การเรียนรู้สหสัมพันธ์ลูกผสมเชิงบวกและเชิงลบ ในขั้นตอนวิธีการประมาณการแจกแจงสำหรับปัญหาการหาค่าเชิงการจัดที่เหมาะที่สุด

นายวรินทร์ วัฒนพรพรหม

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรดุษฎีบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2553

HYBRID POSITIVE AND NEGATIVE CORRELATION LEARNING IN ESTIMATION OF DISTRIBUTION ALGORITHM FOR COMBINATORIAL OPTIMIZATION PROBLEMS

Mr. Warin Wattanapornprom

A Dissertation Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy Program in Computer Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2010

Thesis Title	HYBRID POSITIVE AND NEGATIVE CORRELATION LEARNING IN ESTIMATION OF DISTRIBUTION ALGORITHM FOR COMBINATORIAL OPTIMIZATION PROBLEMS
By	Mr. Warin Wattanapornprom
Field of Study	Computer Engineering
Thesis Advisor	Professor Prabhas Chongstitvatana, Ph.D.

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Doctoral Degree

..... Dean of the Faculty of Engineering
(Associate Professor Boonsom Lerdhirunwong, Dr.Ing.)

THESIS COMMITTEE

.....Chairman
(Associate Professor Wiwat Vatanawood, Ph.D.)

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

.....Examiner
(Associate Professor Parames Chutima, Ph.D.)

.....Examiner
(Assistant Professor Sukree Sinthupinyo, Ph.D.)

..... External Examiner
(Kata Praditwong, Ph.D.)

วรินทร์ วัฒนพรพรหม : การเรียนรู้สหสัมพันธ์ลูกผสมเชิงบวกและเชิงลบในขั้นตอนวิธีการประมาณการแจกแจงสำหรับปัญหาการหาค่าเชิงการจัดที่เหมาะที่สุด. (HYBRID POSITIVE AND NEGATIVE CORRELATION LEARNING IN ESTIMATION OF DISTRIBUTION ALGORITHM FOR COMBINATORIAL OPTIMIZATION PROBLEMS) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : ศ.ดร.ประภาส จงสถิตย์วัฒนา, 172 หน้า.

วิทยานิพนธ์ฉบับนี้ศึกษาบทบาทของความรู้เชิงลบในขั้นตอนวิธีเชิงวิวัฒน์ บทสมุติฐานส่วนประกอบเชิงลบมีใจความว่า "ขั้นตอนวิธีใด ๆ สามารถค้นหาคำตอบที่ดีขึ้นได้โดยการหลีกเลี่ยงการประกอบกันของโครงสร้างทางความรู้ที่มีขนาดเล็กที่มีประสิทธิภาพต่ำ ที่เราเรียกกันว่าส่วนประกอบเชิงลบ" สมมุติฐานดังกล่าวถูกทดสอบโดยการพัฒนาขั้นตอนวิธีการประมาณการแจกแจงจากเส้นเชื่อม ที่มีชื่อว่า ขั้นตอนวิธีการบรรจบ (COIN) ขั้นตอนวิธีดังกล่าว ใช้ประโยชน์จาก ส่วนประกอบเชิงลบที่ซ่อนอยู่ในคำตอบที่ต่ำว่าค่าเฉลี่ย เพื่อใช้ในการหลีกเลี่ยงการประกอบกันของส่วนประกอบเชิงลบนั้นๆ ขั้นตอนวิธี COIN ถูกทดสอบในปัญหาเชิงการจัดที่มีคำตอบหลายรูปแบบ ทั้งนี้ ผลการทดลองสรุปได้ว่าขั้นตอนวิธี COIN มีส่วนช่วยในการสร้างคำตอบทั้งในเชิงปริมาณ และ ในเชิงคุณภาพ บทบาทของความรู้เชิงลบในปัญหาเชิงการจัดที่ได้จากการทดลองอาจสรุปได้ดังนี้ (๑) ความรู้เชิงลบบังคับในขั้นตอนวิธีค้นหาคำตอบนอกพื้นที่ที่ถูกห้ามเอาไว้ (๒) ความรู้เชิงลบช่วยให้ขั้นตอนวิธีผลิตคำตอบที่หลากหลาย ทั้งนี้ต้องมีความแตกต่างจากคำตอบที่มีคุณภาพต่ำ (๓) ความรู้เชิงลบเมื่อใช้ร่วมกันกับความรู้เชิงบวกแล้วมีส่วนช่วยในการแยกแยะส่วนประกอบที่ดีและไม่ดีได้ (๔) ความรู้เชิงลบช่วยให้ขั้นตอนวิธีเชิงการสร้างค้นหาและประกอบโครงสร้างที่ดีขึ้นได้ ท้ายที่สุด ขั้นตอนวิธี COIN ถูกนำไปทดสอบบนปัญหาที่ใช้งานจริงในหลายวัตถุประสงค์ และ แสดงศักยภาพที่เหนือกว่าขั้นตอนวิธีที่ถูกนำมาเปรียบเทียบ

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต วรินทร์ วัฒนพรพรหม

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก ประภาส จงสถิตย์วัฒนา

ปีการศึกษา 2553

4971874821 : MAJOR COMPUTER ENGINEERING
KEYWORDS : NEGATIVE KNOWLEDGE /COMBINATORIAL OPTIMIZATION/ ESTIMATION OF DISTRIBUTION ALGORITHMS.

WARIN WATTANAPORNPROM : HYBRID POSITIVE AND NEGATIVE CORRELATION LEARNING IN ESTIMATION OF DISTRIBUTION ALGORITHM FOR COMBINATORIAL OPTIMIZATION PROBLEMS. / ADVISOR : PROF. PRABHAS CHONGSTITVATANA, Ph.D., 172 pp.

This dissertation studies the roles of negative correlation learning of building blocks in evolutionary algorithms. It is based on a hypothesis called a *Negative Building Block Hypothesis* (NBBH) simply states that "*An algorithm can seeks new-optimal performance by avoiding the juxtaposition of short, low-order, low-performance schemata, called the negative building blocks*". The hypothesis is tested by developing of a new edge based estimation of distribution algorithm named Coincidence algorithm (COIN). Such algorithm utilizes the negative building blocks contained in the below average solution in order to avoid the composition of bad substructures. COIN is tested in several multimodal combinatorial problems. The results conclude that the negative correlation capability of COIN contributes in both quantity and quality of the solutions. In summary, the roles of negative knowledge in combinatorial optimization extracted from the experiments are as follows: (i) The negative knowledge forces the algorithm to explore out of the search space marked as forbidden areas. (ii) The negative knowledge helps producing more diverse solutions, however dissimilar to the solutions considered to be bad quality. (iii) In cooperating with the positive knowledge, the negative knowledge contributes in discrimination of good and bad building blocks. (iv) The negative knowledge enhances a constructive algorithm to recognize better substructures and to compose better solutions. Finally, COIN is tested in several real world multiobjective applications and has shown competitive results compared to the other algorithms in the experiments.

Department : Computer Engineering Student's Signature Warin Wattanapornprom

Field of Study : Computer Engineering Advisor's Signature P. Chongstitvatana

Academic Year : 2010

ACKNOWLEDGEMENTS

There are some people to whom I am deeply indebted for their help in the process that ends with this PhD dissertation. This dissertation could not be completed without the welcome help of certain people throughout the whole study period. I take this opportunity to acknowledge their never faltering encouragement and support.

First of all, I wish to thank my Ph.D. supervisors Professor Dr. Prabhas Chongstitvatana for offering me the opportunity to do this Ph.D. Special thanks also for his wise supervision, guidance, patience, and friendly encouragement.

In particular, I would like to thank Assoc. Professor Dr. Parames Chutima and his students including Dr. Mai, House, Nop, Yok, Pad, Rat and Guide for believing in me and my underage Coincidence Algorithm (COIN) and giving me a chance to try my algorithm in all your works. I would like to gratefully thank Assoc. Professor Dr. Wiwat Vatanawood, Asst. Professor Dr. Sukree Sinthupinyo and Dr. Kata Praditwong for their valuable comments and contributions throughout this study. I appreciate their willingness to talk to me at any time I need them.

I also owe a debt of gratitude to my colleagues of the Intelligent System Lab (ISL) and my Ph.D. Seminar Group especially P'Ning, P'Chat, P'Yod, P'Wak, P'Pong, Dae, Kawtip, Puck, Dej, Aui, Petch, Nan, Palm, Kuk, Woon, Komate, Vit, Peam, Yui, Lei, Ploy, Joe and Nui for useful discussions and encouragements during my entire Ph.D. life. In addition, I also thank Zhang Qing Fu, Kalyanmoy Deb, Marco Dorigo, Xin Yao and Chen Shi Chin for fruitful discussions and helpful suggestions on my research during CEC2008. These discussions have sharpened my understanding of different research issues, provided new points of view and directions for my research or simply made research more enjoyable.

Finally, the greatest thanks must dedicate to my family, my dad, my mom, my sisters and all the members of my family for believing in me when I found it difficult to believe in myself, for saying what I've needed to hear sometimes, instead of what I wanted to hear, for siding with me and for giving me another side to consider. My special appreciation goes to my beloved for putting so much thought and care and imagination into our relationship, for sharing so many nice times and making so many special memories with me.

CONTENTS

Page

Page

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figures **Page**

Figures **Page**

CHAPTER I

INTRODUCTION

1.1 General Background

Combinatorial optimization plays an important role for application in real world problems including scheduling, balancing, timetabling and routing problems, where the domains of feasible solutions are discrete. Combinatorial problems are intriguing as they are easy to state but often very difficult to solve. There is no algorithm exists to find the optimal solution to these classes of problems within polynomial time. Moreover, these optimization problems can have both single or multiple solutions in both single or multiple objectives. Most of them find a natural mapping in permutation spaces where mathematical programming models are inappropriate as they rather produce infeasible solutions than produce feasible ones.

The algorithm approaches to combinatorial optimization problems can be classified as *exact* and *approximate,* or sometimes called *stochastic* and *heuristics*. Exact algorithms are guaranteed to find one or more optimal solutions in finite time by systematically searching the solution space. Unfortunately, due to the *NP*-completeness nature of the problems, the time needed to solve them may grow exponentially in the worst case, for a reasonable problem size, exact algorithms are no longer feasible. To practically solve these problems, one often has to satisfied with finding good approximately solutions within a given reasonable polynomial time. Therefore *approximate algorithms* or sometimes called *metaheuristics* are more preferable. Normally, approximate algorithms cannot guarantee optimality of the solutions, anyhow, in many cases, they are able to find optimal solutions in short computation time.

Over the past few decades, many metaheuristics algorithms have been designed and applied to a wide variety of combinatorial problems. Unfortunately, the scalability of such algorithms has been very poorly investigated. Since many of them use ad hoc

techniques for both representations and operators, they do not scale up. While, typical industrial problems are often large and complex, traditional optimization methods are expected to fail or yield inacceptable solutions.

1.2 Problem Difficulties for Combinatorial Optimizations

This dissertation addresses the ineffectiveness's of combinatorial optimization methods mainly based on representation of candidate solution and its consequences when constructive and improvement methods are applied.

1.2.1 Cartesian and permutation spaces

This section describes the differences between two types of problems [1] in the discrete domain. The problems where the domains of parameters to be optimized take on sets of independent values belong to *Cartesian* or *vector* spaces, while the problems with domains that are permutations of items belong to *permutation* spaces. In Cartesian space, the parameters are independent from each other and the optimization function can be represented geometrically in a multidimensional space, while the parameters in the permutation spaces at a given position in the *n*-tupla are dependent on all the others and constitutes the *n*-tupla of values differentiate one input from another. Moreover, the parameters of the problems in mapped in the Cartesian space are directly used as absolute numbers in order to evaluate a function, while the parameters of permutation problems are indirectly used. In order to evaluate a function, one or more properties of the items are used. In addition, the properties of an item can depend on an absolute position of the item or a position related to the other items. Example 1.1 and 1.2 exemplify the different between the optimization function in discrete Cartesian space and the optimization function in the permutation space. Particularly, the example 1.2 show the indirectly use of the parameters in which the values of the input items depend on their absolute positions.

Example 1.1: A two variable function to optimize (Discrete Cartesian space)

$$F(x,y) = (x-y)^4 - (x-y)^2 \quad (1.1)$$

where $x \in [0\,..\,5], y \in [1\,..\,4\,]$

Example 1.2: A three variables function described by a permutation (Discrete permutation space)

$$Q(x,y,z) = x \times P(x) + y \times P(y) + z \times Pz) \quad (1.2)$$

where $x \in [1\,..\,3]$ and $P(x)$ is the position of x in the permutation

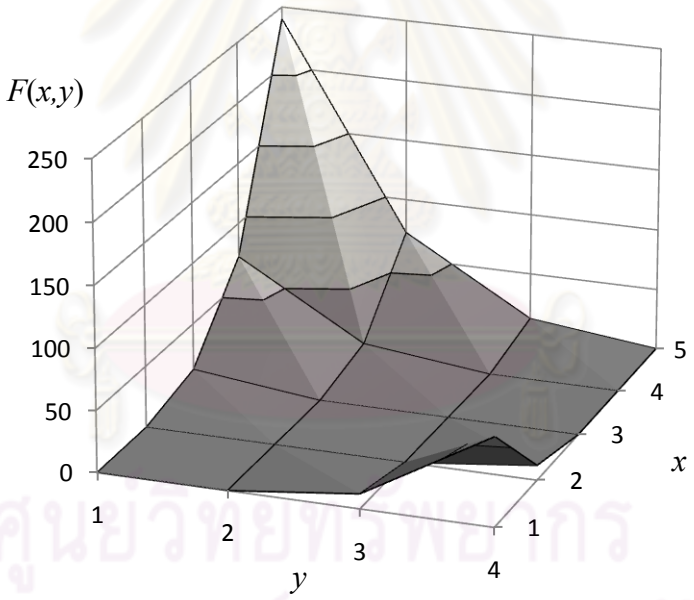


Figure 1.1 Value of $F(x,y)$

Table 1.1 Value of $Q(x,y,z)$ ： $Q(1,2,2), Q(1,1,3), Q(3,3,2)$ … etc. are non-valid permutations

	Value					
x,y,z	1,2,3	1,3,2	2,1,3	2,3,1	3,1,2	3,2,1
$Q(x,y,z)$	1+4+9 = 14	1+6+6 = 13	2+2+9 = 13	2+6+3 = 11	3+2+6 = 11	3+4+3 =10

Consequently, in combinatorial problems, the concept of distance between different coordinates seems to be senseless and cannot be applied in order to estimate or predict the goodness of a solution. In permutation spaces, most optimization methods rely on *Proximate Optimality Principle* (POP) [2] which consider the continuity of each candidate solution based on its neighborhoods.

There are many approaches to classify metaheuristics, which will be precisely described later in Chapter II. In this dissertation, we focus on the ways a solution is generated, that are *constructive* and *improvement*. Constructive methods generate a solution by joining together "pieces" or "components" of a solution, while improvement methods generate a new solution from a pre-existent one and try to improve it by modifying some of its component. Constructive strategies are sometimes called *recombination*, while improvement strategies are usually known as *local search*. The constructive methods have an advantage over the improvement methods as they usually produce more diversity of solutions whereas the improvement methods have advantage on the quality of solutions.

1.2.2 Neighborhoods and their similarities

As already mentioned in section 1.2.1, combinatorial problems cannot be represented geometrically in a multidimensional space. However, there are researches on geometric permutations [3][4][5], which try to represent models to traverse in the permutation spaces. Somehow, geometric permutations are not mature to be applied in metaheuristics.

In an improvement scheme, the continuity of each solution depends on its neighborhoods. The neighbors of a solution depend on one or more move operators defined by the dedicated algorithm. Example 1.3 illustrates two different ways to define move operators of an order-3 permutation problem that are neighborhood based on swap operator and neighborhood based on rotation operator. It is exemplified that using a rotation operator alone cannot traverse to some of the solution in the search space. However, the rotation operator preserves larger sequences of concatenated items than the swap operator. Consequently, many researches need to design such move operator to suit the problems.

Example 1.3: neighborhood for a permutation based on different move operator

Figure 1.2 shows the neighborhood of permutation s of x where $x \in [1\,..\,3]$, where figure (a) shows the neighborhood based on swap operator and (b) shows the neighborhood based on rotation operator. The neighbors of the solution (2,3,1) in (a) are (3,2,1), (2,1,3), and (1,3,2) while in (b) are only (1,2,3) and (3,1,2).

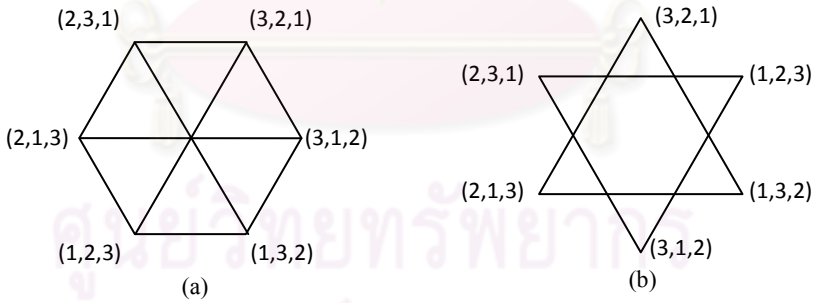


Figure 1.2 Example of neighborhood for a permutation problem of size 3.
(a) neighborhood based on swap operator.
(b) neighborhood based on rotation operator.

Foundations of local search methods are based on a principle called Proximate Optimality Principle (POP) which assumes that good solutions share similar substructure. Therefore, move operators are designed in order to generate the solutions that are considered to share similarity in many ways which will be discussed more in the

next Chapter. Figure 1.3 illustrates two types of similarities. The highlighted blocks indicate the similarity the two sequences are sharing. (a) indicates the similarity of items align in the same column call absolute order while (b) indicates the similarity of the maximum sequence found in two candidate solutions.

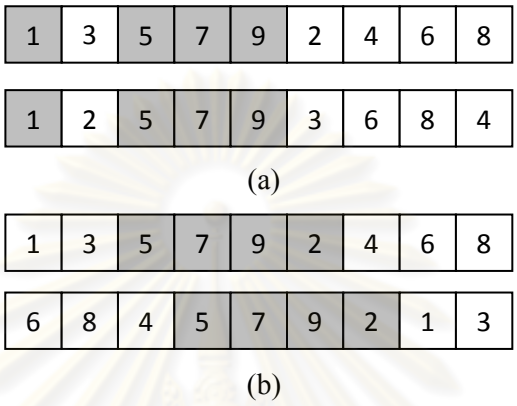


Figure 1.3. Absolute and relative positioning based similarity in permutation representation

Clearly, the “neighborhood” concept emphasizes local search. In seeking ever better solutions, local search methods employ a sensible tenet: solutions that are similar in structure will generally be similar in fitness. With a little thought, especially given that suitably good solutions tend to make up only a tiny fraction of the search space, this implies that it is best to search locally in the region of the best solutions found so far.

Local search methods therefore work via exploitation of the best candidate solutions attained so far. That is, the structures of such candidates are exploited constantly as a template for potentially better solutions. In contrast, there is usually little exploration in local search. Such emphasis on exploitation corresponds to a very high selection pressure strategy, with consequent well-known pitfalls. In particular, local search techniques are highly prone to become “trapped” at solutions that are locally optimal, with no means of escape toward better solutions that may exist elsewhere in the fitness landscape.

1.2.3 Recombination and disruption in permutation representation

Constructive methods differ from improvement methods in that they balance effort between exploitation and exploration in a way that turns out to be more effective in many applications. In combinatorial optimization, constructive schemes usually refer to genetic algorithms. (not all evolutionary algorithms) The idea of crossing over aiding the search process by recombining short, high fitness sections of the genotype called *building blocks*. The aim of crossing over is to propagate these high fitness building blocks throughout the population, raising average fitness by steering the population towards promising areas of the search space. Difficulties quickly arise when a simple genetic algorithm is applied. In particular, the encoding of a solution as a bit string is not convenient as most sequences in the search space would not correspond to the feasible solutions. Thus the permutation representation rather preferred in this class of problem. However, directly applying classical recombination operators such as simple one-point or two-point crossover to permutations will generate solutions that are invalid and needed to be fixed. Accordingly, specialized permutation operators must be developed. A disruption caused by a simple crossover is exemplified in example 1.3

> **Example 1.3**: Application of the one-point crossover on the two permutation chromosomes.
>
> Applying the one-point crossover operator at position 2 creates two infeasible offspring, as illustrated in figure 1.4, none of the two offspring is a valid permutation solution. The darker blocks indicate the redundant components in the offspring which are no longer considered being permutations.

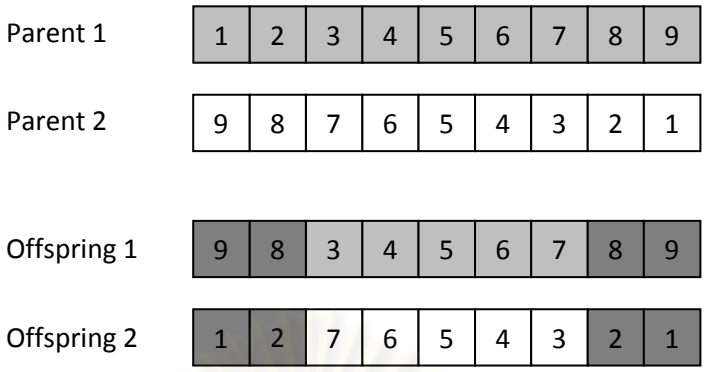


Figure 1.4 Application of the one-point crossover on the two permutation chromosomes

Consequently, various approaches have been proposed to avoid the disruption problems in permutation representation which can be broadly grouped into two classes: (*i*) those that focus on improving the crossover operator by adjusting the overall crossover procedure in such a way that it is less likely to disrupt any distributed knowledge stored in the genetic representation, and (*ii*) those that focus on improving the genetic representation by implementing a one-to-one mapping between genotype and phenotype so that several genetic permutations of the same phenotypic solution cannot co-exist in the same solution. These approaches will be discussed more in detail in the Chapter II.

1.2.4 Degree of freedom, exploitation and exploration

According to the *convergence argument* proposed by [6], the characteristics of permutation encoding GA are (*i*) genetic convergence occurs during the initial generations after which most members of the population will have similar genetic representations, and therefore (*ii*) several significantly distinct permutations of the same solution are unlikely to co-exist. In this case using an absolute order preservation crossover operator is likely to produce offspring with similar fitness to their parents.

To see a better perspective, we illustrate the effect of order based crossovers for permutation encoding GA. With a comparable decision space, a solution encoded in permutation has much less degree of freedom than a solution encoded in binary. For instance, an ordering problem with search space equal to 16! or 2.09228×10^{13} feasible solutions would need only 16 degrees of freedom for a permutation representation while it takes up to 64 degrees of freedom for a binary representation. Let the feasible spaces be equivalent and let the schema domination rate for a degree of freedom be equivalent, the trend of the search space reduction would be in the figure 1.5.

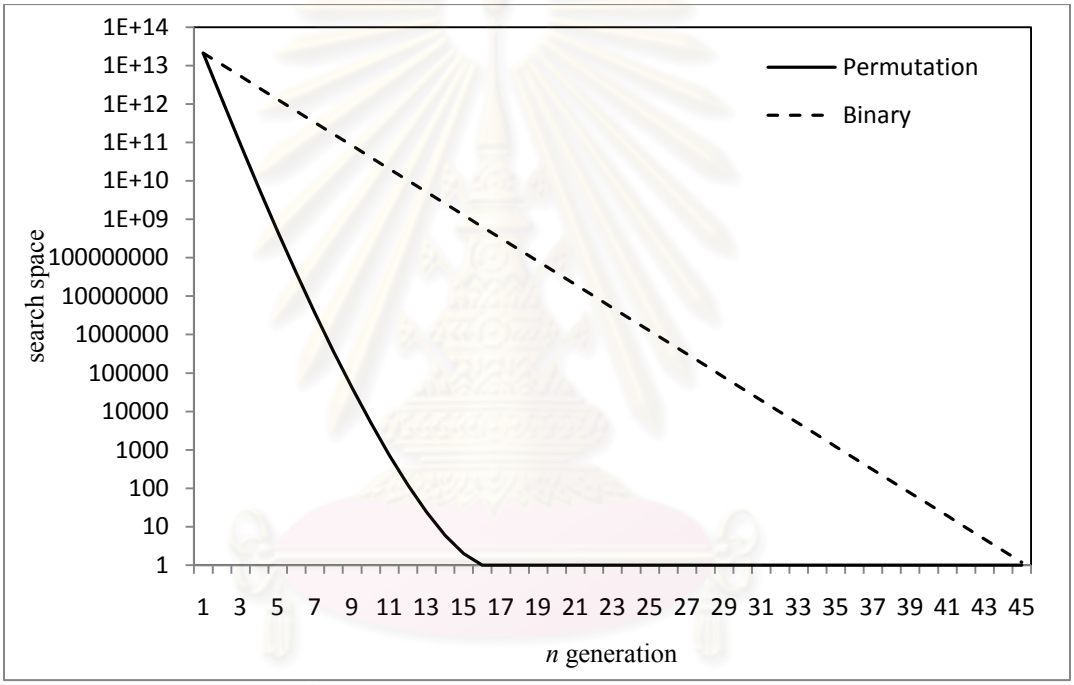


Figure 1.5 A comparison of search space reduction by permutation vs. binary schema at a same schema domination rate

Allowing the candidate solutions to be encoded in permutation rather than binary representation would increase the convergence rate due to the higher significant number of the degree of freedom of each candidate position. The higher order of the permutation size indicates the higher degree of freedom. In addition, high fitness substructures of a genotype are likely to dominate the population faster due to the constraint of the permutation does not allow the redundancy of an item elsewhere in the

permutation solution. This property can either be advantage or disadvantage of the constructive method. If some absolute order substructures show more outstanding fitness than the other, they would cause high exploitation rate, yet lack of diverse solution. On the other hand, if the substructures show similar fitness's at any position, the algorithm would not be able to converge to any single optima. For these reasons, the researchers need to take good care of the population size and the diversity of the initial population. If the initial population is not well diversified, for especially in an ordering problem with larger degrees of freedom, a premature convergence can occur for any population based method. If using an oversize population, the algorithm would waste too many function evaluations in exploring the search space.

1.2.5 Building blocks and linkage learning

Further, when crossover is employed as a variation operator, the GA increasingly samples *combinations* of building blocks, possibly discovering new ones as a result[7]. There is, however, much debate over this hypothesis. For example, in some problems there are verifiably no building blocks at the genotype level [8] and has been shown to be somewhat problem dependent. Despite the lack of building blocks in some problems, crossover may still be an effective search operator. Accordingly, this implies that crossover operators are able to have beneficial effects that do not involve the recombination of building blocks. It is shown that for many problems where crossover was believed to be recombining building blocks, it was in fact performing a macromutation [9][10].

Additionally, other aspects of how genetic algorithms work have been questioned with the result that long-held principles have been shown to be false or incomplete. Jones [11] presents a means by which the possible existence of building blocks in a genotype can be ascertained in a less ambiguous manner than with previous methods. The problem is first attempted using a process of proportional selection and crossover, then compared to the same process using random crossover. The aim is to disrupt the building blocks (if any are indeed present) using the random crossover operator and see how performance is affected. Random crossover entails performing

crossover on one fit individual and a randomly generated individual. Given that the second parent has been generated randomly, its fitness will on average be very low. Combining a fit individual with a random individual effectively removes the implicit information sharing offered by a population which clearly violates the idea of crossover. If this approach is at least as effective as traditional crossover then it suggests that we do not require the idea of crossover, but that its mechanics may be effective.

At first sight this lack of empirical evidence may seem odd, in particular because the permutation problem appears to encapsulate the reasonable claim that it usually makes little sense to recombine individuals who are genetically very dissimilar. As Watson and Pollack [12] point out that "*parents selected from two different fitness peaks are likely to produce an offspring that lands in the valley in between*". For this reason, constructive methods for permutation problems are not receiving good attention by many researchers.

In addition to the building block hypothesis, Holland [13] has also suggested that operators learning linkage information to recombine alleles might be necessary for genetic algorithm success. Afterward, many methods have been developed to solve the linkage problem. The linkage model can be implicit [14] or explicit [15], probabilistic [16] (probabilistic model building genetic algorithms), or estimation of distribution algorithm [17]. Unfortunately, most of the works are based on binary and continuous representation. The outstanding algorithms used to solve the problem in permutation representation domains are Edge histogram based sampling Algorithms (EHBSAs) [18] and Node histogram based sampling Algorithms (NHBSAs) [19] proposed by Tsutsui.

1.3 Research Motivations

Over a few decades, many metaheuristics have been proposed in order to solve both single and multiple objective combinatorial optimization problems especially those which can be represented in permutation. However, many researches turn to be somewhat problems dependent. For example, local search methods need well designed move operators in order to produce effective neighborhoods or genetic algorithms need different appropriate crossover operators to many specific problems. Even though most metaheuristics rely on either *Proximate Optimality Principle* (POP) [2] or *Building Blocks Hypothesis* (BBH) [7], anyhow, due to the constraint of the representation, linkage learning models are rarely been applied. The only few metaheuristics considered to learn the linkage of the substructure contained in the solutions are *Ant Colony Optimization algorithms* (ACO)[20], *Edge Histogram Based Sampling Algorithms* (EHBSA)[18] and *Node Histogram Based Sampling Algorithms* (NHBSA)[19].

The motivation of this research mainly based on a question whether the below average solutions that population based metaheuristics usually discard contain useful information and can be used in optimization or not. As a result, we raise a model capable to learn the linkage of bad substructures in order to produce solutions not containing them. We propose a hypothesis called a *Negative Building Block Hypothesis* (NBBH) simply states that "*An algorithm can seeks new-optimal performance by avoiding the juxtaposition of short, low-order, low-performance schemata, called the negative building blocks*". Further, we expect this hypothesis could fulfill as a counterpart of the BBH.

1.4 Doctoral Framework

According to Bassett's observation [21] that the crossover operator works in the problems in which there is no building blocks exist in the genotype level, we suspect that there might be bad building blocks that the crossover operator might filter out in order to form the better solutions. From this observation, we propose the Negative Building Block Hypothesis (NBBH) and try to test this hypothesis using a simple scientific method. This doctoral framework is as follows: literature surveying, making hypothesis, design and perform experiments to test the hypothesis and conclude the results.

1.5 Research Objectives

The research objectives are to develop a new evolutionary algorithm for single and multiple objectives combinatorial optimization problems and to study the role of applying negative knowledge in evolutionary algorithm for combinatorial optimization problems.

1.6 Scope of the Study

This research proposed to utilize the negative knowledge in combinatorial optimization; however, limited to the design, implementation and testing of edge based EDAs. The benchmarks in this research include multimodal artificial combinatorial problems and some real world applications.

Some broad issues are ignored in the scope of the study and can be developed further.

1. The studies of negative knowledge in the absolute order such as the node based estimation of distribution algorithms.
2. The studies of negative knowledge in the geometry based algorithm.
3. The studies of parameter tuning of such algorithm.
4. The studies of local search and hybridization with other metaheuristics.

1.7 Research Contributions

The outcomes derived from this research include:

1. The first contribution of this research is a new estimation of distribution algorithm (EDA) based on permutation representation call *Coincidence Algorithm* (COIN) which is naturally more suitable with most combinatorial problems. This contribution is twofold. (*i*) a probabilistic model based on Markov chain matrix and (*ii*) an incremental learning method that allow negative correlation learning of the samples.

2. The second contribution is a negative building block hypothesis (NBBH) simply state that "*An algorithm can seeks new-optimal performance by avoiding the juxtaposition of short, low-order, low-performance schemata, called the negative building blocks*"

3. Thirdly, a set of benchmark to test the performance of algorithm in solving globally multimodal optimization problems including both permutation and selection problems. In addition, an alternative method to solve the fix-size combination problems which most metaheuristics are not able to solve is also proposed. The results indicate that negative correlation learning capability contributes in both quantity and quality of the solutions, however, depends mainly on the quantity of building blocks being shared and the quantity of building blocks being in conflict. The insight discussion can be seen in the Chapter V.

4. As a highlight, the roles of negative correlation learning specific in edge based EDA are extracted as followed:

1) The negative knowledge forces the algorithm to explore out of the search space marked as forbidden areas.
2) The negative knowledge helps the algorithm to produce more diverse solutions, however dissimilar to the solutions considered to be bad quality.
3) In cooperating with the positive knowledge, the negative knowledge contributes in discrimination of good and bad substructure.

4) The negative knowledge should enhance a constructive algorithm to recognize better substructures and to compose better solutions.

5. Finally, the most important contribution is the extension of COIN in solving multi-objective problems by applying the non-dominated sorting and crowding distance adopted from NSGA-II. The new algorithm was test in several real world problems. The results state that multi-objective version of COIN can defeat NSGA-II for all performance indicators.

1.8 Dissertation Structure

The outline of this dissertation is organized as follows. Chapter I states the general background, objective, scope of study, and contributions. The state of the art algorithms are reviewed in Chapter II. The negative knowledge which is the inspiration of this research is presented in Chapter III. The proposed algorithm is presented in Chapter IV. In Chapter V, a set of empirical study are discussed. Then we show the application of the proposed algorithm in some real world applications in Chapter VI. Finally, the conclusions and discussions of this research are presented and the future directions are also suggested in Chapter VII.

CHAPTER II
METAHEURISTICS FOR COMBINATORIAL OPTIMIZATION

2.1 Introduction

This chapter provides some necessary knowledge on solution methodologies for solving both single and multi-objective combinatorial problems. However, this dissertation focuses on the methods which apply to the permutation representation, which is naturally more suitable with combinatorial problems. The solution methodologies involving the transformation of representations are not in the scope of the review.

This chapter can be divided into two main parts. The first part is the review of the state of the art algorithms that are designed to solve combinatorial problems. The second part is the additional techniques needed to solve the problems with multi-objectives.

2.1.1 Combinatorial Optimization

Combinatorial optimization problems (COPs) [22] are characterized by the consideration of a selection or permutation of a finite or a countable discrete set of structures. This class of problems arises in many areas of pure mathematics, notably in algebra, probability theory, topology and geometry.

In order to prevent the ambiguous of the term "Combinatorial optimization problems" with any other literatures, we should first define this term [23]. A *combinatorial optimization problem* is either a minimization problem or a maximization problem with an associated set of *instances*.

Definition 2.1: **Combinatorial Optimization Problem**

An instance of a combinatorial optimization problem is a pair (S, f) where S is the finite set of candidate solutions and $f: S \rightarrow \mathbb{R}$ is a function which assign to every $x \in S$ a value $f(x)$ where $x = (x_1, \dots, x_k)$ is a feasible solution belong to the discrete solution set S. $f(x)$ is also called an objective function.

Combinatorial optimization considers the following problem:

Definition 2.2: **Combinatorial Optimization** is defined by

$$z(S) = \min_{x \epsilon S} f(x) \tag{2.1}$$

where $x = (x_1, \dots, x_k)$ is a feasible solution belong to the discrete solution set S, usually called the decision space or solution space. The function f maps S to $\mathbb{R}$ is called the objective function. Therefore $f(x)$ describes the objective function value of the solution x.

2.1.2 Solution Methods for Combinatorial Problems

According to Talbi[24], combinatorial optimization is a special class of optimization distinct from the mathematical programming models. Nevertheless, many literatures consider this class of optimization as a subclass of integer programming. This class of problems is characterized by discrete decision variables and a finite search space, moreover, the objective function and constraints may take any form[25]. Figure 2.1 shows the classification of optimization models.

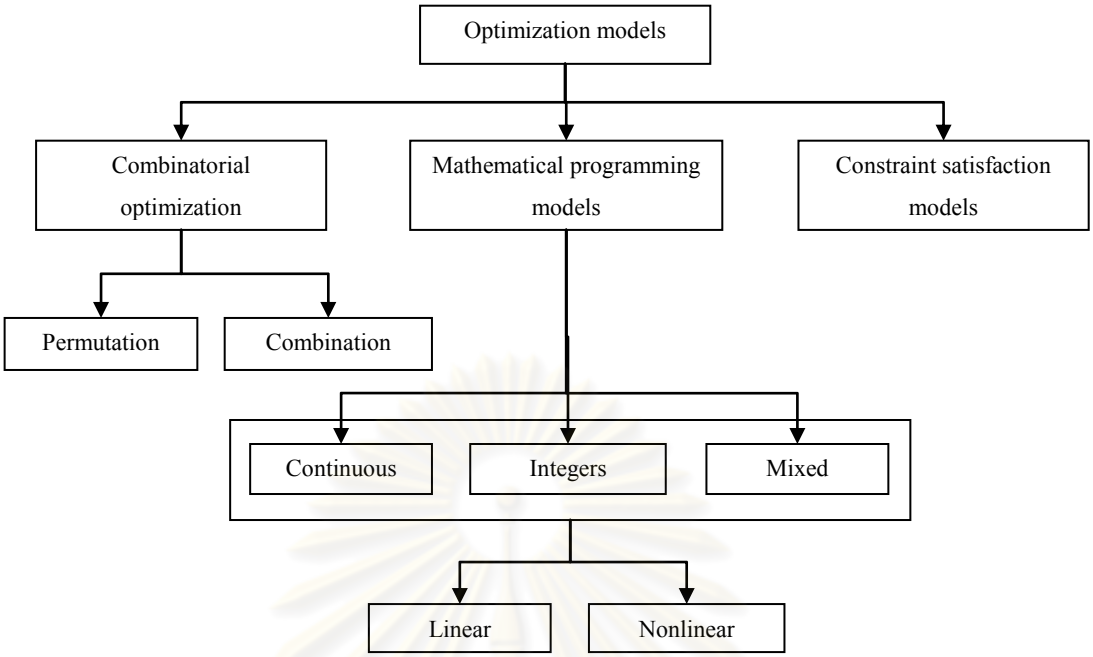


Figure 2.1 Classical optimization models [24]

As mentioned in Chapter I, the COPs are different from the mathematical programming problems as the optimize variables are usually indirectly used in order to evaluate a function. Therefore, the solution methodologies are also different to those mathematical programming models. In addition, many state of the art algorithms applied to different representations such as evolution strategy, differential evolution and most estimation of distribution algorithms (EDA) are inappropriate to solve these kinds of problems.

The algorithms to solve combinatorial problems can be divided into two classes called exact algorithms and approximate algorithms.

2.1.2.1 Exact algorithms

The exact algorithms are designed to find the optimal solution to the combinatorial problems. They are usually computationally expensive because they must (implicitly) consider all solutions in order to identify the optimum. These exact algorithms are typically derived from the integer linear programming (ILP) [26]. Branch and X (refer to Branch and Bound, Branch and Cut and Branch and Price and more variation) algorithms are commonly used to find an optimal solution to many combinatorial problems, however in many cases, partial fitness cannot be determined, thereby, without a heuristic to guide a search, such methods are not applicable.

2.1.2.2 Approximate algorithms

Many COPs are belonging to the class of NP-hard optimization problems [27]. This means that there the algorithms that guarantee to find the optimal solution within bounded time or exact algorithms might require the exponential computational time. Therefore, running an exact algorithm for hours on a powerful computer may not be very cost-effective. Accordingly, heuristic or approximate algorithms are often preferred to exact algorithms for solving the COPs. Heuristic strategies are receiving more and more interest as they can find the reasonable good solution (but not necessarily an optimal one) compared to the given computational time.

The term "*heuristic*" derives from the Greek verb "*heuriskein*" (*ενρισκειν*) which means "to find" or "to discover" it is in optimization not so much used to describe how to *find* as how to *search* for good solutions [28]. Generally, the exact algorithms can apply heuristic strategies, for example, to guide the search in a branch and bound procedure. However, the term heuristic is preferred to denote the approximate algorithm. There are mainly two types of heuristics, "*Constructive Algorithms*" and "*Improvement Algorithms*". Constructive algorithms build a solution by joining together "pieces" or "components" of a solution, while Improvement algorithms start from a pre-existent solution and try to improve it by modifying some of its component. Some heuristics algorithm combines both constructive and improvement strategies altogether, we call these algorithms "*Composite Algorithms*", "*Hybrid Algorithms*" or "*Memetic Algorithms*". These composite algorithms are now the most powerful heuristics for solving COPs. Among the new generation of composite heuristics, the most outstanding ones are the CCAO heuristic [29], the iterated Lin-Kernighan heuristic [30][31], and the GENIUS heuristic [32]. Unfortunately, most of the algorithms are problem-specific and are not in the scope of review. Somehow, they usually combine the existing constructive and improvement strategies found in this chapter.

The Greek suffix "*meta*" used in the word metaheuristics means "beyond, in an upper level". The term "metaheuristics" was first used by Glover [33] to describe a heuristic that is superimposed on another heuristic. Generally speaking, metaheuristics are algorithms that combine heuristics (that are usually problem specific solvers) in a more general framework.

According to Blum and Roli [34], metaheuristics are high level concepts for exploring search spaces by using different strategies. These strategies should be chosen in such a way that a dynamic balance is given between the exploitation of the accumulated search experience and the exploration of the search space. This balance is necessary on one side to quickly identify regions in the search space with high quality solutions and on the other side not to waste too much time in regions of the search space which are either already explored or don't provide high quality solutions.

The different metaheuristics approaches can be characterized by different aspects concerning the search path they follow or how memory is exploited. In this section, we discuss these aspects according to some general criteria which may be used to classify the presented algorithms. For a more formal classification of local search algorithms based on an abstract algorithmic skeleton we refer to [34].

Trajectory methods vs. discontinuous methods: An important distinction between different metaheuristics is whether they follow one single search trajectory corresponding to a closed walk on the neighborhood graph or whether larger jumps in the neighborhood graph are allowed.

Population-based vs. single-point search: Related to the distinction between trajectory methods and discontinuous walk methods is the use of a population of search points or the use of one single search point. In the latter case only one single solution is manipulated at each iteration of the algorithm.

Memory usage vs. memoryless methods: Another possible characteristic of metaheuristics is the use of the search experience (memory, in the widest sense) to influence the future search direction.

One vs. various neighborhood structures: Most local search algorithms are based on one single neighborhood structure which defines the type of allowed moves.

Nature-inspired vs. non-nature inspiration: A minor point for the classification of metaheuristics is to take into account their original source of inspiration. Many methods are actually inspired by naturally occurring phenomena. The algorithmic approaches try to take advantage of these phenomena for the efficient solution of combinatorial optimization problems.

2.2 Single-Solution Based Algorithms

2.2.1 Neighborhood and local search

Local search seems to be the oldest and simplest metaheuristics method [25,35]. It starts at a given initial solution. At each iteration, the heuristic replaces the current solution by a neighbor that improves the objective function. The search stops when all candidate neighbors are worse than the current solution, meaning that a local optimum is reached. For large neighborhoods, the candidate solutions may be a subset of the neighborhood. The main objective of this restricted neighborhood strategy is to speed up the search. Variants of LS may be distinguished according to the order in which the neighboring solutions are generated (deterministic/stochastic) and the selection strategy (selection of the neighboring solution).

```
PROCEDURE BasicLocalSearch

1. s <- GenerateInitialSolution()

2. Repeat

3.     s <- Improve(N(s))

4. Until no improvement is possible
```

Algorithm 2.1: **Basic Local Search**

Definition 2.4: **The neighborhood of a permutation [35]**

The neighborhoods $N(s)$ of a permutation string s is represented by the set $\{s'/d(s',s) \leq \epsilon\}$ where d represents a given distance that is related to the move operator.

Definition 2.5: **A locally minimal solution (or local minimum) [35]** with respect to a neighborhood structure N is a solution $\hat{s}$ such that $\forall\, s \in N(\hat{s}) : f(\hat{s}) \leq f(s)$. We call $\hat{s}$ a strict locally minimal solution if $f(\hat{s}) < f(s) \forall\, s \in N(\hat{s})$.

2.2.1.1 Move operators

For permutation-based representations, a usual neighborhood is based on the swap operator that consists in exchanging or swapping the position of two items S_i and S_j of the permutation. For a permutation of size n, the size of this neighborhood is $n(n-1)/2$.

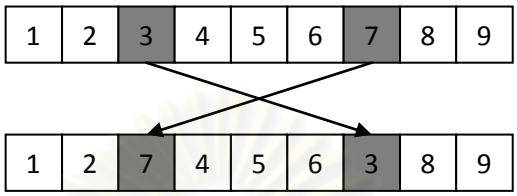


Figure 2.2 Swap operator

However, a swap operator might not be able to produce the neighborhoods that share some similarity apart from the absolute positioning of items. For instance, the insertion operator shown in figure 2.3 preserves both absolute and relative similarities. Figure 2.4 shows a rotation operator which preserve relative similarity and Figure 2.5 shows inversion operator which is a generalization version of 1-Opt, 2-Opt, 3Opt and Double Bridge operators respectively.

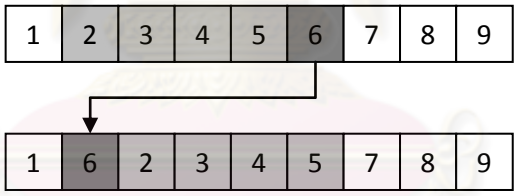


Figure 2.3 Insertion operator

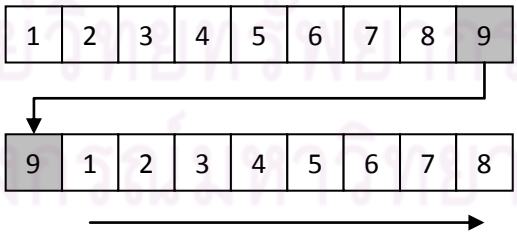


Figure 2.4 Rotation operator

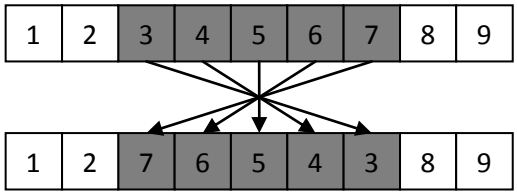


Figure 2.5 Inversion operator

2.2.1.2 Selection of the Neighbor

There are many strategies to select a better neighbor [24][35] including best improvement, first improvement and random select. A compromise in terms of quality of solutions and search time may consist in using the first improvement strategy when the initial solution is randomly generated and the best improvement strategy when the initial solution is generated using a greedy procedure. In practice, on many applications, it has been observed that the first improvement strategy leads to the same quality of solutions as the best improving strategy while using a smaller computational time. Moreover, the probability of premature convergence to a local optima is less important in the first improvement strategy.

In general, local search is a very easy method to design and implement and gives fairly good solutions very quickly. This is why it is a widely used optimization method in practice. One of the main disadvantages of LS is that it converges toward local optima. Moreover, the algorithm can be very sensitive to the initial solution; that is, a large variability of the quality of solutions may be obtained for some problems. Additionally, there is no means to estimate the relative error from the global optimum and the number of iterations performed may not be known in advance. Even if the complexity is acceptable, the worst case complexity of LS is exponential. Local search works well if there are not too many local optima in the search space or the quality of the different local optima is more or less similar.

As already mentioned in the Chapter I that the main disadvantage of local search algorithms is the convergence toward local optima, many alternatives algorithms have been proposed to avoid becoming stuck at local optima.

2.2.2 Simulated annealing

Simulated annealing (SA) emerges from the work of Kirkpatrick et al. [36] and Cerny [37]. Previously, SA has been applied to graph partitioning [36] and VLSI design [37]. In the 1980s, SA had a major impact on the field of heuristic search because of its simplicity and efficiency in solving combinatorial optimization problems. Then, it has been extended to deal with continuous optimization problems [38][39].

SA is based on the principles of statistical mechanics whereby the annealing process requires heating and the slowly cooling a substance to obtain a strong crystalline structure. The strength of the structure depends on the rate of cooling metals. If the initial temperature is not sufficiently high or a fast cooling is applied, imperfections (metastable states) are obtained. In this case, the cooling solid will not attain thermal equilibrium at each temperature. Strong crystals are grown from careful and slow cooling. The SA algorithm simulates the energy changes in a system subjected to acooling process until it converges to an equilibrium state (steady frozen state).

Table 2.1 Analogy between the physical system and the optimization problem [24]

Physical System	Optimization Problem
System state	Solution
Molecular positions	Decision variables
Energy	Objective function
Ground state	Global optimal solution
Metastable state	Local optimum
Rapid quenching	Local search
Temperature	Control parameter T
Careful annealing	Simulated annealing

Table 2.1 illustrates the analogy between the physical system and the optimization problem. The objective function of the problem is analogous to the energy state of the system. A solution of the optimization problem corresponds to a system state. The decision variables associated with a solution of the problem are analogous to the molecular positions. The global optimum corresponds to the ground state of the system. Finding a local minimum implies that a metastable state has been reached.

SA is a stochastic algorithm that enables under some conditions the degradation of a solution. The objective is to escape from local optima and to delay the convergence. From an initial solution, SA proceeds in several iterations. At each iteration, a random neighbor is generated. Moves that improve the cost function are always accepted. Otherwise, the neighbor is selected with a given probability that depends on the current temperature and the amount of degradation ΔE of the objective function. ΔE represents the difference in the objective value (energy) between the current solution and the generated neighboring solution. As the algorithm progresses, the probability that such moves are accepted decreases. This probability follows, in general, the Boltzmann distribution:

$$P(\Delta E\,, T) = e^{\frac{f(s')-f(s)}{T}} \tag{2.2}$$

It uses a control parameter, called temperature, to determine the probability of accepting nonimproving solutions. At a particular level of temperature, many trials are explored. Once an equilibrium state is reached, the temperature is gradually decreased According to a cooling schedule such that few nonimproving solutions are accepted at the end of the search. Algorithm 2.2 describes the template of the SA algorithm.

```
PROCEDURE SimulatedAnnealing

1. s <- GenerateInitialSolution()

2. Initialize Temperature T

3. Repeat

4.   select a random solution y from Neighborhood(x)

5.   if f(y) > f(x) then x <= y

6.   else if exp((f(y)-f(x))/Temp) < random[0;1] then x <- y

7.   if f(x) > BestFx then {BestX <- x and BestFx <- f(x)}

8.   Update(Temp)

9. Until Termination Condition is met
```

Algorithm 2.2: Simulated Annealing

2.2.3 Iterated Local Search

A major problem for local search algorithms is that they may get trapped in local optima in the search space. In such a situation, an action should take place that allows the local search to leave local minima and to continue the search for possibly better solutions. One straightforward possibility is to modify the current locally optimal solution s using a modification larger than those used in the local search algorithm. The application of such a move yields some intermediate solution s_0 beyond the neighborhood searched by the local search algorithm and allows to leave local minima. The local search is then continued from s_0. Iterated local search (ILS) [30][40] systematically uses this idea to solve combinatorial optimization problems. In ILS a local search algorithm is applied repeatedly from initial solutions obtained by modifications to one of the previously visited locally optimal solutions.

ILS is a simple, yet powerful metaheuristic to improve the performance of local search algorithms. The simplicity stems from the underlying principle and the fact that only few lines of code have to be added to an already existing local search procedure to implement an ILS algorithm. ILS also can be expected to perform better than to restart local search from a new, randomly generated solution. This is emphasized by the fact that ILS algorithms are currently among the best performing approximation methods for many combinatorial optimization problems like the traveling salesman problem [40].

To apply an ILS algorithm to a given problem, three “ingredients” have to be defined. One is a procedure Modify, that perturbs the current solution s (usually a local optimum) leading to some intermediate solution s0. We will refer to the perturbation also as *kick-move* in the following. Next, LocalSearch is applied taking s0 to a local minimum s00. Finally, one has to decide which solution should be chosen for the next modification step. This decision is made according to an AcceptanceCriterion that takes into account the previous solution s, the new candidate solution s00 and possibly the search *history*.

```
PROCEDURE IterateLocalSearch

1. generate initial solution s

2. s <- LocalSearch(s)

3. sBest <- s

4. repeat

5.   s' <- Modify(s,history)

6.   s" <- LocalSearch(s')

7.   if (f(s")<f(sBest)) then sBest <- s"

8. Until Termination Condition is met
```

Algorithm 2.3: Iterated Local Search

2.2.4 Tabu Search

Tabu search (TS) is an iterative local search metaheuristic [33,2]. The most distinctive feature of TS compared to other metaheuristics is the systematic use of a *memory* to guide the search process. For the detail discussions of its features, we refer to the recently published book by Glover and Laguna [2].

The most widely applied feature of Tabu search is the use of a short term memory to escape from local minima. TS typically uses an aggressive local search that in each step tries to make the best possible move from s to a neighbor s_0 even if that move worsens the objective function value. To prevent the local search to immediately return to a previously visited solution and to avoid cycling, moves to recently visited solutions are forbidden. This can be implemented by explicitly memorizing previously visited solutions and forbidding moving to those. More commonly, reversing recent moves is forbidden by disallowing the introduction of move attributes to a solution. In particular, reverse moves are forbidden for tl iterations; the parameter tl is called the *tabu tenure*. Forbidding possible moves has the same effect as restricting dynamically the neighborhood $N(s)$ of the current solution s to a subset of admissible solutions. Thus, Tabu search can also be considered as a dynamic neighborhood search technique. Yet, the Tabu conditions

may be too restrictive and they may forbid moves to attractive, unvisited solutions. *Aspiration criteria* are used to override the tabu status of certain moves and to avoid such situations. Most commonly, the aspiration criterion drops the tabu status of moves leading to a better solution than the best one visited so far.

```
PROCEDURE TabuSearch

1. Find a feasible solution x

2. BestX <- x and BestFx <- f(x) and TabuList <- {}

3. Repeat

4.   y <- Argmax {f(y) | y ∈ Neighbor(x)∧MoveAttribute(x,y) ∉
Tabulist}

5.   if length(tl) > TabulistLength then remove the oldElement
from TabuList

6.   add MoveAttribute(y,x) as the newest element to TabuList

7.   x <- y

8.   if f(x) > BestFx then { BestX <- x and BestFx <- f(x) }

9. Until Termination Condition is met
```

Algorithm 2.4: Tabu Search

To increase the efficiency of Tabu search, techniques exploiting the *long-term memory* of the search process are used. These methods are used to achieve intensification or diversification of the search process. Intensification strategies correspond to efforts of revisiting promising regions of the search space either by recovering elite solutions (that is, the best solutions obtained so far) or attributes of these solutions. Diversification refers to exploring new search space regions corresponding to the introduction of new attribute combinations. Many long term memory strategies in the context of TS are based on a frequency memory on the occurrence of solution attributes.

TS appears to be one of the most successful metaheuristics. For many problems, TS implementations are among the algorithms giving the best tradeoff between solution quality and the computation-time required [41,42].

2.2.5 GRASP

Greedy randomized adaptive search procedures (GRASP) [43][44] allow escaping from local minima by generating new starting solutions. Each GRASP iteration consists of two phases, a *construction phase* and a *local search phase*. In the construction phase a solution is constructed from scratch, adding one solution component at a time. At each construction iteration the components to be added are contained in a *restricted candidate list* which is defined according to a *greedy function*. However, not necessarily the best component is added. Instead, in each solution construction step one of the components of the restricted candidate list is chosen at random according to a uniform distribution. The algorithm is called adaptive because the greedy function value for each component is updated reflecting the changes due to the previously added component. The constructed solutions are not guaranteed to be locally optimal with respect to some simple neighborhood definition. Hence, in the second phase local search is applied to improve solutions.

```
PROCEDURE GRASP

1. generate initial solution s

2. repeat

3.   s <- ConstructGreedyRandomizeSolution()

4.   s' <- LocalSearch(s)

5.   if f(s')< f(sBest) then sBest <- s'

6. Until Termination Condition is met
```

Algorithm 2.5: GRASP

2.3 Population-Based Algorithms

2.3.1 Genetic Algorithms

Genetic algorithm (GA) [13][7] is a specific type of *evolutionary algorithms* [45]. Evolutionary algorithms are population-based, adaptive search algorithms designed to attack optimization problems. They are inspired by models of natural evolution of species and use the principle of natural selection which favors individuals that are more adapted to a specific environment for survival and further evolution. Each individual in an evolutionary algorithm typically represents a solution with an associated fitness value. The three main operators used are *selection*, *mutation*, and *recombination*. Selection prefers fitter individuals to be chosen for the next generation and for the application of the mutation and recombination operator. Mutation is a unary operator that introduces random modifications to an individual. Recombination combines the genetic material of two individuals, also called *parents*, by means of a *crossover* operator to generate new individuals, called *offsprings*.

The three main algorithmic developments within the field of evolutionary algorithms are genetic algorithms, evolution strategies [46] [47] and evolutionary programming [48]. These algorithms have been developed independently and, although these algorithms initially have been proposed in the sixties and seventies, only in the beginning of the nineties the researchers became aware of the common underlying principles of these approaches [45]. (For a detailed discussion of similarities and differences between these approaches we refer to [45].) Here we focus on genetic algorithms since they appear to be the best suited evolutionary algorithms for combinatorial optimization problems, which are the target of this dissertation. Evolution strategies and evolutionary programming differ from genetic algorithms by representing solutions directly as real valued parameters (in case of genetic algorithm applications to continuous parameter optimization problems the numbers are coded in binary form) and the much stronger reliance on mutation as a primary search operator. Indeed, in evolutionary programming only mutation is used for modifying solutions.

In the first GA applications, individuals were represented by bit strings of fixed length [13]. Yet, this type of representation proved to be insufficient to

efficiently attack certain types of combinatorial problems [49] like permutation problems. Therefore, for such problems usually more general, problem specific encodings are applied. The *crossover* operator is usually understood as the main operator driving the search in genetic algorithms. The idea of crossover is to exchange useful information between two individuals and in this way to generate a hopefully better offspring. Mutation is understood as a background operator which introduces small, random modifications to an individual. Yet, recent results suggest that the role of mutation has been underestimated [45]. The selection operator is used to keep the population at a constant size, choosing preferably individuals with higher fitness (survival of the fittest). The complete cycle of recombination, mutation and selection is called *generation*.

```
PROCEDURE GeneticAlgorithm

1. generate initial population p

2. repeat

3.   p' <- Recombination(p)

4.   p' <- Mutation(p)

5.   p <- Selection(p,p')

8. Until Termination Condition is met
```

Algorithm 2.6: Simple Genetic Algorithm

2.3.1.1 Crossover operators

As mentioned in Chapter I, the recombination of two permutation sequences is not straight forward. Applying GAs in combinatorial problems turns to be somewhat problem dependent. Choosing an inappropriate crossover operator would not improve the populations, yet disrupt the schemas as well. Many permutation based crossover operators have been consecutively proposed since 1985. They are broadly categorized to preserve the schemas in the parent solutions in three manners including absolute order, relative order and edge. The permutation based crossover operators are reviewed as followed:

2.3.1.1.1 Partially-mapped crossover (PMX) [50] was first proposed by Goldberg and Lingle. This operator first randomly selects two cut points on both parents. In order to create an offspring, the substring between the two cut points in the first parent replaces the corresponding substring in the second parent. Then, the inverse replacement is applied outside of the cut points, in order to eliminate duplicates and recover all positions.

In figure 2.6, the offspring is created by first replacing the substring 4-3-7-6 in parent 1 by the substring 1-7-5-3. Then, the redundancy items in the parent 1 are mapped and were replaced by the matched items in the substring. Those are item 1 was replaced by item 4 while item 5 was replaced by item 6. However, the item 5 was mapped to item 7 which would be redundant to the items in the exchanged substring as well, therefore, the mapping procedure repeats until an available item is found. The item 5 finally mapped and replaced with the item 6 in the parent 2.

Clearly, PMX tries to preserve the absolute position of the items when they are copied from the parents to the offspring. In fact, the number of items that do not inherit their positions from one of the two parents is at most equal to the length of the string between the two cut points. From the above example, in the offspring 1, only item 1 and 5 do not inherit their absolute position from one of the two parents.

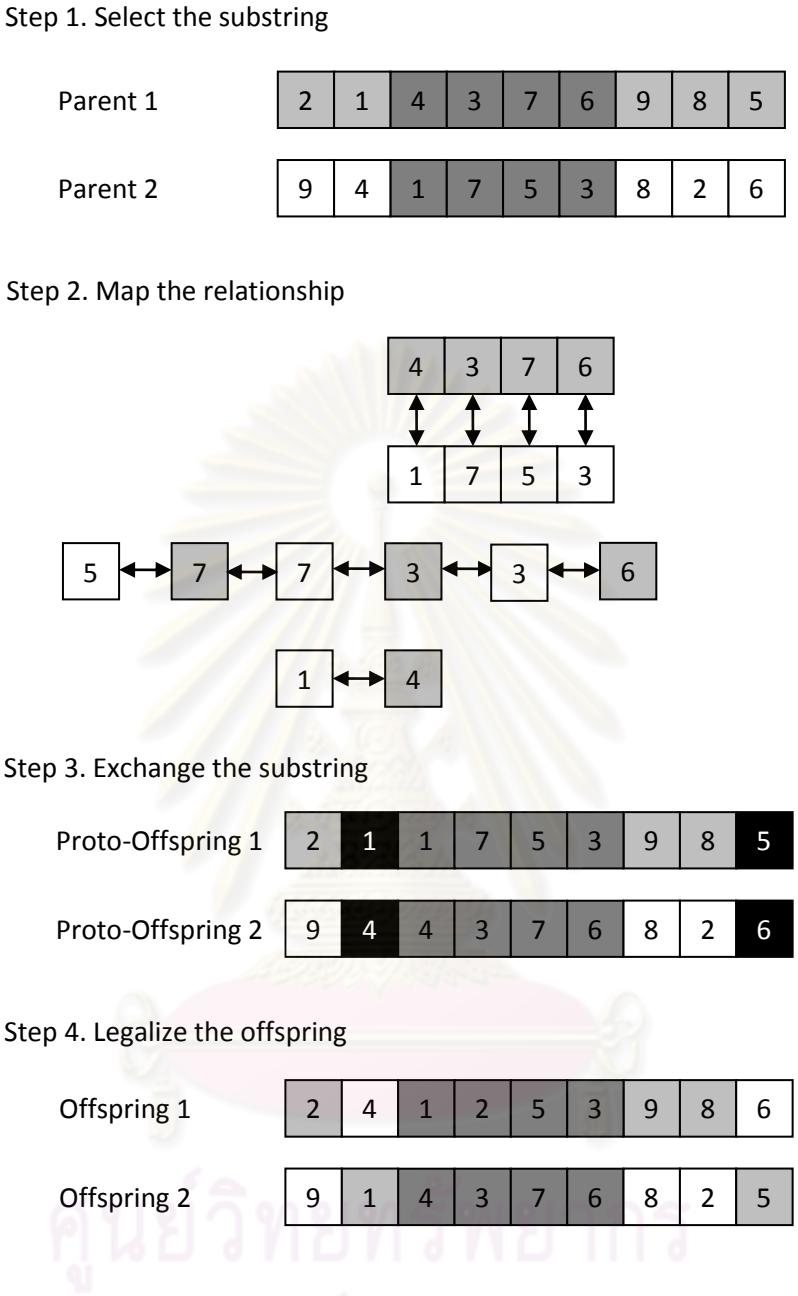


Figure 2.6 The partially-mapped crossover

2.3.1.1.2 Cycle crossover (CX) [51] was introduced by Oliver. The cycle crossover focuses on subsets of items that occupy the same subset of positions in both parents. Then, these items are copied from the first parent to the offspring (at the same position), and the remaining positions are filled with the items of the second parent. In this way, the position of each item is inherited from one of the two parents, However, many edges (connection between each item) can be broken in the process, because the initial subset of items is not necessarily located at consecutive positions in the parent strings.

In figure 2.7, In order to construct offspring 1, the subset of items $\{2, 9, 8\}$ occupies the subset of positions $\{1, 7, 8\}$ in both parents. Hence, an offspring is created by filling the positions 1, 7, and 8 with the items found in the parent 1, and filing the remaining positions with the items found in the parent 2.

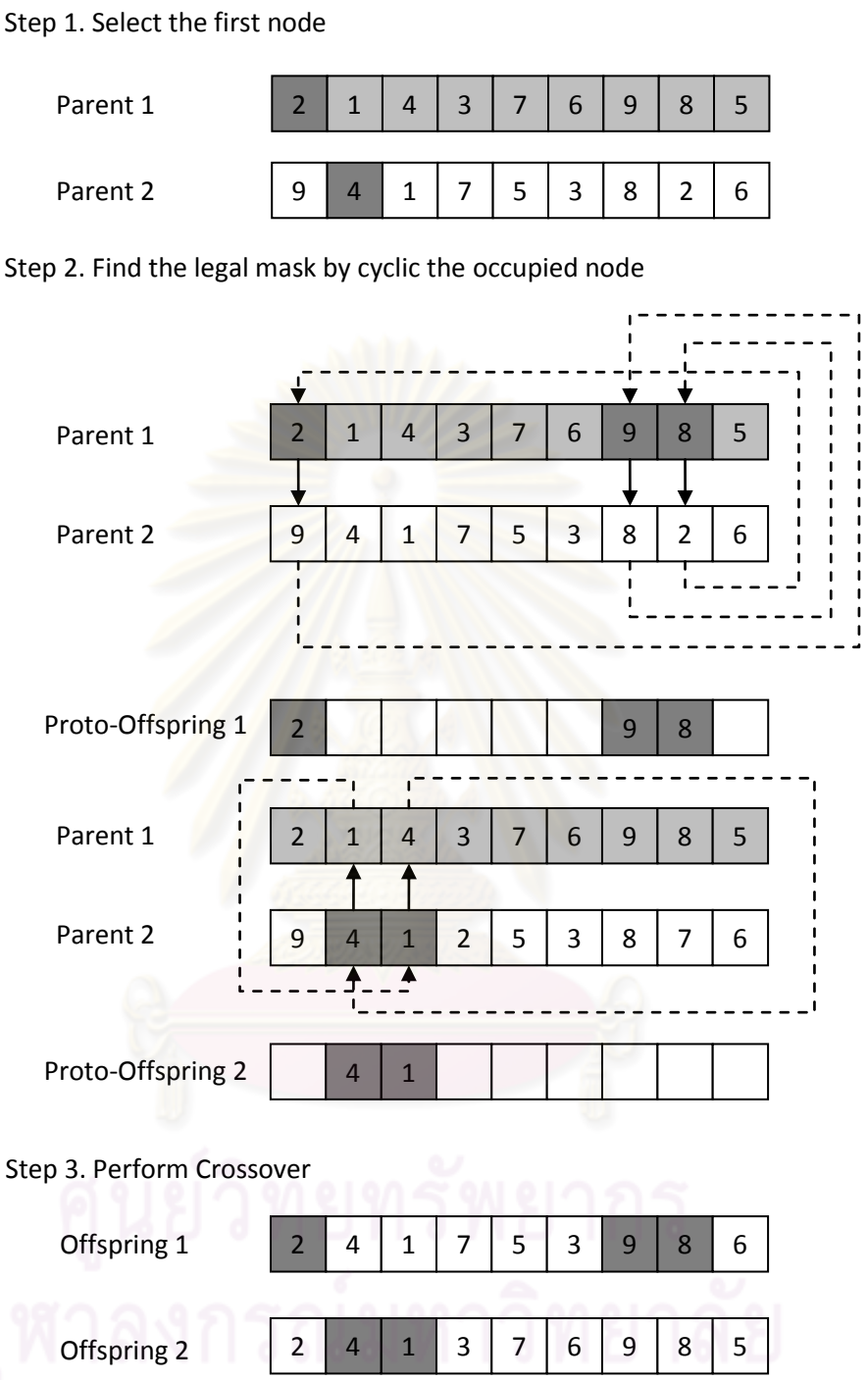


Figure 2.7 The cycle crossover

2.3.1.1.3 Modified crossover [52] was proposed by Davis. This crossover operator is an extension of the one-point crossover for permutation problems. A cut position is chosen at random on the first parent chromosome. Then, an offspring is created by appending the second parent chromosome to the initial segment of the first parent (before the cut point), and by eliminating the duplicates. An example is provided in figure 2.8.

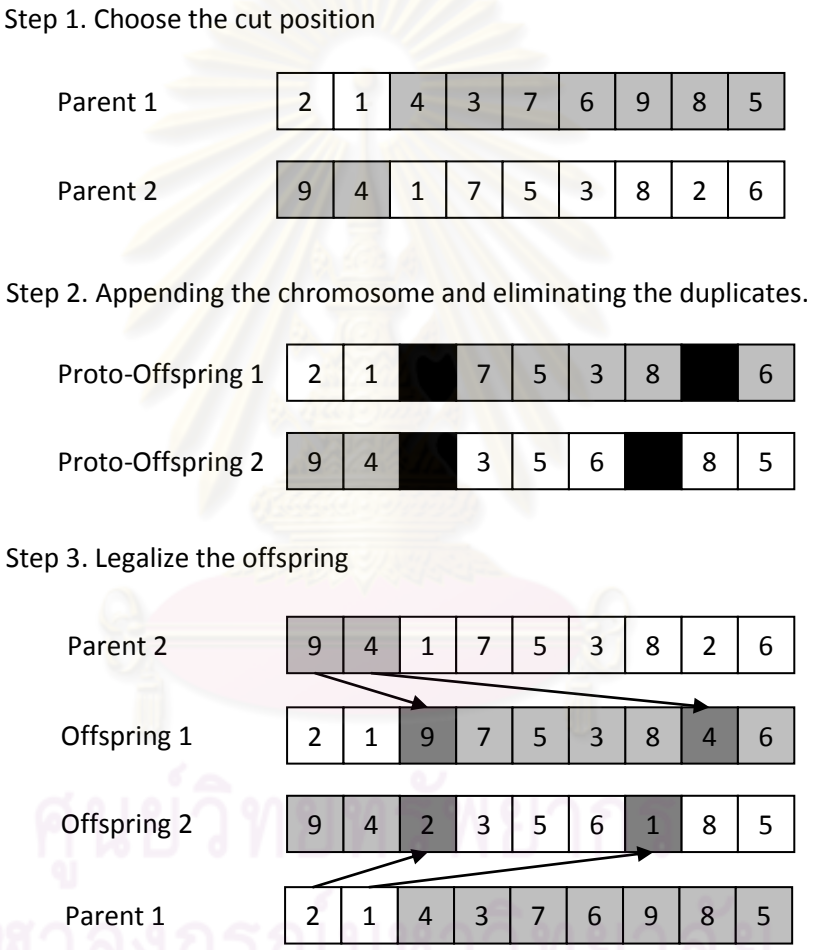


Figure 2.8 The modified crossover

2.3.1.1.4 Order crossover (OX) proposed by Oliver [47] and Goldberg [7]. This crossover operator extends the modified crossover of Davis by allowing two cut points to be randomly chosen on the parent chromosomes. In order to create an offspring, the string between the two cut points in the first parent is first copied to the offspring. Then, the remaining positions are filled by considering the sequence of items in the second parent, starting after the second cut point (when the end of the chromosome is reached, the sequence continues at position 1).

In figure 2.9, the substring 4-3-7 in parent 1 is first copied to the offspring. Then, the remaining positions are filled one by one after the second cut point, by considering the corresponding sequence of items in parent 2, namely 9-4-1-7-5-3-8-2-6. Hence, item 9 is first considered to occupy position 1, the item 4 is secondly considered to occupy the position 2 but it is discarded because it is already included in the offspring. Item 1 is the next item to be considered, and it is inserted at position 2. The procedure repeats as item 5 fills the position 6, item 3 is discarded, item 8 fills the position 7, and item 2 fills the position 8. Finally, the last item 5 fills the last sequence.

Clearly, OX tries to preserve the relative order of the items rather than their absolute position. In figure 2.9, the offspring2 does not preserve the position of most items in parent 1. The variant of OX, known as the *maximal preservative crossover* [MPX], is also described in [53].

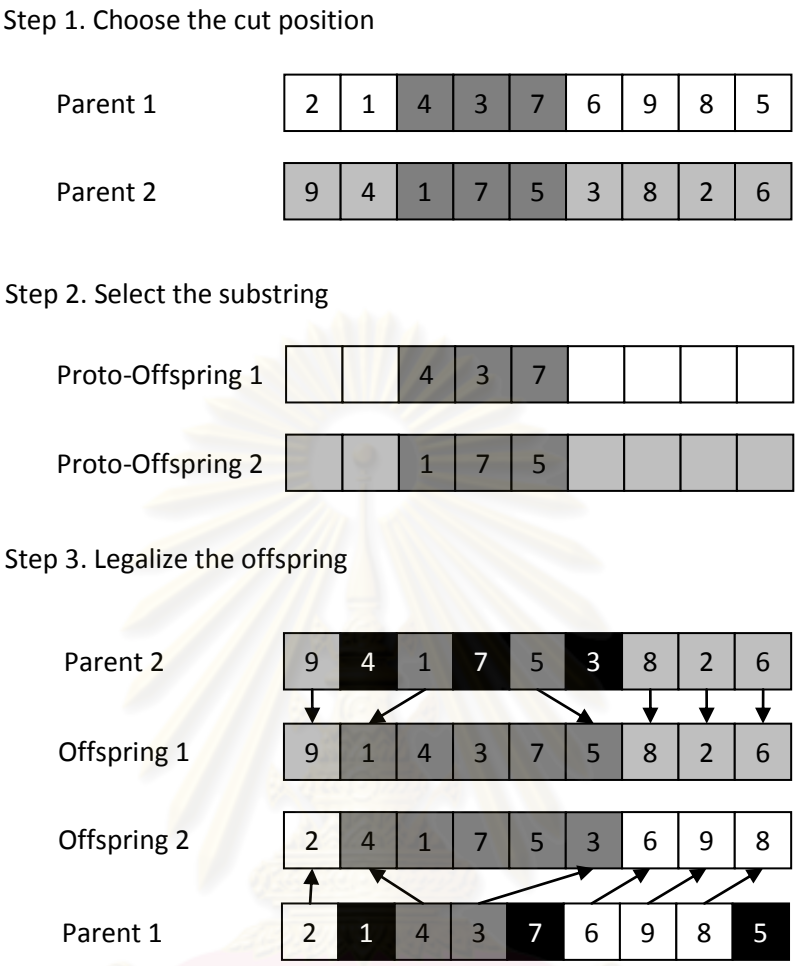


Figure 2.9 The order crossover

2.3.1.1.5 Order-based crossover (OBX) [54] was first introduced by Syswerda. This crossover also focuses on the relative order of the items on the parent chromosomes. First, a subset of items is selected in the first parent. In the offspring, these items appear in the same order as in the first parent, but at positions taken from the second parent. Then, the remaining positions are filled with the items of the second parent.

In figure 2.10, item 4, 7, 9 are first selected in parent 1, and must appear in this order in the offspring. Actually, these items occupy positions 1, 2 and 4 in parent 2. Hence, items 9, 4 and 7 occupy positions 1, 2 and 4, respectively, in the offspring. The remaining positions are filled with the items found in parent 1.

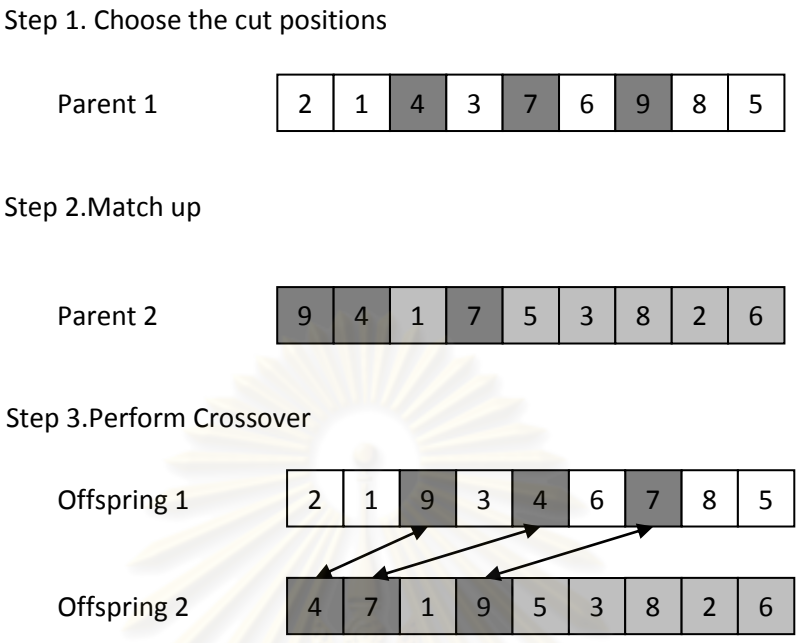


Figure 2.10 The order-based crossover

2.3.1.1.6 Position-based crossover (PBX) [54] was also invented be Syswerda. Here, a subset of positions is selected in the first parent. Then, the items found at these positions are copied to the offspring (at the same positions). The other positions are filled with the remaining items, in the same order as in the second parent.

The name of this operator is a little bit misleading, because it is the relative order of the items that is inherited from the parents (the absolute position of the items inherited from the second parent is rarely preserved). This operator can be seen as an extension of the order crossover OX, where the items inherited from the first parent do not necessarily occupy consecutive positions.

In figure 2.11, positions 3, 5 and 7 are first selected in parent 1. items 4, 7 and 9 are found at these positions, and occupy the same positions in the offspring. The other positions are filled one by one, starting at position 1, by inserting the remaining items according to their relative order in parent 2.

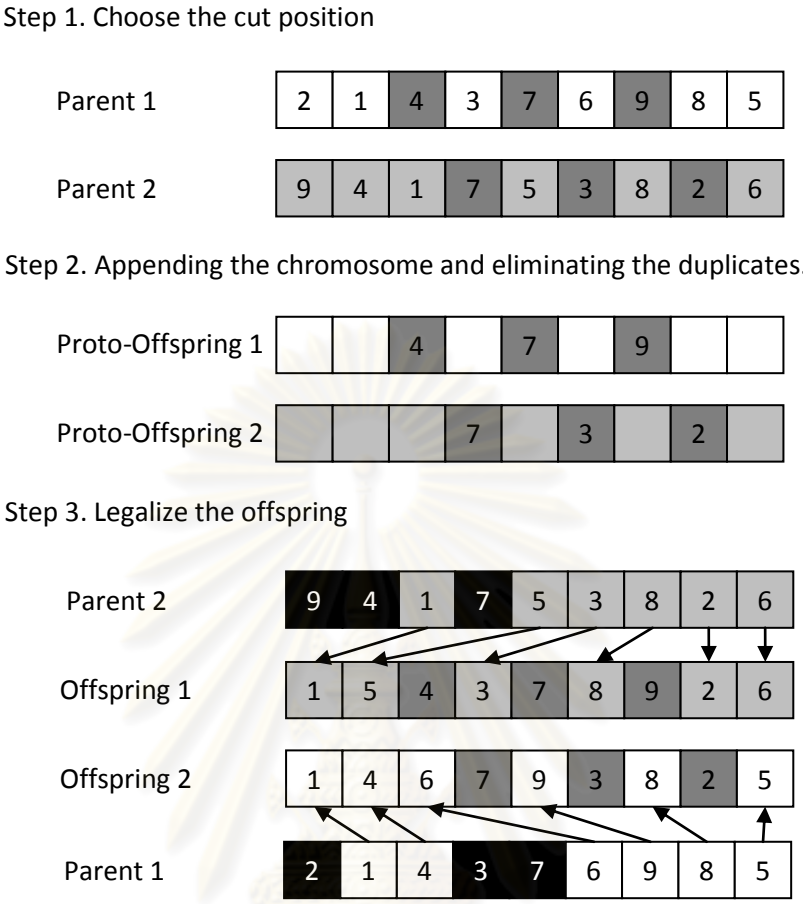


Figure 2.11 The position-based crossover

2.3.1.1.7 Weight mapping crossover (WMX) was recently proposed by Lee el al. [55][56] In many approaches, the mechanism of the crossover is not the same with that of the conventional one-cut point crossover. Some offspring may generate new chromosomes that are not possible to succeed the character of the parents, thereby retarding the process of evolution. For this reason weight mapping crossover is invented.

In figure 2.12, The weight mapping crossover begins with identifying the cut position, then map the items in the substring according to their weight. In this case, the weight of an item is equal to its value. For instance, an item 3 has a weight equal to 3 as well. Thus, a substring 6-9-8-5 from the parent 1 is sorted according to their weight and result is 5-6-8-9 then is map to the sorted substring 2-3-6-8 from parent 2. Then the substring 6-9-8-5 is rearranged according to the sequence of substring 3-8-2-6. The final step simply legalizes the offspring according to the sequence of the mapped weight.

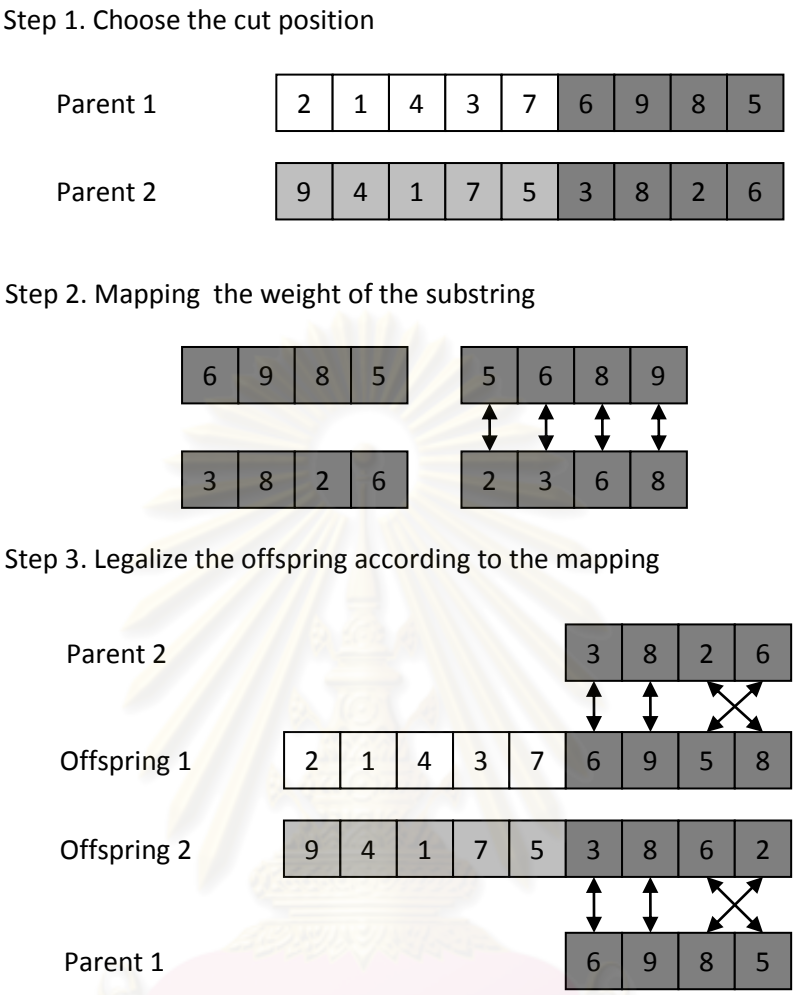


Figure 2.12 The weight-mapping crossover

2.3.1.1.7 Edge recombination (ER) was proposed by Whitley et al. [57] The adjacency representation is designed to facilitate the manipulation of edges. The crossover operators based on this representation generate offspring that inherit most of their edges from the parent chromosomes. The adjacency representation can be described as follows: node j occupies position i in the chromosome if there is an edge from item i to item j in the permutation string. This representation usually considers the relationship between edges as they are symmetrical. For instance an edge 3-2 is equivalent to 2-3. Various crossover operators are designed to manipulate this representation. These operators are aimed at transferring as many edges as possible from the parents to the offspring; however, the effective and most powerful one has shown to be only edge recombination.

The edge recombination operator reduces the myopic behavior of the alternate edge crossover [58] approach with a special data structure called *edge map*. The edge map maintains the list of edges that are incident to each node item of the parent and that lead to the nodes not yet included in the offspring. Hence, these edges are still available for extending the search and are said to be active. The strategy is to extend the search by selecting the edge that leads to the node item with the minimum number of active edges. In the case of equality between two or more item nodes, one of these nodes is selected at random. With this strategy, the approach is less likely to get trapped in a dead end.

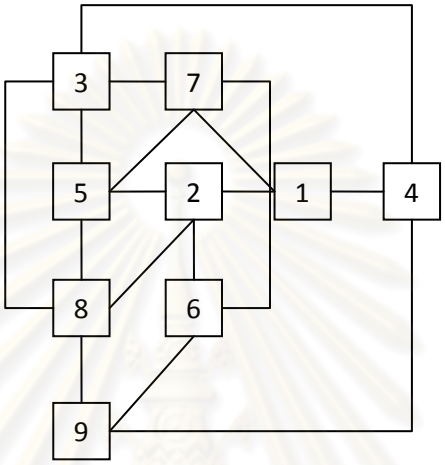
For the parent 2-1-4-3-7-6-9-8-5 and 9-4-1-7-5-3-8-2-6 (path representation), the initial edge map is shown in figure 2.13 Let us assume that node 1 is selected as the starting node. Accordingly, all edges incident to node 1 must first be deleted from the initial edge map. From node 1, we can go to nodes 2, 4 or 7. Each node has three active edges; hence, a random choice is made between nodes 2, 4 and 7. We assume that node 2 is selected. From 2, we can traverse to nodes 5, 6 and 8. Node 5 and 8 has up to three active edges while node 6 only has got two, so the latter is selected. From node 6, there are two choices 7 and 9 with the same amount of active edges, thus a random choice is made. The procedure repeats until there is no node left. From the figure 2.13 the final candidate is generated one node by one node start from node 1 and end up with 8. The final candidate is 1-2-6-9-4-3-7-5-8 which inherits edge 2-1, 6-9, 4-3, 3-7 and 8-5 from parent 1 while inherits edge 2-6, 9-4, 7-5 from parent 2.

More variant of edge recombination which focuses on edges common to both parents is described in [57]. Recently, a descendant of ER called Sequential Constructive Crossover (SCX) [58] proposed by Armed Z.H. integrated the edge weight matrix to the edge map in order to choose a better path.

.

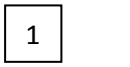
Step 1. Constructing an edge map

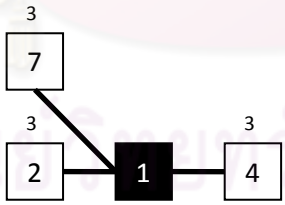
Parent 1	2	1	4	3	7	6	9	8	5
Parent 2	9	4	1	7	5	3	8	2	6



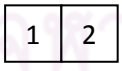
Step 2. Constructing a candidate solution

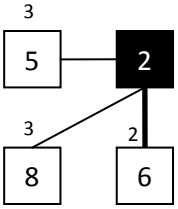
RandomSelect(1,2,3,4,5,6,7,8,9) = 1



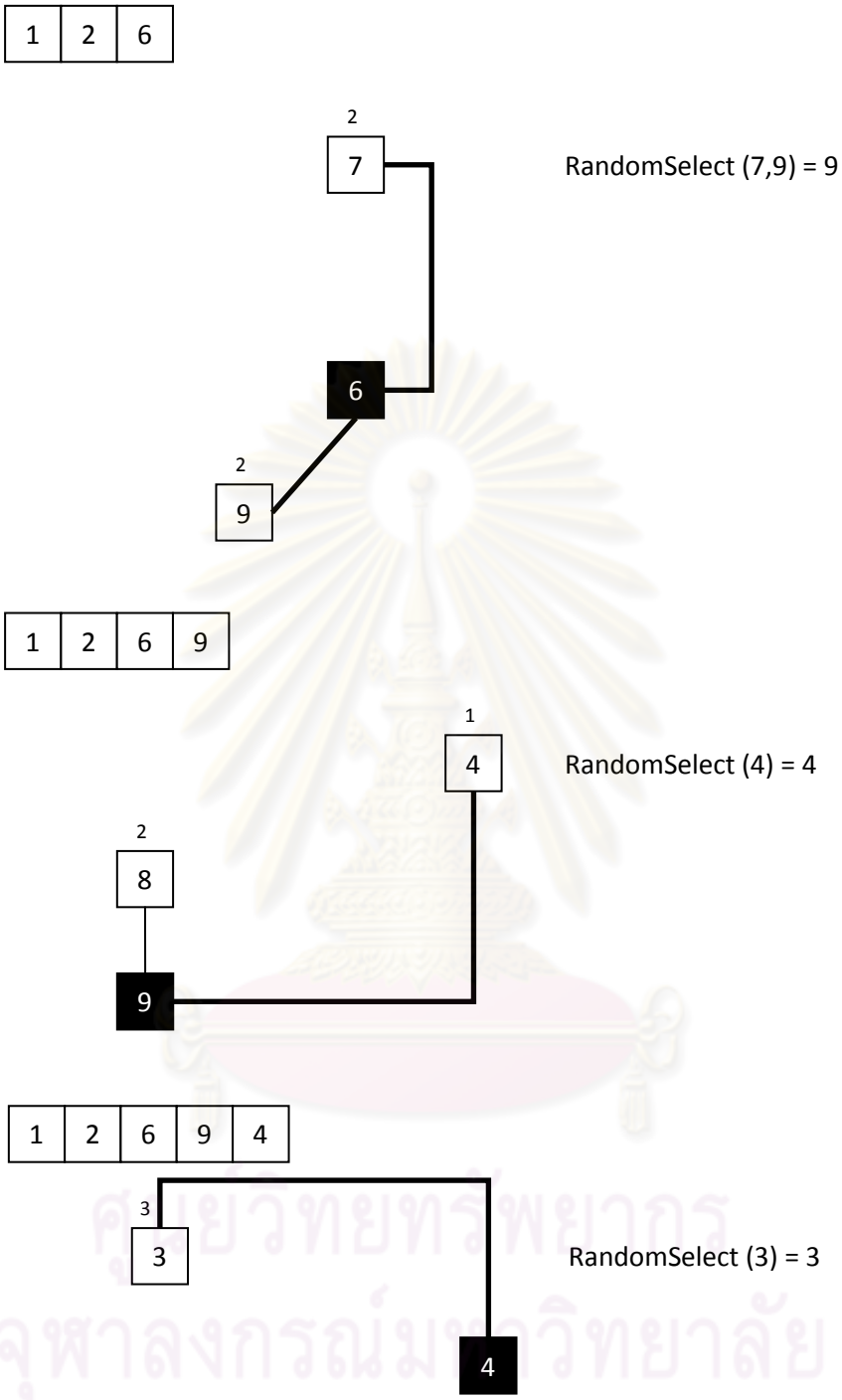


RandomSelect (2,4,7) = 2





RandomSelect (6) = 6



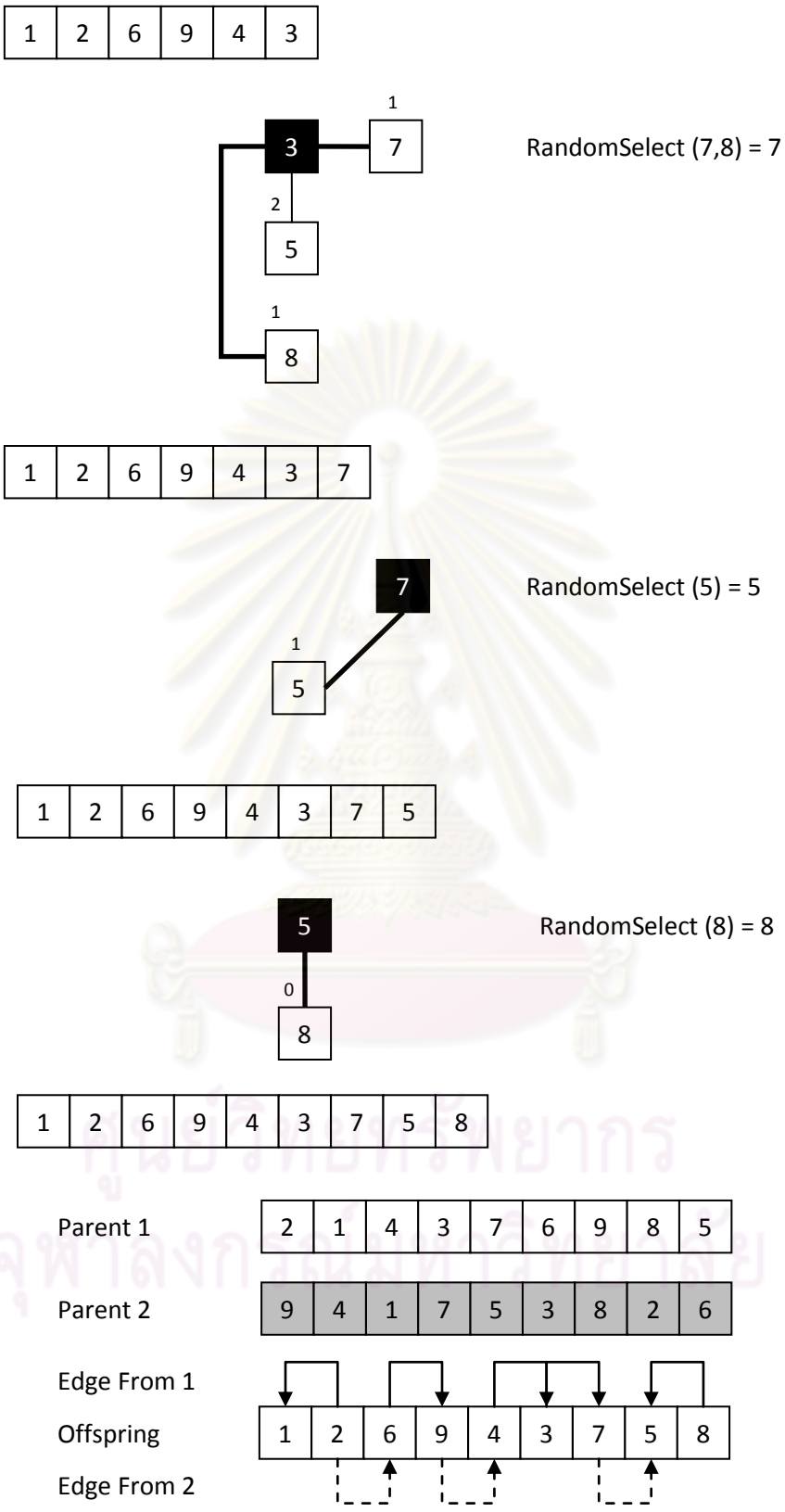


Figure 2.13 The edge recombination

2.3.1.2 Mutation operators

Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With these new gene values, the genetic algorithm may be able to arrive at better solution than was previously possible. However, mutation operators in permutation representation are also needed to be design such that the operators always generate the feasible solutions. Typically, the move operators of local search are adopted as the mutation operators for permutations.

2.3.1.3 Selection operators

Selection is a genetic operator that chooses a chromosome from the current generation's population for inclusion in the next generation's population. Usually, selection operators are not restricted to the representations. The three most commonly used selection methods are proportional (roulette wheel), tournament, and ranking. A proportional selection operator selects the population from the probabilities in which the chance of a chromosome getting selected is proportional to its fitness (or rank). This is where the concept of survival of the fittest comes into play. A tournament selection operator randomly divides the populations in to subsets, and then selects the best candidates among the member of such set. A ranking selection operator selects the top *N* percent of the population based on their rank. The variant of these operators can be found in [7].

2.3.2 Ant colony optimization

Ant colony optimization (ACO) is a population-based search inspired by the behavior of ants [59][60][61][62]. Ants are simple insects that live in colonies and show their amazing capabilities through their cooperative behavior like finding shortest paths from a food source to their colony. The ants exchange information via pheromones. The pheromones are chemical substances which the ants lay down in varying quantities to mark a path. While isolated ants move essentially at random, an ant encountering a previously laid pheromone trail can detect it and may follow the pheromone trail. The ants' probability to follow the pheromone trail depends on the pheromone intensity. The higher the pheromone intensity indicates the larger the possibility to follow. At the same time the ants following the pheromone trail may lay down additional pheromone and a positive feedback loop results. The more ants previously have chosen the pheromone trail, the more ants will follow it in the future. One of the basic ideas of ant colony optimization is to use an algorithmic counterpart of the pheromone trail as a medium for cooperation and communication among a colony of artificial ants which is guided by *positive feedback*.

The most important part in ACO algorithms, in general, is how the pheromone trails are used to generate better solutions in future iterations of the algorithm. The idea is to combine the solution components that in previous iterations have shown to be part of good solutions, even better solutions may be generated. Thus, ACO algorithms can be seen as adaptive sampling algorithms – adaptive in the sense that they consider past experience to influence future iterations.

```
PROCEDURE AntColonyOptimization

1. Initialize PheromoneTrails, calculate HeuristicInformation

2. Repeat

3.   p <- ConstructSolutions(PheromoneTrails,
HeuristicInformation)

4.   GlobalUpdateTrails(p)

5. Until Termination Condition is met
```

Algorithm 2.7: Ant Colony Optimization

We give an algorithmic skeleton into which fit the ACO algorithm applications for static combinatorial optimization problems. For an outline of the more general *ACO metaheuristics* we refer to [65]. In the main loop of the algorithm, first solutions are generated for all ants of the colony (the colony is indicated by p) by a function *ConstructSolutions*. The solution construction typically uses the pheromone information and problem specific local heuristic information. The solutions are then improved by a local search phase (*LocalSearch*). This local search phase is optional; in fact, it is not used in all applications of ACO algorithms to combinatorial optimization problems. Finally, the solutions are used to update the pheromone trails in a function *GlobalUpdateTrails*.

2.3.3 Particle swarm optimization

Particle swarm optimization (PSO) is a population-based metaheuristic inspired from swarm intelligence [63]. It mimics the social behavior of natural organisms such as bird flocking and fish schooling to find a place with enough food. Indeed, in those swarms, a coordinated behavior using local movements emerges without any central control. Originally, PSO has been successfully designed for continuous optimization problems. Its first application to optimization problems has been proposed in Ref. [64].

In the basic model, a swarm consists of N particles flying around in a D-dimensional search space. Each particle i is a candidate solution to the problem, and is represented by the vector x_i in the decision space. A particle has its own position and velocity, which means the flying direction and step of the particle. Optimization takes advantage of the cooperation between the particles. The success of some particles will influence the behavior of their peers. Each particle successively adjusts its position x_i toward the global optimum according to the following two factors: the best position visited by itself ($pbest_i$) denoted as $p_i = (p_{i1}, p_{i2}, \dots, p_{iD})$ and the best position visited by the whole swarm ($gbest$) (or $lbest$, the best position for a given subset of the swarm) denoted as p_g The vector ($p_g - x_i$) represents the difference between the current position of the particle i and the best position of its neighborhood.

```
PROCEDURE ParticleSwarmOptimization

1. Initialize p[N]

2. Repeat

3.   Evaluate f(p[N])

4.   UpdateVelocities(p[N])

5.   UpdatePosition(p[N])

6.   UpdateBestFoundPaticle(pBest[N],gbest)

5. Until Termination Condition is met
```

Algorithm 2.8: Particle Swarm Optimization

Update the velocity: The velocity v_i that defines the amount of change that will be applied to the particle is defined as

$$v_i(t) = v_i(t-1) + \rho_i C_1 \times \left(p_i - x_i(t-1)\right) + \rho_2 C_2 \times \left(p_g - x_i(t-1)\right) \quad (2.5)$$

where ρ_1 and ρ_2 are two random variables in the range $[0, 1]$. The constants C_1 and C_2 represent the learning factors. They represent the attraction that a particle has either toward its own success or toward the success of its neighbors. The parameter C_1 is the cognitive learning factor that represents the attraction that a particle has toward its own success. The parameter C_2 is the social learning factor that represents the attraction that a particle has toward the success of its neighbors. The velocity defines the direction and the distance the particle should go

Update the position: Each particle will update its coordinates in the decision space.

$$x_i(t) = x_i(t-1) + v_i(t) \quad (2.6)$$

Then it moves to the new position.

Update the best found particles: Each particle will update (potentially) the best local solution:

$$\text{if } f(x_i) < pbest_i, \text{ then } p_i = x_i \quad (2.7)$$

Moreover, the best global solution of the swarm is updated:

$$\text{if } f(x_i) < gbest_i, \text{ then } g_i = x_i \tag{2.8}$$

Hence, at each iteration, each particle will change its position according to its own experience and that of neighboring particles. As for any swarm intelligence concept, agents (particles for PSO) are exchanging information to share experiences about the search carried out. The behavior of the whole system emerges from the interaction of those simple agents. In PSO, the shared information is composed of the best global solution *gbest*.

Traditionally, PSO algorithms are applied to continuous optimization problems. Some adaptations must be made for discrete optimization problems. They differ from continuous models in mapping between particle positions and discrete solutions: Many discrete representations such as binary encodings [65] and permutations can be used for a particle position. The velocity models may be real valued, stochastic, or based on a list of moves. In stochastic velocity models for permutation encodings, the velocity is associated with the probability for each item to be generated in a position. Further information can be found in [66][67]. Velocity models for discrete optimization problems have been generally inspired from mutation and crossover operators of EAs.

2.3.4 Estimation of distribution algorithms

Estimation of distribution algorithms (EDA) are a recent class of optimization techniques based on the concept of using probability distribution of the population in reproducing new offsprings [68][69]. EDAs construct a probability distribution of desired population and then create new individuals by sampling from this probability distribution. This class of algorithms is classified as non-Darwinian evolutionary algorithms as they replace Darwinian operators with probability distributions. The first algorithms belonging to this class have been proposed in Refs. [70].

The principal idea in EDAs is to transform the optimization problem into a search over probability distributions. They maintain a population of individuals. A probabilistic model for promising individuals is constructed. For instance, EDA estimates the probability distribution of each decision variable of the optimization problem. The probabilistic model represents an explicit model of promising regions of the search space. The induced probabilistic model will be used to generate new solutions. The generated individuals will replace the old population in a full or a partial manner. This process iterates until a given stopping criteria. The general EDA can be sketched as follows (Algorithm 2.9):

```
PROCEDURE EDA

1. Initialize ProbabilisticModel

2. Repeat

3.   p <- Sampling(ProbabilisticModel)

4.   p <- Selection(p,p')

5.   Update(ProbabilisticModel,p)

6. Until Termination Condition is met
```

Algorithm 2.9: Estimation of Distribution Algorithm

Various algorithms in EDA class have been proposed since 1994, the famous EDAs including PBIL (*population based incremental learning*)[71], CGA (*compact genetic algorithm*)[72], UMDA (*univariate marginal distribution algorithm*)[73], MIMIC (*mutual information maximizing input clustering*)[74], and BOA (*Bayesian optimization algorithm*)[75]. These algorithms differ from each other in encoding, probability models and the methods to update the models. Further information of these algorithms can be found in Refs.[68][69]

The results obtained from the EDA family of algorithms are not yet competitive compared to more traditional metaheuristics especially in combinatorial problems. The simple explanation is that all the mentioned EDAs use inappropriate encoding to solve permutation problems. Recently, two EDAs that naturally represent the permutation in the genotype have emerged. They are called EHBSA (*edge histogram based sampling algorithm*) [18] and NHBSA (*node histogram based sampling algorithms*) [19].

2.3.4.1 Edge Histogram Based Sampling Algorithm

Edge Histogram Based Sampling Algorithm (EHBSA) [18] was proposed by Tsutsui in 2002. EHBSA was designed to solve combinatorial problems and has shown the competitive performances in solving many real world applications including traveling salesman problems (TSP), flow shop scheduling problems (FSSP) and capacitated vehicle routing problems (CVRP).

In permutation scheme, the models of solutions can be represented as a graph of nodes connected by edges. EHBSA utilizes Edge Histogram Matrix (EHM) to learn the mutual information of edges contained in the selected solutions and then construct new solutions by sampling from it. The idea of EHBSA is to use the edge recombination (ER) [57] in genetic algorithms with the whole selected population instead of tradition two-parent recombination.

Constructing Edge Histogram Matrix: Edge histogram matrix is a matrix that simply store the summation of edge counted from the selected population plus a bias. Let string of kth individual in population $P(t)$ at generation t represent as $s_k^t = (\pi_k^t(0), \pi_k^t(1), \dots, \pi_k^t(L-1))$. $\pi_k^t(0)$, $\pi_k^t(1), \dots,$ and $\pi_k^t(L-1)$ are the permutation of $(0,1, \dots, L-1)$ where L is the length of the permutation. Edge histogram matrix $EHM^t\ (e_{i,j}^t)(i,j = 0,1, \dots, L-1)$ of population $P(t)$ is symmetrical and consists of L^2 items as follows:

$$e^t{}_{i,j} = \begin{cases} \sum_{k=1}^{N} (\delta_{i,j}(s_k^t) + \varepsilon \text{ if } i \neq j \\ 0 \text{ if } i = j \end{cases} \tag{2.9}$$

where N is the population size, $\delta_{i,j}(s_k^t)$ is a delta function defined as

$$\delta_{i,j}(s_k^t) = \begin{cases} 1 \text{ if } \exists h[h \in \{0,1, \dots L-1\} \wedge \\ \pi_k^t(h) = i \wedge \pi_k^t\big((h+1) \bmod L\big) = j] \\ 0 \text{ otherwise} \end{cases} \tag{2.10}$$

and ε $(\varepsilon > 0)$ is a bias to control pressure in sampling nodes just like those used for adjusting the selection pressure in the proportional selection in GAs. The average number of edges of item $(e_{i,j}^t)$ in $EHM^t$is $2LN/(L^2 - L) = 2N/(L-1)$. So, ε is determined by a bias ratio B_{ratio} $(B_{ratio} > 0)$ of this average number of edges as

$$\varepsilon = \frac{2N}{L-1} B_{ratio} \tag{2.11}$$

A smaller of value of B_{ratio} reflects the real distribution of edges in sampling of nodes and a bigger value of B_{ratio} will give a kind of perturbation in the sampling. An example of EHM^t is shown in figure 2.14.

$s_1^t = (0,1,2,3,4)$
$s_2^t = (1,3,4,2,0)$
$s_3^t = (3,4,2,1,0)$
$s_4^t = (4,0,3,1,2)$
$s_5^t = (2,1,3,4,0)$

node j

node i

$$\begin{bmatrix} 0 & 1.1 & 0.1 & 1.1 & 0.1 \\ 1.1 & 0 & 2.1 & 2.1 & 0.1 \\ 1.1 & 2.1 & 0 & 1.1 & 0.1 \\ 0.1 & 1.1 & 0.1 & 0 & 4.1 \\ 2.1 & 0.1 & 2.1 & 0.1 & 0 \end{bmatrix}$$

EHM^t

Figure 2.14 An example of asymmetric edge histogram matrix for $\boldsymbol{N = 5, L = 5, B_{ratio} = 5}$.

Sampling from Edge Histogram Matrix: The sampling algorithm of EHBSA is similar to Ant Colony Optimization[20]. A new individual permutation $\boldsymbol{c}[]$ is generated straightforwardly as follows:

```
PROCEDURE SamplingfromEHM

1. Set position counter p ← 0

2. Obtain first node c[0] randomly from [0, L − 1]

3. Construct a roulette wheel vector rw[] from matrix as
rw[j] ← e^t_{c[p],j} (j = 0,1, … , L − 1)

4. Set to 0 previously sampled nodes in rw[] (rw[c[i]] ← 0 for i =
0, … , p)

5. Sample next node c[p + 1] with probability rw[x]/∑_{j=0}^{L−1} rw[j] using
roulette wheel rw[].

6. Update the position counter p ← p + 1.

7. If  p < L − 1, go to Step 3.
```

Algorithm 2.10: Sampling Algorithm of EHBSA

In this review, we exemplify only the simple version of EHBSA. However, there are variations of EHBSA such as hybridization with absolute order base crossover obtained from genetic algorithm to improve the quality of the results. Further information can be found in Ref. [18][19]

2.3.4.2 Node Histogram Based Sampling Algorithm

Node Histogram Based Sampling Algorithm [19] (NHBSA) was also proposed by Tsutsui. NHBSA was also designed to solve combinatorial problems and have shown the competitive performances in many more combinatorial problems. However, NHBSA differs from EHBSA as NHBSA is more suitable to the problems where fitness's depend on the absolute order of item. NHBSA utilize a Node Histogram Matrix (NHM) to construct a solution.

Constructing Node Histogram Matrix: Node histogram matrix is a matrix that simply store the summation of node counted from the selected population plus a bias. Let string of kth individual in population $P(t)$ at generation t represent as $s_k^t = (\pi_k^t(0), \pi_k^t(1), \dots, \pi_k^t(L-1))$. $\pi_k^t(0)$, $\pi_k^t(1), \dots,$ and $\pi_k^t(L-1)$ are the permutation of $(0,1, \dots, L-1)$ where L is the length of the permutation. Node histogram matrix NHM^t $(n_{i,j}^t)(i,j = 0,1, \dots, L-1)$ of population $P(t)$ consists of L^2 items as follows:

$$n^t{}_{i,j} = \sum_{k=1}^{N} (\delta_{i,j}(s_k^t) + \varepsilon \tag{2.12}$$

where N is the population size, $\delta_{i,j}(s_k^t)$ is a delta function defined as

$$\delta_{i,j}(s_k^t) = \begin{cases} 1 \text{ if } \pi_k^t(i) = j \\ 0 \text{ otherwise} \end{cases} \tag{2.13}$$

and ε $(\varepsilon > 0)$ is a bias to control pressure in sampling nodes just like those used for adjusting the selection pressure in the proportional selection in GAs. The average number of edges of item $(n_{i,j}^t)$ in NHM^t is $LN/(L^2) = N/L$. So, ε is determined by a bias ratio B_{ratio} $(B_{ratio} > 0)$ of this average number of nodes as

$$\varepsilon = \frac{N}{L} B_{ratio} \tag{2.14}$$

A smaller of value of B_{ratio} reflects the real distribution of nodes in sampling of positions and a bigger value of B_{ratio} will give a kind of perturbation in the sampling. An example of NHM^t is shown in figure 2.15.

$$
\begin{array}{ll}
s_1^t = (0,1,2,3,4) & \\
s_2^t = (1,3,4,2,0) & \\
s_3^t = (3,4,2,1,0) & \\
s_4^t = (4,0,3,1,2) & \\
s_5^t = (2,1,3,4,0) &
\end{array}
\qquad
\begin{array}{c}
\text{position } j \\
\text{node } i \begin{bmatrix}
1.1 & 1.1 & 0.1 & 0.1 & 3.1 \\
1.1 & 2.1 & 0.1 & 2.1 & 0.1 \\
1.1 & 0.1 & 2.1 & 1.1 & 1.1 \\
1.1 & 1.1 & 2.1 & 1.1 & 0.1 \\
1.1 & 1.1 & 1.1 & 1.1 & 1.1
\end{bmatrix} \\
NHM^t
\end{array}
$$

Figure 2.15 An example of node histogram matrix for $\boldsymbol{N = 5, L = 5, B_{ratio} = 5}$.

Sampling from Node Histogram Matrix: Although the $\boldsymbol{NHM^t}$ is simpler to construct than the $\boldsymbol{EHM^t}$, the sampling algorithm of NHBSA is a little bit more complicate. In EHBSA, each node is constructed one position by one position in a sequence. However, in NHBSA, each node is constructed with a random position sequence. A new individual permutation $\boldsymbol{c}[]$ is generated straightforwardly as follows:

PROCEDURE SamplingfromNHM

1. Set counter $p \leftarrow 0$

2. Construct a roulette wheel of all positions.

3. Set to 0 previously sampled positions

4. Sample a position j from the roulette wheel of position

5. Construct a roulette wheel vector $rw[]$ from matrix as $rw[i] \leftarrow n^t{}_{c[p],i} (i = 0,1, \dots, L-1)$

6. Set to 0 previously sampled nodes in $rw[]$ $(rw[c[i]] \leftarrow 0$ for $i = 0, \dots, p)$

7. Sample next node $c[p+1]$ with probability $rw[x] / \sum_{j=0}^{L-1} rw[j]$ using roulette wheel $rw[]$.

6. Update the counter $p \leftarrow p+1$.

7. If $p < L$, go to Step 2.

Algorithm 2.11: Sampling Algorithm of NHBSA

2.3.5 Scatter Search and Path Relinking

Scatter search has its origin in the paper of F. Glover [76]. SS is a deterministic strategy that has been applied successfully to some combinatorial and continuous optimization problems. Even if the principles of the method have been defined since 1977, the application of SS is in its beginning.

SS is a evolutionary and population metaheuristic that recombines solutions selected from a reference set to build others [77]. The method starts by generating an initial population satisfying the criteria of diversity and quality. The reference set (*RefSet*) of moderate size is then constructed by selecting good representative solutions from the population. The selected solutions are combined to provide starting solutions to an improvement procedure based on a S-metaheuristic. According to the result of such procedure, the reference set and even the population of solutions are updated to incorporate both high-quality and diversified solutions. The process is iterated until a stopping criterion is satisfied.

The SS approach involves different procedures allowing to generate the initial population, to build and update the reference set, to combine the solutions of such set, to improve the constructed solutions, and so on. SS uses explicitly strategies for both search intensification and search diversification. It integrates search components from P-metaheuristics and S-metaheuristics. The algorithm starts with a set of diverse solutions, which represents the reference set (initial population) This set of solutions is evolved by means of recombination of solutions as well as by the application of local search (or another S-metaheuristic).

```
PROCEDURE ScatterSearch

1. Generate RefSet

2. Construct a roulette wheel of all positions.

3. Repeat

4.    SolutionRecombinationMethod(RefSet)

5. Until a termination condition is met
```

Algorithm 2.11: Scatter Search

Search components of scatter search algorithms: The search component of SS including diversification generation method, improvement method, reference set update method, subset generation method and solution combination method.

The diversification generation method generates a set of diverse initial solutions. In general, greedy procedures are applied to diversify the search while selecting high-quality solutions. The improvement method transforms a trial solution into one or more enhanced trial solutions using any S-metaheuristic. In general, a local search algorithm is applied and then a local optimum is generated. In reference set update method, a reference set is constructed and maintained. The objective is to ensure diversity while keeping high-quality solutions. The subset generation method operates on the reference set, to produce a subset of solutions as a basis for creating combined solutions. This procedure is similar to the selection mechanism in EAs. However, in SS, it is a deterministic operator, whereas in GAs, it is generally a stochastic operator. Finally, the solution combination method recombined the subset of solutions produced by the subset generation method. In general, weighted structured combinations are used via linear combinations and generalized rounding to discrete variables. This operator may be viewed as a generalization of the crossover operator in GAs where more than two individuals are recombined.

Path relinking method: [78] [79] The path relinking method is simply a solution combination method. However, the main idea of path relinking is to generate and to explore the trajectory in the search space connecting a starting solution s and a target solution t. The idea is to reinterpret the linear combinations of points in the Euclidean space as paths between and beyond solutions in a neighborhood space. The path between two solutions in the search space (neighborhood space) will generally yield solutions that share common attributes with the input solutions. A sequence of neighboring solutions in the decision space is generated from the starting solution to the target solution. The best found solution in the sequence is returned. The path relinking method becomes popular in finding the set of multi-objective solution in the Pareto frontier. The variants of this method can be found in the Ref.[80][81]

2.4 Multi-Objective Combinatorial Optimization

As far as real world decision making is concerned, it is also well known, that decision makers have to deal with more than one objective. The growth in the interest of theory and methodology of multi-criteria decision making (MCDM) over the last thirty years [82] [83][84][85] is well aware.

Definition 2.3: **Multi-objective Combinatorial Optimization [82]**

$$MOP = \min_{x \epsilon S} F(x) = (f_1(x), f_2(x), \dots, f_n(x)) \qquad (2.15)$$

where n $(n \geq 2)$ is the number of objectives, $x = (x_1, \dots, x_k)$ is a feasible solution belong to the discrete solution set S defined in the definition 1.1. $F(x) = (f_1(x), f_2(x), \dots, f_n(x))$ is the vector of objectives to be optimized.

Surprisingly, multi-criteria or multi-objective combinatorial optimization (MOCO) has not been widely studied. There is a lack of "standard" benchmarks even if recently there is an interest in providing test instances for classical combinatorial MOPs.

In this dissertation, we preferred to review the multi-objective technique in a form of addition to the common concepts of single-objective metaheuristics, since the multi-objective techniques that applied to combinatorial problems are rarely studied. A unified view of multi-objective metaheuristics [24] is presented in an attempt to provide a common terminology and classification mechanisms. The goal of the general classification is to provide a mechanism that allows a common description and comparison of multi-objective metaheuristics in a qualitative way. It also allows the design of new multi-objective metaheuristics, borrowing ideas from current ones. The multi-objective metaheuristics contains three main search components:

Fitness assignment: The main role of this procedure is to guide the search algorithm toward Pareto optimal solutions for a better convergence. It assigns a scalar-valued fitness to a vector objective function.

Diversity preserving: The emphasis here is to generate a diverse set of Pareto solutions in the objective and/or the decision space.

Elitism: The preservation and use of elite solutions (e.g., Pareto optimal solutions) allows a robust, fast, and a monotonically improving performance of a metaheuristic.

The following sections discuss how these three search components can be defined independently to design a multi-objective metaheuristic.

2.4.1 Fitness assignment strategies

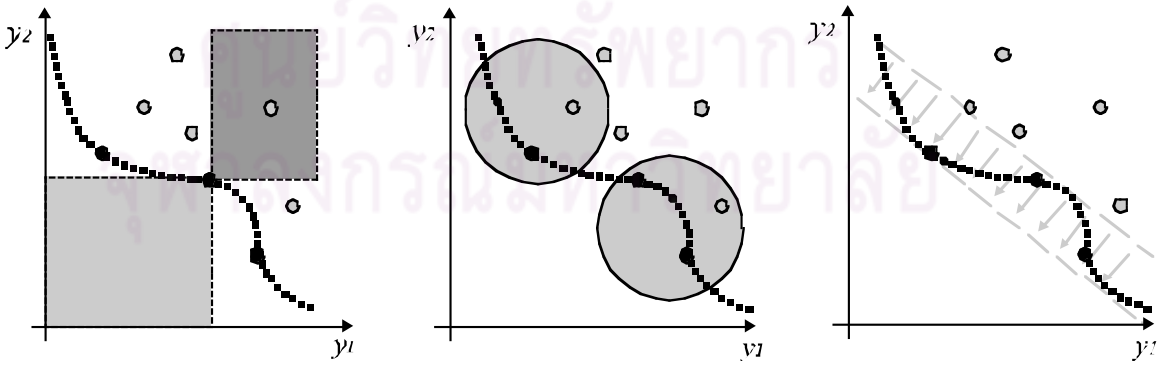
For a given solution, a fitness assignment procedure maps a fitness vector to a single value. The fitness scalar value measures the quality of the solution. According to the fitness assignment strategy, multi-objective metaheuristics can be classified into four main categories including scalar approaches, criterion-based approach, dominance-based approach and indicator-based approsed. Figure 2.16 show overviews of dominance-based, criterion-based and scalar approaches.

Scalar approaches: They are based on the MOP problem transformation into a single objective problem. This class of approaches based on aggregation that combine the various objectives f_i into a single objective function$F$. These approaches require for the decision maker to have a good knowledge of his problem.

Criterion-based approaches: In criterion-based approaches, the search is performed by treating the various noncommensurable objectives separately.

Dominance-based approaches: The dominance-based approaches[12] use the concept of dominance and Pareto optimality to guide the search process. The objective vectors of solutions are scalarized using the dominance relation.

Indicator-based approaches: In indicator-based approaches, the metaheuristics use performance quality indicators to drive the search toward the Pareto front.



(a) Dominance-based (b) Criterion-based (c) Scalar

Figure 2.16 Fitness assignment strategies

2.4.1.1 Scalar approaches

This class of multi-objective metaheuristics contains the approaches that transform a MOP problem into a single objective. Among these methods one can find the aggregation methods, the weighted metrics, the goal programming methods, the achievement functions, the goal attainment methods, and the ε-constraint methods. The use of scalarization approaches is justified when they generate Pareto optimal solutions.

2.4.1.1.1 Aggregation method The aggregation (or weighted) method is one of the first and most used methods for the generation of Pareto optimal solutions. It consists in using an aggregation function to transform a MOP into a single objective problem by combining the various objective functions f_i into a single objective function F in a linear way [86][87]:

$$F(x) = \sum_{i=1}^{n} \lambda_i f_i(x), \quad x \in S \tag{2.16}$$

where the weights $\lambda_i \in [0 \dots 1]$and $\sum_{i=1}^{n} \lambda_i = 1$. The solution of the weighted problem is weakly Pareto optimal.

2.4.1.1.2 Weighted metrics In this approach[88], the decision maker must define the reference point z to attain. Then, a distance metric between the referenced point and the feasible region of the objective space is minimized. The aspiration levels of the reference point are introduced into the formulation of the problem, transforming it into a single objective problem.

2.4.1.1.3 Goal programming It is one of the oldest and most popular methods dealing with MOPs [89]. The decision maker defines aspiration levels $\bar{z}_i$for each objective f_i, which define the goals $f_i(x) \leq \bar{z}_i$.Then, the deviations δ_i associated with the objective functions f_i are minimized.

2.4.1.1.4 Achievement functions Achievement functions have been introduced by Wierzbicki [90]. Unlike the previous methods where the reference point must be chosen carefully (e.g., ideal point), an arbitrary reference point $\bar{z}$ can be

selected in the objective space. A Pareto optimal solution is produced for each location of the reference point. Different Pareto optimal solutions may be generated by using various reference points. Without using the augmentation factor, weakly Pareto optimal solutions are generated.

2.4.1.1.5 Goal attainment [91] The goal attainment method constitutes another approach that is based on the preference specification of the intermediary of a goal to reach. In this approach, a weight vector and the goals for all the objectives have to be chosen by the decision maker.

The major drawback of this method is the possible absence of the selection pressure of the generated solutions. For instance, if there are two solutions that have the same value for one objective and different values for the other objective, they have the same fitness. The search algorithm cannot differentiate them in the problem resolution.

2.4.1.1.6 ϵ-Constraint method [92] In the popular ϵ-constraint, the problem consists in optimizing one selected objective f_k subject to constraints on the other objectives $f_j, j \in [1,n], j \neq k$ of a MOP. Hence, some objectives are transformed into constraints. The new considered problem may be formulated as follows:

$$(MOP_k(\epsilon)) \begin{cases} \min f_k(x) \\ x \in S \\ s.c.\,(f_j(x) \leq \epsilon_j, j = 1, \dots, n, j \neq k \end{cases} \tag{2.17}$$

Where the vector $\epsilon = (\epsilon_1, \dots \epsilon_n)$ represents an upper bound for the objectives. Thus, A single objective problem subject to constraints on the other objectives is solved.

2.4.1.2 Criterion-based approaches In this class that is mainly based on P-metaheuristics, the search is carried out by treating the various objectives separately. Few studies belonging to this class exist in the literature. Among them one can find the parallel selection in evolutionary algorithms, parallel pheromone update in ant colony optimization, and lexicographic optimization.

2.4.1.2.1 Parallel Approach [93][94] In this approach, the objectives are handled in parallel. P-metaheuristics may be transformed easily to parallel criterion-based multi-objective optimization algorithms. Indeed, the generation of new solutions will be carried out independently according to the objectives.

2.4.1.2.2 Sequential or Lexicographic approach [95] In this approach, the search is carried out according to given preference order of the objective defined by the decision maker. This order defines the significance level of the objectives. Let us suppose that the objective indices of the functions also indicate their priority; the function f_1 has the greatest priority. Then, a set of single objective problems are solved in a sequential manner.

If the problem associated with the most significant objective function has a unique solution, the search provides the optimal solution found and stops. Otherwise, the problem associated with the second most significant objective function is solved, including the constraint that the most significant objective function preserves its optimal value (i.e., an equality constraint is associated with the already optimized functions).

The solution obtained with lexicographic ordering of the objective is Pareto optimal. A relaxation may be applied to the constraint regarding the previous objective functions. For instance, a small decrease in the performance of the most significant objective functions may be allowed to obtain trade-off solutions.

2.4.1.3 Domainance-based approaches

The dominance-based approaches use the concept of dominance in the fitness assignment, contrary to the other approaches that use a scalarization function or treat the various objectives separately. This idea was introduced initially by Goldberg [7]. The main advantage of dominance-based approaches is that they do not need the transformation of the MOP into a single objective one. In a single run, they are able to generate a diverse set of Pareto optimal solutions and Pareto solutions in the concave portions of the convex hull of feasible objective space.

P-metaheuristics seem particularly suitable to solve MOPs, because they deal simultaneously with a set of solutions that allow to find several members of the Pareto optimal set in a single run of the algorithm. Most of Pareto approaches use evolutionary multi-objective algorithms (EMOs). One can mention the commonly used ones: NSGA-II (nondominated sorting genetic algorithm) [96] , and SPEA2 (strength Pareto evolutionary algorithm) [97]. Many other competent EMOs have been developed, such as MOMGA (multi-objective messy GA) [98], MOMGA-II [99][100] and neighborhood constraint GA [515]. Moreover, Pareto P-metaheuristics are less sensitive to the shape of the Pareto front (continuity, convexity).

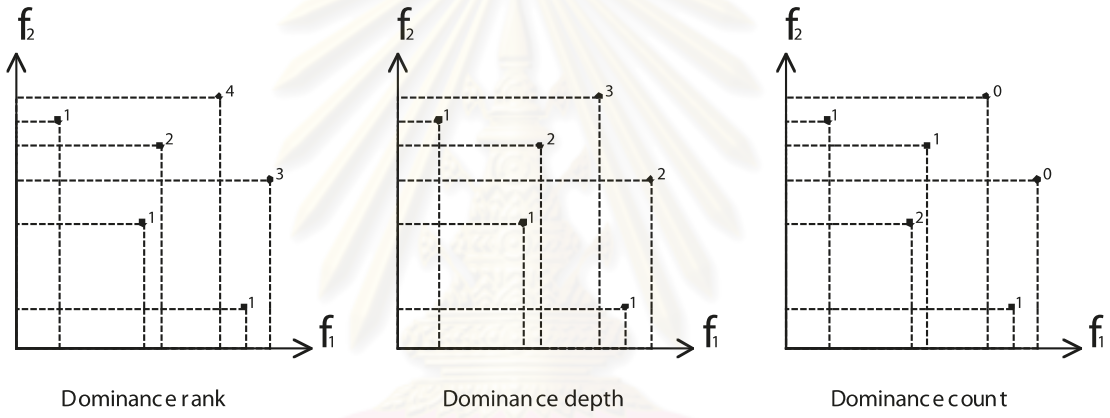


Figure 2.17 Fitness assignment: some dominance-based ranking methods.

Our concern here is to design a fitness assignment procedure to guide the search toward the Pareto border. Ranking methods are usually applied to establish an order between the solutions. This order depends on the concept of dominance and thus directly on Pareto optimality. Most of these fitness assignment procedures have been proposed in the EMO community. The most popular dominance-based ranking procedures are as follows (see Figure. 3.17) [102]:

2.4.1.3.1 Dominance rank In this strategy, the rank associated with a solution is related to the number of solutions in the population that dominates the considered solution [103]. This strategy was first employed in the MOGA algorithm (multi-objective genetic algorithm) [103]. In the MOEA algorithm, the fitness of a solution is equal to the number of solutions of the population that dominate the considered solution, plus one.

2.4.1.3.2 Dominance depth The population of solutions is decomposed into several fronts. The nondominated solutions of the population receive rank 1 and form the first front E_1. The solutions that are not dominated except by solutions of E_1 receive rank 2; they form the second front E_2. In a general way, a solution receives the row k if it is only dominated by individuals of the population belonging to the unit $E_1 \cup E_2 \cup ... \cup E_{k-1}$. Then, the depth of a solution corresponds to the depth of the front to which it belongs. For instance, this strategy is applied to the NSGA-II algorithm [96].

2.4.1.3.3 Dominance count The dominance count of a solution is related to the number of solutions dominated by the solution. This measure can be used in conjunction with the other ones. For instance, in the SPEA algorithm family, the dominance count is used in combination with the dominance rank [97].

Since a single value fitness (rank) is assigned to every solution in the population, any search component of a single objective metaheuristic can be reused to solve MOPs. For instance, the selection mechanism in EAs can be derived from the selection mechanisms used in single objective optimization. The interest of Pareto-based fitness assignment, compared to scalar methods, is that they evaluate the quality of a solution in relation to the whole population. No absolute values are assigned to solutions.

2.4.1.4 Indicator-Based Approaches

In indicator-based approaches, the search is guided by a performance quality indicator [104]. The optimization goal is given in terms of a binary quality indicator I that can be viewed as an extension of the Pareto dominance relation. A value $I(A, B)$ quantifies the difference in quality between two approximated efficient sets A and B. So, if R denotes a reference set, the overall optimization goal can be formulated as

$$\arg\min_{A \in \Omega} I(A, R) \tag{2.18}$$

where Ω represents the space of all efficient set approximations.

The reference set R does not have to be known, it is just required for the formalization of the optimization of the optimization goal. Since R is fixed, the indicator I actually represents a unary function that assigns a fitness reflecting the quality of each approximation set according to the optimization goal I. If the quality indicator I is dominance preserving, $I(A,R)$ is minimum for $A = R$[842]. Indicator-based multi-objective algorithms have several advantages:

- The decision maker preference may be easily incorporated into the optimization algorithm.
- No diversity maintenance; it is implicitly taken into account in the performance indicator definition.
- Small sensitivity of the landscape associated with the Pareto front.
- Only few parameters are defined in the algorithm.

2.4.2 Diversity Preservation

P-metaheuristics are reputed to be very sensitive to the choice of the initial population and the biased sampling during the search. Diversity loss is then observable in many P-metaheuristics. To face this drawback related to the stagnation of a population, diversity must be maintained in the population. The fitness assignment methods presented previously tend to favor the convergence toward the Pareto optimal front. However, these methods are not able to guarantee that the approximation obtained will be of good quality in terms of diversity, either in the decision or objective space.

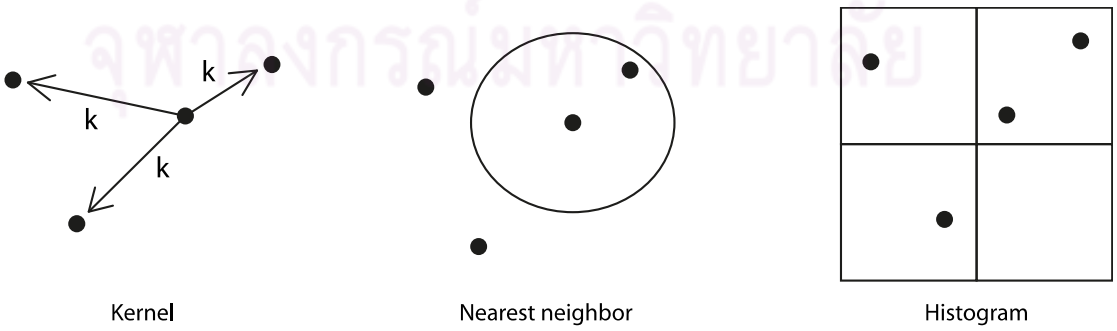


Figure 2.18 Diversity maintaining strategies.

Thus, diversity preservation strategies must be incorporated into multi-objective metaheuristics. In general, diversification methods deteriorate solutions that have a high density in their neighborhoods. As suggested in Ref. [102], the diversity preservation methods may be classified into the same categories used in statistical density estimation.

2.4.2.1 Kernel methods

Kernel methods define the neighborhood of a solution I according to a Kernel function K, which takes the distance between solutions as the argument. For a solution i, the distances d_{ij} between i and all other solutions of the population j are computed. The kernel function $K(d_i)$ is applied to all distances. Then, the density estimate of the solution i is represented by the sum of the Kernel function $K(d_i)$.

2.4.2.2 Nearest-Neighbor methods

In the nearest-neighbor approach, the distance between a given solution I and its k^{th} nearest neighbors is taken into account to estimate the density of the solution. For instance, this approach is used in the SPEA2 algorithm [97], where the estimator is a function of the inverse of this distance.

2.4.2.3 Histograms

The histograms approach consists in partitioning the search space into several hypergrids defining the neighborhoods. The density around a solution is estimated by the number of solutions in the same box of the grid. For instance, this approach is used in the PAES (Pareto archived evolution strategy) algorithm [105]. The hypergrid can be fixed a priori (statically) or adaptively during the search with regard to the current population.

One of the most important issues in the diversity preservation approaches is the distance measure. Many metrics can be used. Moreover, the distance may be computed in the decision and/or objective space of the problem. In general, in MOPs, the diversity is preserved in the objective space. However, for some problems, the diversity in the decision space may be important in terms of decision making, and may also improve the search.

2.4.3 Elitism

In general terms, elitism consists in archiving the "best" solutions generated during the search (e.g., Pareto optimal solutions). A secondary population, named archive, is used to store these high-quality solutions. First, elitism has been used to prevent the loss of the obtained Pareto optimal solutions. In this passive elitism strategy, the archive is considered as a separate secondary population that has no impact on the search process. Elitism will only guarantee that an algorithm has a monotonically nondegrading performance in terms of the approximated Pareto front. Then, elitism has been used in the search process of multi-objective metaheuristics (active elitism), that is, the archived solutions are used to generate new solutions. Active elitism allows to achieve faster and robust convergence toward the Pareto front for a better approximation of the Pareto front. However, a care should be taken to be trapped by a premature convergence if a high-elitist pressure is applied to the generation of new solutions.

The archive maintains a set of "good" solutions encountered during the search. The strategy used in updating the archive (elite population) relies on size, convergence, and diversity criteria.

2.5 Chapter Summary

This Chapter, we review the metaheuristics for solving combinatorial problems particularly in the permutation representation. Each metaheuristic approach is designed with the aim of both intensification and diversification or sometimes called exploitation and exploration. However, the terms exploitation and exploration have a more restricted meaning. In fact, the notions of exploitation and exploration often refer to rather short term strategies tied to randomness, whereas intensification and diversification refer to rather medium and long term strategies based on the usage of memory. As the various different ways of using memory become increasingly important in the whole field of metaheuristics, the terms intensification and diversification are more and more adopted and understood in their original meaning.

Table 2.2 summarizes them in term of their feature. The symbol $\forall$ indicates the existence of the features. While symbol $\exists$ presents partial feature and symbol $\neg$ indicates nonexistence of the features. Table 2.3 summarizes all the reviewed metaheuristics in term of intensification and diversification component.

Table 2.2 Feature classification of metaheuristics

Feature	Algorithm								
	SA	Tabu	ILS	GRASP	GA	ACO	PSO	EDA	SS
Trajectory	∀	∀	¬	¬	¬	¬	¬	¬	∀
Population	¬	¬	¬	¬	∀	∀	∀	∀	∀
Memory	¬	∀	∃	¬	∃	∀	∀	∃	∃
Multiple Neighborhood	¬	¬	∀	¬	∃	¬	¬	¬	∃
Nature-inspired	∀	¬	¬	¬	∀	∀	∀	¬	¬

Among the presented metaheuristics, SA, Tabu search and SS are typical examples of trajectory methods. These methods usually allow moves to worse solutions to be able to escape from local minima. Also local search algorithms which perform more complex transitions which are composed of simpler moves may be interpreted as trajectory methods. In ant colony optimization, iterated local search, genetic algorithms, GRASP, PSO and EDA starting points for a subsequent local search are generated. This is done by constructing solutions with ants, modifications to previously visited locally optimal solutions, applications of genetic operators, randomized greedy construction heuristics, driving particles and sampling from the probabilistic model respectively. The generation of starting solutions corresponds to jumps in the search space; these algorithms, in general, follow a discontinuous walk with respect to the neighborhood graph used in the local search. SS is consider to contain both trajectory and non-trajectory methods as the algorithm can be customized.

Table 2.3 Intensification and diversification component of metaheuristics

Metaheuristic	I&D component
SA	acceptance criterion + cooling schedule
ILS	black box local search
	kick-move
	acceptance criterion
TS	neighbor choice (tabu lists)
	aspiration criterion
GA	recombination
	mutation
	selection
ACO	pheromone update
	probabilistic construction
VNS	black box local search
	neighborhood choice
	shaking phase
	acceptance criterion
GRASP	black box local search
	restricted candidate list
PSO	Velocity
	Information Sharing
EDA	Sampling from distribution
SS	diversification generation method
	improvement method
	reference set update method
	subset generation method
	solution combination method

Tabu search, simulated annealing, iterated local search, and GRASP are such single-point search methods. On the contrary, in ACO, GA, PSO, EDA and SS algorithms, a population of individuals is used. Using a population-based algorithm provides a convenient way for the exploration of the search space. However, the final performance depends strongly on the way the population is manipulated.

Memory is explicitly used in Tabu search. Short term memory is used to forbid revisiting recently found solutions and to avoid cycling, while long term memory is used for diversification and intensification features. In ACO and EDA an indirect kind of adaptive memory of previously visited solutions is kept via the pheromone trail matrix which is used to influence the construction of new solutions. Also, the population of the GA, PSO and SS could be interpreted as a kind of memory of the recent search experience. Recently, the term adaptive memory programming [245] has been coined to refer to algorithms that use some kind of memory and to identify common features among them. Also ILS could be classified as an adaptive memory programming algorithm. On the contrary, SA and GRASP do not use memory functions to influence the future search direction.

ILS algorithms typically use at least two different neighborhood structures N and N_0. The local search starts with neighborhood N until a local optimum is reached and in such a situation a kick-move is applied to catapult the search to another point. The behavior of GA and SS has the same effect as the kick-move in ILS and therefore may also be interpreted as a change in the neighborhood during the local search. On the other side, the solution construction process in ACO, EDA and GRASP is not based on a specific neighborhood structure. Nevertheless, one could interpret the construction process used in ACO, EDA and GRASP as a kind of local search, but this interpretation does not reflect the basic algorithmic idea of these approaches.

Among the presented methods, ACO, SA, and GA belong to the nature-inspired algorithms. The others, have been inspired more by considerations on the efficient solution of combinatorial problems.

This dissertation mainly focuses on how a solution is generated. There are two main methods that are constructive methods and improvement methods. Table 2.4 summarize the methods in term of their capability to preserve the schema order of the initial solution.

Table 2.4 Information inheritance for generation methods

Generate methods		Preservation		
		Relative order	Absolute order	Edge
Improvement	Swapping		X	
	Insertion	X		
	Rotation	X		
	Inversion			X
Constructive	Modified (MX)	X		
	Order (OX)	X		
	Order based (OBX)	X		
	Position based (PBX)	X		
	Partially mapped(PMX)		X	
	Cycle (CX)		X	
	Weight Mapping (WMX)		X	
	Edge recombination (ER)			X

For the multi-objective metaheuristics, we present a unified view of their feature in an attempt to provide a common terminology and classification mechanisms including fitness assignment, diversity preserving and elitism.

The fitness assignment can be classified as scalar, criterion-based, dominance based and indicator based. The scalar methods are more suitable to S-metaheuristic. They are easy to implement, yet require a decision maker to fine-tune the parameter. Criterion based methods are used in both single solution and population based algorithms. These methods are good at finding the extreme solutions for each objective as they treating each objective separately, however lack of the coverage in the Pareto frontier. The dominance-based methods are more suitable to population based algorithms. They do not need the transformation of the MOP into a single one. Moreover, they are able to generate a diverse set of Pareto optimal solutions. Indicator-based method use indicator to determine the quality of a solution. The main advantage of this method is that it implicitly embedded the diversity mechanism into the indicator; therefore, no diversity maintenance is required.

The diversity preservation mechanism is needed to prevent the genetic drifting in constructive based algorithms. One of the most importance issues in the diversity preservation is the distance measure. Many metrics can be use. In Kernel methods and histogram based methods, the distances are usually determined in the objective spaces. While the nearest neighbor methods, the distances are usually determine by the genotypes of the solutions. Histogram based methods are easiest to implement, however, the grids sizes need to be appropriate to the fitness landscape. The nearest neighbor methods provide the best diverse solutions compared to the other methods, but require expensive computation.

One of the main issues of the multi-objective optimization is elitism. Archives are needed to maintain in order to improve both quality and quantity of the solution. The strategy used in updating the archive depends on size, convergence and diversity criteria.

CHAPTER III

NEGATIVE KNOWLEDGE

3.1 Introduction

In nature, the potential differences between two reference points produce potential energies including gravity, electricity, elasticity, pressure and temperature. The higher the potential difference causes the higher energy and thus higher acceleration rate of a particle carrying the energy. It is interesting to find out if there exists a potential difference of knowledge in order to accelerate the learning process in machine learning.

Machine learning methods usually use the empirical data, such as from sensor data or databases to shape the behaviors of computers. In classification tasks, the negative samples usually help the learner to learn a concept faster and more accurately. However, in optimization tasks, the negative information is rarely utilized. In improvement methods such as tabu search[2]. memories are used in order to forbid the search process to search in the landscape considered to be inferior. However, in constructive methods, the algorithms such as GA[13][7] and SS[76] try to extract only good substructures in order to compose a better solution, therefore positive information are usually kept in form of a population of desired solution, while the negative information are simply neglected. This research tries to study how the concept of negative knowledge contributes to the construction of a candidate in permutation representation.

3.2 Negative Knowledge

Negative knowledge has been previously discussed as developed in the field of artificial intelligence, education and business philosophy [106] [107] [108]. Before we get to know more about the negative knowledge, we need to clarify the definition of negative knowledge which commonly misunderstood to be bad, disadvantageous, or malign. Just like negative numbers, they are not being either good or bad. The term "negative" in negative knowledge does not imply a valuation. Instead, the term

“negative” refers to characteristic attributes which will be described later. To clarify an important attribute of negative knowledge requires considering the use of the term “knowledge” in constructivist theorization which is considered to be “propositional knowledge”. In constructivism, knowledge is not seen as an exact representation of reality, but rather as a

> *map of what reality allows us to do. It is the repertoire of concepts conceptual relations, and actions or operations that have proven to be viable in the pursuit of our goals.(von Glasersfeld)* [109]

In that sense, knowledge in a constructivist understanding is regarded as a system of representations and assumptions about reality that is closely related to individual goals or individual achievement. Roughly speaking, this means that individuals‟ assumptions can be called viable if they do not contradict previous knowledge and turn out to be useful for reaching goals. Yet, non-viable knowledge is knowledge that somehow stands in contradiction to prior knowledge or is counterproductive with regard to a certain goal.

The basic idea that is pursued through the concept of negative knowledge is that just because knowledge is non-viable in the described understandings, it is not necessary worthless or superfluous. This is because in order to reach a goal, there are often different ways that seem possible and the task of identifying the right one is very complex and demanding. Therefore, it is seen as having a heuristic advantage through knowing what is wrong in regard to a certain task: that is, to have negative knowledge. Hence, negative knowledge can be described as *non-viable knowledge* that is *heuristically valuable*.

According to the quote of von Glaserfeld, knowledge is compared to a map. Remaining with that metaphor, negative knowledge can be seen as an indicator of adverse ways, wrong turns or disadvantageous routes in order to reach a certain destination.

In 1994, Minsky[106] introduced explicitly the idea of knowing negatively in his literature “Negative Expertise”. He points out that competence often requires one to know what one must do, but it also requires one to know what not to do. Living things more often learn to avoid disaster rather than how to succeed in order to survive. In the process of learning, even experts seem to have negative rather than positive goals, namely that we seem to learn what should not be done. Minsky also states that the creativity of the machine does not only come from the randomization, but also come from the reduction of the search space. The performance of a smart or creative problem-solver is not how many trials precede a success, but how few. So the secret lies not in disorderly search, but in pre-shaping the search space so as reduce the numbers of useless attempts.

In 2005, Oser and Spychiger [107] define negative knowledge as knowledge about “*what something is not, (in contrast to what it is), and how something does not work, (in contrast to how it works), which strategies do not lead to the solution of complex problems (in contrast to those, that do so) and why certain connections do not add up (in contrast to why they add up)*”

In 2006, Parviainen and Eriksson[108] extended Minsky‟s idea. They further characterize the negative knowledge by identifying four features of negative knowing as follows:-

1) *to know what one does not know*: experts are usually aware of their own competence, but they must also know what they do not know and what they should know
2) *to know what not to do*: experts must know both how to achieve goals and how to avoid disasters, namely „learning what not to do‟
3) *unlearning and bracketing knowledge*: experts may get into a situation when they have to give up some parts of their knowing and „unlearn‟ or „bracket‟ their skills and know-how
4) *failures and mistakes*: experts should also regard the value of failures and disappointments as emotions, as well as recognize the creativity that emerges from making mistakes

Apart from the negative knowing, they also argue that negative is not considered as the mere empty opposite to the positive. They purpose that positive and negative knowledge can be considered as independent areas, which overlap one another in the following way:

Table 3.1 Linking positive and negative knowledge[108]

Positive knowledge	*Positive and Negative knowledge*	*Negative knowledge*
True justified beliefs	To know what one does not know	Unlearning and bracketing knowledge
Constructive, cumulative, paradigmatic	To know what not to do	Failures and mistakes ignorance

In 2008, Gartmeier et al.[110] describe the three functions of negative knowledge as follows:-

1) *negative knowledge increases certainty*: It is assumed that negative knowledge helps to increase individuals‟ certainty through awareness of possible positive as well as negative outcomes of their actions and through the capability to judge their respective probabilities under given circumstances.
2) *negative knowledge increases efficiency*: Negative knowledge is assumed to contribute to effective action.
3) *negative knowledge promotes reflection*: Negative knowledge is assumed to promote detailed reflective processes, because an essential component of reflection comprises an engagement with individual‟s prior and episodic knowledge. Remembering and being aware what is inappropriate in a given situation should enhance the ability to precisely discriminate similar phenomena.

The negative knowledge has shown to be beneficial in many ways. This dissertation presumes that it could be used in a constructive algorithm where building blocks are combined to form a better solution. On the other hand, bad building block should be avoided to construct a solution as well.

3.3 Related Concepts and Methodologies

3.3.1 Opposition-based learning

In 2005, the concept of opposition-based learning (OBL)[111] has emerged. Tizhoosh introduces the idea of learning toward the opposite states, opposite weights, opposite actions and many more opposition ways. The secret behind OBL is the simultaneous consideration of an estimate and its corresponding opposite estimate in order to achieve a better approximation of the current candidate solution. The opposition-based learning has been successfully applied to accelerate reinforcement learning[112], back propagation learning[111], and differential evolution[113].

OBL algorithms are considered to be utilizing of the negative knowledge to accelerate the optimization process. However the opposition-based extension idea of genetic algorithm is still too far from the negative knowledge. The anti -chromosome with inverted sub mutation can partially describe some of the negative information. The negative concept of a binary representation of a sub-chromosome [*101*] is not as simple as [*010*]. The complete negative concept should includes [*001*], [*011*], [*100*], [*110*] and [*000*] as well. This example indicates that the size of the negative concept in probabilistic-based learning is unimaginable large compared to the positive one.

3.3.2 Artificial immune system

Artificial immune systems (AIS) are computational systems inspired by the principles and processes of the vertebrate immune system. The algorithms typically exploit the immune system's characteristics of learning and memory to solve a problem. The negative selection techniques[114] in AIS try to output the complementary concept of the real target concept. Algorithms in this class are used in many areas including classification and optimizations.

3.3.3 Evolutionary Algorithms

There are some works based on evolutionary algorithms that try to utilize the negative knowledge hidden in the below average solutions by applying classification techniques in optimization. These algorithms include:-

3.3.3.1 Learnable Execution Model (LEM)

In 2000, Michalski[115] proposed algorithms that apply classifiers to develop a population of solutions. The candidates of a population are divided as the fittest and the less fitted ones. Then the characteristics of the good ones are strengthened while bad ones are avoided.

3.3.3.2 Statistical Learning + Inductive Learning (SI3E)

Later in 2003, Llorà and Goldberg[116] proposed an algorithm that combined the Induction of Decision Tree (ID3) and evolutionary algorithm using statistical approaches.

3.3.3.3 Evolutionary Bayesian Classifier-Based Optimization Algorithm (EBCOA)

In 2004, Miquelez, Bengoetxea, and Larrañaga[117] introduced a new estimation of distribution algorithm based on Bayesian classifiers and later extended in the continuous optimization domains[118].

3.3.3.4 Population Based Incremental Learning (PBIL)

In 1994, Baluja [71] proposed the first estimation of distribution algorithm called PBIL. This algorithm allows learning from the below average solutions. Variations of negative learning rate were used in the original paper. From the empirical experiments, different negative learning rates show their effectiveness in different behaviors in different benchmarks. Moreover, negative correlation learning in PBIL contributes to find the optimal solutions in hard deceptive problems faster than the other benchmark algorithms in term of function evaluations.

3.3.3.5 Compact Genetic Algorithm (cGA)

In 1998, Harik et al. presented and EDA called CGA [72]. In each iteration, cGA generates two candidate solutions from the current probability vector. Then, the two generated solutions compete against each other. The the winner and the loser are used to update the probabilistic vector using reward and punishment model. Performance of cGA can be expected to be similar to that of PBIL.

3.3.3.6 Incremental Bayesian Optimization Algorithm (iBOA)

In 2008, Pelikan et al. proposed another EDA called iBOA [119]. This algorithm modify the original Bayesian Optimization Algorithm (BOA) to estimate the probabilistic model in an incremental manner. Similar to cGA, the loser of a tournament selection lost their probability to be regenerated to the winner. The main benefit of using iBOA instead of the standard BOA is that iBOA eliminates the population and it will thus reduce the memory requirements of BOA. iBOA also provides the first incremental EDA capable of maintaining multivariate probabilistic models built with the use of multivariate statistics.

3.3.4 Particle Swarm Optimization

In 2005, Yang and Simon proposed a new version of PSO that utilized only the negative knowledge in optimization where each particle adjusts its position according to its own previous worst solution and its group's previous worst to find the optimal value. The strategy is to avoid a particle's previous worst solution and its group's previous worst based on similar formulae of the regular PSO. In their experiments, the results show that the NPSO[120] always finds better solutions than PSO in every benchmark.

3.3.5 Evolutionary Ensemble with Negative Correlation Learning (EENCL)

In 2000, Liu, Yao and Higuchi proposed ensemble techniques [121][122] which composed of neuron networks (NN). The negative correlation learning and fitness sharing were adopted to encourage the formation of diverse species in the population.

3.4 Schema Theorem and Order Schema

The Schema Theorem is defined by Holland[13] represented a mile stone in the development of Genetic Algorithms. In schema theorem, the search space is partitioned into subspaces of varying levels of generality, and mathematical models are constructed which estimate how the number of individuals in the population belonging to certain schema can be expected to grow in the next generation.

3.4.1 Schema theorem

Theorem 3.1: Schema Theorem [13]

The expected number E of schema H at generation $t+1$ when using a canonical GA with proportional selection, single point crossover and gene wise mutation is,

$$E[m(H,t+1)] \geq \frac{m(H,t)f(H,t)}{\bar{f}(t)}\left\{1 - p_c \frac{\delta(H)}{1-l} p_{diff}(H,t) - o(H)p_m\right\} \quad (3.1)$$

where $p_{diff}(H,t)$ is the probability that parent does not match schema H; p_c is the selected threshold of applying crossover and p_m is the threshold of applying mutation.

Schema theorem implies that schema with fitness greater than the average population fitness are likely to account for proportionally more of the population at the next generation. From this Theorem, Goldberg arose the building block hypothesis (BBH), which attempted to explain how a GA solves a problem by positing that near optimal solutions were forged from small, low-order, better-than-average schemata.

3.4.2 Order schema

In 1992, Kargupta, Deb and Goldberg [123] discussed about schemata in permutation problems, the so called order schema or o-schema is defined by assigning a sequence characteristic to a similarity subset. It has unique alleles at all of its fixed positions and contains all permutations of other alleles at don‟t care positions. In general, an o-schema can be classified into two broad categories – *absolute ordering schema* and *relative ordering schema.*

Table 3.2 A Comparison of some statistics for binary,*l*-ary and size-*l* permutation problems[123]

	$binary$	$l-ary$	$size-l\ permutation$
Solution Space	2^l	l^l	$l!$
String in an order $o(H)$ schema	$2^{l-o(H)}$	$l^{l-o(H)}$	$(l-o(H))!$
Number of $o(H)$ schemata	$2^{o(H)}$	$l^{o(H)}$	$Pm(l,o(H))$
Total schemata	3^l	$(l\ +\ 1)^l$	$\sum_{i=0}^{l}\binom{l}{i}Pm(l,i)$

3.4.1.1 Absolute order schema The absolute o-schema defines a similarity subset having some common allelic position characteristics. For instance, the absolute o-schema [! ! ! 1 ! 8 ! !] defines the subset of all valid permutation string that have alleles 1 and 8 in fourth and sixth positions respectively. The string $S1\ =$ [4 3 2 1 5 8 7 6] is contained in the schema, while $S2\ =$ [1 5 8 2 4 3 5 6 7] is not contained in the schema. This schema representation is useful in the problems where the placement of certain position is important. In this type of schema, both ordering and position of the defined alleles are important.

3.4.1.2 Relative order schema The relative o-schema defines a similarity subset having some common order allelic characteristics which in between the order there can be some gap containing any size of the permutation substring. For example, the relative o-schema [! ! ! 1 ! 4 ! !] represents the subset of all valid permutation string that have allele 1 happens before 4 in any configuration without having any restriction on the specific allelic position of genes. This coding is important for problems where ordering among alleles is the only matter.

According to the literatures [123] and [7], the definitions of relative o-schema are inconsistent. The definition defined in 1989[7] allow only the fixed distance between the defined schemata. For example, the strings containing 4 placed

after 1 with in a fixed order of gaps. In this dissertation, we refer to the relative o-schema proposed in 1989 as type I relative o-schema and the one in 1992[123] as type II relative o-schema.

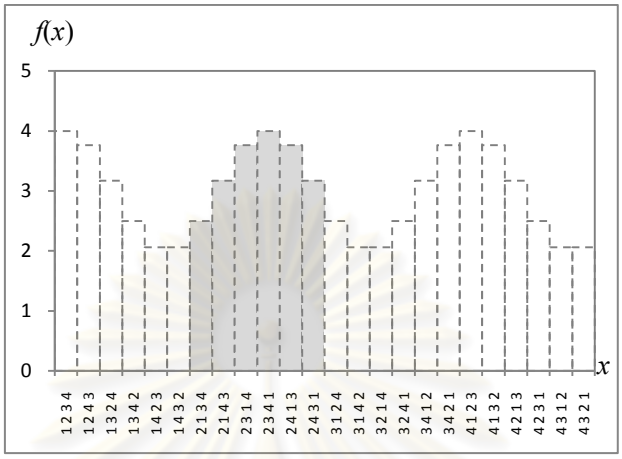
Size of a schema can be used to describe the searching boundary of a GA. If a schema dominates a population, the available solution that a simple crossover operator can search is reduced according to the size of the schema. Table 3.2 summarizes the statistics of three different types of schemas, that are binary, unary with size l and permutation with size l. For example, in a binary representation GA, let $[**\ 1\ **]$ be a binary schema with order 1, given size l of the solution strings equal to 5, the solution space contained only $2^{l-o(H)} = 2^{5-1} = 14$ total strings. The number of solution per space ration is 1/2. The larger number of order, the smaller number of available space is.

The size of a schema in higher based unary is relatively less than the size of a schema in lower based unary. For example, let $[**\ 4\ **]$ be a based 5 l-ary schema with order 1, given size l = 5. The solution space is equal to $5^5 = 3{,}125$, while the number of solution strings in an order 1 schema is equal to $5^{5-1} = 625$. The number of solution per space ratio is 1/5 which is relatively less than the binary"s.

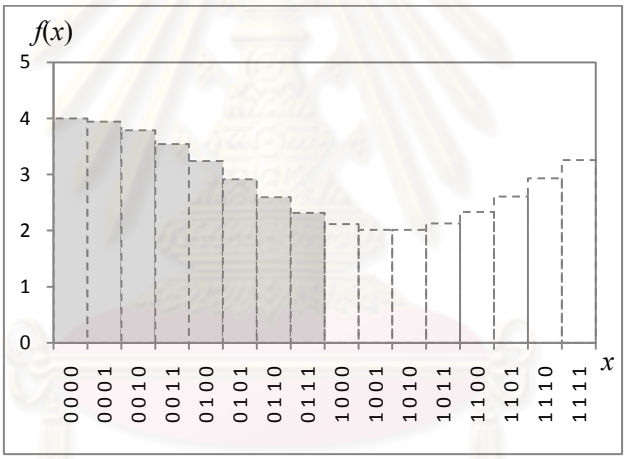
Once a schema dominates the population, a GA reduces the search space according to the based l-ary schema. This higher based schema indicates the higher degree of exploitation. Consequently, many researches in GA prefer to represent the genotype of a solution string using the lowest order as possible, in order to fine grain the search and to preserve the diversity.

In permutation representation, the redundancy of an item is not allowed. This means that an item can occupy only one position at a time. Therefore, if an o-schema dominate the population, the number of solution per space would be worse than a unary schema with the equal based. Let $[**\ 1\ **]$ be a binary schema, there would be only 16 strings in the order 1 schema which is $1/2$ of the solution space. While an absolute o-schema $[!!\ 1\ !!]$ reduce the solution space down from 120 to only 24 strings. Figure 3.1 shows the effect of how a schema with greater

fitness dominates the population in permutation and binary representation. The gray area indicates the number of solutions in the schema, while the white area indicates the deducted solution spaces.



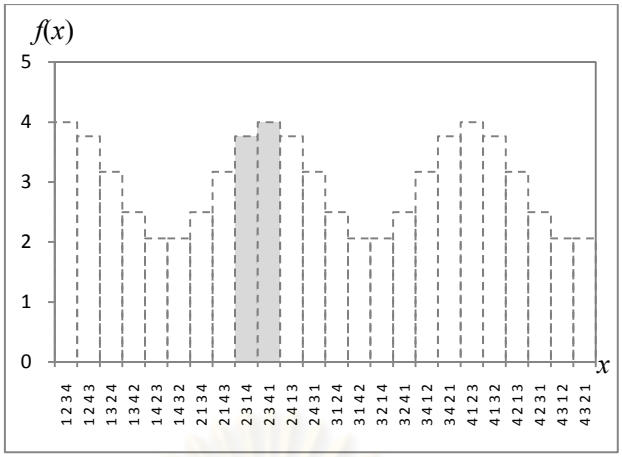
(a) absolute order schema



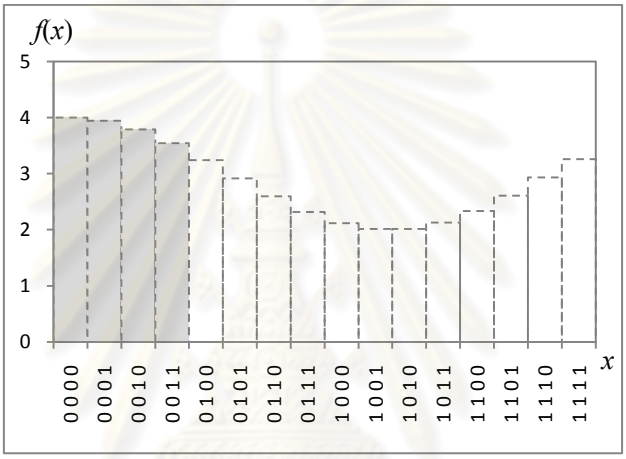
(b) binary schema

Figure 3.1 Effect of how a schema dominate the search space

In figure 3.1 (a), an absolute order schema [2 ! ! !] dominate the population causes the reduction of the search space by 3/4, while a binary schema [0 ∗ ∗ ∗] reduces only 1/2 of the whole space in figure 3.1 (b). As generation progressed, the more specific order schema [2 3 ! !] and binary schema [0 0 ∗ ∗] dominates the population. The order schema reduces the search space at a much greater rate compared to the binary schema. In figure 3.2 (a) the order schema with order 2 reduces the space down to only 1/12, while the binary with order 2 reduces the search space down to 1/4 .



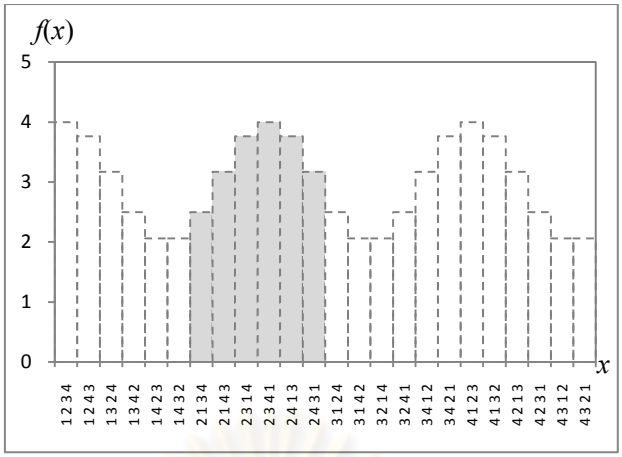
(a) absolute order schema



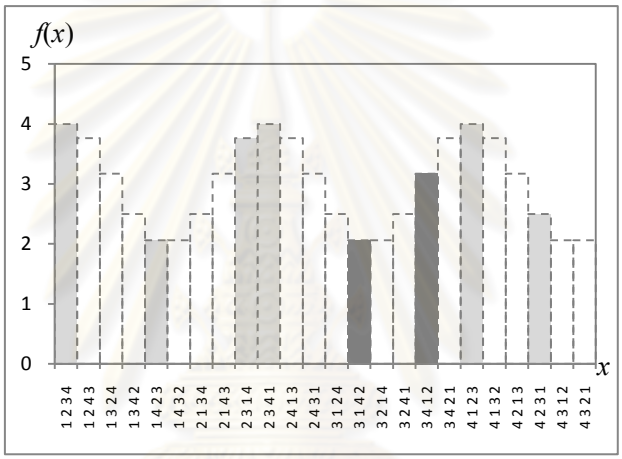
(b) binary schema

Figure 3.2 Effect of how a more specific schema dominate the search space

From the example, binary encoding let the algorithm to fine grain the search. However, such encoding could consequence a big trouble as the recombination and mutation of one or more solution strings are most likely to generate infeasible solutions. Therefore, encoding a solution string in permutation is unavoidable. Unfortunately, as mention in Chapter I, using a permutation encoding could easily lead to a premature convergence. This is the explanation why GAs with permutation encoding do not scale.



(a) absolute order schema



(b) edge schema

Figure 3.3 A comparison of absolute order schema and edge schema

3.4.2.3 Edge schema In this dissertation, we aim to study the roles of negative knowledge mainly on edge based estimation of distribution algorithm. Therefore we propose a new schema call edge schema. An edge is a link or connection between two nodes and has important information about the permutation string. However, edge can be symmetry. This research refers to the edge schema as the edges are asymmetry. The set of edge schema can be considered as a subset of relative o-schema, yet the gaps are not taken into account. Figure 3.3 illustrate the comparison of an absolute order schema and an edge schema. In a fitness landscape, two types of schema conquer different geometries. The figure 3.3 (a) illustrates how an absolute o-schema [2 ! ! !] dominate the population, while (b) illustrates an edge schema[! 2 3 !]. This can be seen that the number of solutions contained in the edge

schema [! 2 3 !] is equal to that in the absolute o-schema [2 ! ! !]. The two darker gray bars in (b) are the solution where the circulated edges are taken into the account. From this example, we also illustrate the different kind of neighborhoods. Using inappropriate move operators or crossover operators could not be able to find such good regions. As illustrated, the edge schema can cover all the global optimal solutions and consider them to be the neighborhood of each others.

3.4.3 Negative Schema

In 2006, Tae and Lee[124] prove that a negated concept is defined implicitly by a hidden feature abstracted from the property common to all the objects not belonging to the concept. This paper proposes based on Minsky that there is a logical schema that enables an agent to perceive a negated concept. However, the negative schema of Tae and Lee is defined based on an assumption in which an agent can recognize only one concept at a time. They also state that the positive and negated concepts, exist together, and two concepts cannot be recognized at the same time.

3.5 Negative Order Schema

In this work, we purpose a way to recognize negative schema in the ordering problems and show the role of the negative schema in search and optimization.

In most evolutionary algorithms, the positive schemas usually dominate the population and assume that the solutions in the inverse set of the schema are negative due to the selection process filters the less fitted solutions. Thus the strings that are not contained in the schema are extinct especially in the ordering problems; the strings satisfied by the schema are not easy to regenerate by chance. The search and optimization procedure might be stuck in some local peaks. Moreover, some of the multimodal and multi-objective solutions might be missing. These problems are well aware therefore many researches try to preserve the diversity and elitist in order to explore more in the uncovered searching areas and prevent the extinction of solutions in the uncovered space.

Negative o-schemas play a different role compared to the positives. They are used to void the search space rather than to limit the search space. The more specific schemas void less space than the general ones as can be seen in Figure 3.4 The negative o-schemas are defined using a "~" in front of the positive o-schema. For example ~[! 3 4 !] is negative edge schema that void the search space from the string containing in the negative schema. The search space after voiding is reduced down from $4!$ to $4! - 3!$. Therefore the remaining space is greater than the space bounded by the positive schema with the equally order.

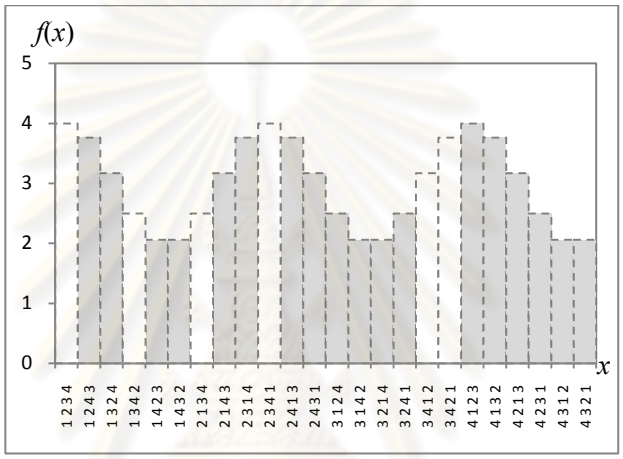


Figure 3.4 An example of a negative edge schema ~[! 3 4 !]

The positive schemas dominated by the selected solution seen so far, thus the uncovered peaks that have not been found in the previous generation were not considered. In contrast, if the negative schemas are identified by the solution seen so far in the very beginning generations, the remaining spaces apart from the set of the strings contained in the negative schemas remain to be explored. This mechanism enhances the preserving of diversity and force the algorithm to search in the unexplored space.

3.6 Applying the Negative Knowledge in Optimization

In most evolutionary algorithms, the schemas are kept in a form of the selected population. Thus the knowledge of which schemas are likely to form the bad solutions, are abandoned with the non-selected population. The evolutionary process repeats searching for the more specific schema within the bounded area assuming that the solutions in the unexplored space would not be able to survive. The solutions in the unexplored area are composed of the unknown to the positive area and known negative area.

Vice versa, the negative schemas void the search space and left behind the unexplored area that composed of unknown quality solutions and the positive known solutions.

In theory, the negative concepts should be the complement of the positive concepts. However, it is impossible to identify the positive and the negative concepts without verifying all of the solutions in the space. The learning process starts from the unknown space. As the learning progresses, the solutions are verified and classified.

In order to combine together the positive and negative schemas, we need to discriminate between the good and the bad schemas. However some average solutions might contain both good and bad schemas in a solution, such structure should not be prejudged and should not be recognized as either good or bad. Therefore the selection methods also need to distinguish between the good solutions, the bad solutions and the solutions that might contain both positive and negative schemas.

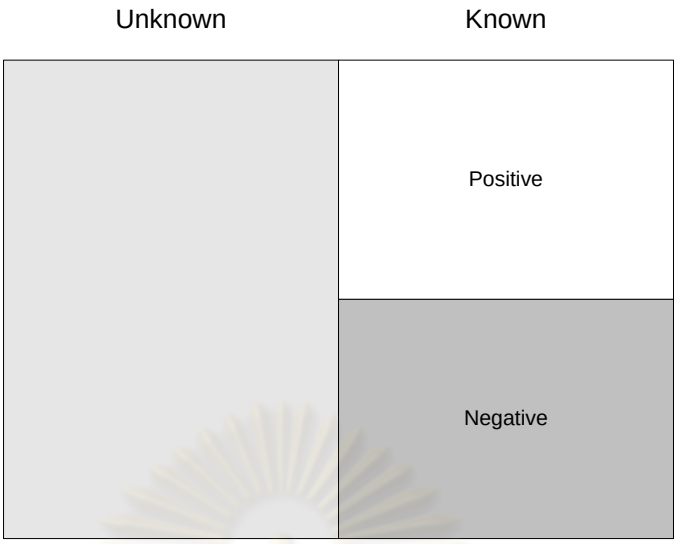


Figure 3.5 The classification of the solution in the space

According to the negative knowledge defined by Parviainen and Eriksson [108], the negative cannot be considered to be the complement of the positive due to the space contain not only the known positive and known negative but also contain the unknown.

In order to utilize the negative knowledge in learning, we apply the four feature of the negative knowing proposed by Parviainen and Eriksson. First of all we need to categorize the solutions in to two categories as known and unknown. Moreover, the known solutions are also divided in to positive and negative. This enable the algorithm to distinguish between what is known and what is not known.

The algorithm should be able to identify the negative schemas in order to pre-shape the search space and try not to waste the function evaluation with the expected undesired geometries containing the bad schemas. Thus, we need a data structure to keep the state of schema.

Moreover, the solutions found in the good solutions might contain the same schemata found in the bad solutions or vice versa, which can be classified as false positive or false negative. Thus, the algorithm should be able to justify the gained knowledge and should be able to unlearn or bracket the knowledge back to unknown information or re-classify or re-justify the old beliefs.

3.7 Chapter Summary

This chapter, we give an introduction to the negative knowledge and give a brief review of optimization methodologies utilizing them. In many literatures, the negative knowledge is proved to be beneficial in many ways including the quality of solutions, the diverse solutions, and the time to convergence.

We adopt the order schema as a tool to explain the behaviors of constructive algorithms based on permutation representation. In addition, we introduce a new subtype of relative o-schema called edge schema and the negative order schema. From the models, we presume that the negative knowledge in edge representation should contribute in the following ways:

1) The negative knowledge forces the algorithm to explore out of the search space marked as forbidden areas.
2) The negative knowledge forces to produce diverse solutions, however dissimilar to the solutions considered to be bad quality.
3) In cooperating with the positive knowledge, the negative knowledge contributes in discrimination of good and bad substructure. From such cooperation, negative knowledge should enhance a constructive method should recognize better substructures and composing better solutions.

Finally, we propose some guidelines to design an adaptive searching algorithm incorporate the negative knowledge as a form of classifier based algorithm. However, such guidelines are not limited to the permutation representation.

CHAPTER IV

COINCIDENCE ALGORITHM

4.1 Introduction

In the Chapter III, we propose some guidelines for designing an evolutionary algorithm that can make use of negative knowledge in optimization. In this chapter, we introduce a new evolutionary algorithm in a form of estimation of distribution algorithm.

4.2 Coincidence Algorithm

The proposed algorithm is explained in this section. The main idea of the algorithm is to allow learning from the below average solutions as well as the traditional learning from the good solutions. The coincidence found in a situation should be able to statistically describe the chance of the situation to be happening whether the situation is good or bad. Thus the learning of the coincidence found in the bad solutions should be used to avoid the bad situation as well.

4.2.1 Design

Coincidence algorithm (COIN) is designed to construct the solutions based on the mutual information in edge schema. We assume that there are linkages between each of the pairwise items. In this algorithm, we rather focus on the pair of permuted objects called incidences than the absolute position of a single object. For example, two candidates with order 5, 1-2-3-4-5 and 4-5-3-2-1 share the common coincidence 4-5 which is considered to be a schema in edge schema.

According to the definition of building blocks hypothesis (BBH), the coincidences can be considered to be the building blocks. However, we would not rather call the coincidences as building blocks due to the coincidences can describe only some partial building blocks in relative o-schema.

COIN adapts the first order matrix of MCMC[125] (Markov Chain Monte Carlo) as a data structure to maintain the joint probabilities, this matrix is used to learn both positive and negatives incidences found in the populations. Then it is used to generate the populations according to the conditional probability. Even though the relative o-schemas with gaps are not easy to be recognized, they are indirectly identified using the conditional probability property of Markov Chain. Therefore COIN can indirectly recognize the relative o-schema as well.

COIN uses the same distribution similar to EHBSA. However, COIN does not estimate the selected population the same way as EHBSA does. COIN rather uses the incremental model based on reward and punishment. When an incidence is found in the above average solutions, it is rewarded more probability to be selected. Otherwise, if an incidence is found in the below average solutions, it would be punished by deducting the probability to be selected. The reward probabilities are gathered from the other incidences equally, while the deducted probabilities are scattered to the other incidences the oppositional way.

4.2.2 Components

Similar to most black box optimization algorithms, the components of the algorithms are composed of data structure and fitness function(s) evaluator. The data structure of this algorithm mimics from the first order matrix of Markov Chain in which we use to learn the positive and negative building blocks in a form of joint probabilities and then use the joint probabilities to generate the candidates. We simply call it a generator.

4.2.2.1 Generator

The COIN algorithm uses a generator to generate the population according to the coincidences found in the good and the bad candidates. The generator H is a matrix of size $n \times n$ where n is the size of a permutation. The sum over each row $H_{i,j}$ where j ranges from 1 to n equals to 1.0. It denotes the probability of the occurrence of ij in the solution string. Each entry of $H_{i,j}$ has a value between 0 to 1.0. The diagonal $H_{i,i}$ are 0.

4.2.2.2 Fitness Function Evaluator

The fitness function evaluator is used to evaluate the fitness of the solution generated by the generator. To maximize the efficiency of the algorithm, the solutions are sorted on the fly (an insertion sort is recommended) as the selection mechanism of the algorithm needs to select the solutions from their ranks.

4.2.3 Mechanics

The mechanism of the COIN algorithm is shown in Algorithm 4.1. It begins by initializing the generator then the population is sampling from the generator. The generator is updated by each of the coincidences found in the selected good and bad candidates according to their evaluated ranks. The generating, evaluating and updating steps are repeated until a termination condition is met.

```
PROCEDURE COIN

1. Initialize Generator

2. Repeat

3.   p <- Sampling(Generator)

4.   p' <- Sort(Evaluate(p))

5.   Reward(Selection(Top(p)),Generator)

     Punnishment(Selection(Bottom(p)),Generator)

6. Until Termination Condition is met
```

Algorithm 4.1: Estimation of Distribution Algorithm

4.2.3.1 Initialization

The generator H is initialized so that each of the joint probabilities $H_{i,j}$ except the $H_{i,i}$ equal to $1/(n-1)$. The summation of all joint probability $H_{i,j}$ where j range from 1 to n equals to 1. This initialization represents the uniform distribution of each joint probability.

4.2.3.2 Generating population

The sampling algorithm of COIN is similar to EHBSA and ACO. Each individual are sampling one position by one position from head to tail. Each position is generated depend on the current and all of the previously generated position. Therefore the permutation sequences are always feasible. The sampling procedure is as follows:

PROCEDURE SamplingfromCOIN

1. Set position counter $p \leftarrow 0$

2. Obtain first node $c[0]$ randomly from $[0, L-1]$

3. Construct a roulette wheel vector $rw[]$ from matrix as $rw[j] \leftarrow H_{c[p],j} (j = 0,1, \dots, L-1)$

4. Set to 0 previously sampled nodes in $rw[]$ $(rw[c[i]] \leftarrow 0$ for $i = 0, \dots, p)$

5. Sample next node $c[p+1]$ with probability $rw[x] / \sum_{j=0}^{L-1} rw[j]$ using roulette wheel $rw[]$.

6. Update the position counter $p \leftarrow p+1$.

7. If $p < L-1$, go to Step 3.

Algorithm 4.2: Sampling Algorithm of COIN

4.2.3.3 Selection

Two selection methods are considered: a uniform method selects from the top and bottom c percent of the population and an adaptive method selects from the population above and below the average band of two standard deviations.

In the adaptive selection process, if the population contains more good candidates, the selector will select more of the bad solutions rather than the good solutions. Conversely the selector would select more of the good solution when the overall candidates in the population are not good. This mechanism maintains the fitness distribution among the candidates in the objective space in which we hope to be corresponded to the diversity of the candidates in the decision space.

4.2.3.4 Updating the generator

In the initialization phase, the joint probabilities $H_{i,j}$ are equally initiated so that the probabilities to be selected are uniform. As the generation progresses, the candidates are ranked, good and bad populations are well separated. In this phase, the mutual information indicating the joint probabilities are used to bias the generator in order to generate the desired candidates being closed to the good concepts and avoid generating the undesired candidates being distant to the opposite concepts.

The reward and punishment schemes are used to bias the generator. The coincidence found in the top ranks are considered as good building blocks and given more probabilities to be chosen as rewards. On the other hand, the coincidence found in the bottom ranks are considered as bad building blocks and punished by deducting the probabilities to be chosen.

The incremental and detrimental models used in the algorithm are different to the other evolutionary algorithms based on probabilistic models as most of them are represented in binary. The good substructures are usually rewarded by deducted from the bads[71][72][119]. In this algorithm, when good and bad coincidences are found, all the other coincidences ij sharing joint probability i are affected. The generator updates the good and bad joint probabilities using two different methods.

4.2.3.4.1 Reward When each coincidence i,j is found in a better group of candidates, the reward is given to $H_{i,j}$ by gathering the probability $\frac{k}{(n-1)^2}$ from the $H_{i,j}$ where j range from 1 to n, $j \neq i$. k is denoting the learning coefficient, and $r_{i,j}$ is the total number of coincidence i,j counted from the good solution. The reward equation is

$$H_{i,j}(t+1) = H_{i,j}(t) + \frac{k}{(n-1)}(r_{i,j}) - \frac{k}{(n-1)^2}\left(\sum_{z=1}^{n} r_{i,z}\right) \tag{4.1}$$

The last term, $\frac{k}{(n-1)^2}\left(\sum_{z=1}^{n} r_{i,z}\right)$, represents the adjustment step for all "others" $H_{i,j}$ (z $\neq$ i, z $\neq$ j) in the opposite direction hence keeping the sum of all probabilities in a row constant to one.

4.2.3.4.2 Punishment Contrary to the rewarding, when each coincidence i, j is found in a worse group of candidates it is used to update the joint probability $H_{i,j}$ by scattering its own probability $\frac{k}{(n-1)^2}$ to every member $H_{i,j}$ where j range from 1 to n, $j \neq i$. k is the coefficient denoting the learning coefficient, and $p_{i,j}$ is the total number of coincidence i,j counted from the bad solution. The punishment equation is

$$H_{i,j}(t+1) = H_{i,j}(t) - \frac{k}{(n-1)}\left(p_{ij}\right) + \frac{k}{(n-1)^2}\left(\sum_{z=1}^{n} p_{i,z}\right) \tag{4.2}$$

The last term, $\frac{k}{(n-1)^2}\left(\sum_{z=1}^{n} p_{i,z}\right)$, also represents the adjustment step for all "others" $H_{i,j}$ (z $\neq$ i, z $\neq$ j) in the opposite direction hence keeping the sum of all probabilities in a row constant to one.

Combining together reward and punishment when a coincidence i, j is found in both good and bad solutions we will get:

$$H_{i,j}(t+1) = H_{i,j}(t) + \frac{k}{(n-1)}\left(r_{i,j} - p_{i,j}\right) + \frac{k}{(n-1)^2}\left(\sum_{z=1}^{n} p_{i,z} - \sum_{z=1}^{n} r_{i,z}\right) \tag{4.3}$$

There is some constraint in updating the generator. Since the joint probability is updated by increasing or decreasing by a constant rate, a joint probability must not become negative. Therefore we need to maintain the probability value by disallowing the punishment if it would decrease the probability down below 0.

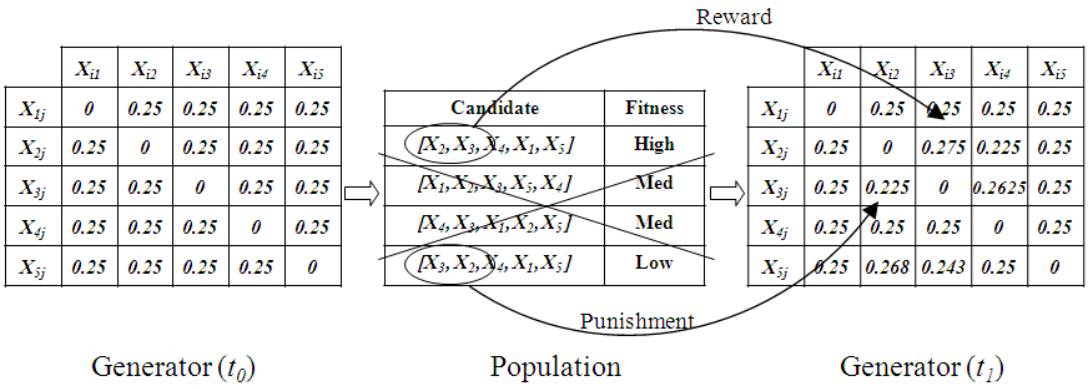


Figure.4.1. Updating the generator *k=0.1*

Figure. 4.1 exemplifies the process of initializing the generator, generating the first population, selection of good and bad candidates and finally updating the generator using the selected candidates. Since the problem size is equal to 5, the generator is initialized so that each joint probability is equal to 0.25. Then, the population is generated from the initiated generator. The candidates are sorted and classified into three classes: high fitness, medium fitness, and low fitness. The high fitness candidates are considered to be the good solutions while the low fitness candidates are considered to be the bad solutions in the population.

As seen in the figure 4.1, the candidate X_2-X_3-X_4-X_1-X_5 is classified as a good solution. The incidences X_2-X_3, X_3-X_4, X_4-X_1 and X_1-X_5 are used to update the generator as rewards. The candidate X_3-X_2-X_4-X_1-X_5 is classified as a bad solution thus the incidences X_3-X_2, X_2-X_4, X_4-X_1 and X_1-X_5 are used to punish the generator in the opposite way. Since the coincidences X_4-X_1 and X_1-X_5 are found in both good and bad solutions, they are counted as a one-time reward and a one-time punishment so the row *1,j* and row *4,j* remain unchanged. While X_2-X_3 and X_3-X_4 are considered to be the coincidences found in the good solutions, hence these coincidences are used to update the row X_2 and X_3. The coincidences X_3-X_2 and X_2- X_4 are used to punish the generator as they are found in the bad solutions.

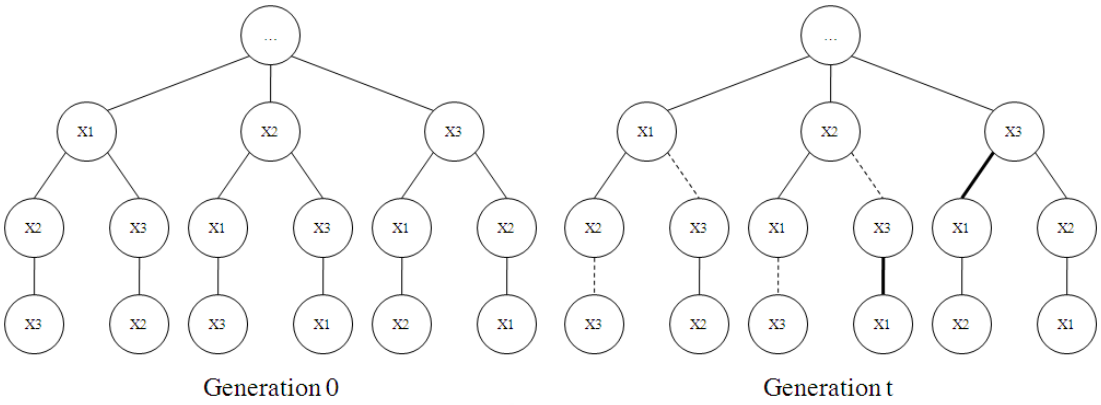


Figure. 4.2. The probability dependency tree of a 3 dimensions combinatorial problem

Figure. 4.2 represents the search space of a 3 dimensional ordering problem. In the generation 0, all joint probabilities are equal. As the generation progresses, the joint probabilities are increased or decreased. It can be seen that some of the connections are weaken as they are statistical found in the bottom ranks. Whereas some of the connections are strengthen as they are found in the top ranks.

4.2.4 Computational cost and space

Let the problem size = n, and there are m candidates in each generation, the computational cost and space complexity are as follow:

1. Generating the population requires time $O(mn^2)$ and space $O(mn)$
2. Sorting the population requites time $O(m\ log\ m)$
3. The generator require space $O(n^2)$
4. Updating the generator require time $O(mn^2)$

4.3 Multiobjective Coincidence Algorithm

The multiobjective version of coin is slightly different from the single objective COIN in the selection method. We adopt the non-dominate sorting and crowding distance of NSGA-II[25] as the way to select the population used in updating the generator. Again, we use the not-good solutions to update the generator. The not-good solutions are defined different from the single-objective COIN. They are obtained from the non-dominated frontier of the opposite side of the objectives we are optimizing.

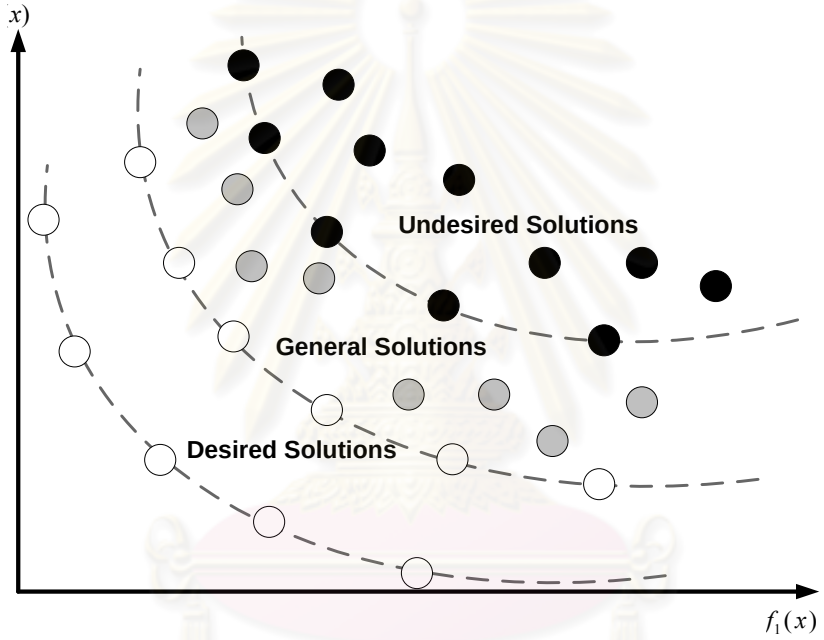


Figure. 4.3. The non-dominate ranking in multiobjective coincidence algorithm

Figure. 4.3 shows the non-dominate ranking in Multi-objective COIN. The number of the selected candidates depends on the rank of the frontiers. In this case the first and the second ranks contain 10 candidates, while the last two ranks contain 11 candidates.

4.4 Discussion

In this Section, we differentiate COIN from the existing algorithms in the Chapter II and III. Figure 4.4 illustrates genealogy of COIN. Clearly, COIN is a population based metaheuristic and is also a constructive algorithm. COIN and EHBSA[18] use the different encoding and different probabilistic model compared to most EDAs [71][72][119]. The data structures of COIN and EHBSA are more similar to ACO[20]. However, the distinctiveness of COIN is that it learns the negative correlation of the worse group of candidates as well as the traditional better group of candidates. COIN and ACO incrementally combines the solution components of the latest generation with all the previous iterations while EHBSA construct a new probabilistic model only from the most recent population. However, ACO uses a more complicate model of pheromone evaporation for exploration purpose in which an appropriate parameter tuning is needed.

The negative knowledge of COIN is different to that used in NPSO [120] as COIN considers the correlation of bad building blocks in the candidate solutions while NPSO considers the negative knowledge of the geometry of neighborhoods.

The incremental model of COIN is also difference to PBIL, cGA and iBOA. PBIL, cGA and iBOA deduct the probability from the opposite values at the same absolute positions. However, COIN gathers the probability from the rest of the edges sharing the same starting nodes. Incremental models of PBIL, cGA and iBOA consider being reward and punishment at the same time, whiling, COIN separately uses different method to the reward. The punishment model of COIN scatters the probability of an edge to the rest of the edges sharing the same starting nodes.

The multiobjective version of COIN (MO-COIN) mimics the non-dominated sorting and crowding distance from NSGA-II. However, COIN is an EDA, therefore COIN does not need to maintain the elitists. The purpose of embedding the crowding distance in MO-COIN is to prevent the premature convergence of the probability model.

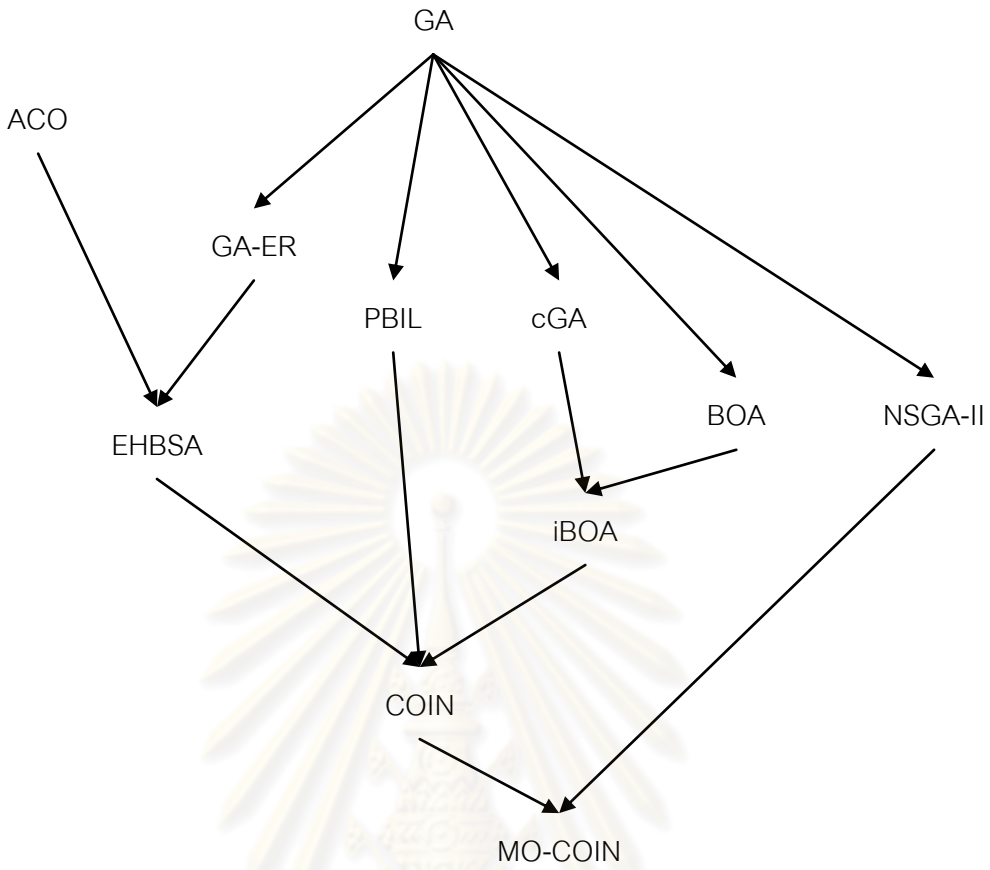


Figure. 4.4. Genealogy of COIN.

4.5 Chapter Summary

In this chapter, we propose a new edge-based estimation of distribution algorithm called COIN, which incorporate the negative correlation learning. The reward and punishment scheme is used to update the probabilistic model adopt from the first order Markov chain matrix. The selection process of COIN discriminate the good and the bad edges found in the better and the worse groups of candidates. This mechanism enhances the algorithm to recognize the better substructures in order to compose them. In addition, the worse substructures are discriminate and can be avoided. The figure 4.5 illustrates the recognition of the better quality substructure by eliminating the substructures found in the worse group.

Figure. 4.5. The differentiation of substructures contain in the good and the bad populations.

We also, propose the extension of COIN to solve the multi-objective problems by adopting the non-dominated sorting and crowding distance from NSGA-II. Finally, the distinctive characteristics of COIN are discussed.

CHAPTER V

EMPIRICAL ANALYSIS

5.1 Introduction

Previously in chapter III and IV, we set a hypothesis that the negative knowledge should (*i*) prevent the composition of bad building blocks and should (*ii*) preserve more diversity. In order to test the hypothesis, this chapter compares two EDAs, COIN and EHBSA which utilize similar probabilistic model in order to construct a candidate solution. The major difference of COIN and EHBSA is that the COIN takes the negative knowledge into account, while EHBSA doesn't consider the negative knowledge at all. In addition, COIN incrementally combines the solution components of the latest generation with all the previous iterations while EHBSA construct a new probabilistic model only from the most recent population.

One approach to investigate the behavior of EAs is to test them on artificial problems where the solutions are known a priori. For this purpose researchers usually used deceptive problems, which are hard globally multimodal optimization problems. In our experiments, we use multimodal combinatorial puzzles where the solutions are known and expected to mislead the testing algorithm to certain local optimal points. We compare COIN with EHBSA in two classes of combinatorial problems that are permutation and combination. The permutation problems include 8-Queens puzzles, 3×3 magic square, 4×4 magic square and knight's tour problems while the combination problems include 8-Queens, 8-rooks, 14-bishops and 32-knights puzzles.

EAs and EDAs are usually ineffective in solving globally multimodal problems as they converge to a single global optimum. The explanation [126] is straight forward. *The basins of different global optima may be represented in the population. As there is no significant selective preference for one of the basins in the population over another, the stochastic variations due to the selection method make the population drift towards one of them and, thus, discover only one global optimum at most. Moreover, this global optimum is randomly chosen from the existing global optima.* This phenomenon is known as *genetic drift* [127][7]. In absence of selective pressure, the stochastic nature of the selection method reduces population diversity.

5.2 Magic Square

5.2.1 Introduction

A magic square of order n is an arrangement of $n \times n$ distinct integers in a square, such that the n numbers in all rows, all column and both diagonals sum to the same constant or magic sum M. which can be calculate by the equation

$$M = \frac{n(n^2 + 1)}{2}$$

For normal magic square of order n = 3, 4, 5 the magic constants are 15, 34, 65 respectively

2	7	6
9	5	1
4	3	8

(a)

12	6	15	1
13	3	10	8
2	16	5	11
7	9	4	14

(b)

Figure 5.1 Sample of magic square solutions
(a) The solution of a 3×3 magic square
(b) The solution of a 4×4 magic square

5.2.2 Related works

Apart from the exact methods, there is no known algorithm to find the solution to magic squares. According to [128] and [129], genetic algorithm with specific crossover cannot always find a solution to these problems. In the works of [128][129], they relax the constraints of the magic squares such that the fitness evaluation are calculated only from the row and the column, while, the works of [130] use multimillion of runs to find a solution and measure the relative error compared to the optimal solution.

Surprisingly, magic square problems are composed with the set of numbers with summand equal to the magic numbers in each axis, thus they are considered to contain building blocks in the genotype. However, constructive algorithms such as genetic algorithms can rarely be able to find such a solution. The competitive algorithms to solve magic square problems become the improvement algorithms that usually are trapped in some local optima.

2	7	6
9	5	8
4	1	3

1	3	4
8	5	9
6	7	2

Figure 5.2 The magic squares that contain the conflict building blocks

The simple explanation is that even if a constructive algorithm can recognize that which squares are better than the others in term of fitness and determine that they might contain the good building blocks which should lead to a better solution, however, the algorithm cannot be able to recognize that which building blocks are good and which ones are not. In addition, even if the algorithm can recognize the good building blocks contained in a candidate solution, the algorithm does not know how to compose them as they might be being in conflict.

Figure 5.2 exemplifies two of the fittest solutions in which none of the crossover operators [50][51][52][53][54][55][56][57][58]can recombine to form a better solution. Moreover, the recombining these two solutions results in generating the worse offspring compared to their parents due to the crossover operator would rather disrupt the building blocks than constructing the higher order, better ones.

In a 3×3 magic square problem, each axis composed of the summands equal to 15. However, not all the summands are the good building blocks. For example, the summands M = 15 can be 1+5+9, 2+4+9, 2+6+7, 3+5+7, 1+6+8, 2+5+8, 3+4+8, 4+5+6 and their permutations, but the subsequence 5+1+9 is not a subcomponent of any solution. We call this kind of subsequence a false building block. Composing a false building block, might mislead the search algorithm into a pitfall. This reason makes the problems become much harder to be solved.

5.2.3 Experimental setup

We compare EHBSA and COIN in 3×3 and 4×4 magic square problems. Both EHBSA and COIN were implemented using Codegear Delphi 2007 and test them using Intel core 2 duo 2.16 GHz with 2 GB of RAM. Ten runs ware performed for each problem, we apply different configurations of population size and number of generation such that the total numbers of evaluation are smaller than the solution/space ratios. For the 3×3 magic square we use to 5,000 evaluations while in the 4×4 magic square we use up to 800,000 evaluations. The bias ratio B_{ratio} of EHBSA is 0.005 were used in all experiments while the learning rate *k* of COIN is set to be 0.05. The selection pressure of EHBSA is 50% of the whole population, while COIN uses 25% for both reward and punishment.

We evaluate the algorithms by measuring their ANE (average number of evaluations to find the first global optimum) #SOL (average number of solution found within the given number of evaluations) and #DSOL (average number of distinct solution found within the given number of evaluations)

Figure 5.3 illustrates the encoding of a 3×3 magic square. The candidate simply formed by concatenation of each of the row. This problem is a problem that the building blocks are explicitly exposed. Apart from the rows, each magic square clearly contains the building blocks for each column and diagonal axes.

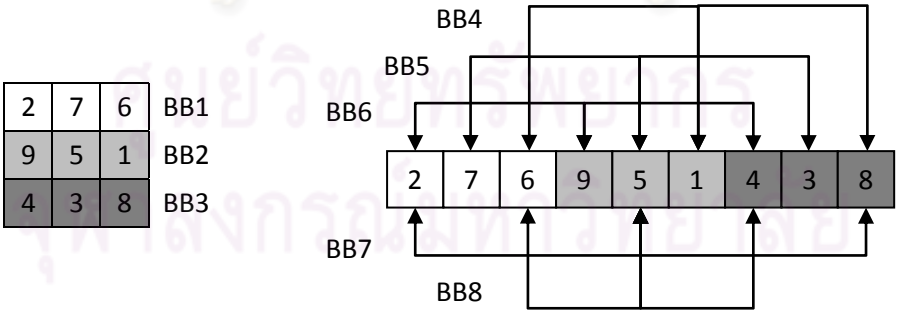


Figure 5.3 Encoding and building blocks (BB) of a 3×3 Magic Square problem

5.2.4 Discussion

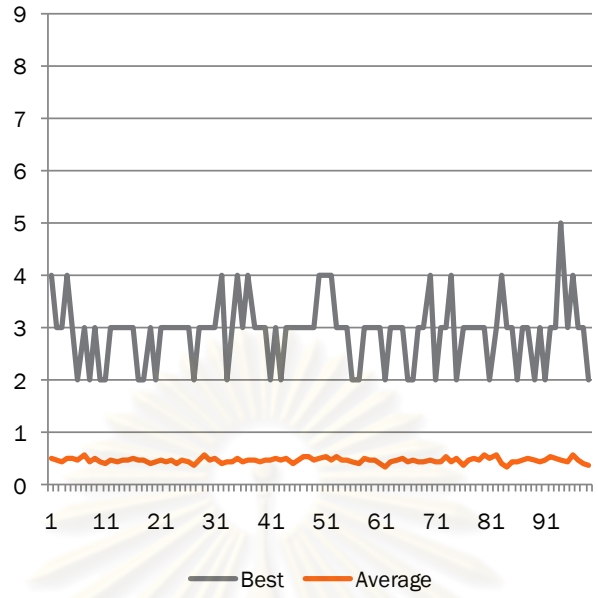


Figure 5.4 Performance of EHBSA in 3×3 magic square problem

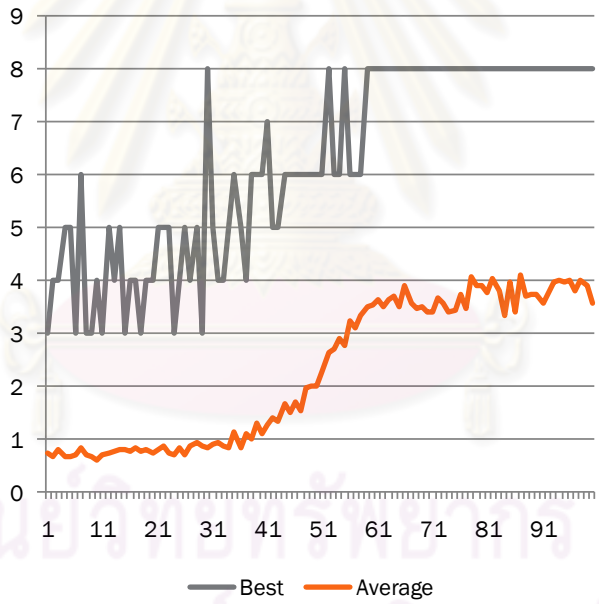


Figure 5.5 Performance of COIN in 3×3 magic square problem

Figure 5.4 and 5.5 show the result of EHBSA and COIN in 3×3 magic square problem respectively. The vertical axes indicate the fitness of the benchmarks while the horizontal axes indicate the number of generation used by the algorithms. The magic squares are the maximization problems, thus the larger number indicates the better performance. In these experiments, COIN performs better than

EHBSA in magic square problems, as COIN can find a solution to the 3×3 magic square, while EHBSA cannot converge to find such a solution. COIN learns the negative correlation of the bad building blocks contained in the bad solutions. COIN prevents the composition of these building blocks, thus easier to converge closer to the optimal solutions. For example, in the 3×3 magic square problem, The total number of 3 combination summand is $\binom{9}{3} = 84 \times 3!$. the summands of magic number $M = 15$ are the permutation of $1+5+9, 2+4+9, 2+6+7, 3+5+7, 1+6+8, 2+5+8, 3+4+8$ and $4+5+6$ altogether $= 8 \times 3!$, otherwise the summand are infeasible. (consequently, there are only 24 good valid summand out of 504 summand which are considered to be the good building blocks). EHBSA cannot determine whether which summand belongs to which model. It only tries to compose a solution from the good found build blocks, while COIN prevents the composition of 480 bad building blocks. However, COIN rarely finds the solution of the 4x4 magic square. The probability to find a solution is 0.5, otherwise got stuck in a local optima.

Magic squares are also considered to be a hard deceptive problem where the canonical magic squares can be generated all the other by rotation and transposition or reflection. The difficult part of this problem is that there are no overlapping relative building blocks between each solution, thus, there is no significant selective preference for one of the basins in the population over another. So the algorithms don't know which direction they should converge to. The figure 5.6 reveals all the possible solutions to the 3×3 magic square.

2	7	6
9	5	1
4	3	8

6	1	8
7	5	3
2	9	4

8	3	4
1	5	9
6	7	2

4	9	2
3	5	7
8	1	6

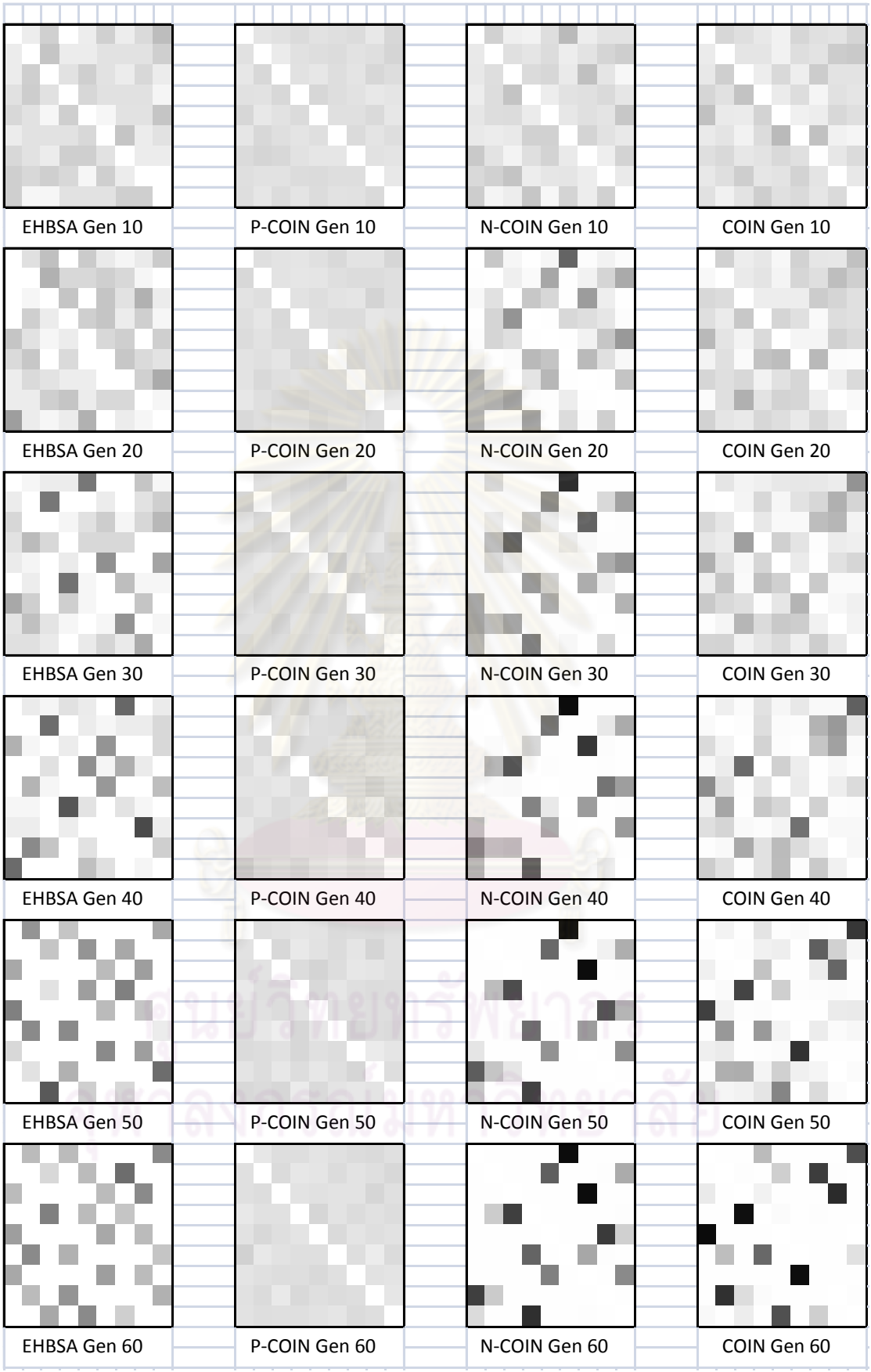
4	3	8
9	5	1
2	7	6

2	9	4
7	5	3
6	1	8

6	7	2
1	5	9
8	3	4

8	1	6
3	5	7
4	9	2

Figure 5.6 All of the 3×3 magic square solutions


EHBSA Gen 10
P-COIN Gen 10
N-COIN Gen 10
COIN Gen 10
EHBSA Gen 20
P-COIN Gen 20
N-COIN Gen 20
COIN Gen 20
EHBSA Gen 30
P-COIN Gen 30
N-COIN Gen 30
COIN Gen 30
EHBSA Gen 40
P-COIN Gen 40
N-COIN Gen 40
COIN Gen 40
EHBSA Gen 50
P-COIN Gen 50
N-COIN Gen 50
COIN Gen 50
EHBSA Gen 60
P-COIN Gen 60
N-COIN Gen 60
COIN Gen 60

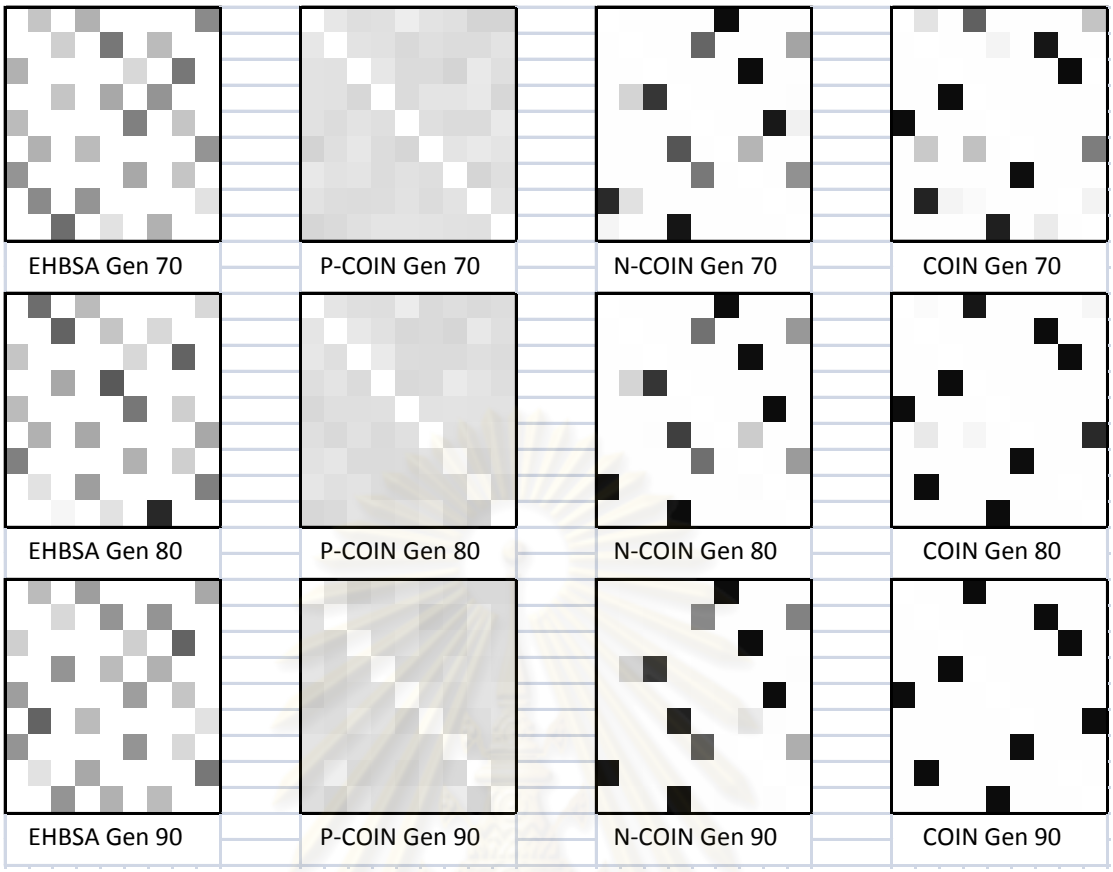


Figure 5.7 Generator snapshots of EHBSA, positive COIN, negative COIN and COIN for the 3×3 magic square problem

Figure 5.7 shows the generator snapshots of EHBSA and COINs for the 3×3 magic square problem. In this experiment, we analyze the effects of negative correlation learning of COIN by comparing EHBSA and the three versions of COIN including the positive correlation learning COIN (P-COIN), the negative correlation learning COIN (N-COIN) and COIN that combines the positive and negative correlation learning altogether. The lighter blocks indicate the lower probabilities while the darker blocks indicate the higher probabilities. The probability ranges from white to black as 0 to 1.

Since the 3×3 magic square is a multimodal problem where the building blocks are rarely shared, EHBSA and P-COIN cannot converge to a single solution. However, EHBSA and P-COIN are different to each other as EHBSA estimates the probabilistic model from the current population only, while P-COIN incrementally combines the results learned from the current generation with all the

previous generations. N-COIN seems to be smarter in exploiting the search space. The negative correlation learning contributes in converging by exploiting the common bad substructures shared by the below average solutions. The combination of positive and negative correlation learning enhances COIN to converge fastest compared to the others.

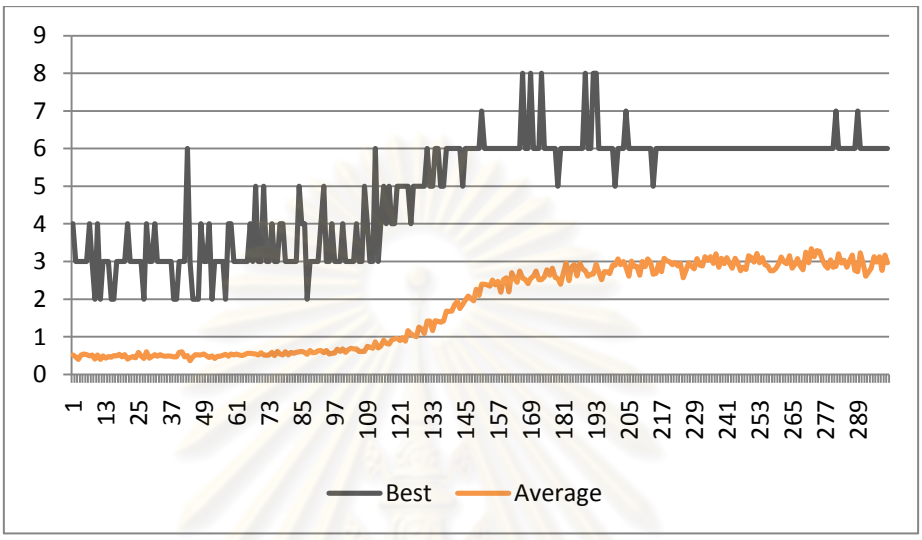


Figure 5.8 Performance of N-COIN in 3×3 Magic Square problem

Figure 5.8 illustrate the behavior of N-COIN in the 3×3 magic square problem. The negative correlation learning can make the overall solution converge towards the optimality. However, after the whole population is dominated by some substructure, the negative correlation learning considers the algorithm to be stuck in a local optima and then try to unlearn the edges containing in the below average solutions in order to jump off the local optima.

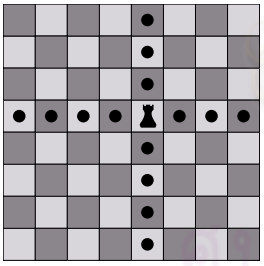
5.3 Combination Chess Puzzle

5.3.1 Introduction

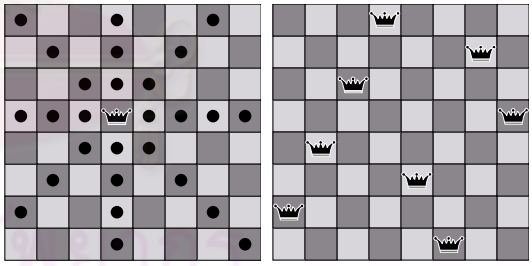
The *n*-pieces chess puzzles are the problem of placing *n* kind of the same class chess pieces on an 8×8 chessboard so that none of them can capture any other using their moves. A standard 8×8 chessboard can place maximum number of 8 queens, or 8 rooks, 14 bishops or 32 knights so that none of them can attack each other. These *n*-pieces chess puzzles are considered to be the combination problems which are formally defined as "to find an optimal *n*-combination of a set *S* is a subset of *n* distinct elements of *S*". These combination problems are more difficult than selections problems because the feasible set of solutions need to be a fixed size of *k*

distinct items which standard mutation and crossover operators in genetic algorithms could produce infeasible solutions even the crossover operators for permutation representation is used. For 8-queens puzzle, the problem can be both combination and permutation as it is possible to reduce the number of possibilities by generating the permutations that are solutions of the eight rooks puzzle and then checking for diagonal attacks further reduces the possibilities from 4,426,165,368 or $\binom{64}{8}$to just 40,320

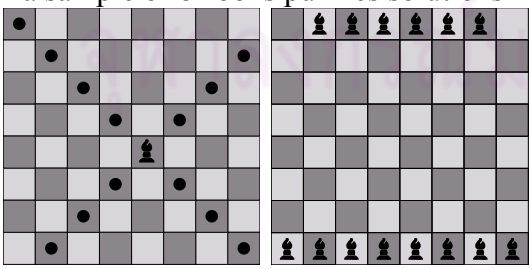
Figure 5.9 shows the available chess moves of the dedicated problems and a sample solution to each problem as follows: (a) shows the available rook's move and a sample of 8-rooks puzzles solutions. (b) shows the available bishop's move and a sample of 14-bishops puzzles solutions. (c) shows the available queesn's move and a sample of 8-queens puzzles solutions. (d) shows the available knight's move and a sample of 32-knights puzzles solutions. The rook can move any number of squares along any rank or file while the bishop can move any number of squares diagonally. The queen combines the power of the rook and bishop and can move any number of squares along rank, file, or diagonal. The knight moves to any of the closest squares that are not on the same rank, file, or diagonal, thus the move forms an "L"-shape two squares long and one square wide.



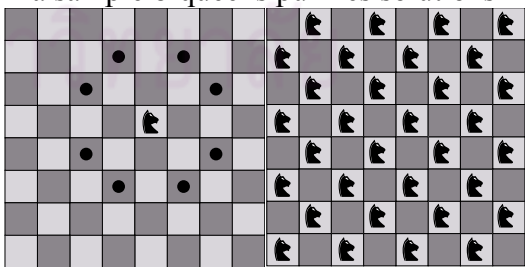
(a) Available rook's move and a sample of 8-rooks puzzles solutions



(c) Available queen's move and a sample 8-queens puzzles solutions



(b) Available bishop's move and a sample 14-bishops puzzles solutions



(d) Available knight's move and a sample 32-knights puzzles solutions

Figure 5.9. Available chess move and sample solutions of combination problems.

According to [131], The 8-Queens puzzle has totally 92 distinct solutions, the 8-Rooks puzzle has up to 40,320 distinct solutions, the 14-Bishops

puzzle has totally 8 distinct solutions and the 32-Knights puzzle has only 2 distinct solutions

5.3.2 Related works

None of the existing literature is found to solve such combination chess puzzle problems. However, *n*-queen problems can be found in many literatures including [132][133] and [134], however, the multimodality of the solutions to the *n*-queen have not been tested in any of them.

5.3.3 Experimental setup

The experiments settings of the combination chess puzzles are similar to that in the experiments setting of magic square problems. However, the only difference is that the encoding of the solution strings. Figure 5.8 presents the encoding of a 8-queen solution. A candidate solution compose of items correspond to the positions label in the chess board. The candidate 4-15-19-32-34-45-49-62 represents the solution in the figure 5.10.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

4	15	19	32	34	45	49	62

Figure 5.10 The sample encoding of a combination 8-queen solution

In these test sets, we also include the permutation version of 8-Queens benchmark. The encoding is shown in the figure 5.11.

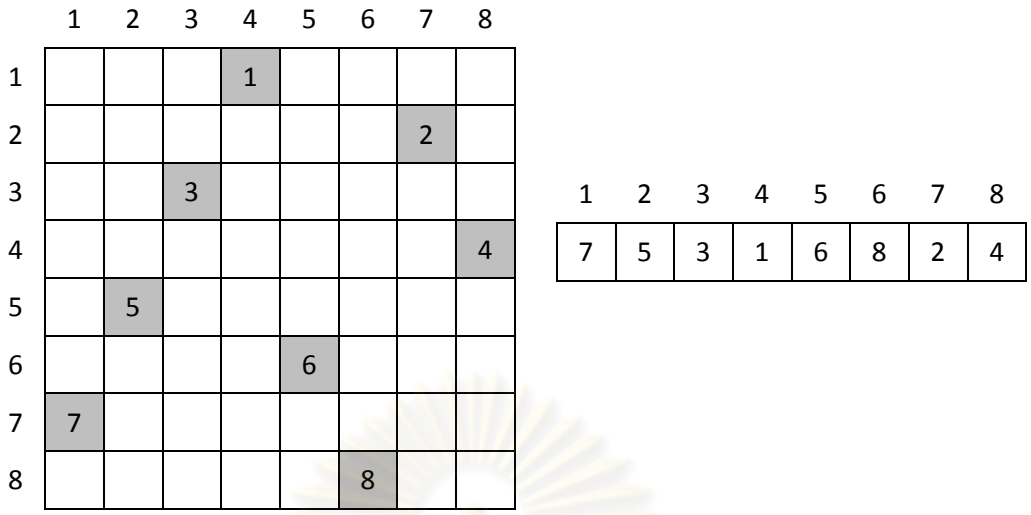


Figure 5.11 The sample encoding of a permutation 8-queen solution

We also evaluate the algorithms by measuring their ANE (average number of evaluations to find the first global optimum) #SOL (average number of solution found within the given number of evaluations) and #DSOL (average number of distinct solution found within the given number of evaluations)

5.3.4 Discussion

Figure 5.12 through Figure 5.21 show the results of EHBSA and COIN in 8 permutation Queens (8-Queens-P), 8-Rooks, 8 combinations Queens (8-Queens-C), 14-Bishops and 32-Knights Respectively. The vertical axes indicate the fitness of the benchmarks while the horizontal axes indicate the number of generation used by the algorithms. These benchmarks are the minimization problems, thus the smaller number indicates the better performance.

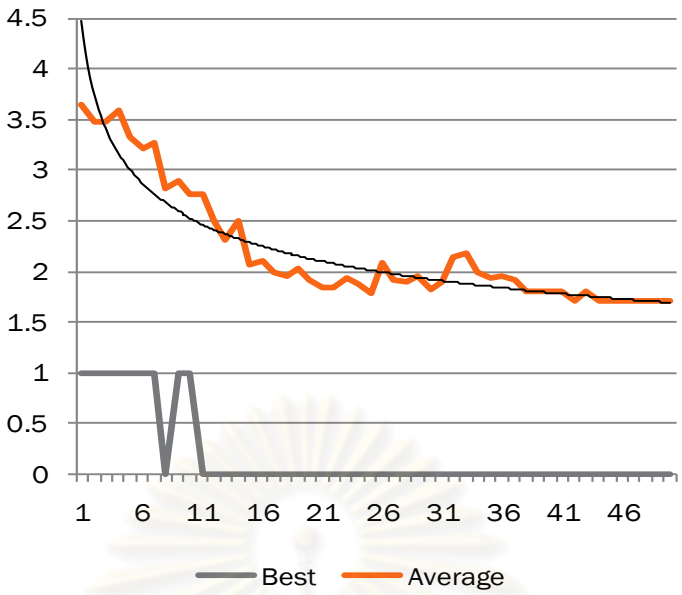


Figure 5.12 Performance of EHBSA in 8-Queens-P problem

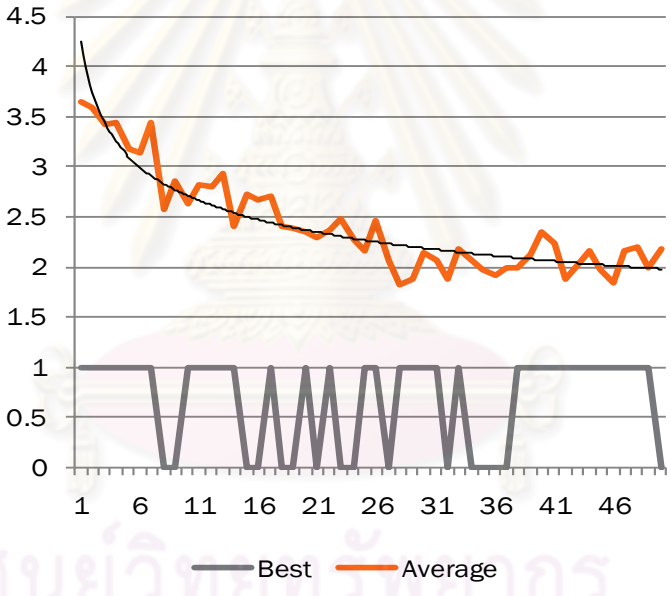


Figure 5.13 Performance of COIN in 8-Queens-P problem

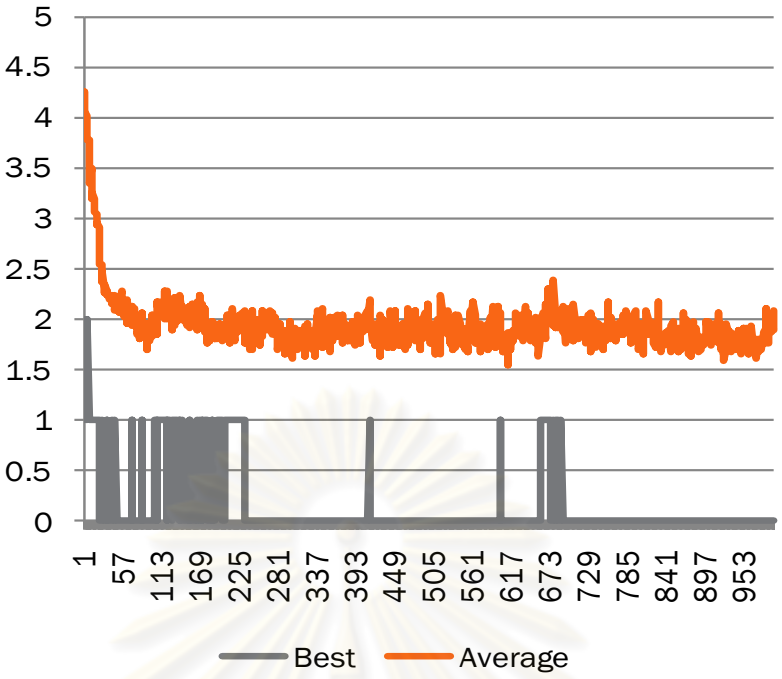


Figure 5.14. Performance of EHBSA in 8-Rooks problem

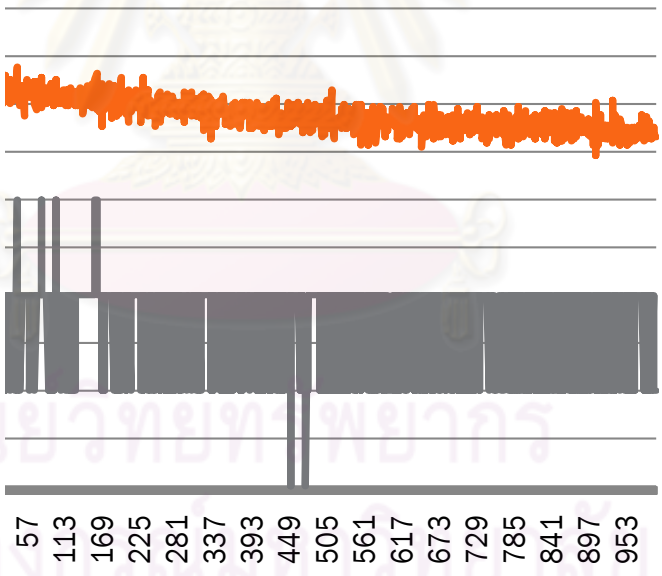


Figure 5.15 Performance of COIN in 8-Rooks problem

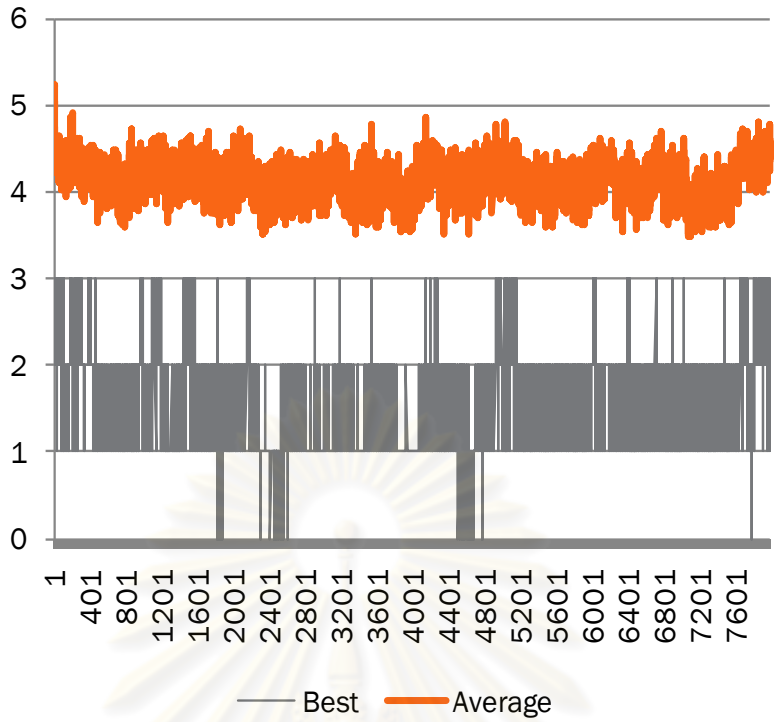


Figure 5.16. Performance of EHBSA in 8-Queens-C problem

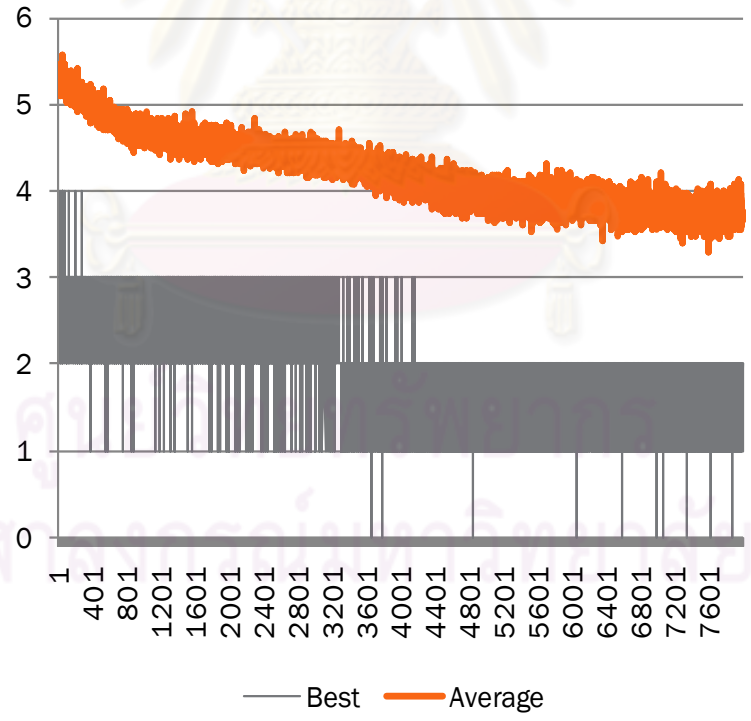


Figure 5.17 Performance of COIN in 8-Queens-C problem

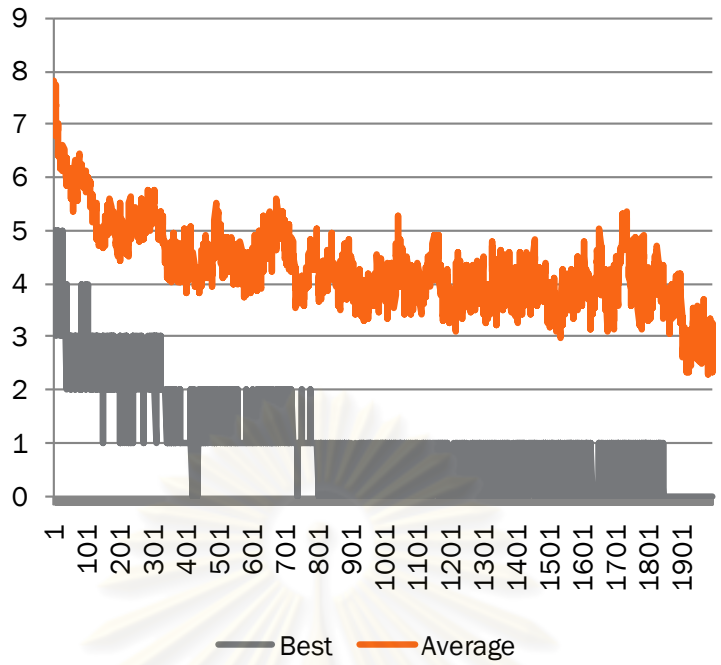


Figure 5.18 Performance of EHBSA in 14-Bishops problem

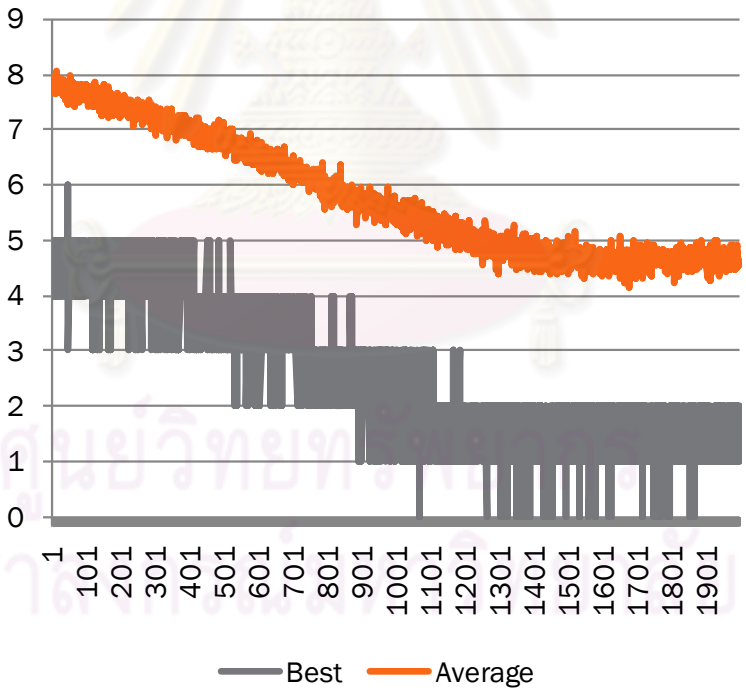


Figure 5.19 Performance of COIN in 14-Bishops problem

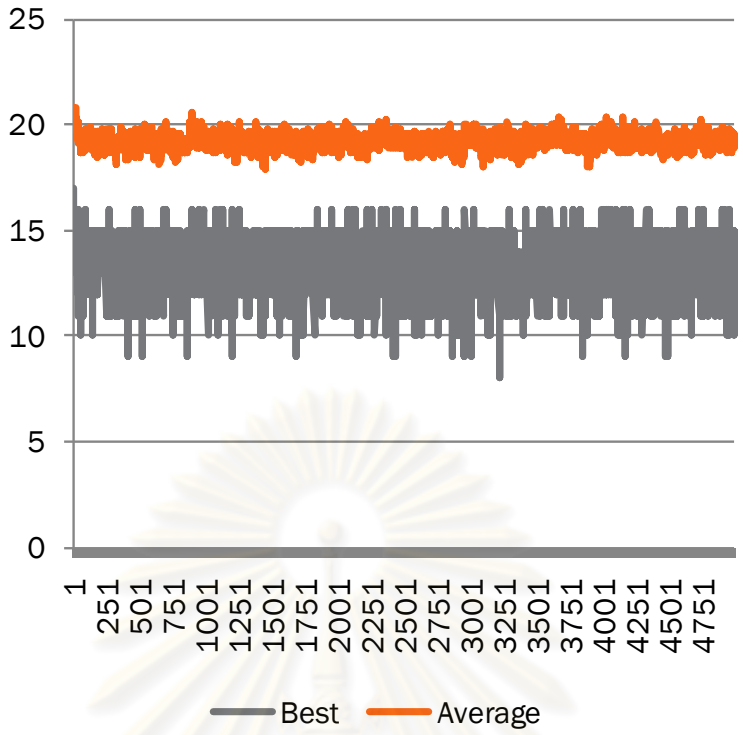


Figure 5.20 Performance of EHBSA in 32-Knights problem

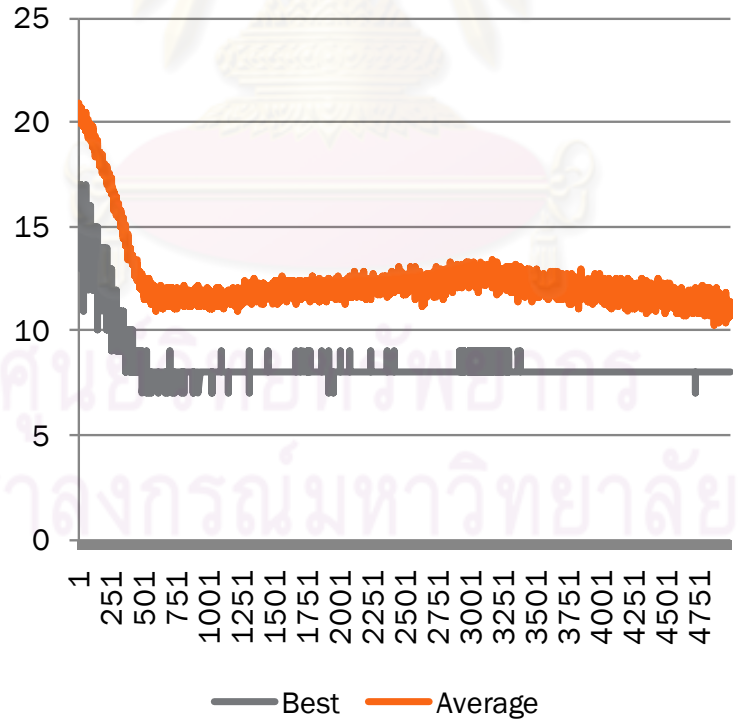


Figure 5.21 Performance of COIN in 32-Knights problem

From the overall perspective, EHBSA seems to outperform COIN in the combination problems as EHBSA can converge to the solution faster than COIN. However, COIN can generate more diverse solution compared to EHBSA.

In 8 Queens-P problems, COIN can find up to 13 distinct solutions while EHBSA can find only 4 distinct solutions. 8-Queens-C and 8-Rooks are the problems with equal feasible solution space. However, in 8-Queens-C problem EHBSA can find an average of 4 distinct solutions while COIN can find up to 9 distinct solutions, however, the average numbers of distinct solutions are equal. While in 8-Rooks problem EHBSA can find more distinct solutions than COIN.

In 14-Bishops problem, COIN can find all the 8 distinct solutions while EHBSA can find only 4 distinct solutions. EHBSA speedily converge to an optima point, while COIN tries to maintain all of the possible good substructures in order to compose them.

The 32-Knights problem is the hardest problem. It is considered to be a deceptive problem. In this problem, there are only two patterns of solutions, where either black or white checkers are all filled. Thus, this implies that there are no overlapping building blocks existing. None of the dedicated algorithm can solve this problem. According to the figure 5.20 and 5.21, COIN can converge closer to the global optimal solution, unfortunately, got stuck in some local optima, where the black and the white checkers are equally filled while EHBSA cannot improve the average population at all. The explanation is that COIN tries to generate a compromise model by filter the bad substructures from the goods. We try to bias the algorithm such that both EHBSA and COIN always select the top black corner as a starting position. And it results that EHBSA always find an optimal solution within a hundred generation, while COIN gives the same compromising results shown in figure 5.22.

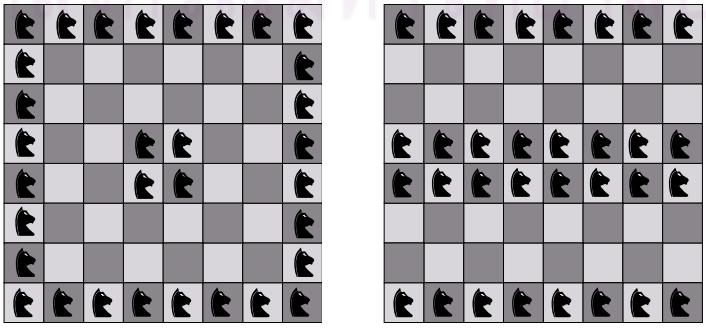
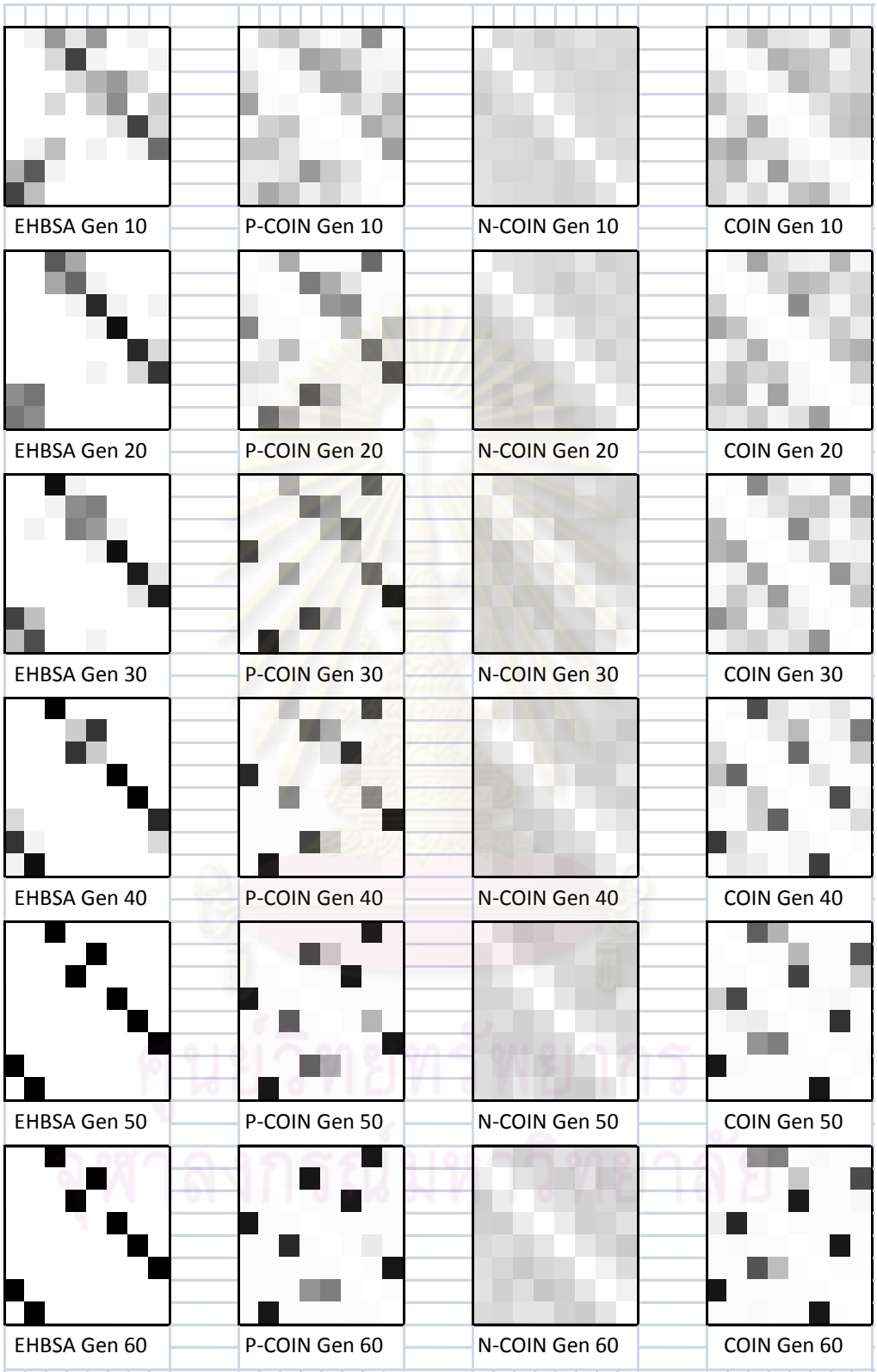


Figure 5.22 Two compromising 32-Knight solutions obtained from COIN


EHBSA Gen 10
P-COIN Gen 10
N-COIN Gen 10
COIN Gen 10
EHBSA Gen 20
P-COIN Gen 20
N-COIN Gen 20
COIN Gen 20
EHBSA Gen 30
P-COIN Gen 30
N-COIN Gen 30
COIN Gen 30
EHBSA Gen 40
P-COIN Gen 40
N-COIN Gen 40
COIN Gen 40
EHBSA Gen 50
P-COIN Gen 50
N-COIN Gen 50
COIN Gen 50
EHBSA Gen 60
P-COIN Gen 60
N-COIN Gen 60
COIN Gen 60

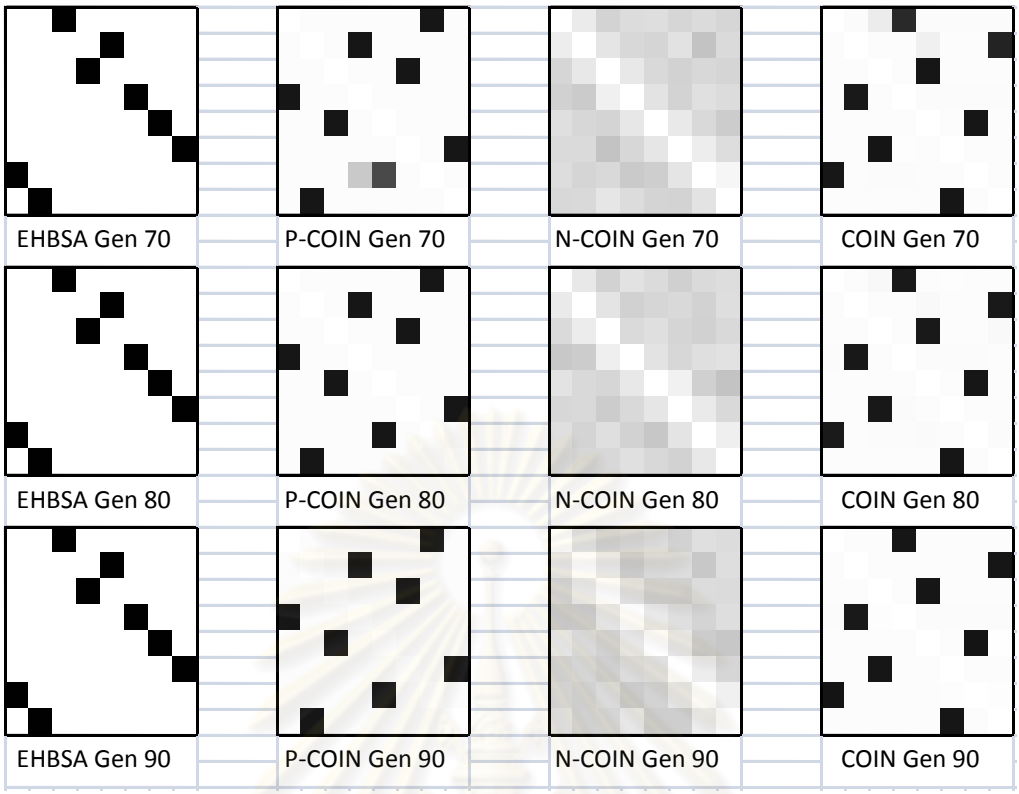


Figure 5.23 Generator snapshots of EHBSA, P-COIN, N-COIN and COIN for the 8 Queens-P problem

Figure 5.23 shows generator snapshots of EHBSA, P-COIN, N-COIN and COIN for the 8 Queens-P problem. These generators do not represent the graphical phenotype of the solutions. In order to generate a solution, we need to sample from the generator. For example, if we begin with the node 3 and then we perform the greedy search within the final generator of COIN, we will get a solution 3-5-7-1-4-2-8-6 which is a solution to the 8-Queens-P. If we begin with the node 4, we will either obtain a solution 4-2-8-5-7-1-3-6 or a solution 4-2-8-5-7-1-6-3. The first case is also a solution to the 8-Queens-P as well while the latter case is not.

In contrast to the 3×3 magic square problem, EHBSA and P-COIN converge to a solution faster than N-COIN. However, EHBSA and P-COIN can find less distinct solutions compared to the COIN with the negative correlation learning embedded. In this problem, the negative correlation learning seems not to be converging at all. However, when combining with the positive correlation learning, it contributes in producing more diverse distinct solutions as it try to preserve all the possible good substructures found in the better population.

5.4 Knight's Tour

5.4.1 Introduction

Knight's Tour is a well-known classical chess puzzle which has been studied over the last thousand years. The objective of the problem is to find a Hamiltonian path in a graph defined by the legal move of a knight on a chess board in which the chess knight has to traverse each square exactly once. Moreover, there are superior solutions called closed tours which are the solutions that the knight can have an extra move to complete the circuit at the starting square. The closed tours are more difficult to find. Murray [16] trace the origin of the problem as the first manuscript written in Arabic text was introduced by Ali C. Mani in 1350. It described the first closed by Ar-Rumi in Baghdad in 840. Later in 1766, Euler [17] proposed the first mathematical analysis of the problem. Other well-known mathematicians who work on the problem include Taylor, de Moivre and Lagrance.

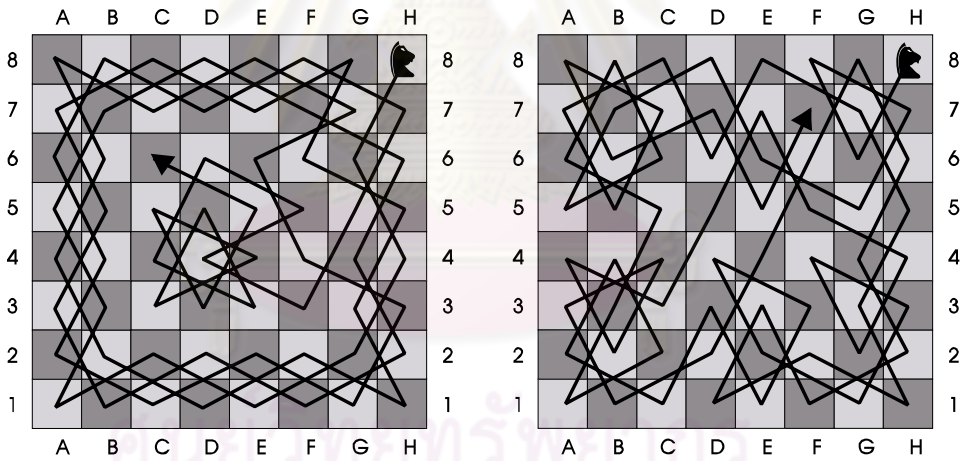


Figure 5.24 Two of the earliest known knight's tour solutions
Left is the solution by Ali C. Mani.
Right is the first closed tour solved by Ar-Rumi in 840A.D.

5.4.2 Related works

Recently, there are many works on solving the knight's tour problem on a standard 8x8 chessboard. Borrell [135] proposed a straight forward depth first search with no bias or heuristic using a Prolog language to develop a brute force algorithm. The work aims to speed up the search time.

In contrast to the exact algorithm, there are metaheuristics approaches try to overcome the problem including Ant Colonization Optimization (ACO) by Hingston and Kendall [136] which augmented the problem specific heuristic [137] in to their algorithms in order to increase the chance to find the solutions. Genetic Algorithm (GA) by Gordon and Slocum [138] and Jarfar Al-Gharaibeh Zakariya Qawagneh and Hiba Al-Zahawi[139] utilized the repair operation and heuristics augmented respectively. The advantage of using the problem specific heuristic is that the search space is reduced down from 63! or approximately $1.98x10^{87}$ solutions down to 8^{65} or approximately $5x10^{58}$ solutions.

Another approach to the knight's tour was proposed by Takefuji and Lee [140]. They utilized a Neural Network to solve the knight's tour problem on a large 26x26 boards. However, standard approaches such as Euler and Warnsdorf [141] heuristics and divide and conquer approach proposed by Parburry [142] can easily find the knight's tours on the board as large as 78x78.

The total number of solutions to the knight's tours problems was researched by Wegener [143] , Mckay [144] and Mordecki [145] using a very large scale computers. On a standard 8x8 chess board, McKay calculated the total number of closed tours to be 13,267,364,410,532 while Mordecki found the open tours to be approximately $1.305x10^{35}$.

5.4.3 Experimental setup

In this section, we report the experiment result of COIN in the knight's tour problem. We conduct the experiment using two algorithms which is EHBSA/WO and COIN and then compare the result with ACO [138] GA with repair [140] and GA with heuristic [141] using the results report in their literatures.

For testing purpose, In order to compare the performance of COIN and EHBSA, We chose the EHBSA/WO which is the standard version of EHBSA in order to contrast the result of edge based sampling algorithm with and without a negative learning. We set the parameters of COIN and EHBSA to the following values as:-

Population size = 400
Number of generation = 800
COIN Learning rate = 0.05
Reward selection pressure = 0.25
Punishment selection pressure = 0.25
EHBSA selection pressure = 0.5

Both COIN and EHBSA were implemented with CodeGear Delphi 7, the testing environment is MS Windows XP on a 2.4 GHz Intel core 2 duo, with 2 GB RAM.

The encoding scheme of both COIN and EHBSA is a straight forward permutation string, where each of the items refers to the position of the chess board ranging from the top left toward right as 1 to 8 and then repeat to the next row until 64 at the bottom right. The evaluation function of our approach is the number of legal move found in a tour.

5.4.4 Discussion

Figure 5.25 compares the performance of COIN and EHBSA. The two red top lines are the result of maximum and average tour generated by COIN, which can converge to the first open tour at the generation 150. Then more of the complete tours are rapidly generated until the first closed tour is found at the generation 301.

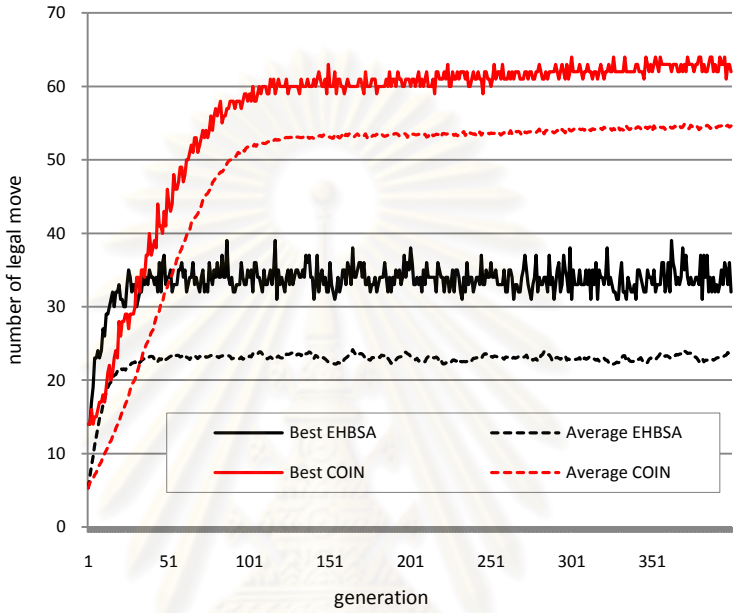


Figure 5.25 Comparison of the performance of COIN vs. EHBSA in the knight's tour problem

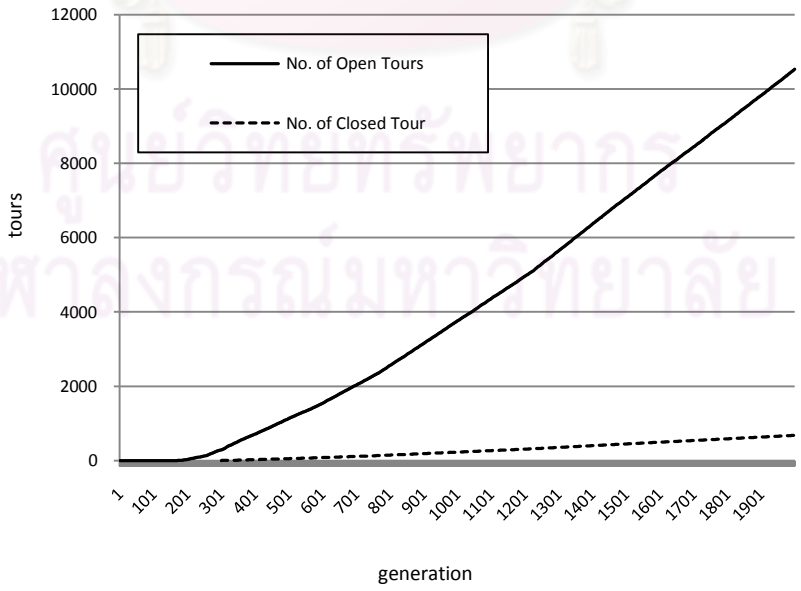


Figure 5.26 Average performance of COIN in the knight's tour problem

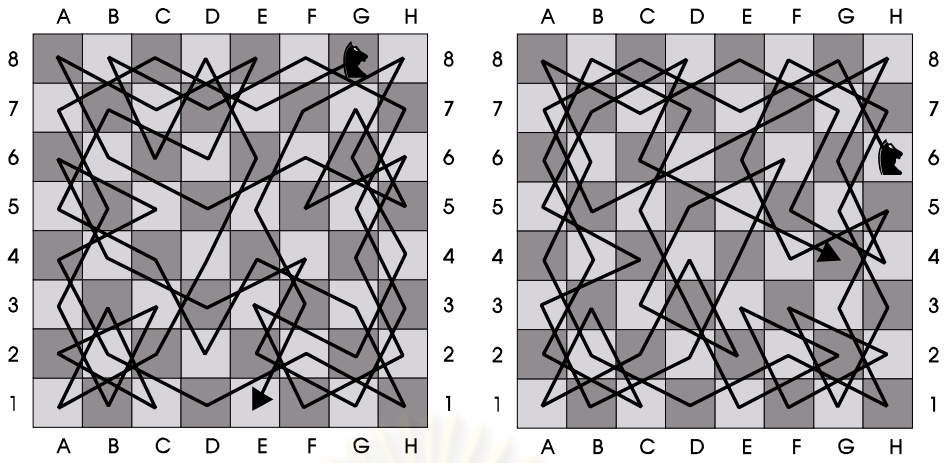
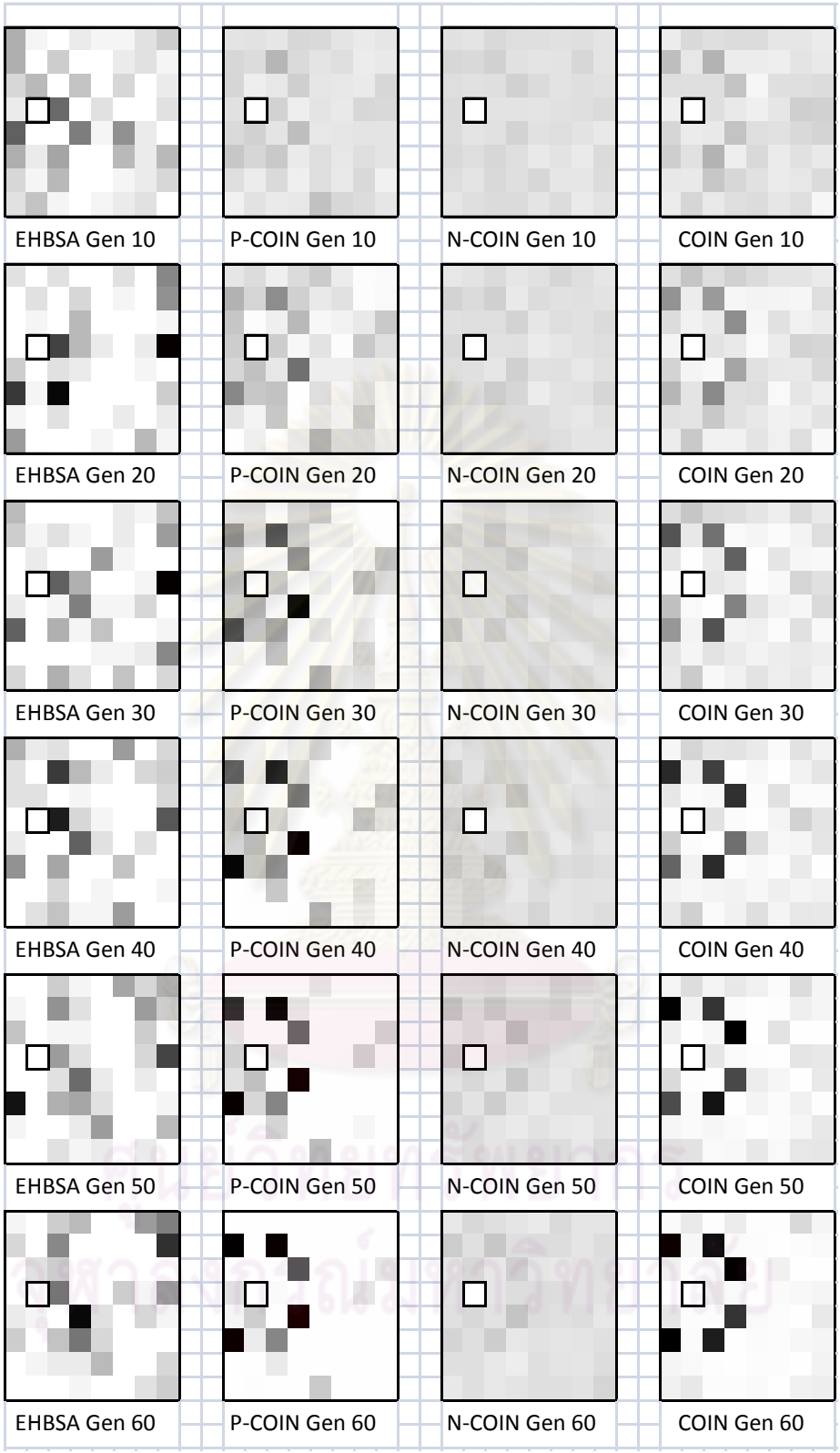


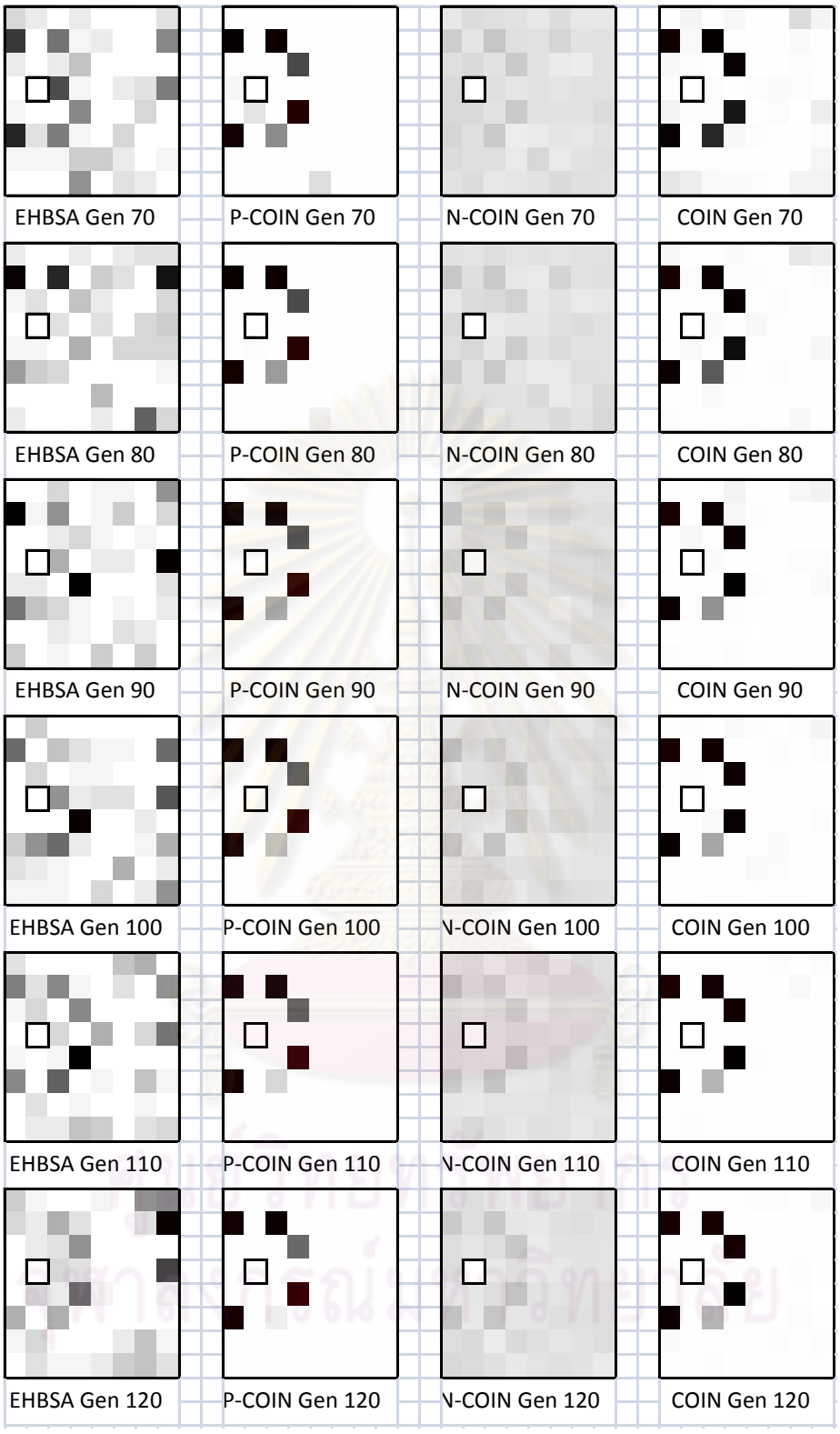
Figure 5.27 Two of the solutions generated by the coincidence algorithm.
Left is the first open-tour found in the generation 150.
Right is the first closed-tour found in the generation 301.

Due to [18] and [146], given a same number of function evaluation, EHBSAs are better than COINs in TSP benchmarks, however in the knight's tour puzzle which is the globally multimodal problem, EHBSA cannot converge to a single optimal tour. The explanation is that in this benchmark, there are substructures in the population in which there is no significant selective preference to differentiate which one is better. EHBSA cannot even make the population drift towards to a single peak.

While EHBSA got stuck in local optima, COIN has an ability to learn the negative knowledge contained in the undesired solutions. In a complete graph, a knight can have at most 8 legal moves while the rest 56 moves are prohibits. Learning the prohibit moves is easier than learning the legal moves as there are the greater number of the prohibit moves.

Negative correlation learning of COIN plays two major roles in this benchmark. First, it helps recognizing and eliminating the illegal knight's path from the complete graph, which leads to the diversity amongst the legal path. Second, once the probability matrix converges, it unlearns some of the occurrence in the previous generation in order to find more of the solution models.


EHBSA Gen 10
P-COIN Gen 10
N-COIN Gen 10
COIN Gen 10
EHBSA Gen 20
P-COIN Gen 20
N-COIN Gen 20
COIN Gen 20
EHBSA Gen 30
P-COIN Gen 30
N-COIN Gen 30
COIN Gen 30
EHBSA Gen 40
P-COIN Gen 40
N-COIN Gen 40
COIN Gen 40
EHBSA Gen 50
P-COIN Gen 50
N-COIN Gen 50
COIN Gen 50
EHBSA Gen 60
P-COIN Gen 60
N-COIN Gen 60
COIN Gen 60


EHBSA Gen 70
P-COIN Gen 70
N-COIN Gen 70
COIN Gen 70
EHBSA Gen 80
P-COIN Gen 80
N-COIN Gen 80
COIN Gen 80
EHBSA Gen 90
P-COIN Gen 90
N-COIN Gen 90
COIN Gen 90
EHBSA Gen 100
P-COIN Gen 100
N-COIN Gen 100
COIN Gen 100
EHBSA Gen 110
P-COIN Gen 110
N-COIN Gen 110
COIN Gen 110
EHBSA Gen 120
P-COIN Gen 120
N-COIN Gen 120
COIN Gen 120

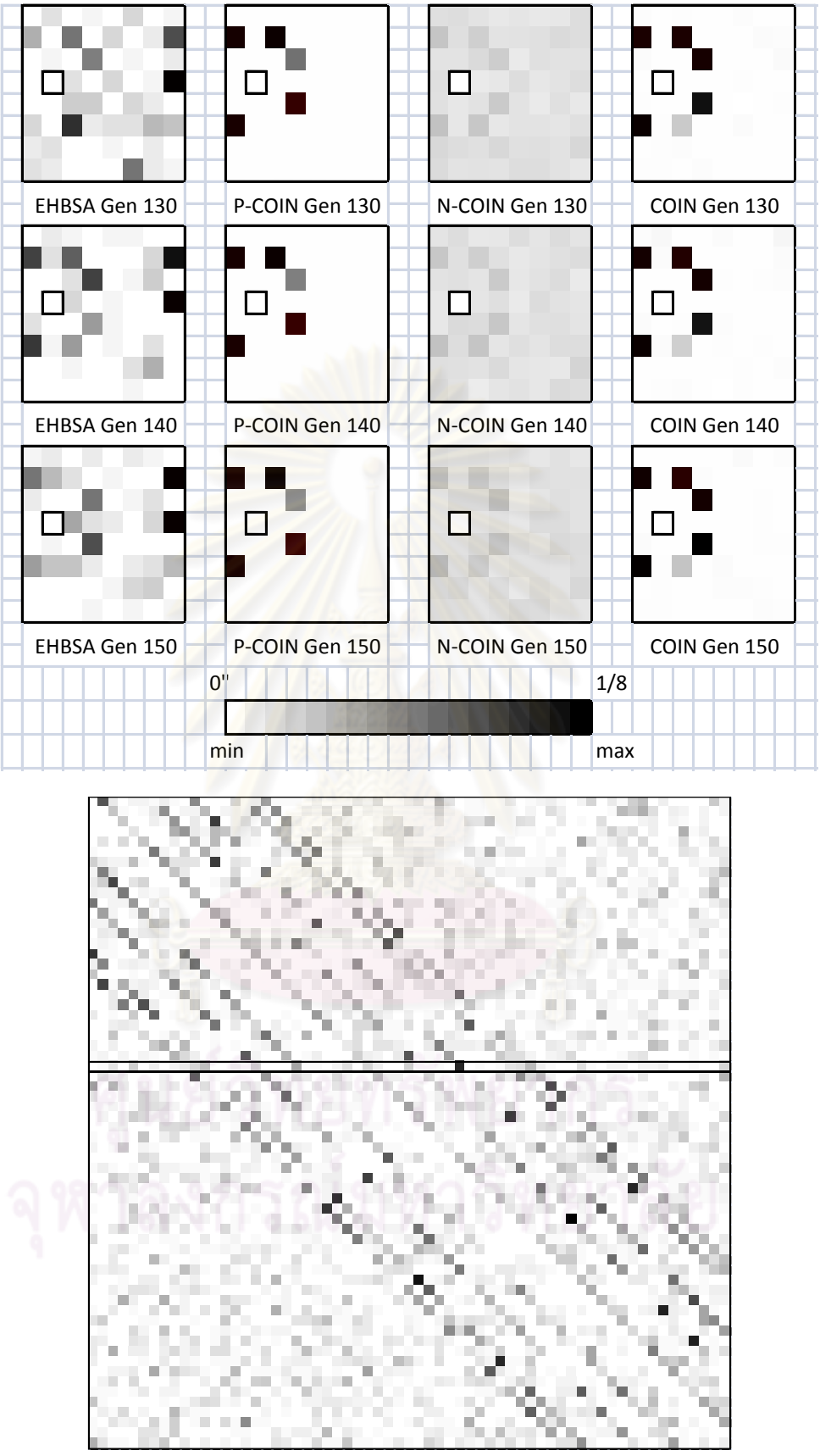


Figure 5.28 The 27th row generator snapshots of
EHBSA, P-COIN, N-COIN and COIN for the knight's tour problem

Figure 5.28 shows the generator snapshots of EHBSA, P-COIN, N-COIN and COIN for the knight's tour problem. Due to the enormous size of the generator, we sample only from the 27th row of the generator. Such row can be transform to the possible path of a knight from the coordinate 5B to all the other coordinates. In this experiment, we do not embed the problem specific heuristic in order to observe the behavior of the 4 algorithms. The probabilistic model of EHBSA cannot converge to a stable stage at all as it estimates the distribution from the latest generation, while P-COIN, N-COIN and COIN incremental learn from all the previous generations. The probabilistic models of P-COIN, N-COIN and COIN slowly adjust themselves toward a stable stage. Such stable stages show that the probability models try to constraint the probabilities of the possible moves to satisfy the knight's legal moves.

According to figure 5.29, compared to the figure 5.25, P-COIN performs almost as the same quality as COIN. Unfortunately, P-COIN cannot find an optimal solution. From the generator snapshot, P-COIN tries to converge to a single solution, while COIN tries to maintain all the possible solution. In this problem, the negative correlation learning contributes in preventing the premature convergence to a single model.

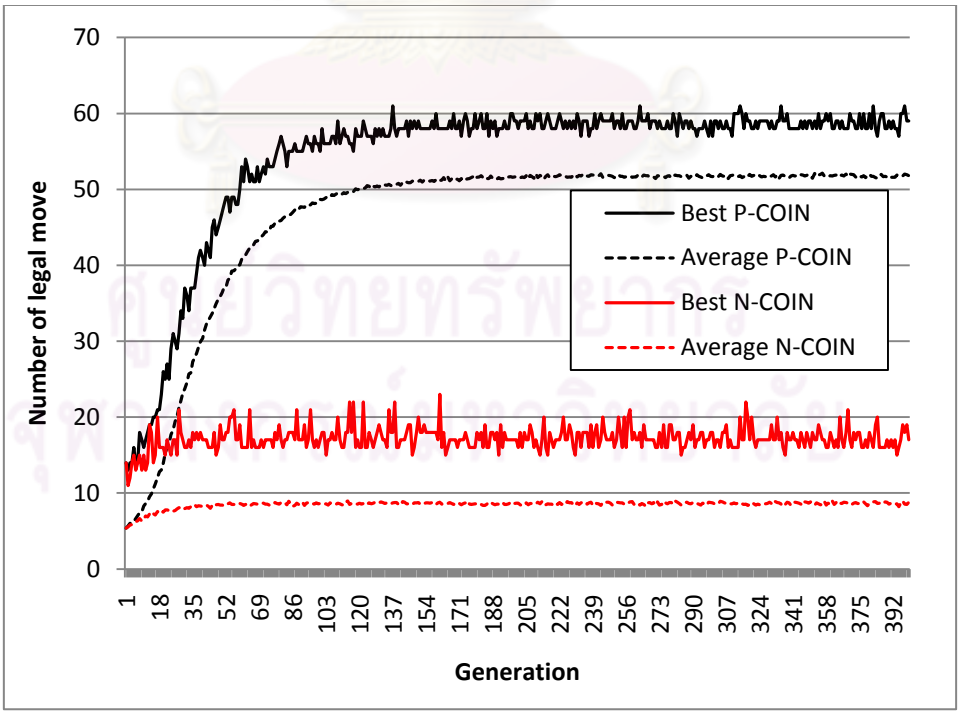


Figure 5.29 Comparison of the performance of
P-COIN vs. N-COIN in the knight's tour problem

According to [140] and [141], pure genetic algorithm was proven to fail to find solutions to the knight's tour problem. One of the reasons is there are too many global optimum which contain diverse of substructure. These substructures are not only hard to recognize but also are in conflict to each other. Even if all of the substructures are identified, the problem of how to compose these substructures remains. The probability matrix of COIN does not only learn to compose substructures in order to form a good solution, it also learns not to compose the substructures likely to form an undesired solution.

For method comparison, we increase the number of generation from the testing propose to 2,000 generations in order to compare the results with GA with heuristic and GA with repair operation proposed by Al-Ghraibah [141] and Slocum [142]. We also compare our work with an improved version of ACO called Multiple-restart Ant Colony Enumeration (MACE) algorithm proposed by Hingston [137] as well, however, the number evaluation used by Hingston is incredibly enormous therefore we prefer to mainly compare the result using the hit ratio obtained from the percentage of tour found from the invested function evaluation.

Table 5.1 Results of applying different approaches to solve Knight's tour.

Algorithms	Evaluations	Tours Found	Hit Ratio
MACE	172,800,000	13,124,464	7.5%
COIN	1,000,000	10,531	1.05%
GA+Repair	1,000,000	5,696	0.57%
Repair only	1,000,000	192	0.02%
GA+Heuristic	800,000	12,084	1.51%
Heuristics only	800,000	1,979	0.25%

From Table 5.1 the performance of MACE is superior. Within 172,800,000 evaluations, MACE has found over 13 millions tours in average as the ants quickly bias the prohibit moves by not laying the pheromones on it. Once a complete tour is found a population of ants performs local search over the shared information. While MACE evolves a tour from the sub-tours containing only the legal moves, COIN learns the whole permutation strings where both legal and illegal moves are mixed and trying to differentiate the legal and illegal moves among the whole populations. This is considered as a totally blind search. Without a problem specific heuristics or bias, COIN can find an average of 10,531 tours within a million function evaluations with 1.05% hit rate compared to 7.5% obtained from MACE. GA with heuristics also perform a great job as it improve the odds to find a solution of a pure heuristics from 0.25 up to 1.51%. GA with repair can improve the odds to find a solution from an iterated repair operation up to only 0.57%.

However, the numbers of closed tours are not mentioned in the literature. In average, COIN can find up to closed 921.4 tours out of 10,531 open tours.

5.4 Discussion

From the 3 sets of experiments, COIN rather gives more diverse solutions compared to EHBSA. However, it sacrifices more function evaluation as it trades of the convergence rate with the diversity. However, in 8-rooks puzzles, EHBSA can find more distinct solutions than COIN, while in 8-queens-C, COIN can find more distinct solutions than EHBSA. The explanation to this phenomenon is that the solution of the 8-rooks solutions share more common substructure than 8-Queens. The 8-Rooks solutions share at most 6 positions while the 8-Queens can share at most 2 positions. Once the probabilistic model of EHBSA converge to a solution, neighbors of such solution are likely to be a solution as well. While 8-Queens puzzle rarely shares the substructure. Moreover, the substructures of 8-Queens solutions are likely to conflict with each others, therefore EHBSA cannot converge to any direction.

5.6 Chapter Summary

This chapter, we investigate the roles of negative correlation learning of COIN compared to EHBSA where negative information is not taken into account. Table 5.2 summarizes all the benchmarks and their properties, while table 5.3 summarizes the performances of such benchmarks. The results conclude that negative correlation learning contributes in preserving diversity and preventing the premature convergence.

Table 5.2 Summary of test suites and their properties

Problem	Problem Type	Search Space	No. of Solution	Solution/Space Ratio	Population Size	Number of Generation	Number of Trial (Evaluation)
8 Queens-P	Permutation	40,320	92	1/438	50	50	2,500
8 Queens-C	Combination	4,426,165,368	92	1/48,110,493	100	1,000	1,000,000
8 Rooks	Combination	4,426,165,368	40,320	1/109,776	100	1,000	1,000,000
14 Bishops	Combination	47,855,6999,958,816	8	1/5,981,962,494,852	100	2,000	2,000,000
32 Knights	Combination	1,832,624,140,942,590,534	2	1/916,312,070,471,295,267	100	5,000	5,000,000
Knight's Tour	Permutation	1.268×10^{89}	1.3×10^{35}	$1/9.72 \times 10^{53}$	400	1,000	4,000,000
3x3 Magic Square	Permutation	326,880	8	1/40860	50	100	5,000
4x4 Magic Square	Permutation	20,922,789,888,000	7,040	1/2,971,987,200	100	800	800,000

Table 5.3 Performance of EHBSA vs. COIN in combinatorial puzzles

Problem	Algorithm					
	EHBSA			COIN		
	ANE	#SOL	#DSOL	ANE	#SOL	#DSOL
8 Queens-P	8	25	4	8	21	13
8 Queens-C	1821	78	4	3651	10	9
8 Rooks	25	2457	2293	454	4	4
14 Bishops	419	408	4	1070	45	8
32 Knights	N/A	0	0	N/A	0	0
Knight's Tour	N/A	0	0	154	2816	2759
3x3 Magic Square	N/A	0	0	35	40	2
4x4 Magic Square	N/A	0	0	N/A*	0	0

CHAPTER VI

REAL WORLD APPLICATIONS

6.1 Introduction

This chapter, we introduce the application of COIN in three major applications including travelling salesman problems (TSP), production line balancing problems and production line sequencing.

6.2 Travelling Salesperson Problem

6.2.1 Introduction

The traveling salesman problem (TSP) is a typical combinatorial optimization problem which is perhaps the most-studied NP-hard combinatorial problem. Given a list of cities and their pairwise distances, the common objective is to find a shortest possible tour that visits each city exactly once. The problem was first formulated as a mathematical problem in 1930 and is one of the most intensively studied problems in optimization. It is used as a benchmark for many optimization methods.

6.2.2 Related works

Even though the problem is computationally difficult, a large number of heuristics and exact methods are known, so that some instances with tens of thousands of cities can be solved. There are some near-optimal or approximate approaches to solve this problem, such as simulated annealing[147], neural networks[148], and tabu search[149]. Integer linear programming approach[150] is an exact algorithm to solve this problem by using additional linear constraints to eliminate the illegal sub-tours. Genetic algorithm is also purposed with the goal of solving the optimization problems, and has been applied to the TSP with varying degree of success.

6.2.3 Experimental setup

To measure the performance of COIN, we perform several benchmarks on single objective TSP problems and compare them to the experiment of Robles and Larrañaga [151]. We aim to measure the performance of the algorithm in two main aspects: quality of the results and the number of function evaluations. This research,

we show the result of the well known Gröstel24, Gröstel48 and Gröstel120 which can be obtained from the TSPLIB [152].

The experiments of Robles and Larrañaga use both of the discrete and continuous EDAs in the following learning methods: UMDA [73], MIMIC[74], TREE [154] and EBNA [155]. Moreover we compare the results with GA in the literature of Larrañaga [156] in 1999 which uses GENITOR [57] algorithm. The parameter of COIN in these experiments depends on the size of the problems. For each of the combinations shown in the experiment, we perform 10 runs and average the results.

To study the effect of negative correlation learning in multi-objective problems, the multi-objective COIN is tested in a multi-objective TSP problem. We setup an experiment using kroa100 and krob100 as a bi-objective 100 tours TSP problem obtained from the TSPLIB. The population size we used in the experiment is 250 and the learning step k is equal to 0.1.

6.2.4 Discussion

6.2.4.1Gröstel24

Table 6.1 Tour length for the Gröstel24 problem

Algorithm	Population & Local Optimization							
	500-without		500-with		1000-without		1000-with	
	Best	Aver	Best	Aver	Best	Aver	Best	Aver
GA-ER*	1272	1272						
GA-OX2*	1300	1367						
UMDA	1339	1495	1272	1272	1329	1496	1272	1272
MIMIC	1391	1486	1272	1272	1328	1451	1272	1272
TREE	1413	1486	1272	1272	1429	1442	1272	1272
EBNA	1431	1528	1272	1272	1329	1439	1272	1272
COIN *unif*	1272	1280			1272	1278		
COIN *adpt*	1272	1272			1272	1272		

* Size of population 200, mutation used SM

unif denotes uniform selection with learning step $k = 0.1$

adpt denotes adaptive selection with learning step $k = 0.1$

Optimum 1272

The TSP coding for continuous EDAs in the original literature uses real numbers which later sorted into the ordering path based on these numbers, is known to be a poor alternative coding compared with path representations based on permutations. Thus, the results obtained from continuous EDAs are not compared in this study due to the use of different representations. Additionally, the detail of local search in the literature is limited to us, thus the result of COIN incorporate with local search cannot be implemented and compared.

Table 6.2 Number of generations for the Gröstel24 problem

Algorithm	Population & Local Optimization			
	500-without	500-with	1000-without	1000-with
	Gen	Gen	Gen	Gen
UMDA	75	19	78	12
MIMIC	47	4	58	4
TREE	37	4	46	2
EBNA	72	16	79	7
COIN	67		48	

Table 6.1 shows the best results and average results obtained for each of population size, with and without local optimization and learning type of EDAs. The table also shows results obtained for the GA using the crossover operators ER and OX2. The results show that COIN with adaptive selection can find the optimum of Gröstel24 without the need of local optimizer and it is competitive with all the EDAs in the experiment.

Figure 6.1 shows the convergence of the Gröstel24 problem using only good solutions, only bad solutions and using both. It shows that the use of both good and bad solutions outperform the use of only either good or bad solutions. Learning from bad solutions creates more diversity amongst the best results but retaining the average results.

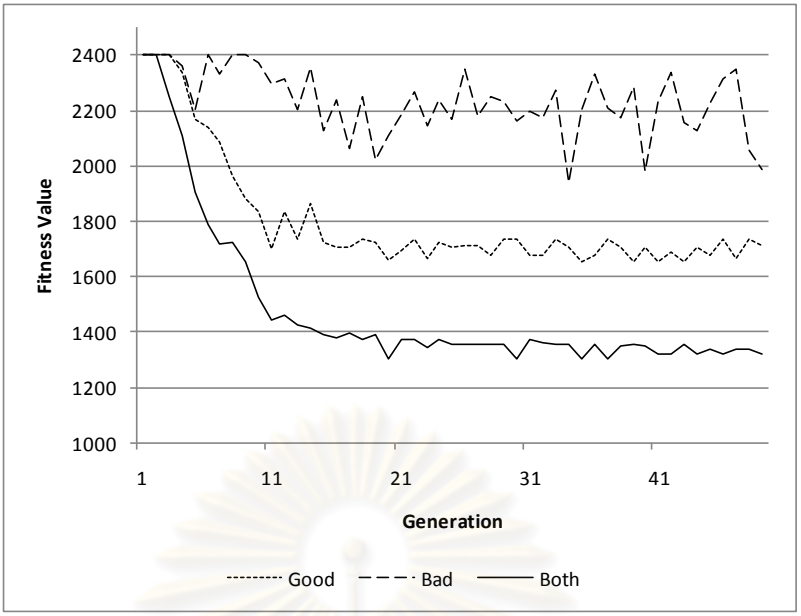


Figure 6.1 The best candidates generated from the generator for Gröstel24

In this experiment, we use the parameter less adaptive selection approach. According to the Figure 6.2, in some generation, there is no reward given to the good solutions as overall fitness's are high. In order to maintain the fitness distribution, the selector rather selects more of the bad individuals than the good ones. When some generations contain more of the bad solutions, the selector rather give more reward than punishment.

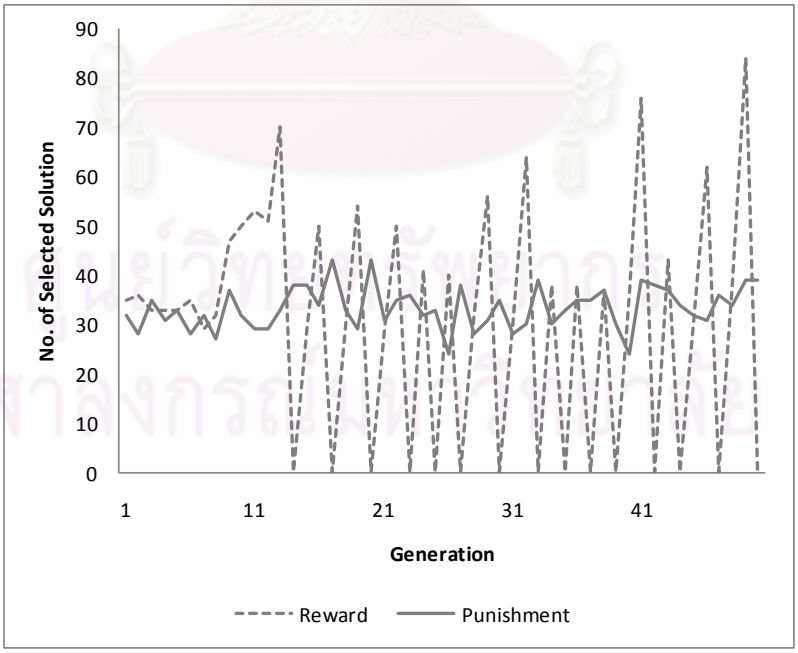


Figure 6.2 The number of good and bad selected solutions using an adaptive selection method in Gröstel24 problems

6.2.4.2 Gröstel48

The results for the Gröstel48 are shown in Table 6.3 and 6.4 COIN sacrifices more generations for a better solution compared to the other discrete EDAs.

Table 6.3 Tour length for the Gröstel48 problem

Algorithm	Population & Local Optimization							
	500-without		500-with		1000-without		1000-with	
	Best	Aver	Best	Aver	Best	Aver	Best	Aver
GA-ER*	5074	5138						
GA-OX2*	5251	5715						
UMDA	6715	7432	5079	5149	6683	7388	5067	5139
MIMIC	6679	7083	5046	5053	6104	6717	5046	5057
TREE	-	-	5046	5071	-	-	5046	5057
EBNA	7044	7476	5165	5193	6398	7336	5114	5146
COIN**	6356	6889			6136	6358		

** Size of population 200, mutation used SM*

*** Learning step k = 0.12, Adaptive Selection*

Optimum 5046

Table 6.4 Number of generations for the Gröstel48 problem

Algorithm	Population & Local Optimization			
	500-without	500-with	1000-without	1000-with
	Gen	Gen	Gen	Gen
UMDA	362	47	218	54
MIMIC	167	23	113	18
TREE	-	8	-	7
EBNA	306	63	261	65
COIN	384		304	

6.2.4.3Gröstel 120

Table 6.5 and 6.6 show the results of the Gröstel120 problem. Again, the COIN algorithm outperforms the rest discrete EDAs. However, more function evaluations were sacrificed.

Table 6.5 Tour length for the Gröstel120 problem

Algorithm	Population & Local Optimization							
	500-without		500-with		1000-without		1000-with	
	Best	Aver	Best	Aver	Best	Aver	Best	Aver
UMDA	14550	15530	7171	7257	14440	15127	7287	7298
MIMIC	13644	14432	7050	7092	12739	13444	7042	7079
COIN*	12298	14307			10162	11273		

** Learning step k = 0.14, Adaptive selection*

Optimum 6942

Table 6.6 Number of generations for the Gröstel120 problem

Algorithm	Population & Local Optimization			
	500-without	500-with	1000-without	1000-with
	Gen	Gen	Gen	Gen
UMDA	385	55	368	42
MIMIC	306	51	348	42
COIN	659		574	

6.2.4.4 KroAB100

We took some snapshots at the number of generations equal to 100 and 500 respectively. The behavior of the algorithm can be seen in Fig. 14 and Fig 15.

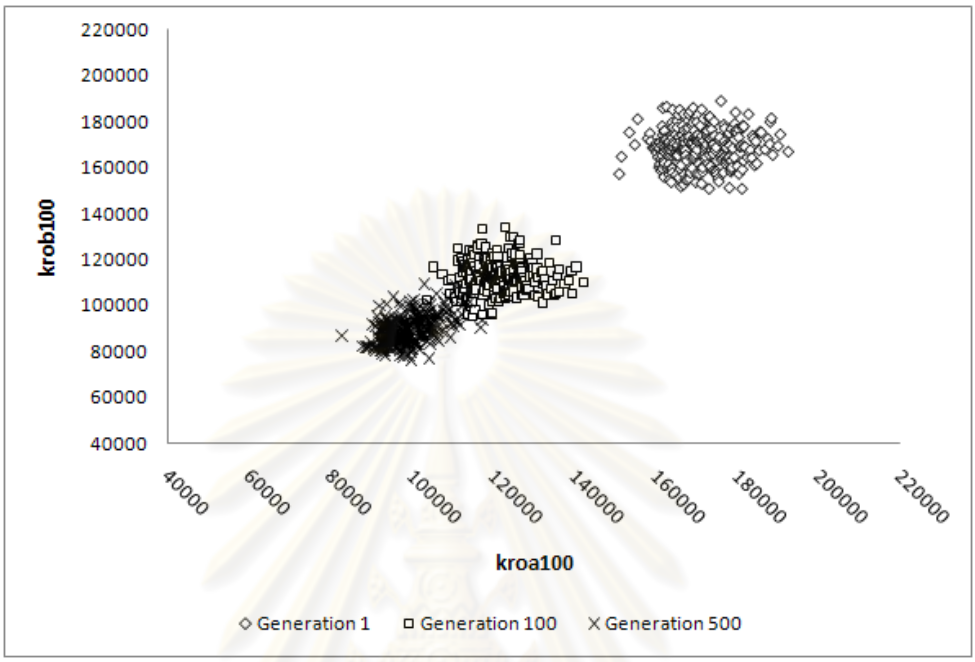


Figure 6.3 The population clouds in a bi-objective kroa/b100 TSP

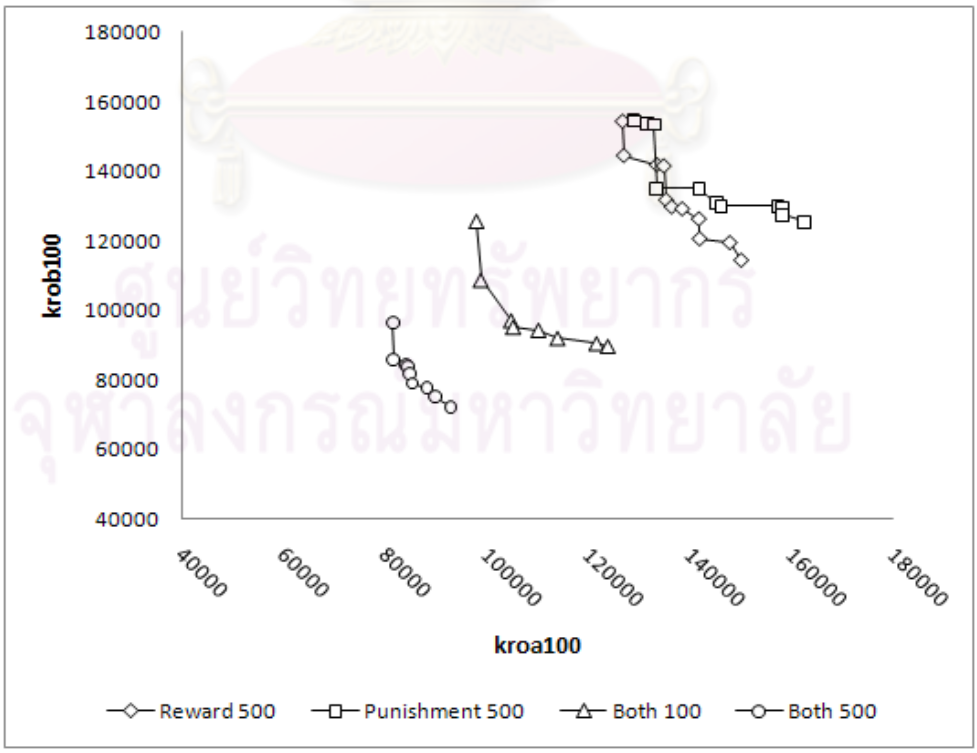


Figure 6.4 The parato frontier obtained from different generation and updating method in a bi-objective kroa/b100 TSP

Figure 6.3 shows the population in the generation 1, 100 and 500 respectively. As the generation progresses, the population migrates towards the optimum fitness area. Figure 6.4 shows the effect of COIN algorithm in using only reward and only punishment compared with using both. Two curves at the upper-right hand corner are the result from using only reward or punishment for 500 generations. Contrast this with the rest of the curves in the lower-left corner which use both reward and punishment together for 100 and 500 generations. The use of both reward and punishment for just 100 generations outperforms the result from using only either one for 500 generations.

However, the result of KroAB100 of COIN cannot be compared with the results of hybrid algorithms in Kumar and Singh's experiments [34]. In their experiment, the local search methods were used in order to improve the quality of the solutions. However, they did not report the number of function evaluation used by local search.

6.3 U-Shape Assembly Line Balancing Problem

6.3.1 Introduction

U-shaped assembly line balancing is considered to be NP-Hard. This kind of assembly line has advantage in reduction of the waste walking time to switch from workstation to workstation, thus enhance reduction of employee and cost. Figure 6.5 illustrates the Jackson's problem[158] with 11 tasks.

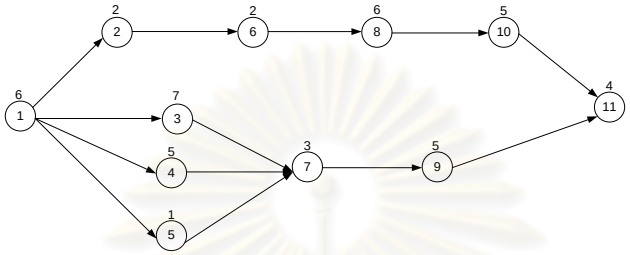
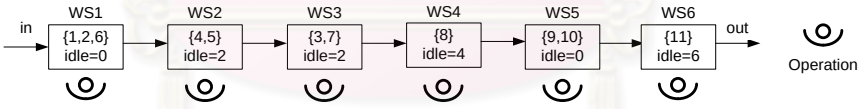
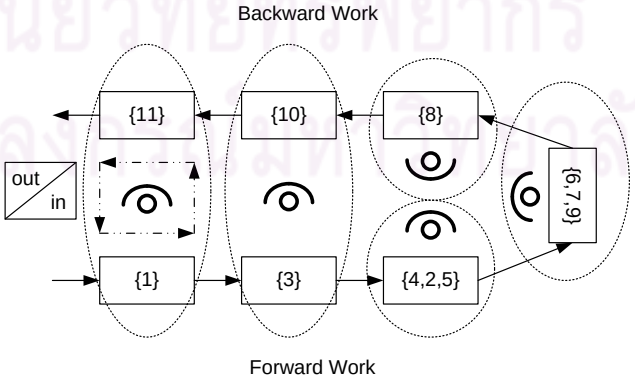


Figure 6.5 The precedence diagram with assembly network (Jacjson 1956)

Given each workstation $ws = 1 \text{ to } m$, number of tasks $i = 1 \text{ to } n$, each task uses time ti. The total time used in the Figure 6.6 (a) is equal to 10 while it used up to 14 if the line is straight. After fitting the tasks and workstations in to the assembly line, we can see that the U-shaped assembly line in the Figure 6.6 (b) use one less workstation than the straight line in the Figure 6.6 (a) 10. As the employee who processes the task number 1 can be shared with the task number 11.



(a) Straight assembly line



(b) U-shape assembly line

Figure 6.6 The comparison of U-Line and straight complete line assignment

6.3.2 Experimental Setup

We carry three experiments based on the work of Hwang and Katayama [159] in three objectives:

Given m is number of workstation

SN_k is number of relatedness of work in the workstation k

S_{max} is total maximum time in the workstation

S_k is total time in the workstation k

, the three objectives are:

1. To minimize the number of workstation.
$$f_1(X) = Min\ m \tag{6.1}$$
2. To minimize the relatedness of the workstations.
$$f_2(X) = m - m \Big/ \sum_{k=1}^{m} SN_k \tag{6.2}$$
3. To minimize the distribution of workload in each station.
$$f_3(X) = Min\ \sqrt{\sum_{k=1}^{m} (S_{\max} - S_k)^2} \Big/ m \tag{6.3}$$

We perform the experiment using Matlab 7.0. The test environment is on Intel Core2 Duo 2.00 GHz with 1.49 GB of RAM. NSGA-II use WMX as a crossover operator. The crossover probability is 0.7 while the mutation probability is 0.3. In this experiment, COIN uses an extra probabilistic model to estimate the first sequence. Moreover, the precedence constraints are integrated into the probability matrix in order to prevent the infeasible solution. In addition, NSGA-II uses a repairing algorithm to detect and repair the infeasible solutions.

The performance indicators use in this experiment including

1. Convergence to the Pareto optimal set.

$$Convergence = \frac{\sum_{i=1}^{|S^*|} d_i}{|S^*|} \tag{6.4}$$

where

$$di = \sqrt{\sum_{i=1}^{k} \left[\frac{f_i(x) - f_i(y)}{f_i^{\max} - f_i^{\min}} \right]} \tag{6.5}$$

and S^* is the set of the solutions, $f_i^{\max}$ and $f_i^{\min}$ are the maximum and minimum value of the objective i and k is the number of objective function.

2. Spread to the Pareto-optimum set.

$$Spread = \frac{d_f + d_l + \sum_{i=1}^{N-1} |d_i - \bar{d}|}{d_f + d_l + (N-1)\bar{d}} \tag{6.6}$$

where d_f and d_l are the distances of the Parato ends, d_i is the distance from the neighbor solutions in the Parato front and N is the total number of solution in the Pareto front.

3. Ratio of non-dominated solution

$$Ratio = \frac{|S_j - \{x \in S_j \mid \exists y \in S : y \prec x\}|}{|S_j|} \tag{6.7}$$

where Sj is the solution set j, S is the union of all Sj, x are the member Sj and y are the member of set S.

Table 6.7 Problem sets of Hwang and Katayama

Problem set	Product	Task	Time)ces(	Density Network
Thomopolous (1970)	3	19	2	0.122807
Kim(2006)	4	61	10	0.036066
Arcus(1963)	5	111	10,000	0.028337

The density network shown in table 6.7 indicates the relationship of the task. If the density network has high value (limited to 1), the possibility of assigning task to workstation is low. In contrast, the possibility of assigning task to workstation is high when the density network is low.

6.3.3 Discussion

According to the Table 6.8, COIN has higher convergence rate than the NSGA-II. NSGA-II give more spread solution in the Pareto-optimal set, but the spread of the solution in this experiment has less significant due to the ratio of non dominated solution of COIN. It is equal to 1 in every test set as none of the NSGA-II's solution can dominate the COIN's solution. Figure 6.7 to 6.9 compares only the Pareto-optimal solution for two objectives since the numbers of workstations in each problem are equal. Moreover, in terms of real CPU time, the multi-objective COIN is much faster than NSGA-II. The total processing time of NSGA-II in Thomopolous (19 tasks), Kim (61 tasks) and Arcus (111 tasks) are 124, 347 and 735minites, while COIN uses only 3, 15, and 40 minutes respectively.

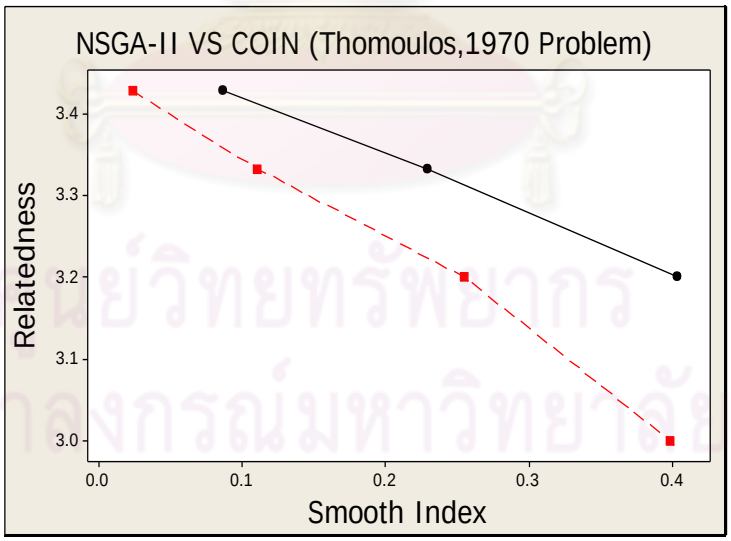


Figure 6.7 The comparison of NSGA-II and COIN in Thomoulos's Problem

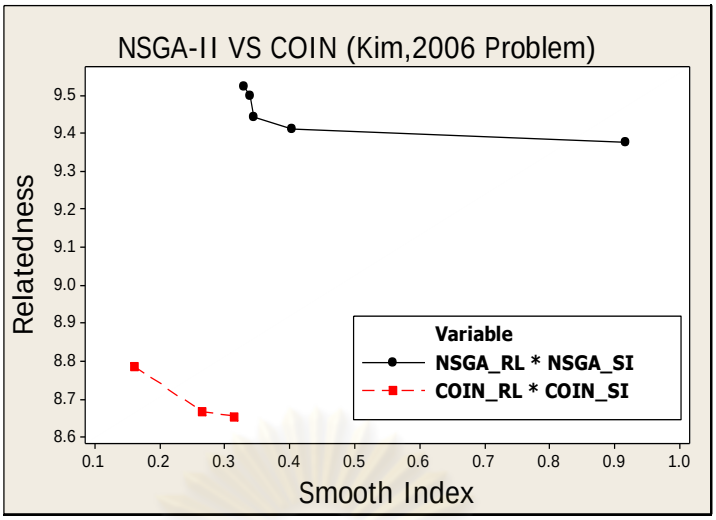


Figure 6.8 The comparison of NSGA-II and COIN in Kim's Problem

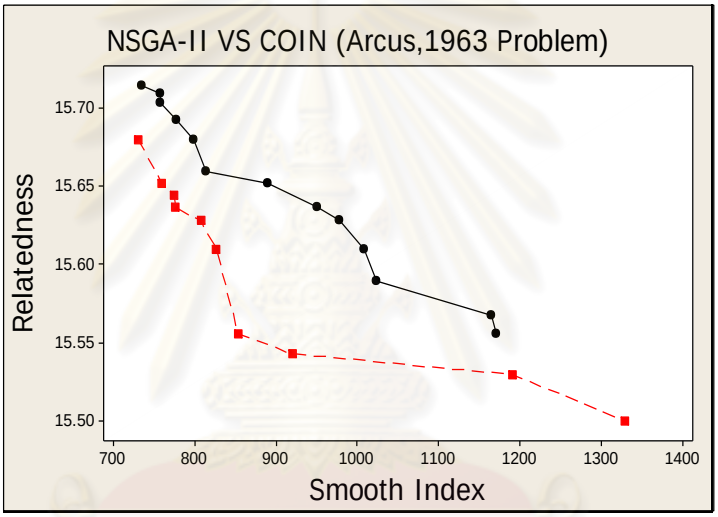


Figure 6.9 The comparison of NSGA-II and COIN in Arcus's Problem

Table 6.8 Result of the experiment in Hwang and Katayama's problems

Benchmarking	Problems and Algorithms					
	Thomopolous (19 task)		Kim (61 task)		Arcus (111 task)	
	NSGA-II	COIN	NSGA-II	COIN	NSGA-II	COIN
Convergence	0.295	**0**	0.847	**0**	0.189	**0**
Spread	0.566	0.523	0.742	0.774	0.485	**0.710**
Ratio of solution	0	**1**	0	**1**	0	**1**
Time (min)	124	**3**	347	**15**	735	**40**

Population size = 100, Generation = 200

NSGA-II: Crossover probability = 0.7, Mutation probability = 0.3

COIN: $k = 0.1$

6.4 U-Shape Assembly Line Sequencing Problem

6.4.1 Introduction

The next case study, the problem sets are sequencing problems on mixed-model U-Shaped assembly lines sequencing. In this experiment, we assume that line balancing is solved and only sequencing problems are considered. Determining the sequence of introducing models to MMUL is of particular importance considering the goals crucial for efficient implementation of JIT, i.e. smoothening workload and setup time reduction.

6.4.2 Experimental setup

We carry three experiments based on the work of Kim and Arcus [159] in two objectives:

Given MPS_i is the minimum part set for a task i,

MS_i is the model sequencing of task i.

s_{ik} is equal to 1 if the task pattern at the position k of MS_i is different from the task pattern at $k-1$, otherwise 0.

t_{ik} is the machine setting up time for task i.

t_{i0} is the machine initialization time.

L_i is the total number of task pattern

N is the total number of task

J is the number of work station

T_{js} is the number of assigned task to the work station j at the s^{th} cycle.

T is the cycle time

n is the number of the product in the production line

d_i is the product demand

, the objective functions are:

1. To minimize the machine setting up time

$$f_1(x) = \sum_{i=1}^{N} (\sum_{k=2}^{L_i} s_{ik} \times t_{ik}) + t_{i0} \tag{6.8}$$

2. To minimize the absolute deviation of the workload

$$f_2(x) = \sum_{j=1}^{J} \sum_{s=1}^{S} \left| \; T_{js} - \overline{T} \; \right| \tag{6.9}$$

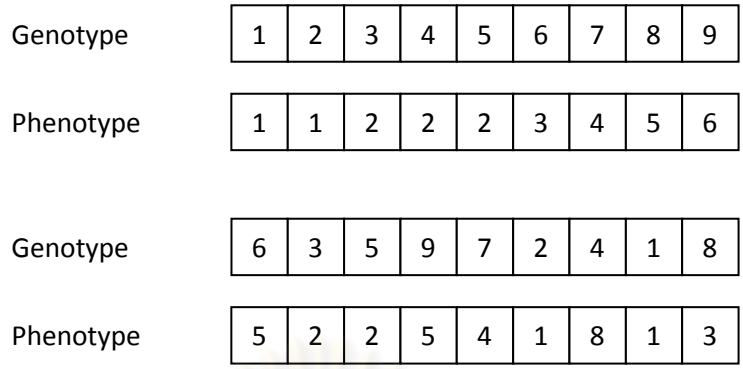


Figure 6.10 Encoding of the sequencing problem.

The experiments settings and performance indicators of this experiment are similar to that in the Multiobjective U-shape assembly line balancing problems. However, the only difference is that the encoding of the solution strings. Figure 6.10 presents the encoding of a sequencing problem. In sequencing problems, the sequence items can be redundant. Therefore, in order to apply the coincidence algorithm, we need to encoding the redundant items in to a permutation of unique items such that the unique items can be mapped to the redundant tasks.

6.4.3 Discussion

According to the Table 6.9, COIN defeat NSGA-II in all performance measurements. Figure 6.11 and 6.12 compares the Pareto-optimal solution obtained from COIN and NSGA-II.

An important aspect of this individual representation based on permutations is that the cardinality of the search space is $n!$.This cardinality is higher than that of the traditional individual representation, but it is tested for its use with EDAs in sequencing problems for the first time here. In addition, it is important to note that a permutation-based approach can create redundancy in the solutions, as two different permutations may correspond to the same solution. An example of this is shown in Figure 6.13, where two individuals with different permutations are shown and the solution they represent is exactly the same.

Table 6.9 Performances of NSGA II and COIN
in U-shaped assembly line sequencing problems

Problem Set	Performance Measure	Algorithm	
		NSGA-II	COIN
Kim 2	Convergence	0.015	**0**
	Spread	0.783	**0.719**
	Ratio of Solution	0.667	**0.778**
	Time (min)	398	**12**
Kim 3	Convergence	0.031	**0**
	Spread	0.532	**0.573**
	Ratio of Solution	0.667	**0.714**
	Time (min)	398	**12**
Kim 5	Convergence	0.025	**0**
	Spread	0.572	**0.643**
	Ratio of Solution	0.315	**0.5**
	Time (min)	398	**12**
Kim 6	Convergence	**0**	**0**
	Spread	**0.427**	**0.427**
	Ratio of Solution	**0.693**	**0.693**
	Time (min)	397	**11**
Arcus 2	Convergence	0.013	**0**
	Spread	0.546	**0.549**
	Ratio of Solution	0.800	**1**
	Time (min)	725	**18**
Arcus 3	Convergence	0.178	**0.076**
	Spread	**0.783**	0.543
	Ratio of Solution	0.750	**0.8**
	Time (min)	725	**18**
Arcus 4	Convergence	1.097	**0**
	Spread	**0.758**	**0.758**
	Ratio of Solution	0	**1**
	Time (min)	725	**19**
Arcus 6	Convergence	0.016	**0.019**
	Spread	0.664	**0.692**
	Ratio of Solution	**0.714**	**0.714**
	Time (min)	725	**18**
Arcus 7	Convergence	**0**	**0**
	Spread	**0.553**	**0.553**
	Ratio of Solution	**1**	**1**
	Time (min)	725	**18**
Arcus 8	Convergence	0.124	**0.049**
	Spread	**0.687**	0.654
	Ratio of Solution	0.333	**0.5**
	Time (min)	725	**18**

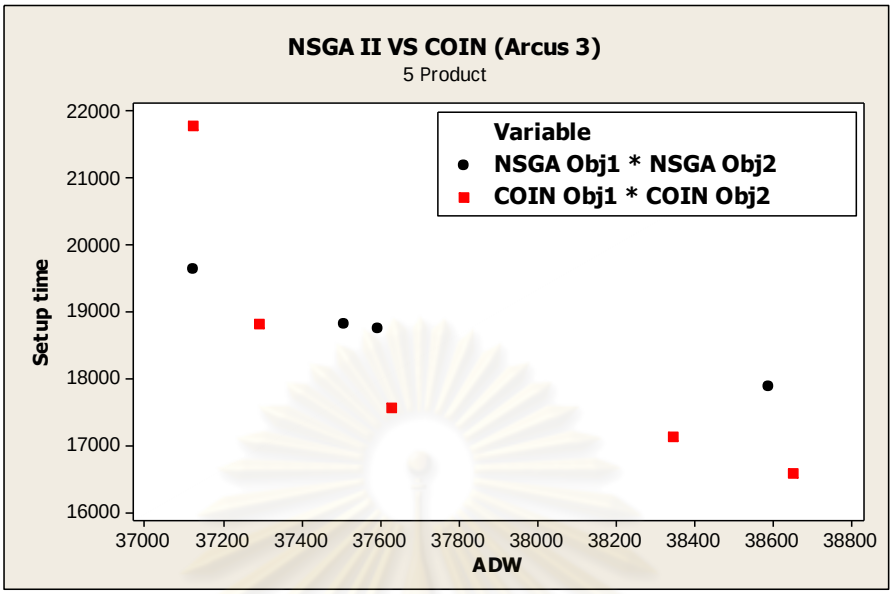


Figure 6.11 The comparison of NSGA-II and COIN in Arcus's Problem

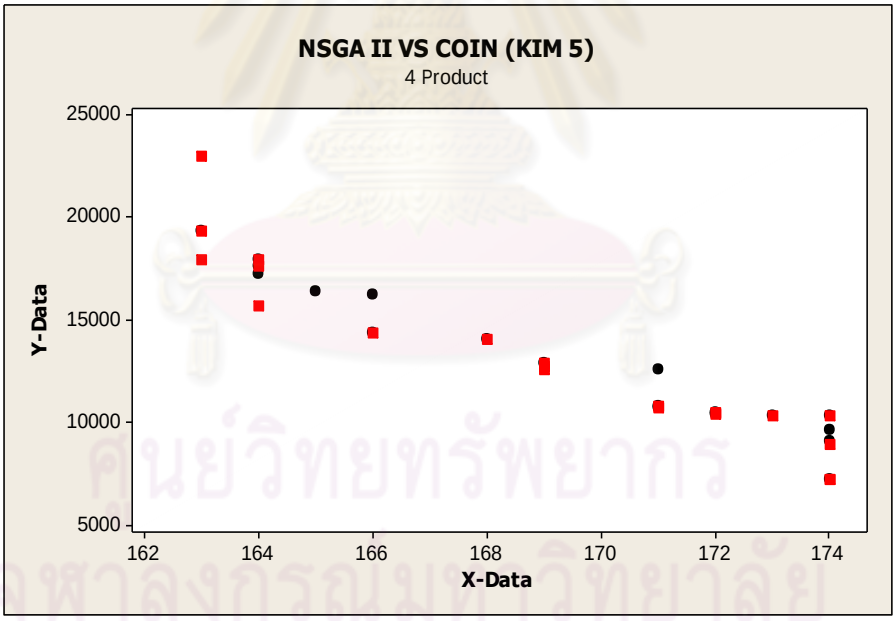


Figure 6.12 The comparison of NSGA-II and COIN in Kim's Problem

Consequently, encoding using such scheme favor NSGA-II to generate more diverse solution than COIN due to COIN might waste the function evaluation to evaluate the redundant solutions.

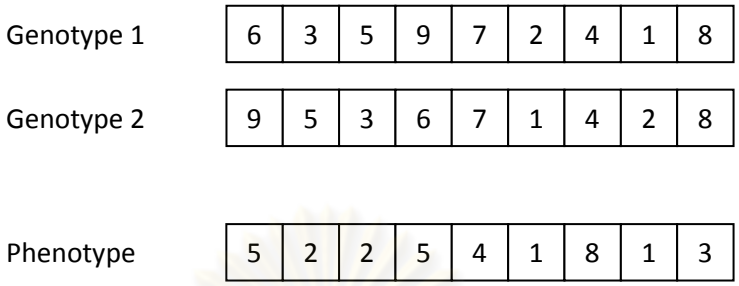


Figure 6.13 Example of redundancy in the permutation-based approach. The two individuals represent the same solution shown in the phenotype.

6.5 Discussion

In this chapter, COIN has proved its efficiency in solving several real world applications. COIN searches and samples candidates for single and multiple objectives problems very effectively compared to GA. The performances of COIN are gained from (*i*) the sampling method and (*ii*) the negative correlation learning. As shown in the TSP benchmarks, the negative correlation learning does not only contribute in preventing the premature convergence, but also contribute in accelerating the search process.

6.6 Chapter Summary

This chapter, we propose some application of COIN in several real world applications including travelling salesperson problems, line balancing problems and sequencing problems. The overall results show that COIN is a competitive algorithm in solving both single and multiple objectives real world applications.

CHAPTER VII

CONCLUSIONS

7.1 Conclusions

In this dissertation, we addressed the difficulties of combinatorial optimization where the main difficulty is the representation and the effective ways to construct a candidate solution.

We presented a new estimation of distribution algorithm (EDA) called Coincidence Algorithm (COIN). Our contribution here is twofold. First, a probabilistic model based on Markov Chain Monte Carlo (MCMC), and second new incremental learning method that involve the negative correlation learning in the model.

From this algorithm, we propose a new hypothesis called the *negative building block hypothesis* (NBBH) which extends the *building block hypothesis* (BBH) previously proposed by Goldberg [7]. The NBBH simply says that avoiding the recombination of short low fitted schemas so called *negative building blocks* should be able to form the average solutions not worse than their ancestors. Searching in a scope of schemas can be considered to be a guided search or a search with direction. However, searching out of a scope of schemas cannot be considered to be an unguided search or a search without a direction but considered to be a multi-direction search. The multi-direction search is expected to maintain more diversity than a guided search; however, the time to converge and the quantity of the result are expected to be poorer. COIN is an algorithm based on both BBH and NBBH. The combination of BBH and NBBH is expected to utilize both of the BBH and NBBH advantages.

Different types and different sizes of the benchmarks have been presented. Our contribution here is a set of globally multimodal benchmarks which has never been tested by any algorithm on the capability to find the multimodal solutions. The results show that the negative correlation learning capability of COIN contributes on both quality and quantity of the solutions. However, the negative correlation learning expresses differently in different benchmarks which depends mainly on the quantity of building blocks being shared and the quantity of building blocks being in conflict.

Finally, one of the best contributions is that we propose an extension of COIN in solving multi-objective problems. We adopt the non-dominated sorting and crowding distance from NSGA-II. The experiments were performed in several real world applications and yield fascinating results. The overall performances of COIN are better than NSGA-II in every benchmark indicators. More results of COIN in solving multi-objective problems can be found in parallel works including a PhD dissertation [160] and five master theses [161][162][163][164][165].

7.2 Recommendation for Future Research

Many different adaptations, tests, and experiments have been left for the future due to lacking of time (i.e. the experiments with real world applications are usually very time consuming, requiring even days to finish a single run). Future work concerns deeper analysis of particular mechanisms, new proposals to try different methods, or simply curiosity.

There are some ideas that the author would like to try during the development of the updating equation in Chapter IV. This dissertation has been mainly focused on the use of negative knowledge in EDAs. However, we used a constant learning rate k for both positive and negative sample. Moreover, the learning rate is static. From the observations, the greater learning rate would lead the algorithm to converge faster, yet easier to get stuck in some local optima. In order to investigate the role of negative knowledge, the constant learning rate is needed to be fixed, leaving the study of dynamic learning rate outside the scope of the dissertation. The following ideas could be tested:

1. It could be interesting to separate the learning coefficient k for reward and punishment.
2. The learning coefficient k could be change dynamically.

The negative correlation could be apply to node based EDA as well. At the very beginning of this research, the incremental node based EDA came up in the author's mind. The prototype of COIN based on absolution position was implemented. However, the algorithm was failed to converge. At the end of this research, we found out that the candidate solutions in node based EDA should not be generated as a sequence, but should be generated according to the random position.

The initial population in all EDAs has been built using a uniform distribution. Other methods could be also tested, as sometimes a pre-processing step could be added such that the search can also start with some specific individuals. Also, other types of statistical initializations such as greedy probabilistic methods could help at directing the search from the beginning, leading to fewer evaluations.

Regarding the application of parallelism to EDAs, an extension for the near future is the use of more powerful multiple instruction multiple data (MIMD) architecture. COIN can make use of the advantage of parallel instruction set to improve the performance in generating the candidates, updating the probabilistic model and evaluation of the population.

REFERENCES

[1] Silvio, T. Optimization in Permutation Spaces. Western Research Laboratory 1996/1

[2] Glover, F., and Laguna, M. Tabu Search. Kluwer Academic Publishers : Norwell, 1997.

[3] Andrei, A. Geometric Permutation in the Plane and in Eucidean Spaces of Higher Dimension. Doctoral dissertation in Mathematics Faculty of Mathematics Israel Institute of Technology, 2005.

[4] Aronov, B. and Smorodinsky. On Geometric Permutations Induced by Lines Transversal through a Fixed Point. Proc. 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) (2005)

[5] Cheong, O. Goaoc, X. anh Na, H.S. Geomatric Permutations of Disjoint Unit Spheres. Comput. Geom.: Theory and Applications 30 (2005) : 253-270.

[6] Stefan, H. On the Significance of the Permutation Problem in Neuroevolution. Doctoral dissertation, Faculty of Engineering and Physical Sciences University of Manchester, 2010.

[7] Goldberg D.E. Genetic Algorithms in Search and Optimization. Addison-Wesley Professional, 1989.

[8] Bassett, JK. Potter, MA. and De Jong, K. Applying Price's Equation to Survival Selection. Proc. of the 2005 Genetic and Evolutionary Computation (GECCO) (2005) : 1371–1378.

[9] Jain, A. and Fogal D. Case Studies in Applying Fitness Distributions in Evolutionary Algorithms: I. Simple Neural Networks and Gaussion Mutation.. Proc. of The International Society for Optical Engineering (SPIE) (2000) : 168-175.

[10] Fodal, D. Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. John Wiley and Sons, 2006.

[11] Jones, T. Crossover, Macromutation, and Population-Based Search. Proceedings of the Sixth International Conference on Genetic Algorithm (1995) : 73-80.

[12] Watson, R. A., and Pollack, J. B. Recombination Without Respect: Schema Combination and Disruption in Genetic Algorithm Crossover. Proc. of the 2000 Genetic and Evolutionary Computation (GECCO) (2000) : 112-119.

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

[14] Harik, G.R., and Goldberg, D.E. Learning Linkage, Foundations of Genetic Algorithms 4, 7-12. 1996.

[15] Munetomo, M., and Goldberg, D. E. Identifying Linkage Groups by Nonlinearity/Non-monotonicity Detection. Proc. of the 2000 Genetic and Evolutionary Computation (GECCO) 1 (1999) : 433-440.

[16] Pelikan, M. and Goldberg, D.E. A Survey of Optimization by Building and Using Probabilistic Models. IlliGAL Report 99018 (1999).

[17] Larrañga, P., and Lozano, J. A. Estimation of Distribution Algorithms. Kluwer Academic Publishers, 2002.

[18] Tsutsui, S. Node Histogram vs. Edge Histogram: A Comparison of Probalistic Model-Building Genetic Algorithms in Permutation Domains. Proc. of the IEEE Congress on Evolutionary Computation (2006).

[19] Tsutsui, S. Probabilistic Model-Building Genetic Algorithms in Permutation Representation Domain Using Edge Histogram. Proc. of the 7th International Conference on Parallel Problem Solving from Nature (PPSN VII) (2002) : 224-233.

[20] Dorigo, M. et al. Ant Algorithms for Distributed Discrete Optimization. Artificial Life, 5 (1999).

[21] Bassett, J.K. Potter, M.A., and De Jong, K. Looking Under the EA Hood with Price's Equation. Proc. of the 2005 Genetic and Evolutionary Computation (GECCO) (2004) : 914–922.

[22] William J.C., William H.C., William R.P. and Alexander S. Combinatorial Optimization. John Wiley & Sons, 1997.

[23] Alexander, S. Combinatorial Optimization. Springer, 2003.

[24] Talbi, E. Metaheuristics : From Design to Implementation. John Wiley and Sons, 2009.

[25] Papadimitriou, C.H. Combinatorial Optimization : Algorithm and Complexity. Prentice-Hall, 1982.

[26] Dmitris, A., and Manfred, W.P. Linear Optimization and Extensions: Problems and Extensions. Springer-Verlag, 2001.

[27] Garey, M.R., and Johnson, D.S. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, 1979.

[28] Reeves, C.R., and Beasley, J.E. Modern heuristic techniques for combinatorial problems. McGraw-Hil, 1995.

[29] Golden, B.L, and Stewart, Jr.W.R. Empirical Analysis of Heuristics. Lawler, E.L., and Shmoys, D.B. (Eds), The Travelling Salesman Problem. A Guided Tour of Combinatorial Optimization, 207-249. Wiley, 1985.

[30] Johnson, David S., and McGeoch, Lyle A. The Traveling Salesman Problem: A Case Study in Local Optimization. Aarts H.L. and Lenstra, J.K. (Eds) Local Search in Combinatorial Optimization, 215-310. London : John Wiley and Son, 1997.

[31] Lin, S. and Kernighan, B.W. An Effective Heuristic Algorithm for the Traveling-Salesman Problem., Operations Research , 21/2 (1973) : 498-516.

[32] Gendreau, M., Hertz, A., and Laporte, G. New Insertion and Post Optimization Procedures or the Traveling Salesman Problem. Operations Research 40 (1992) : 1086-1094.

[33] Glover, F. Future Paths for Integer Programming and Links to Artificial Intelligence. Computers and Operations Research, 13 (5): 533–549.

[34] Blum, C., and Roli, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys 35/3 (2003) : 268-308.

[35] Aarts H.L., and Lenstra, J.K. Local Search in Combinatorial Optimization. London : John Wiley and Son, 1997.

[36] Kirkpatrick, S. et al. Optimization by Simulated Annealing. Science 220 (1983) : 671-680.

[37] Cerny, V. A Thermodynamics Approach to the Travelling Salesman Problem: An Efficient Simulation Annealing. Journal of Optimization Theory and Applications 45 (1985) : 41-51.

[38] Dekkers, A., and Aarts, E. Global Optimization and Simulated Annealing. Mathematical Programming 50 (1991) : 367-397.

[39] Locatelli, M. Simulated Annealing Algorithms for Continuous Global Optimizatin: Convergence Conditions. Journal of Optimization Theory and Applications 29/1 (2000) : 87-102.

[40] Martin, O. Otto, S.W., and Felten, E.W. Large-Step Markov Chains for the Traveling Salesman Problem. Complex Systems, 5/3(1991) : 299–326.

[41] Nowicki, E. and Smutnicki, C. A Fast Tabu Search Algorithm for the Permutation Flow-Shop Problem. European Journal of Operational Research 91 (1996) : 160–175.

[42] Vaessens, R.J.M., Aarts, E.H.L., and Lenstra, J.K. Job Shop Scheduling by Local Search. INFORMS Journal on Computing, 8 (1996) : 302–317.

[43] Feo, T.A. and Resende, M.G.C. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. Operations Research Letters 8 (1989) : 67–71.

[44] Feo, T.A. and Resende, M.G.C. Greedy Randomized Adaptive Search Procedures. Journal of Global Optimization 6(1995) : 109–133.

[45] Back, T. Evolutionary Algorithms in Theory and Practice. Oxford University Press, 1996.

[46] Rechenberg, I. Evolutionsstrategie— Optimierung technischer Systeme nach Prinzipien der biologischen Information. Fromman Verlag, Freiburg, 1973.

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

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

[49] Michalewicz, Z. Genetic Algorithms + Data Structures = Evolution Programs 3rd edition Springer Verlag, 1996.

[50] Goldberg, D.E., and Lingle, R. Alleles, Loci and the Traveling Salesman Problem. Proc. of the 1st International Conference on Genetic Algorithm (ICGA) (1985) : 154-159.

[51] Oliver, I.M., Smith, D.J., and Holland, J.R.C. A Study of Permutation Crossover Operators on the Travelling Salesman Problem. Proc. of the 2nd International Conference on Genetic Algorithm (ICGA) (1986) : 224-230.

[52] Davis, L. Applying Adaptive Algorithms to Epistactic Domains. Proc. of International Conference on Artificial Intelligence (IJCAI) (1985) : 162-164.

[53] Gorges, S.M. ASPARAGOS: An Asynchronous Parallel Genetic Optimization Strategy. Proc. of the 3rd International Conference on Genetic Algorithm (ICGA) (1987) : 422-427.

[54] Syswerda, G. Schedule Optimization Using Genetic Algorithm. Davis L. (Eds) Handbook of Genetic Algorithm 332-349 Van Nostrand Reinhold, 1990.

[55] Lee, J., Gen M., and Rhee, K. Designing a Multistage Reverse Logistics Network Problem by Hybrid Genetic Algorithm. Proc. of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO) (2008).

[56] Lee, J., Gen, M. and Rhee, K. Network Model and Optimization of Reverse Logistics by Hybrid Genetic Algorithm. Computers and Industrial Engineering 56/3 (April 2009).

[57] Whitley, D. Starkweather T. and Fuquay D. Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator. Proc. of the 3rd International Conference on Genetic Algorithm (ICGA) (1987) : 133-140.

[58] Grefenstette, J. et al. Genetic Algorithms for Traveling Salesman Problem. Proc. of the 1st International Conference on Genetic Algorithm (ICGA) (1985) : 160-168.

[57] Starkweather, T. et al. A Comparison of Genetics Sequencing Operator. Proc. of the 4th International Conference on Genetic Algorithm (ICGA) (1991) : 69-76.

[58] Armed, Z.H. Genetic Algorithm for the Traveling Salesman Problem using Sequential Constructive Crossover Operator. International Journal of Biometrics & Bioinformatics (IJBB) 3/6 (2010) : 96-105.

[59] Dorigo, M. Optimization, Learning, and Natural Algorithms. Doctoral thesis, Dip. Elettronica e Informazione, Politecnico di Milano, Milano, Italy, 1992.

[60] Dorigo, M., Maniezzo, V., and Colorni, A. Positive Feedback as a Search Strategy. Technical Report 91-016, Politecnico di Milano, Milano, Italy, 1991.

[61] Dorigo, M., Maniezzo, V., and Colorni, A. The Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transactions on Systems, Man, and Cybernetics – Part B, 26/1 (1996) : 29–42.

[62] Dorigo, M. et al. Ant Algorithms for Distributed Discrete Optimization. Artificial Life, 5 (1999).

[63] Kennedy, J. and Eberhart, R.C. Swarm Intelligence. San Francisco, CA : Morgan Kaufmann, 2001.

[64] Kennedy, J. and Eberhart, R.C. Particle Swarm Optimization. Proc of IEEE Conference on Neural Networks (1995) : 1942-1948.

[65] Kennedy, J. and Eberhart, R.C. A Discrete Binary Version of the Particle Swarm Algorithm. Proc. of IEEE Conference Systems, Man and Cybernetic (1997) : 4104-4108.

[66] Li, B., Wang, L. and Liu, B. An Effective PSO-Based Hybrid Algorithm for Multiobjective Permutation Flow Shop Scheduling. Proc. of IEEE Conference Systems, Man and Cybernetic (2008) : 818-831.

[67] Hu, X., Eberhart, R. C., and Shi, Y. Swarm Intelligence for Permutation optimization: a case study on n-queens problem. Proc. of the IEEE Swarm Intelligence Symposium 2003 (SIS 2003) (2003) : 243-246.

[68] Lozano, J.A. Larrañga, P., Inza, I., and Bengoetxea, E. Towards a New Evolutionary Computation. Advances in Estimation of Distribution Algorithms. Springer, 2006

[69] Pelikan, M., Sastry, K., and Cantu-Paz, E. Scalable Optimization via Probabilistic Modeling: From Algorithm to Applications. Springer, 2006.

[70] Baluja, S., and Caruana, R. Removing the Genetics from the Standard Genetic Algorithm. Proc. of International Conference on Machine Learning. (1995) : 38-46.

[71] Baluja, S. Population Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA, 1994.

[72] Harik, G.R., Lobo, F.G., and Goldberg, D.E. The Compact Genetic Algorithm. IEEE Transaction on Evolutionary Computation 3/4 (Nov 1999) : 287-297.

[73] Ghosh, A., and Muehlenbein, H. Univariate Marginal Distribution Algorithms for Non-Stationary Optimization Problems. International Journal of Knowledge-based and Intelligent Engineering Systems 8/3, (August 2004) : 129-138.

[74] De Bonet, J.S., Isbell, C.L., and Viola, P. MIMIC: Finding Optima by Estimating Probability Densities. Advance in Neural Information Processing Systems, 9 (1997).

[75] Pelikan, M. Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms. Springer, 2005.

[76] Glover, F. Heuristics for Integer Programming Using Surrogate Constraints. Decision Science, 8 (1997) : 156-166.

[77] Laguna, M. and Marti, R. Scatter Search: Methodology and Implementations in C. Boston, MA, Kluwer Academic Publishers. 2003.

[78] Glover, F. Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems. Discrete Applied Mathematics 65(1996) : 223-253.

[79] Glover, F. A Template for Scatter Search and Path Relinking. Lecture Notes in Computer Science, 1363(1997) : 13-54.

[80] Glover, F. Scatter Search and Path Relinking. New Ideas in Optimisation Wiley, (1999).

[81] Glover, F. Fundamentals of Scatter Search and Path Relinking. Control and Cybernatics 29/3 (2000) : 653-684.

[82] Ehrgott, M., and Gandibleux, X. A survey and annotated bibliography of multiobjective combinatorial optimization. OR Spektrum, 22 (2000) : 425-460.

[83] Diaz, J.A. Solving multiobjective transportation problems. Ekonomicko Mathematicky Obzor 14 (1978) : 267–274.

[84] Steuer, R.E., Gardiner, L.R., and Gray, J. A Bibliographic Survey of the Activities and International Nature of Multiple Criteria Decision Making. Journal of Multi-Criteria Decision Analysis 5 (1996) : 195–217.

[85] White, D.J. A Bibliography on the Application of Mathematical Programming Multiple-Objective Methods. Journal of the Operational Research Society 41/8 (1990) : 669–691.

[86] Hwang, C.L., and Masud, A.S.M. Multiple Objective Decision Making – Methods and Application: A State of the Art Survey. Lecture Notes in Economics and Mathematical Systems 164 (1979).

[87] Srinivas, N., and Deb, K. Multi-Objective Function Optimization Using Non-Dominated Sorting Genetic Algorithms. Evolutionary Computation 2/3(1995) : 221–248.

[88] Sandgren, E. Multicriteria design optimization by goal programming. Advances in Design Optimization, 225-265. London, Chapman&Hall 1994.

[89] Kosmidou, K., and Zopounidis, C. Goal Programming Techniques for Bank Asset Liability Management. Kluwer Academic Publishers, 1994.

[90] Wierzbicki, A. The use of reference objectives in multiobjective optimization. Multiple Criteria Decision making, Theory and Application, Springer, 1980.

[91] Liao, W.B., Chen. Y.-L., and Wang, S.C. Goal-Attainment Method for Optimal Multi-objective Harmonic Filter Planning in Industrial Distribution Systems. IEEE Proc. on Generation, Transmission and Distribution. 149/5 (2002) : 557-563.

[92] Becerra, R.L. Coello, C.C., and Alfredo, G. Alternative Techniques to Solve Hard Multi-Objective Optimization Problems. Proc. of the 2007 Genetic and Evolutionary Computation (GECCO) (2007) : 757–764.

[93] Schaffer, J.D. Multiple Objective Optimization With Vector Evaluated Genetic Algorithms. Doctoral thesis, Vanderbilt University, Nashville, TN, USA, 1984.

[94] Parsopoulos, K.E., Tasoulis, D.K., and Vrahatis, M.N. Multiobjective Optimization Using Parallel Vector Evaluated Particles Swarm Optimization. Proc. of the 22nd International Conference on Artificial Intelligence and Applications (IASTED) (2004).

[95] Fishburn, P.C. Lexicographic Orders, Utilities and Decision Rules: A survey. Management Science 20/11 (1974) : 1442-1471.

[96] Deb, K., Pratap, A., Agrawal, S., and Meyarivan, T. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6/2(April 2002) : 182-197.

[97] Zitzler, E., Laumanns, M., and Thiele, L. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. TIK-Report 103 (May 2001).

[98] David, A., van Veldhuizen., and Lamont, G.B. Multiobjective evolutionary algorithms: classifications, analyses, and new innovations. Doctoral Dissertation. Air Force Institute of Technology Wright Patterson AFB, OH, USA 1999.

[99] Zydallis, J.B., David, A. van Veldhuizen., and Lamont, G.B, A Statistical Comparison of Multiobjective Evolutionary Algorithms Including the MOMGA-II. Proc. of the 1st International Conference on Evolutionary Multi-Criterion Optimization (March 2001) : 226-240 .

[100] David, A., van Veldhuizen., and Lamont, G.B. Evolutionary Algorithms for Solving Multi-Objective Problems, Norwell, MA : Kluwer Academic Publishers, 2002.

[101] Ranjithan, S. R., Chetan, S. K., and Dakshina, H. K. Constraint Method-Based Evolutionary Algorithm (CMEA) for Multiobjective Optimization. Proc of Evolutionary Multi-Criterion Optimization: First International Conference, EMO 2001, Zurich, Switzerland, March 2001. Lecture Notes in Computer Science 1993 (2001) : 299–313.

[102] Zitzler, E., Laumanns, M., and Bleuler, S. A Tutorial on Evolutionary Multiobjective Optimization. Workshop on Multiple Objective Metaheuristics (MOMH 2002) Springer Verlag 2004.

[103] Fonseca, C. M., and Fleming, P. J. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. Proc. of the 5th International Conference on Genetic Algorithms (1993) : 416–423.

[104] Zitzler, E., and Künzli, S. Indicator-Based Selection in Multiobjective Search. Proc. of Conference on Parallel Problem Solving from Nature (PPSN VIII) (2004) : 832-842.

[105] Knowles, J. D., and Corne, D. W. The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Pareto Multiobjective Optimisation. Proc. of Congress on Evolutionary Computation (CEC99), 1 (1999) : 98–105.

[106] Minsky, M. Negative Expertise. International Journal of Expert Systems 7(1994) : 13-19.

[107] Oser, F., and Spychiger, M. Learning is Painful. On the Theory of Negative Knowledge and the Practice of Error Culture. Weinheim: Beltz, 2005.

[108] Parviainen, J. & Eriksson, M. Negative Knowledge, Expertise and Organisations. International Journal of Mangement Concepts and Philosophy 2(2006) : 140-153.

[109] von Glasersfeld, E. Constructivism in Mathematics Education. Kluwer Academic Publishers, 1991.

[110] Gartmeier, M. et al. Negative knowledge: Understanding Professional Learning and Expertise. Vocations and Learning: Studies in Vocational and Professional Education. (2008) : 87-103.

[111] Tizhoosh, H.R. Opposition-Based Learning: A New Scheme for Machine Intelligence. Proc.of International Conference on Computational Intelligence for Modelling Control and Automation - CIMCA'2005 1(2005) : 695-701.

[112] Tizhoosh, H.R. Reinforcement Learning Based on Actions and Opposite Actions. Proc. of International Conference on Artificial Intelligence and Machine Learning (AIML-05) (2005)

[113] Rahnamayan, S., Tizhoosh, H.R., and Salama, M.M. Opposition-Based Differential Evolution Algorithms. IEEE Congress on Evolutionary Computation proceeding 2006 (WCCI 2006) (2006) : 7363-7370.

[114] Ji, Z., and Dasgupta, D. Revisiting Negative Selection Algorithms. Evolutionary Computation. 15/2 (2007) : 223-251.

[115] Michalski, R.S. Learnable Execution Model: Evolutionary Processes Guided by Machine Learning. Machine Learning. 38 (2000) : 9-40.

[116] Llorà, X. and Goldberg, D.E. Wise Breeding GA via Machine Learning Techniques for Function Optimization. Proc. of Genetic and Evolutionary Computation Conference (GECCO-03) (2003) : 1172-1183.

[117] Miquélez, T., Bengoetxea, E., and Larrañaga, P. Evolutionary Computation Based on Bayesian Classifiers. International. Journal of Applied Mathematics and Computer Science 14/3 (2007) : 101-115.

[118] Miquélez, T., Bengoetxea, E., and Larrañaga, P. Combining Bayesian Classifiers and Estimation of Distribution Algorithms for Optimization in Continuous Domains. Connection Science 19/4 (2007) : 297-319.

[119] Pelikan, M., Sastry, K., and Goldberg, D.E. iBOA: The Incremental Bayesian Optimization Algorithm. Proc. of the Genetic and Evolutionary Computation Conference (GECCO08) (2008) : 445-456.

[120] Yang, C., and Simon, D. A New Particle Swarm Pptimization Technique. Proc of the International Conference on Systems Engineerin 2005 (2005) : 164-169.

[121] Liu, Y., Yao, X., and Higuchi, T. Evolutionary Ensembles with Negative Correlation Learning. IEEE Transaction on Evolutionary Computation. 4 (September 2000) : 380–387.

[122] Liu, Y., and Yao, X. Ensemble Learning via Negative Correlation. Neural Networks 12 (1999) : 1399–1404.

[123] Kargupta, H., Deb, K., and Goldberg, D.E.Ordering genetic algorithms and deception. Parallel Problem Solving form Nature 2(1992) : 47-56.

[124] Tae, K.S. and Lee, S.S. On cognitive role of negative schema. Lecture Notes in Computer Science Springer 2006.

[125] Andrieu, C. et al. An Introduction to MCMC for Machine Learning. Machine Learning 50(2003) : 5-43.

[126] Peña, J.M., Lozano, J.A., and Larrañaga, P. Globally Multimodal Problem Optimization Via an Estimation of Distribution Algorithm Based on Unsupervised Learning of Bayesian Networks. Journal of Evolutionary Computation 13/1 (January 2005) : 43-66.

[127] De Jong, K.A. An Analysis of the Behavior of a Class of Genetic Adaptive Systems. Doctoral dissertation, University of Michigan. 1975.

[128] Barahona, J., and Fonseca. The Magic Square as a Benchmark: Comparing MIP to Improved GA and to a High Performance Minimax AI Algoritm. Proc. of WSEAS Evolutionary Computation Conference (2005) : 486-492.

[129] Barahona, J., and Fonseca. From the Magic Square to the Optimization of Networks of AGVs and from MIP to a Hybrid Algorithm and from this one to the Evolutionary Computation. Proc. of the 5th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases (2006) : 117-122.

[130] Heckman, I. Empirical Analysis of Solution Guided Multi-Point Constructive Search. Master Thesis in Computer Science. Department of Computer Science, University of Toronto, 2007.

[131] Watkins, J.J. Across the Board: The Mathematics of Chess Problems. Princeton: Princeton University Press, 2004.

[132] Božikovic, M., Golu, M., and Budin, L. Solving n-Queen Problem Using Global Parallel Genetic Algorithm. Proceedings of Eurocon 2003 (2003) : 104-107.

[133] Martinjak, I., and Golub, M. Comparison of Heuristic Algorithms for the N-Queen Problem. Proc. of International Conference on Information Technology Interfaces (ITI 2007). (2007) : 759 – 764.

[134] Zeng, C., and Gu, T. A Novel Assembly Evolutionary Algorithm for n-Queens Problem. Proc. of International Conference on Computational Intelligence and Security Workshops (CISW 2007). (2007) : 171-174.

[135] Borrel, R. A Brute Force Approach to solving the Knight's Tour Problem using Prolog. Proc. of The World Congress in Computer Science, Computer Engineering and Applied Computing (2009).

[136] Hingston, P., and Kendall, G. Enumerating Knight's Tours using an Ant Colony Algorithm Proc. of the IEEE Congress on Evolutionary Computation 2005 (2005).

[137] Cordon, O. et al. A Review on the Ant Colony Optimization Metaheuristic: Basis, Models and New Trends. Mathware and Soft Computing 9 2/3 (2002) : 141-175.

[138] Gordon, V.S., and Slocum, T.J. The Knight's Tour – Evolutionary vs. Depth-First Search Proc. of the IEEE Congress on Evolutionary Computations 2004 (2004).

[139] Al-Gharaibeh, J., Qawagneh, Z., and Al-zahawi, H. Genetic Algorithms with Heuristic – Knight's Tour Problem. Proc. of International Conference on Genetic and Evolutionary Methods 2007 (2007).

[140] Takefuji, Y. and Lee, K.-C. Finding Knight's Tours on an MxN Chessboard with O(MN) hysteresis McCulloch-Pitts Neurons. IEEE Transactions on Systems, Man and Cybernetics 24/2 (1994) : 300-306.

[141] Warnsdorff, H.C.V. Des Rösselsprungs einfachste und allgemeinste Lösung Schmalkalden (1823).

[142] Parberry, I. An Efficient Algorithm for the Knight's Tour Problem. Discrete and Applied Mathematics 73 (1997) : 251-260.

[143] Löbbing, M. and Wegener, I. The Number of Knight's Tours equals 33,439,123,484,294 – Counting with Binary Decision Diagrams. Electronic Journal of Combinatorics. 3/1 (1996) : R5.

[144] McKay, B.D. Knight's tours of an 8x8 chessboard. Doctoral Thesis. Department of Computer Science, Australian National University. 1997.

[145] Mordecki, E. On the number of Knight's tours. Pre-publicaciones de Matematica de la Universidad de la Republica, Uraguay (2001).

[146] Wattanapornprom, W., and Chongstitvatana, P. Multiobjective Combinatorial Optimization with Coincidence Algorithm. Proc. of IEEE Congress on Evolutionary Computation 2009 (CEC2009) (2009).

[147] Kirkpatrick, S.et al. Configuration Space Analysis of Traveling Salesman Problem. Journal Physique 46 (1985) : 1277–1292.

[148] Shirrish, B., Nigel, J., and Kabuka, M.R. A Boolean Neural Network Approach for the Traveling Salesman Problem. IEEE Transactions on Computers 42 (1993) : 1271–1278.

[149] Glover, F. Artificial Intelligence. Heuristic Frameworks and Tabu Search. Managerial & Decision Economics 11(1990) : 365–378.

[150] Wong, R. Integer Programming Formulations of the Traveling Salesman Problem. Proc. of IEEE International Conference of Circuits and Computers. (1980) : 149–152.

[151] Robles, V., and Larrañaga P. Solving the Traveling Salesman Problem with EDAs. In Estimation of Distribution Algorithm: A New Tool for Evolutionary Computation. (2006)

[152] Bixby, B., and Reinelt, G. 2008 [Online] Available from: http://softlib.rice.edu/tsplib.html. [2009, August 12]

[153] Mühlenbein, H. The Equation for Response to Selection and Its Use for Prediction. Evolutionary Computation, 5 (1998) : 303-346.

[154] Chow, C., and Liu, C. Approximating Discrete Probability Distributions with Dependency Trees. IEEE Transactions on Information Theory, 14 (1967) : 462-467.

[155] Etxeberria, R., and Larranga, P. Global Optimization with Bayesian Networks. Proc. of the 2nd Symposium on Artificial Intelligence. (CIMAF99). Special Session on Distributions and Evolutionary Optimization. (1999) : 322-339.

[156] Larrañaga, P., Kujipers, C.M. H., Murga, R.H., Inza, I., and Dizdarevic, S. Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. Artificial Intelligence Review 13 (1999) : 129-170.

[157] Kumar, R., and Singh, P.K. Pareto Evolutionary Algorithm Hybridized with Local Search for Biobjective TSP. Studies in Computational Intelligence 75 (2007) : 361-389.

[158] Jackson, J.R. A Computing Procedure for a Line Balancing Problem. Management Science 2/3 (1956) : 261-271.

[159] Hwang, R. K., Katayama, H., and Gen, M. U-shaped assembly line balancing problem with genetic algorithm. International Journal of Production Research, 46/16 (2008) : 4637-4649.

[160] Sirovetnukul, R. Multi-Objective Worker Allocation in U-Shaped Assembly Lines. Doctoral dissertation Faculty of Engineering, Chulalongkorn University 2010.

[161] Kidarn, S. Application of Mimetic Algorithms for Multi-Objective Balancing Problem on Mixed-Model U-Shaped Assembly Line with Parallel Workstation in JIT Production Systems. Master thesis Faculty of Engineering, Chulalongkorn University 2010.

[162] Jirakomate, C. Application of Mimetic Algorithms for Multi-Objective Worker Allocation in U-Shaped Assembly Lines. Master thesis Faculty of Engineering, Chulalongkorn University 2010.

[163] Chimklay, P. Application of Particle Swarm Optimization Algorithms for Multi-Objective Balancing Problem on Mixed-Model Two-Side Assembly Line. Master thesis Faculty of Engineering, Chulalongkorn University 2010.

[164] Olanwichai, P. Application of Mimetic Algorithms for Multi-Objective Balancing Problem on Mixed-Model U-Shaped Assembly Line in JIT Production Systems. Master thesis Faculty of Engineering, Chulalongkorn University 2008.

[165] Kampirom, N. Application of Mimetic Algorithms for Multi-Objective Sequencing Problem on Mixed-Model U-Shaped Assembly Line in JIT Production Systems. Master thesis Faculty of Engineering, Chulalongkorn University 2008.

VITA

Mr. Warin Wattanapornprom was born on September 24, 1978 in Bangkok, Thailand. He earned his Bachelor of Science in Information Technology from Sirindhorn International Institute of Technology, Thammasat University in 1999. After he graduated his Master degree in Computer Science from Chulalongkorn University in 2003, he had worked as a Chief Information Officer (CIO) at Thai Ocean Industries Co.,Ltd. His research interests are metaheuristics, supply chain management, parallel computing, robotics, production line balancing, sequencing and scheduling.

During his doctoral study at Chulalongkorn University, he presented two international conference papers at the IEEE International Congress on Evolutionary Computation (CEC2008) at Norway in 2008 and Genetic and Evolutionary Computation Conference (GECCO2011) at Ireland in 2011. At present, he works as a Chief Information Officer at Thai Ocean Industries Co.,Ltd. His email address is yongkrub@gmail.com. His mobile phone is 080-935-6789.