the 500-peak problems, the results of APCA, AGA and the strategy adaptation algorithm are presented in Tables 5.5, 5.6 and 5.7 respectively.

Table 5.6: Results of AGA

Problem	Performance	Unuse factor
1-peak problems	0.8	0.08
500-peak problems	0.8	0.08

Table 5.7: Results of the strategy adaptation algorithm

Problem	Performance	Unuse factor
1-peak problems	0.2	0.10
500-peak problems	0.0	0.0

For the 1-peak problems, APCA has the highest performance. The algorithm can find all optimal solutions, like the use of a static parameter set. AGA performs quite well although it cannot obtain all optimal solutions. The strategy adaptation algorithm has the lowest performance. For the 500-peak problems, APCA and AGA have equal performance, while APCA has slightly higher unuse factor. The strategy adaptation algorithm cannot find any optimal solutions in these instances.

The convergence curves between APCA and the use of a static parameter are shown in Figures 5.22 and 5.23. It is clear that the use of a static parameter converges faster on the 1-peak problems. However, both algorithms can find all optimal solutions. For the 500-peak problems, the use of a static parameter converges faster in the beginning. After that, the convergence rate of the static algorithm reduces substantially, while APCA has a nearly constant convergence rate. Finally, APCA overtakes the static algorithm.

Although APCA is a bit slower than the use of a highly fit parameter set, its generalization is better. The results also confirm a common intuition about the use of a static parameter set. The parameter set that performs well on particular instances may not generalize to other instances.

5.3 An Observation of the Evolved Parameters

This section proposes a framework for analyzing the internal mechanism of APCA. The objective is to verify whether the evolved parameter sets during the run are better than the initial parameter sets. To achieve this, the framework is divided into two stages. In the first stage, the parameter sets in all nodes

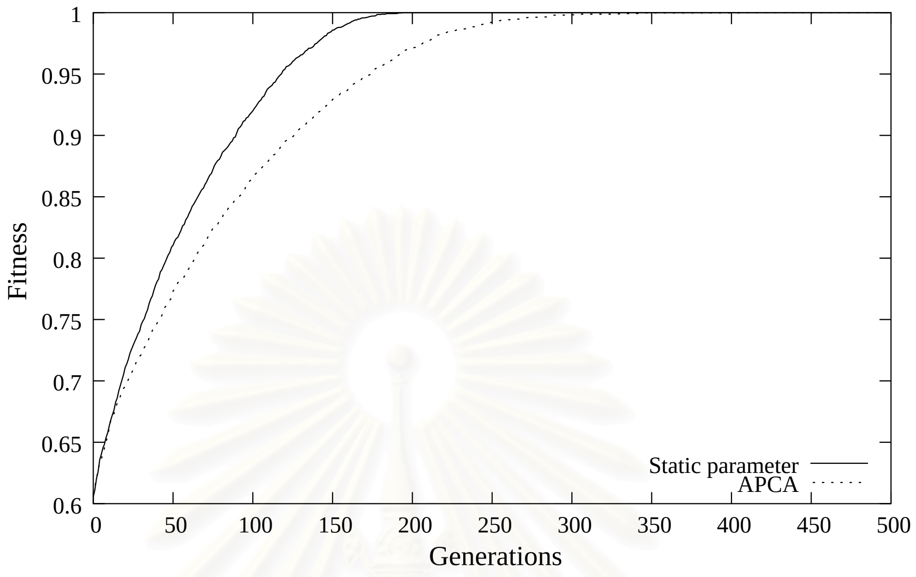

Figure 5.22: Convergence on the 1-peak problems

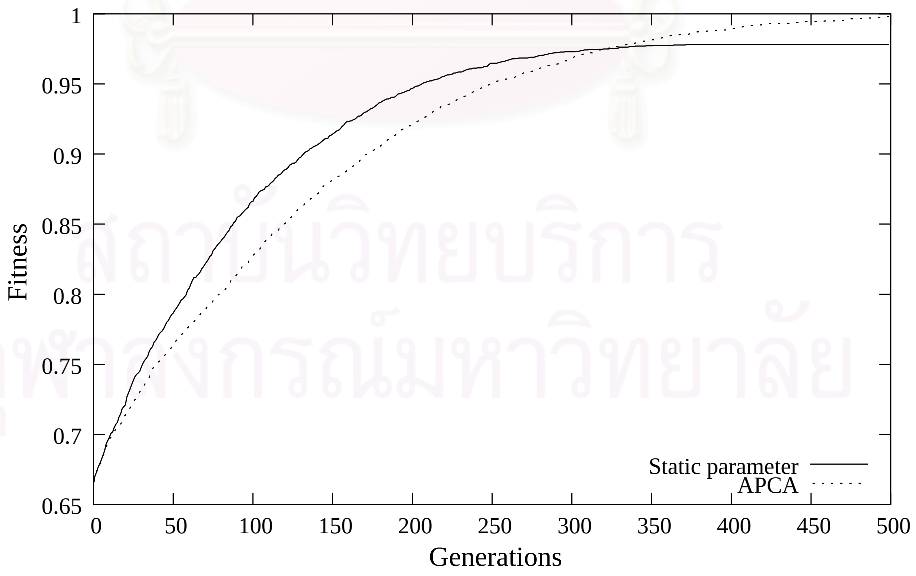

Figure 5.23: Convergence on the 500-peak problems

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

are recorded while APCA is running. If the parameter sets at generation i are considered, one parameter set ($\mathcal{S}$) is randomly selected from all parameter sets in each trial. We run APCA 20 times. Thus, there are 20 parameter sets, $\{\mathcal{S}_n, n = 1, \ldots, 20\}$.

In the second stage, each parameter set ($\mathcal{S}_n$) is evaluated by assigning $\mathcal{S}_n$ to a coarse-grained parallel GA. The configuration of the parallel GA is identical to that is used earlier. All processors use the same parameter set, $\mathcal{S}_n$. This is repeated for every ($\mathcal{S}_n$). By using the same performance measurement, the performance factor is calculated.

We apply this method to the parameter sets at generations 0, 25 and 250. The parameters found in generation 0 represent the initial parameters from the meta-level GA in APCA, while the parameters at generations 25 and 250 represent the evolved parameters at 5% and 50% of a run respectively. Figures 5.24, 5.25, 5.26, 5.27, and 5.28 show the results on five test problems. To make the results clearer to see, the performance of the evolved parameters are compared to that of the initial parameters by using the Wilcoxon signed rank sum test for paired difference at significance $\alpha = 0.05$. Table 5.8 shows the critical values, where n is the number of unequal samples. The results are shown in Table 5.9. The '>' symbol denotes the evolved parameters at that generation are better than the initial parameters, while the '=' symbol means there is no significant difference.

Apart from the results on the knapsack problem at generation 250, all evolved parameters are better than the initial parameters. The results confirm that the meta-level algorithm is capable of transforming the initial parameters to the better ones.

5.4 Summary

This chapter presents the experimental results of APCA. The algorithm is compared against five algorithms on five test problems. The experiments are

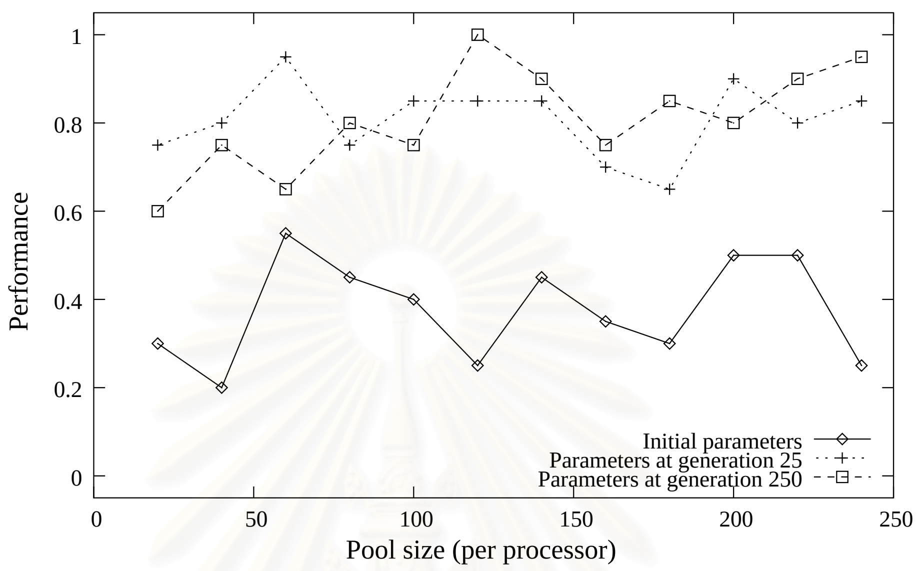

Figure 5.24: Parameters at generations 0, 25 and 250 on the onemax problem

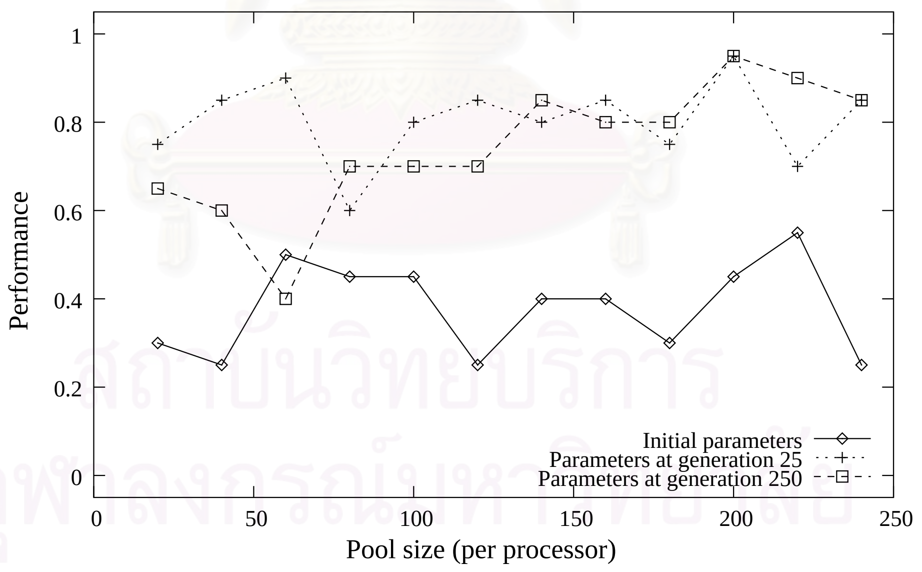

Figure 5.25: Parameters at generations 0, 25 and 250 on the contiguous bits problem

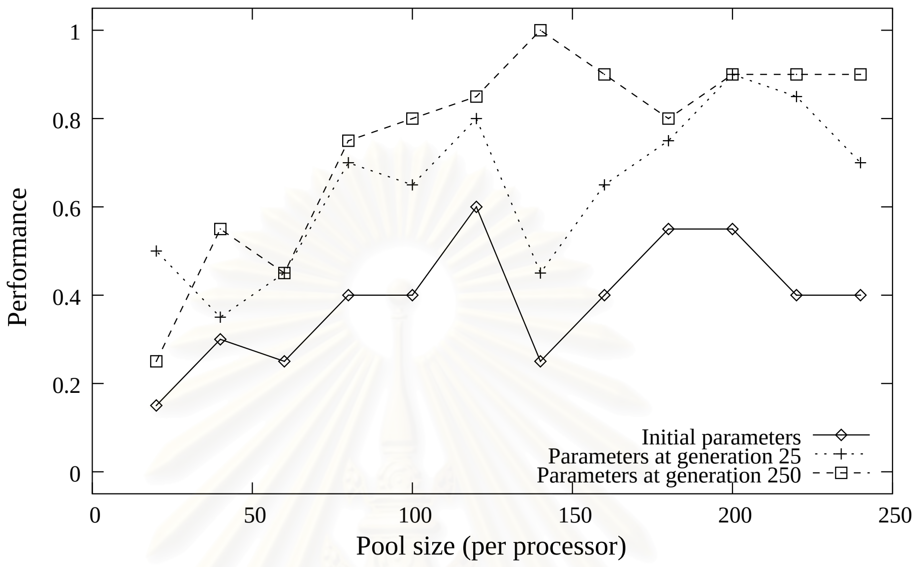

Figure 5.26: Parameters at generations 0, 25 and 250 on the minimal deceptive problem

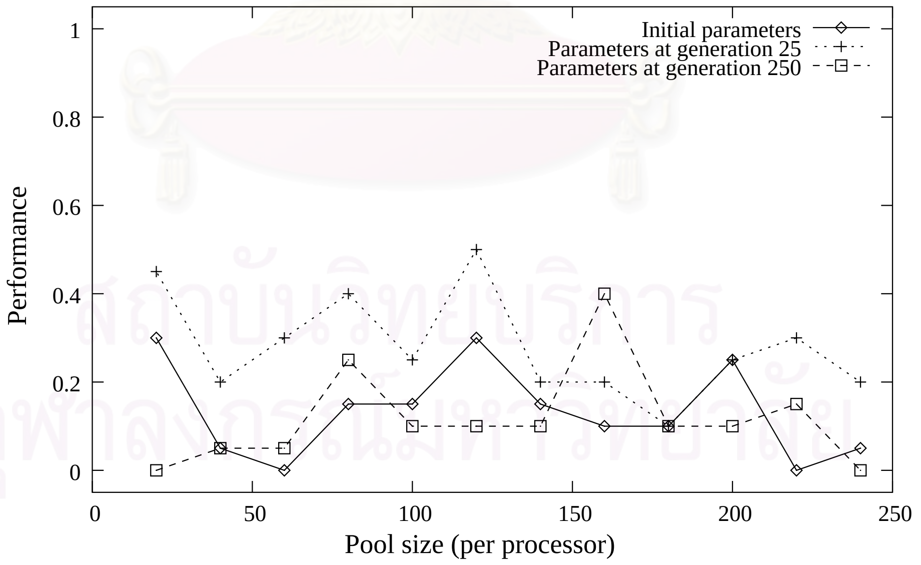

Figure 5.27: Parameters at generations 0, 25 and 250 on the knapsack problem

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

Table 5.8: Critical values of T_L in the Wilcoxon signed rank sum test, $P(T \leq T_L) = 0.05$

n	T_L
5	1
6	2
7	4
8	6
9	8
10	11
11	14
12	17

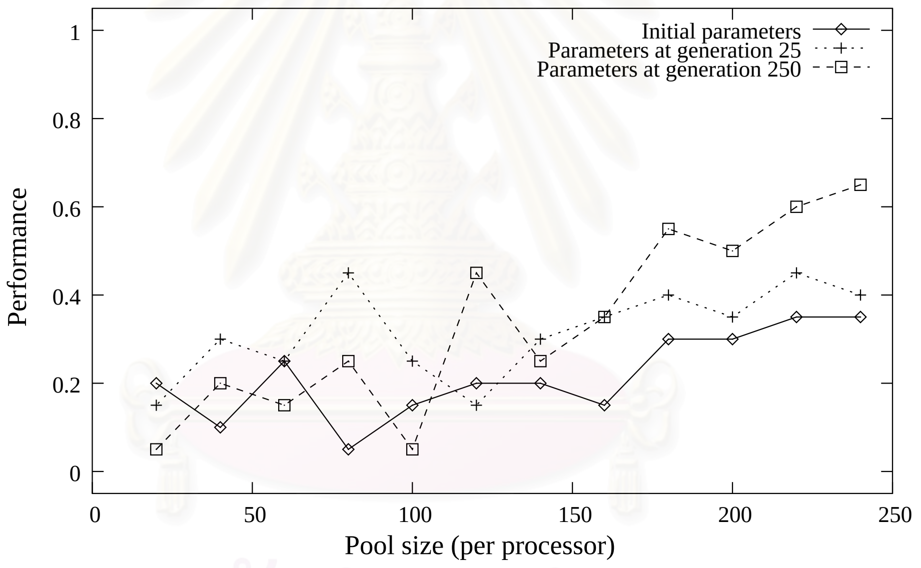

Figure 5.28: Parameters at generations 0, 25 and 250 on the royal road problem

conducted on a wide range of population sizes in order to reduce bias associated with the setting of population size. Regarding the population sizes, the results show that the proposed method offers two advantages over the other competing methods on the test problems: the reliability in finding the optimal solution and the time required for finding the optimal solution.

We also use a multimodal problem generator to examine how well APCA adapts itself to new problem instances. The algorithm is a bit slower than the

Table 5.9: A comparison between the evolved parameters and the initial parameters by using the Wilcoxon signed rank sum test ($\alpha = 0.05$)

Problem	Parameters at generation 25	Parameters at generation 250
Onemax	$>$	$>$
Contiguous bits	$>$	$>$
MDP	$>$	$>$
Royal road	$>$	$>$
Knapsack	$>$	$=$

use of a highly fit parameter set in some cases, but the algorithm is more general to different problem instances. We also test other adaptive algorithms with this problem generator. The results suggest that APCA performs better than the others for this problem.

The results show that the performance of APCA is less sensitive to the availability of the crossover operator in the meta-level GA. The disable of crossover does not introduce any reduction of the performance of APCA. Conversely, the performance is drastically reduced when the mutation operator is disabled. In case of no prior knowledge, it is safe to use both operators.

We argue that the evolution of parameters in the meta-level GA is not random. We compare APCA with the random walk algorithm that randomly changes the parameters with the same frequency of APCA. The results of APCA are different from those of the random walk algorithm. The finding is later confirmed by the analysis of the evolved parameters in the last section.

CHAPTER VI

A Case Study in Examination Timetabling

The previous chapter presents the results of APCA on some artificially constructed problems. This is a usual practice since many researchers have also tested their algorithms with this kind of problems. However, it is questionable whether the results on such problems are really justified. This is due to a gap between laboratory problems and real-world problems. The goal of this chapter is to apply APCA to a more complicated and realistic problem. The findings will provide some insights about the usefulness of APCA on a real-world problem.

The problem used in this chapter is an examination timetabling problem. This problem arises in many universities and colleges. Solving this problem is not trivial since the problem has a large search space and is highly constrained. Fang [86] pointed out that a problem with 44 examinations and 28 timeslots (4 per day over 7 days) had 28^{44}, or c. 5×10^{63} possible solutions. This was the complete space of possibilities. If the number of impossible and infeasible cases was omitted from the count, there were still $44!/(7! \times 7!^2 \times 6!^5)$ or c. 1.486×10^{26} candidate solutions, which was still a very large search space.

This chapter starts by describing an examination timetabling problem. Then, it presents how to apply a GA for this problem. This section also describes how to apply APCA to the problem. Thereafter, we present some experimental results and discussion. Finally, some concluding remarks are presented.

6.1 Problem Description

A simple timetabling problem can be regarded as a problem of assigning v events to s timeslots. The events may be examinations, lectures, meetings, or seminars, depending on the type of problems. Many timetabling problems also involve the assignment of resources (e.g., rooms, items of equipment) to each event. In addition, the problems also incorporate of many constraints that should be satisfied. Typical timetabling problems have long been acknowledged to belong to the class of problems called NP-complete [87]. Several approaches have been proposed to tackle timetabling problems, for instance, genetic algorithms, memetic algorithms, simulated annealing, tabu search, constraint logic programming. A comparison of the performance of different algorithms on timetabling was presented in [88]. Some researchers also proposed hybrid algorithms incorporating several search algorithms into unified frameworks (e.g. [89]).

A particular instance of timetabling problems used in this chapter is examination timetabling. For examination timetabling problems, the events are examinations. Let $\mathcal{E}$ be the set of v examinations $\{e_1, e_2, ..., e_v\}$ and let $\mathcal{T}$ be the set of s timeslots $\{t_1, t_2, ..., t_s\}$. An assignment (a, b), such that $a \in \mathcal{E}$ and $b \in \mathcal{T}$, can be interpreted as 'event a occurs during period b'. Solving this problem can be regarded as finding v assignments (each for one event) that can satisfy all constraints.

The set of constraints can be divided into two groups: hard constraints and soft constraints. The difference between both types is that hard constraints must not be violated, while soft constraints may be violated if necessary. The hard constraints of typical examination timetabling problems are that no student must take more than one examination at a time. These constraints are normally found in most examination timetabling. Soft constraints always vary between problems. An example of soft constraints is to let examinations spread out across the timetable in the way that reduces the pressure of most students.

The constraints used in this study are as follows:

- **Hard constraints**
  - *Clash*: No student may have two examinations in the same period.
- **Soft constraints**
  - *Near-clash*: If examinations for one student are too close and are on the same day, it is supposed to violate one of the near-clash constraints.

Two examinations have to be separated with one whole timeslot, unless it is considered to violate one of the constraints.

– *Exclusion*: Some examinations should not be scheduled on specified timeslots.

– *Order*: Some examinations should precede some specific examinations.

6.1.1 An Example of Examination Timetabling

This section provides an example of a simple examination timetabling problem and a solution to this problem. Assume there are 8 timeslots $(t_1, t_2, ..., t_8)$ allocated on two days (See Table 6.1), 12 examinations $(e_1, e_2, ..., e_{12})$ and 8 students. Table 6.2 shows the list of examinations taken by each student. For example, the first student takes four examinations (namely, 1, 6, 11 and 12), while the second student takes three examinations (namely, 8, 9 and 10).

Table 6.1: Timeslot allocation

Day	Period			
	9:00-10:00	10:30-11:30	13:00-14:00	14:30-15:30
Monday	t_1	t_2	t_3	t_4
Tuesday	t_5	t_6	t_7	t_8

Table 6.2: Student data

Student	Examination
s1	e_1 e_6 e_{11} e_{12}
s2	e_8 e_9 e_{10}
s3	e_2 e_3 e_5 e_7
s4	e_7 e_9
s5	e_{10} e_{11}
s6	e_5 e_9 e_{10}
s7	e_2 e_3 e_7
s8	e_5 e_6

Assume that only clash constraints and near-clash constraints are considered. A solution to this problem consists of 12 assignments. One of the solutions is as follows:

$\{(e_1,t_7)(e_2,t_6)(e_3,t_4)(e_4,t_3)(e_5,t_8)(e_6,t_1)(e_7,t_1)(e_8,t_5)(e_9,t_4)(e_{10},t_2)(e_{11},t_5)(e_{12},t_4)\}$

This solution means that the examination e_1 is scheduled to the slot t_7, the examination e_2 is scheduled to the slot t_6 , and so on. The solution is illustrated in Table 6.3.

Table 6.3: A possible solution

Day	Period			
	9:00-10:00	10:30-11:30	13:00-14:00	14:30-15:30
Monday	e_6, e_7	e_{10}	e_4	e_3, e_9, e_{12}
Tuesday	e_8, e_{11}	e_2	e_1	e_5

6.2 Genetic Algorithms for Examination Timetabling

6.2.1 Representation and Evaluation

The representation of a chromosome is a list of numbers. The length of a chromosome is equal to the number of examinations (v). The i_{th} element of a chromosome represents the timeslot for the examination i. Thus, the solution in the previous section can be represented as follows:

$\{7, 6, 4, 3, 8, 1, 1, 5, 4, 2, 5, 4\}$

The fitness function is adopted from the study by Fang [86]. The fitness evaluation is based on the number and types of constraints violated. Let $c_i(a)$ be the number of violated constraints of type i in the individual a and let w_i be the weight associated with the violations of constraints of type i. The weight for important constraints will be high, whereas constraints that are less important will have low weight. The fitness calculation for the individual a is as follows:

$$f(a) = 1/(1 + \sum_{i=1}^{n} w_i c_i(a)) \tag{6.1}$$

where n is the number of constraint-types.

6.2.2 Search Operators

There are three steps of applying search operators. The first two steps are identical to that used in the previous chapter. The first step is the application of crossover operators. Then, the second step applies random mutation. Finally, the third step is the application of local search operators, called *smart mutation*.

1. *Crossover*: The application of these operators is controlled by the crossover rate (P_c). There are five choices of crossover, namely, (1) one-point crossover, (2) two-point crossover, (3) uniform crossover, (4) uniform crossover with a probability of 0.1 and (5) uniform crossover with a probability of 0.2. All choices are identical to that used in the previous chapter. It is interesting to note that all crossover operators do not cause any problem with the integer representation used in the problem. Offspring produced by the crossover operators are still valid. The research by Abramson and Abela [90] used a different representation. Each position represented a timeslot, and a set of events occurring in the timeslot were grouped into this position. This representation was close to a real timetable. However, it suffered from a problem called the *label replacement* problem. That is, the produced offspring from the crossover operators may be invalid. Hence, it is necessary to have an operator to repair invalid offspring.
2. *Random Mutation*: The application of these operators is controlled by the mutation rate (P_m). Specifically, each position in an individual has a probability (P_m) to be mutated. There are two choices of mutation that are suitable for the representation used here.
   - *Swap mutation*: Another point is randomly selected. Then, the values of two positions are swapped.
   - *Change mutation*: The value at the mutation point is set to a new value.
3. *Smart Mutation*: When solving complex problems, it is quite common to use some heuristic operators to improve the quality of solutions. These operators

incorporate some problem-specific knowledge. For instance, three operators based on the 2-opt technique were proposed for improving the solutions of the three-matching problem [91]. These operators perform local improvement in a small portion of an individual, without reference to the global picture. For examination timetabling problems, some heuristic operators have been proposed to improve the solutions produced by genetic operators. Four heuristic operators used in this chapter are adopted from the study by Fang [86]. These operators are called the smart mutation.

- *Violation-directed Mutation* (VDM): One event with a maximal violation score is chosen. Then, a new random timeslot is assigned to this event.
- *Event-freeing Mutation* (EFM): One event with a maximal violation score is chosen. Then, a new timeslot that will maximally reduce the violation score is assigned to this event.
- *Stochastic Violation-directed Mutation* (SVDM): One event is stochastically selected with bias toward events with higher violation scores. Then, a new random timeslot is assigned to this event.
- *Stochastic Event-freeing Mutation* (SEFM): One event is stochastically selected with bias toward events with higher violation scores. Then, a new timeslot with bias toward timeslots that will maximally reduce the violation score is stochastically selected for this event.

6.2.3 Applying APCA to Examination Timetabling

APCA is used for controlling five parameters, namely, crossover operators, crossover rates, random mutation operators, random mutation rates and smart mutation operators. The representation of the population of the meta-level GA is slightly modified from that used in the previous chapter. An example of an individual in the population of the meta-level GA is illustrated in Figure 6.1. This individual means the application of uniform crossover with a probability of 0.6, swap mutation with a probability of 0.2, SEFM as a smart mutation operator.

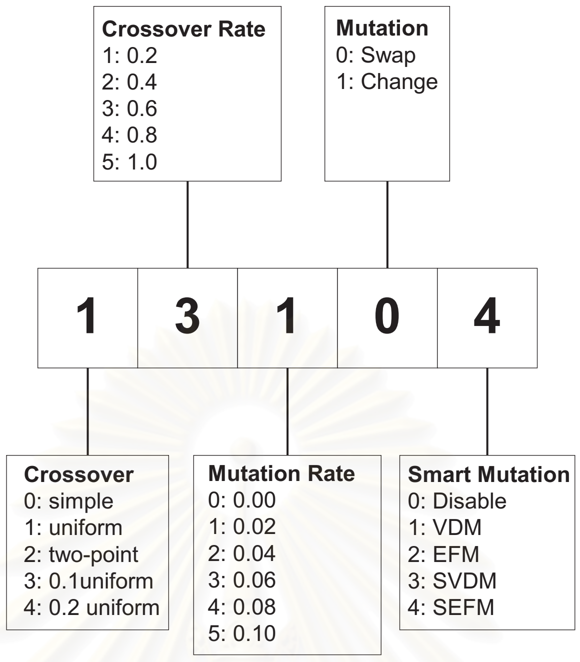

Figure 6.1: An example of a meta-level individual

The search operators for the meta-level algorithm are close to those used in the previous chapter, except that the mutation rate is simply set to the inverse of an individual length (i.e., 1/5 or 0.2).

6.3 Experiments

The program used in the experiments is modified from PGA written by Ross [92]. Three problem instances are used in the experiments (Table 6.4). The first instance (EDAI 92/93) is the timetabling data from Department of Artificial Intelligence, The University of Edinburgh in 1992/1993. However, this instance is quite small. Thus, we use a problem generator written by Ross to generate two additional problem instances. This problem generator creates a solution first, by assigning all examinations to slots at random. Then, a number of constraints are produced. Hence, it can guarantee that some optimal solutions exist. In the first instance, the penalties for each constraint are set according to the default setting of PGA [92]. This setting is also applied to the second and the third instances.

Table 6.4: Instances of the examination timetabling problem

Instance	Problem information	
EDAI 92/93	Number of examinations	44
	Number of timeslots	36
	Number of days	9
	Number of clash constraints	414
	Number of exclusion constraints	302
	Weight for clash constraints	100
	Weight for exclusion constraints	100
	Weight for near-clash constraints	10
GEN1	Number of examinations	200
	Number of timeslots	36
	Number of days	9
	Number of clash constraints	3766
	Weight for clash constraints	100
	Weight for near-clash constraints	10
GEN2	Number of examinations	200
	Number of timeslots	36
	Number of days	9
	Number of clash constraints	4672
	Weight for clash constraints	100
	Weight for near-clash constraints	10

Eight processing nodes are used in the experiments. Due to the limitation of computational power, a fixed population size is used rather than a range of population sizes. The population size in each node for the first problem instance is 20, while the population size in each node for the second and third problem instance is 40. The parameters that are not mentioned here are identical to those used in the previous chapter. From the number of examinations and constraints, the first instance seems to be the easiest instance, while the third instance seems to be the hardest one.

Table 6.5 shows the results on three instances. Note that we compare only the competitive algorithms from the results in the previous chapter. For a small problem instance, all algorithms are nearly identical. The performance of APCA slightly reduces when the complexity of the problem instances increases.

Table 6.5: Experimental results for the examination timetabling problem

Instance	Algorithm	Performance	Unuse factor
EDAI 92/93	APCA	1.0	0.954
	Diverse random algorithm	1.0	0.953
	Strategy adaptation	1.0	0.943
GEN1	APCA	1.0	0.744
	Diverse random algorithm	0.65	0.307
	Strategy adaptation	0.25	0.140
GEN2	APCA	1.0	0.623
	Diverse random algorithm	0.2	0.112
	Strategy adaptation	0.1	0.063

In contrast, the performance of other two algorithms dramatically reduces when the problem instances become harder.

An observation is made from the evolved parameters from the meta-level algorithm of APCA. It is observed that the value of the random mutation rate is uniform across all runs. APCA always assigns zero to this parameter. This means that APCA normally disables the use of random mutation. One possible interpretation is that random mutation operators deteriorate performance, and then they are disabled by APCA. In order to verify that random mutation operators really deteriorate performance, some additional experiments are conducted. All algorithms use the uniform crossover with a probability 0.8. Two smart operators (SVDM, SEFM) are tested. For the random mutation, the change mutation is used with probabilities 0.0, 0.04 and 0.1. Table 6.6 shows the experimental results on all instances. It is clear that the use of a random mutation operator reduces the performance. In addition, SEFM is obviously better than SVDM, especially on the complex instances.

6.4 Summary

This chapter shows how to apply APCA to some instances of the examination timetabling problem. The characteristics of the problem are quite different from

Table 6.6: Algorithms with and without random mutation

Instance	Algorithm	Performance	Unuse factor
EDAI 92/93	no random mutation, SVDM	1.0	0.924
	Change mutation (0.04), SVDM	1.0	0.856
	Change mutation (0.1), SVDM	1.0	0.555
	no random mutation, SEFM	1.0	0.965
	Change mutation (0.04), SEFM	1.0	0.956
	Change mutation (0.1), SEFM	1.0	0.901
GEN1	no random mutation, SVDM	1.0	0.272
	Change mutation (0.04), SVDM	0.0	0.0
	Change mutation (0.1), SVDM	0.0	0.0
	no random mutation, SEFM	1.0	0.758
	Change mutation (0.04), SEFM	0.0	0.0
	Change mutation (0.1), SEFM	0.0	0.0
GEN2	no random mutation, SVDM	0.0	0.0
	Change mutation (0.04), SVDM	0.0	0.0
	Change mutation (0.1), SVDM	0.0	0.0
	no random mutation, SEFM	1.0	0.639
	Change mutation (0.04), SEFM	0.0	0.0
	Change mutation (0.1), SEFM	0.0	0.0

the test problems in the previous chapter. The representation is based on lists of integer values rather than simple binary strings. In addition, some problem-specific operators are essential for this problem. The results in this chapter would suggest how APCA performs on a complex and highly constrained problem. In the examination timetabling problem, the application of search operators is divided into three stages: (1) crossover operators, (2) random mutation operators and (3) smart mutation operators. APCA is used for deciding and selecting the choices of search operators in these stages. Three instances of the examination timetabling problem are used in the experiments. APCA performs quite well, even the complexity of the problem instances increases. Interestingly, the results also suggest that APCA can detect the ineffectiveness of the random mutation operators.

CHAPTER VII

Conclusions

7.1 Summary

A general practice for setting the parameters of GAs relies on trial and error experiments. That is, users use their intuition in selecting a proper parameter set, and try it with their problems. If the results are unsatisfactory, then users attempt to adjust the parameter set and try it again. This method is obviously a time-consuming operation.

This thesis proposes a method to overcome this problem. The parameter setting is controlled by an additional GA. This algorithm is denoted as 'an Adaptive Parameter Control Algorithm' (APCA). The characteristics of APCA can be summarized as follows:

- **Meta-level optimization**
  APCA employs two levels of optimization. The task level algorithm (lower level algorithm) solves the given problem, while the meta-level algorithm (upper level algorithm) optimizes the parameters of the task level. The optimization of the meta-level algorithm performs concurrently with that of the task level algorithm. This feature of concurrent optimization makes the proposed algorithm differ from many existing meta-level optimization approaches.
- **Multiple parameters**
  Several parameters are controlled by the proposed method. APCA is used to control four parameters for the artificial problems, while five parameters are controlled for the timetabling problem. Note that many existing methods can handle only one parameter at a time.
- **Coarse-grained model of parallelization**
  The proposed method can be regarded as an extension to a conventional

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

coarse-grained model for parallelization. This model of parallelization has been acknowledged for its advantages. Thus, we would expect that the proposed method can inherit the advantages of a coarse-grained model.

Five test problems are used to conduct a comparison between APCA and other five algorithms. The experiments are carried out on a range of population sizes in order to prevent bias associated with the setting of population size. Regarding the population size used, the results show that APCA is not only faster than other algorithms, but APCA also more reliably finds optimal solutions.

In order to test whether the change of a problem can affect the algorithm, the study then conducts an experiment by using a multimodal problem generator. The problem generator can create a number of problems with controllable characteristics. The results indicate that APCA can adapt to the change of this problem. Moreover, the use of a highly fit parameter set (if it is available) is slightly faster than APCA. However, it has a limit generalization. This can be interpreted that APCA is promising when prior knowledge about a feasible parameter set is not known.

The study also examines the evolution of the task level parameters during the run. Apart from directly comparing the evolved parameters to the initial parameters, the study compares their effectiveness in order to verify that the meta-level algorithm can improve the task level parameters. By assigning the parameters to a coarse-grained parallel GA, the effectiveness of the parameters is taken from the final result of the run. The results show that the evolved parameters are better than the initial ones. This indicates that the meta-level algorithm can improve the task level parameters.

Another set of experiments is to test the sensitivity of the meta-level parameters. The results indicate that the performance of APCA is less sensitive to the availability of the crossover operator in the meta-level GA. However, it is still safe to use both crossover and mutation in the meta-level GA.

Finally, the proposed algorithm is examined on a more complex and realistic problem. Three instances of the examination timetabling problem are used to test the algorithms. The results not only suggest the advantage of APCA, but also show that APCA can detect the inferior performance caused by the random mutation operators.

7.2 Future Research

There are several possible extensions to the study in this thesis. Some of them are outlined next.

- **Introduction of new adaptive parameters**
  This study focuses on the choices of search operators and their rates. Thus, one of the obvious extensions is to introduce other adaptive parameters to the algorithm. The introduction of new adaptive parameters obviously increases the size of the search space. This may require additional subpopulations in order to effectively optimize the parameters.
- **Optimize the parameters of other evolutionary algorithms**
  The task level algorithm in this study is a GA. However, it is possible to apply this technique to other evolutionary algorithms. Indeed, most evolutionary algorithms also have several control parameters.
- **Evolve search strategies rather than parameters**
  The parameter optimization is the first step. It is not limited to optimize only the parameters of the task level algorithm. The idea is to use the meta-level algorithm to construct more complex search strategies rather than only the parameters. Some researchers have used meta-level algorithms to evolve genetic operators [61, 64]. Thus, we may use a meta-level algorithm to evolve other components of an algorithm, e.g., selection operator, replacement policy, migration scheme.
- **Apply to real-world applications**
  After evaluating the algorithm on test problems, the next step would be the

application of the algorithm to real-world problems. Although APCA has been applied to the examination timetabling problem, it would be better to find more real-world applications in order to make a judgment about the usefulness of the algorithm in real situations.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

References

[1] De Jong, K. A., Potter, M. A., and Spears, W. M. Using problem generators to explore the effects of epistasis. In Bäck, Thomas, ed., Proceedings of the Seventh International Conference on Genetic Algorithms, pp. 338–345, San Francisco, July 19–23 1997. Morgan Kaufmann.

[2] Schaffer, J. D., Caruana, R. A., Eshelman, L. J., and Das, R. A study of control parameters affecting online performance of genetic algorithms for function optimization. In Proceedings of the Third International Conference on Genetic Algorithms, pp. 51–60, 1989.

[3] Spears, W. M. and De Jong, K. A. Dining with GAs: Operator lunch theorems. In Banzhaf, W. and Reeves, C. R., eds., Proceedings of the Fifth Workshop on Foundations of Genetic Algorithms, pp. 85–101. Morgan Kaufmann, 1998.

[4] Spears, W. M. The Role of Mutation and Recombination in Evolutionary Algorithms. Doctoral dissertation, George Mason University, Fairfax, Virginia, 1998.

[5] Goldberg, D. E., Deb, K., and Clark, J. H. 1992. Genetic algorithms, noise, and the sizing of populations. Complex Systems 6:333–362.

[6] Goldberg, D. E., Deb, K., and Thierens, D. 1993. Toward a better understanding of mixing in genetic algorithms. Journal of the Society of Instrument and Control Engineers 32(1):10–16.

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

[8] Lobo, F. G. The parameter-less genetic algorithm: Rational and automated parameter selection for simplified genetic algorithm operation. Doctoral dissertation, Universidad Nova de Lisboa, Lisboa, Portugal, 2000.

[9] Cicirello, V. A. and Smith, S. F. Modeling GA performance for control parameter optimization. In Proceedings of the Genetic and Evolutionary Computation Conference, pp. 235–242, 2000.

[10] Lis, J. Parallel genetic algorithm with the dynamic control parameter. In Proceedings of IEEE International Conference on Evolutionary Computation, pp. 324–329, 1996.

[11] Bäck, T. Self-adaptation in genetic algorithms. In Proceedings of the First European Conference on Artificial Life, pp. 263–271, 1992.

[12] Fogel, L. J., Owens, A. J., and Walsh, M. J. Artificial intelligence through a simulation of evolution. In Maxfield, M., Callahan, A., and Fogel, L. J., eds., Biophysics and Cybernetic Systems: Proceedings of the 2nd Cybernetic Sciences Symposium, pp. 131 – 155, Washington DC, 1965. Spartan Books.

[13] Fogel, L. J., Owens, A. J., and Walsh, M. J. Artificial Intelligence through Simulated Evolution. New York:John Wiley & Sons, 1966.

[14] Rechenberg, I. Cybernetic solution path of an experimental problem. Library translation 1122, Royal Aircraft Establishment, Farnborough, UK, 1965.

[15] Beyer, H.-G. and Schwefel, H.-P. 2002. Evolution strategies — A comprehensive introduction. Natural Computing 1(1):3 – 52.

[16] Holland, J. H. 1973. Genetic algorithms and the optimal allocation of trials. SIAM Journal on Computing 2(2):88–105.

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

[18] Goldberg, D. E. Genetic Algorithm in search, optimization and machine learning. Addison-Wesley, 1989.

[19] Yao, X. 2002. Evolutionary computation. Evolutionary Optimization, pp. 27–53.

[20] Whitley, D. The GENITOR algorithm and selection pressure. In Schaffer, J. D., ed., Proceedings of the Third International Conference on Genetic Algorithms, pp. 116–121. Morgan Kaufmann, 1989.

[21] Yao, X. 1996. An overview of evolutionary computation. Chinese Journal of Advanced Software Research 3(1):12–29.

[22] Cantú-Paz, E. 1998. A survey of parallel genetic algorithms. Calculateurs

Parallèles, Reseaux et Systems Repartis 10(2):141–171.

[23] Alba, E. and Troya, J. M. 1999. A survey of parallel distributed genetic algorithms. Complexity 4(4):31–52.

[24] Alba, E. and Tomassini, M. 2002. Parallelism and evolutionary algorithms. IEEE Transactions on Evolutionary Computation 6(5):443–462.

[25] Koza, J. R. and Andre, D. Parallel genetic programming on a network of transputers. Technical Report CS-TR-95-1542, Department of Computer Science, Stanford University, January 1995.

[26] Dracopoulos, D. C. and Kent, S. Speeding up genetic programming: A parallel BSP implementation. In Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L., eds., Proceedings of the First Annual Conference on Genetic Programming, p. 421. MIT Press, 1996.

[27] Punch, B. How effective are multiple populations in genetic programming. In Proceedings of the Third Annual Conference on Genetic Programming, pp. 308–313, 1998.

[28] Tongchim, S. and Chongstitvatana, P. 2001. Parallel genetic programming: synchronous and asynchronous migration. Journal of Artificial Life and Robotics 5(4):189–194.

[29] Riessen, G. A., Williams, G. J., and Yao, X. PEPNet: parallel evolutionary programming for constructing artificial neural networks. In Proceedings of the Sixth Annual Conference on Evolutionary Programming (EP97), pp. 35–45, 1997.

[30] Tongchim, S. and Yao, X. Parallel evolutionary programming. In Proceedings of the 2004 Congress on Evolutionary Computation, pp. 1362–1367. IEEE Press, 2004.

[31] Belding, T. C. The distributed genetic algorithm revisited. In Eshelman, L. J., ed., Proceedings of the Sixth International Conference on Genetic Algorithms, pp. 114–121. Morgan Kaufmann, 1995.

[32] Alba, E. 2002. Parallel evolutionary algorithms can achieve super-linear performance. Information Processing Letters 82(1):7–13.

[33] Lin, S.-C., Punch, W. F., and Goodman, E. D. Coarse-grain genetic algorithms, categorization and new approaches. In Proceedings of the Sixth IEEE Parallel and Distributed Processing, pp. 28–37, 1994.

[34] Andre, D. and Koza, J. R. 1998. A parallel implementation of genetic programming that achieves super-linear performance. Information Sciences 106(3–4):201–218.

[35] Alba, E. and Troya, J. M. 2001. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. Future Generation Computer Systems 17(4):451–465.

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

[37] Reed, J., Toombs, R., and Barricelli, N. A. 1967. Simulation of biological evolution and machine learning. Journal of Theoretical Biology 17:319–342.

[38] Eiben, Á. E., Hinterding, R., and Michalewicz, Z. 1999. Parameter control in evolutionary algorithms. IEEE Transactions on Evolutionary Computation 3(2):124–141.

[39] Fogel, D. B. The advantages of evolutionary computation. In Proceedings of the Bio-computing and Emergent Computation conference, pp. 1–11, 1997.

[40] Angeline, P. J. 1995. Adaptive and self-adaptive evolutionary computations. Computational Intelligence: A Dynamic Systems Perspective, pp. 152–163.

[41] Smith, J. E. and Fogarty, T. C. 1997. Operator and parameter adaptation in genetic algorithms. Soft Computing 1(2):81–87.

[42] Harik, G. R. and Lobo, F. G. A parameter-less genetic algorithm. In Proceedings of the Genetic and Evolutionary Computation Conference, pp. 258–265, 1999.

[43] Schlierkamp-Voosen, D. and Mühlenbein, H. Strategy adaptation by competing subpopulations. In Parallel Problem Solving from Nature (Lecture

Notes in Computer Science volume 886), pp. 199–208, 1994.

[44] Schlierkamp-Voosen, D. and Mühlenbein, H. Adaptation of population sizes by competing subpopulations. In Proceedings of International Conference on Evolutionary Computation, pp. 330–335, 1996.

[45] Hinterding, R., Michalewicz, Z., and Peachey, T. C. Self-adaptive genetic algorithm for numeric functions. In Parallel Problem Solving from Nature (Lecture Notes in Computer Science volume 1141), pp. 420–429, 1996.

[46] Eiben, Á. E., Sprinkhuizen-Kuyper, I. G., and Thijssen, B. A. Competing crossovers in an adaptive GA framework. In Proceedings of 1998 IEEE International Conference on Evolutionary Computation, pp. 787 –792, 1998.

[47] Pham, Q. T. Competitive evolution: A natural approach to operator selection. In Yao, X., ed., Progress in Evolutionary Computation, Lecture Notes in Artificial Intelligence, volume 956, pp. 49–60. Springer-Verlag, Heidelberg, 1995.

[48] Schnecke, V. and Vornberger, O. An adaptive parallel genetic algorithm for VLSI-layout optimization. In Parallel Problem Solving from Nature, pp. 859–868, 1996.

[49] Wang, G., Goodman, E. D., and Punch, W. F. Simultaneous multi-level evolution. Technical Report 96-03-01, Genetic Algorithms Research and Applications Group, Michigan State University, 1996.

[50] Wang, G., Goodman, E. D., and Punch, W. F. Toward the optimization of a class of black box optimization algorithms. In Proceedings of the Ninth IEEE International Conference on Tools with Artificial Intelligence, pp. 348–356, November 3–8 1997.

[51] Wang, G., Dexter, T. W., Punch, W. F., and Goodman, E. D. Optimization of a GA and within a GA for a 2-dimensional layout problem. In Proceedings of the First International Conference on Evolutionary Computation and its Applications, Presidium, Russian Academy of Sciences, pp. 18–29, 1996.

[52] Tanese, R. Parallel genetic algorithm for a hypercube. In Grefenstette, J. J., ed., Proceedings of the Second International Conference on Genetic Algorithms and their Application, pp. 177–183. Lawrence Erlbaum Associates, 1987.

[53] Adamidis, P. and Petridis, V. Co-operating populations with different evolution behavior. In Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, pp. 188–191, 1996.

[54] Miki, M., Hiroyasu, T., Kaneko, M., and Hatanaka, K. A parallel genetic algorithm with distributed environment scheme. In Proceedings of the 1999 IEEE International Conference on Systems, Man, and Cybernetics, volume 1, pp. 695–700, 1999.

[55] Herrera, F. and Lozano, M. 2000. Gradual distributed real-coded genetic algorithms. IEEE Transactions on Evolutionary Computation 4(1):43–63.

[56] Freisleben, B. 1997. Metaevolutionary approaches. Handbook of Evolutionary Computation, pp. C7.2:1–8.

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

[58] Shahookar, K. and Mazumder, P. 1990. A genetic approach to standard cell placement using meta-genetic parameter optimization. IEEE Transactions on Computer-Aided Design 9(5):500–511.

[59] Mernik, M., Črepinšek, M., and Žumer, V. A metaevolutionary approach in searching of the best combination of crossover operators for the TSP. In Hamza, M.H., ed., Proceedings of the IASTED international conference on Neural networks, pp. 32–36, Pittsburgh, Pennsylvania, May 15–17 2000. IASTED/ACTA Press.

[60] B̈ack, T. Parallel optimization of evolutionary algorithms. In Parallel Problem Solving from Nature, pp. 418–427, 1994.

[61] Teller, A. 1996. Evolving programmers: The co-evolution of intelligent

recombination operators. Advances in Genetic Programming II, pp. 45–68.

[62] Kantschik, W., Dittrich, P., Brameier, M., and Banzhaf, W. Meta-evolution in graph GP. In Proceedings of the Second European Workshop on Genetic Programming, pp. 15–28, 1999.

[63] Kantschik, W., Dittrich, P., Brameier, M., and Banzhaf, W. Empirical analysis of different levels of meta-evolution. In Proceedings of Congress on Evolutionary Computation, volume 3, pp. 2086–2093, 1999.

[64] Edmonds, B. 2001. Meta-genetic programming: Co-evolving the operators of variation. Electrik 9:13–29.

[65] Tongchim, S. and Chongstitvatana, P. 2002. Parallel genetic algorithm with parameter adaptation. Information Processing Letters 82(1):47–54.

[66] Tongchim, S. and Chongstitvatana, P. Adaptive parameter control in parallel genetic algorithm. In Proceedings of International Conference on Intelligent Technologies, pp. 100–109, 2000.

[67] Ackley, D. H. A Connectionist Machine for Genetic Hillclimbing. Kluwer Academic Publishers, Boston, MA, 1987.

[68] Syswerda, G. Uniform crossover in genetic algorithms. In Proceedings of the Third International Conference Genetic Algorithms and Their Applications, pp. 2–8, 1989.

[69] Spears, W. M. and De Jong, K. A. On the virtues of parameterized uniform crossover. In Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 230–236, 1991.

[70] Huang, W.-C., Kao, C.-Y., and Horng, J.-T. A genetic algorithm approach for set covering problems. In Proceedings of the First IEEE Conference on Evolutionary Computation, pp. 569–574, 1994.

[71] Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. A racing algorithm for configuring metaheuristics. In Proceedings of the Genetic and Evolutionary Computation Conference, pp. 11–18, 2002.

[72] Senyu, S. and Toyoda, Y. 1967. An approach to linear programming with

0-1 variables. Management Science 15:B196–B207.

[73] Beasley, J. E. 1990. OR-library: distributing test problems by electronic mail. Journal of the Operational Research Society 41(11):1069–1072.

[74] Khuri, S., Bäck, T., and Heitkötter, J. The zero/one multiple knapsack problem and genetic algorithms. In Proceedings of the 1994 ACM Symposium on Applied Computing, pp. 188–193, 1994.

[75] Holland, J. H. Royal road functions. Internet Genetic Algorithms Digest, vol. 7, issue 22, 12 August 1993.

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

[77] Vafaie, H. and De Jong, K. Robust feature selection algorithms. In Proceedings of the 1993 IEEE International Conference on Tools with Artificial Intelligence, pp. 356 – 363. IEEE Press, November 8–11 1993.

[78] Fang, H., Liang, S., and Kuusk, A. 2003. Retrieving leaf area index using a genetic algorithm with a canopy radiative transfer model. Remote Sensing of Environment 85(3):257–270.

[79] Chen, S.-H., Chen, C.-F., and Tan, C.-W. Toward an effective implementation of genetic algorithms in financial data mining: Retraining plus validating. In Proceedings of the 1998 International Symposium on Intelligent Data Engineering and Learning, pp. 99–105. Springer-Verlag, October 14–16 1998.

[80] Bonissone, P. P., Khedkar, P. S., and Chen, Y. Genetic algorithms for automated tuning of fuzzy controllers: A transportation application. In Proceedings of the Fifth IEEE International Conference on Fuzzy Systems, pp. 674–680. IEEE Press, September 8–11 1996.

[81] Campoy, A. Martí, Ivars, A. Perles, and Mataix, J. V. Busquets. Dynamic use of locking caches in multitask, preemptive real-time systems. In Proceedings of the 15th World Congress of the International Federation of Automatic Control. Elsevier Science, 2002.

[82] Srinivas, M. and Patnaik, L. M. 1994. Adaptive probabilities of crossover

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

and mutation in genetic algorithms. IEEE Transactions on System, Man and Cybernatics 24(2):656–667.

[83] Deb, K. and Agrawal, S. Understanding interactions among genetic algorithm parameters. In Banzhaf, W. and Reeves, C. R., eds., Proceedings of the Fifth Workshop on Foundations of Genetic Algorithms, pp. 265–286. Morgan Kaufmann, 1998.

[84] Corcoran, A. L. and Wainwright, R. L. 1995. Using LibGA to develop genetic algorithms for solving combinatorial optimization problems. Practical Handbook of Genetic Algorithms, Applications, 1:143–172.

[85] Kennedy, J. and Spears, W. M. Matching algorithms to problems: An experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator. In Proceedings of the IEEE International Conference on Evolutionary Computation, pp. 78–83. IEEE Press, May 4–9 1998.

[86] Fang, H.-L. Genetic Algorithms in Timetabling and Scheduling. Doctoral dissertation, University of Edinburgh, UK, 1994.

[87] Burke, E. K., Jackson, K. S., Kingston, J. H., and Weare, R. F. 1997. Automated timetabling: The state of the art. The Computer Journal 40(9):565–571.

[88] Rossi-Doria, O., Sampels, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L. M., Knowles, J., Manfrin, M., Mastrolilli, M., Paechter, B., Paquete, L., and Stützle, T. A comparison of the performance of different metaheuristics on the timetabling problem. In Burke, E. and De Causmaecker, P., eds., Proceedings of the fourth International Conference on Practice and Theory of Automated Timetabling, pp. 329–351. Springer, 2002.

[89] Merlot, L. T.G., Boland, N., Hughes, B. D., and Stuckey, P. J. A hybrid algorithm for the examination timetabling problem. In Burke, E. and De Causmaecker, P., eds., Proceedings of the fourth International Conference on Practice and Theory of Automated Timetabling, pp. 207–231.

Springer, 2002.

[90] Abramson, D. and Abela, J. A parallel genetic algorithm for solving the school timetabling problem. Technical report, Division of Information Technology, C.S.I.R.O., April 1991.

[91] Magyar, G., Johnsson, M., and Nevalainen, O. July 2000. An adaptive hybrid genetic algorithm for the three-matching problem. <u>IEEE Transactions on Evolutionary Computation</u> 4(2):135–146.

[92] Ross, P. The PGA v4.0 manual. September 2001.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

Biography

Shisanu Tongchim was born in Khon Kaen, Thailand on December 2, 1977. He received the B.Eng. degree from Chulalongkorn University in 1998, the M.Eng. degree from Chulalongkorn University in 2000. Then, he continued his study in the Ph.D. program at the department of computer engineering, Chulalongkorn University. His research was supported by the Royal Golden Jubilee Ph.D. Program of the Thailand Research Fund.

In 2003, he was a visiting scholar in the School of Computer Science, University of Birmingham, U.K. He has published a number of papers in international journals and conferences in the fields of evolutionary computation. His research interests include evolutionary algorithms and parallel computing.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย