การเรียนวิธีการเลือกตัวแปรทวิภาคเพื่อเพิ่มประสิทธิภาพของเวลาที่ใช้หาผลเฉลยของ ปัญหากำหนดการเชิงเส้นจำนวนเต็มผสมในการหาค่าเหมาะที่สุดของการจัดวางผังอาคารทางสถาปัตยกรรม

นายกมล เกียรติเรืองกมลา

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาวิทยาการคอมพิวเตอร์ ภาควิชาคณิตศาสตร์

คณะวิทยาศาสตร์

จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2550

LEARNING BINARY VARIABLES SELECTIONS TO IMPROVE THE MIP SOLUTION TIME IN ARCHITECTURAL LAYOUT DESIGN OPTIMIZATION

Mr. Kamol Keatruangkamala

A Dissertation Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctoral of Philosophy Program in Computer Science
Department of Mathematics
Faculty of Science
Chulalongkorn University
Academic Year 2007

Thesis Title	LEARNING BINARY VARIABLE SELECTIONS TO IMPROVE THE MIP SOLUTION TIME IN ARCHITECTURAL LAYOUT DESIGN OPTIMIZATION
By	Mr. Kamol Keatruangkamala
Field of Study	Computer Science
Thesis Advisor	Assistant Professor Krung Sinapiromsaran, Ph.D.

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

[illegible] Dean of the Faculty of Science
(Professor Supot Hannongbua, Ph.D.)

THESIS COMMITTEE

[illegible] Chairman
(Professor Chidchanok Lursinsap, Ph.D.)

[illegible] Thesis Advisor
(Assistant Professor Krung Sinapiromsaran, Ph.D.)

[illegible] Member
(Associate Professor Somchai Prasitjutrakul, Ph.D.)

[illegible] Member
(Associate Professor Ekachai Leelarasmee, Ph.D.)

[illegible] External Member
(Acharawan Chutarat, Ph.D.)

กมล เกียรติเรืองกมลา : การเรียนวิธีการเลือกตัวแปรทวิภาคเพื่อเพิ่มประสิทธิภาพของเวลาที่ใช้หาผลเฉลยของปัญหากำหนดการเชิงเส้นจำนวนเต็มผสมในการหาค่าเหมาะที่สุดของการจัดวางผังอาคารทางสถาปัตยกรรม. (LEARNING BINARY VARIABLE SELECTIONS TO IMPROVE THE MIP SOLUTION TIME IN ARCHITECTURAL LAYOUT DESIGN OPTIMIZATION) อ.ที่ปรึกษา : ผศ.ดร.กรุง สินอภิรมย์สราญ, 111 หน้า.

เทคนิคการแก้ปัญหาการหาค่าเหมาะที่สุดด้วยวิธีการต่างๆ ได้ถูกนำมาใช้แก้ปัญหาการหาค่าเหมาะที่สุดของการจัดวางผังอาคารทางสถาปัตยกรรมมากว่าทศวรรษ อาทิเช่น ระบบผู้เชี่ยวชาญ, ขั้นตอนวิธีวิวัฒนาการ, การจำลองการอบเหนียว และวิธีกำหนดการเชิงคณิตศาสตร์ วิทยานิพนธ์นี้เน้นเทคนิคการแก้ปัญหากำหนดการเชิงคณิตศาสตร์ด้วยวิธีการแก้ปัญหาจำนวนเต็มผสม (MIP) เพื่อใช้แก้ปัญหาการหาค่าเหมาะที่สุดของการจัดวางผังอาคารทางสถาปัตยกรรม เรียก AL-MIP โดยความสัมพันธ์ที่ไม่เชิงเส้นระหว่างองค์ประกอบของการออกแบบถูกอธิบายด้วยสมการและอสมการเชิงเส้นที่สอดคล้องกัน สืบเนื่องจากลักษณะธรรมชาติเชิงการจัดของผลเฉลย MIP วิธี AL-MIP สามารถแก้ปัญหาการหาค่าเหมาะที่สุดสำหรับปัญหาขนาดเล็ก (2-5 ห้อง) ภายในระยะเวลาจำกัดที่ยอมรับได้ เพื่อที่จะจัดการกับสถานการณ์นี้ สองอสมการอย่างสมเหตุสมผล (valid inequalities) เรียก AL-MIP+ ที่ได้จากการเชื่อมต่อแบบไม่วนกลับของลำดับห้องที่เรียงต่อกัน และการกำหนดความพึงพอใจของสถาปนิก ได้ถูกนำมาใช้เพื่อลดเวลาการคำนวณอย่างมีนัยสำคัญ นอกจากนี้เพื่อเพิ่มความเร็วการคำนวณของ AL-MIP+ การเรียนรู้ด้วยเครื่องที่ใช้ขั้นตอนวิธีพันธุกรรม (GA) ได้ถูกนำมาประยุกต์ใช้ เพื่อหาลำดับของตัวแปรแตกกิ่งที่เหมาะสมของ เซตอันดับพิเศษ (SOS) เรียก AL-MIP+GA การลดลงของปริภูมิการค้นหามาจากการตัดทอนจำนวนการค้นหาด้วยการใช้ผลลัพธ์ที่ดีกว่าของเซตอันดับที่เหมาะสม เ ทคนิคการเพิ่มความเร็วด้วยวิธีการเหล่านี้ แสดงให้เห็นจำนวนรอบของการคำนวณและเวลาที่ลดลงมากกว่าร้อยละ 80 ซึ่งประสบความสำเร็จสำหรับปัญหาขนาดกลาง (5-10 ห้อง) ผลเฉลยที่เหมาะที่สุดที่เป็นไปได้จากรูปแบบของห้องขนาด 10 ห้องสามารถแก้ได้ภายในเวลาไม่กี่นาที การผสานสองวิธีระหว่างอสมการอย่างสมเหตุสมผลและการเรียนรู้ด้วยเครื่อง เป็นการเสนอแนวทางใหม่เพื่อแก้ปัญหาการหาค่าเหมาะที่สุดสำหรับการจัดวางผังการออกแบบทางสถาปัตยกรรม

ภาควิชา	คณิตศาสตร์	ลายมือชื่อนิสิต [illegible]
สาขาวิชา	วิทยาการคอมพิวเตอร์	ลายมือชื่ออาจารย์ที่ปรึกษา [illegible]
ปีการศึกษา	2550	

##4673802723 : MAJOR COMPUTER SCIENCE
KEYWORDS : ARCHITECTURAL LAYOUT DESIGN / MIXED INTEGER PROGRAMMING / BRANCH AND BOUND ALGORITHM / LEARNING VARIABLES / GENETIC ALGORITHM

KAMOL KEATRUANGKAMALA: LEARNING BINARY VARIABLES SELECTIONS TO IMPROVE THE MIP SOLUTION TIME IN ARCHITECTURAL LAYOUT DESIGN OPTIMIZATION. THESIS ADVISOR: ASST. PROF. KRUNG SINAPIROMSARAN, Ph.D., 111 pp.

Varieties of optimization techniques have been used to solve an architectural layout design optimization for more than a decade such as an expert system, an evolutionary algorithm, a simulated annealing and a mathematical programming method. This thesis will concentrate on the mathematical programming technique that formulates an architectural layout design optimization as the architectural layout Mixed Integer Programming (MIP) model called AL-MIP. All non-linear relationship among design components will be captured using the corresponding linear equalities and linear inequalities. Due to the combinatorial nature of the MIP solutions, the AL-MIP can be solved optimally for a small size, (2-5 rooms), within a reasonable time limit. To remedy this situation, both valid inequality constraints called AL-MIP+ from non-circular connectivity of consecutive room connections and the architect's preference constraints have been adopted that reduces the computational time significantly. Moreover, to speed up the computational time of AL-MIP+, the machine learning using Genetic Algorithm (GA) has been applied to determine the best sequences of branching variables, the Special Order Set (SOS) called AL-MIP+GA. The search space reduction comes from the better candidate solution used to prune the search tree. These combinations of speeding up technique illustrate the computational MIP iterations and time reduction more than 80% that is now achievable for a medium size (5-10 rooms). The global solutions from 10 room patterns have been solved within a few minute. Indeed, both valid inequality MIP and learning methodology present a novel mathematical concept to optimize MIP for an architectural layout design problem.

Department	Mathematic	Student's signature ......[illegible]......
Field of study	Computer Science	Advisor's signature ......[illegible]......
Academic year	2007	

Acknowledgements

This work can not complete in this present stage without numerous contributions, and help from many people. I would like to use this invaluable section to express my gracious gratitude to all of them.

First and foremost, I would like to thank my advisor, Asst. Prof. Dr. Krung Sinapiromsaran, for his through guidance and support throughout the research, his whole-hearted efforts in providing the necessary help and many useful discussions and suggestions for my dissertation and his encouragement in many ways. He has shown me the impression of Mathematical programming since my early days of computer science education several year ago. With gratitude I would also acknowledge my dissertation committee Prof. Dr. Chidchanok Lursinsap, Assoc. Prof. Dr. Somchai Prasitjutrakul, Assoc. Prof. Dr. Ekachai Leelarasmee, and Asst. Prof. Dr. Acharawan Chutarat for their valuable times of reading and making constructive comments on this dissertation.

Most important, I would like to thank the faculty of architecture, Rangsit university for the financial support during my study here.

Finally, I would like to thank my parents (Mrs. Saraya S. and Mr. Jirasak K.), my sister and my brother (Ms. Arpaporn K. and Mr. Chalearmchai K.), Miss Hong Raksakulkeat, Miss Keerati Boonchote, Miss Aujjima Tritanon and my college friends Mr. Sittisak S., Mr. Yuttana L., Miss Kodchakorn N., Mr. Matee B. and Mr. Thirawut K. for supporting, inspiring, and understanding throughout my academic years.

Contents

List of Tables

List of Figures

Figure Page

CHAPTER I
Introduction

1.1 Background and Motivation

Architectural layout design is one of the most important and complex parts of any architectural design process. In order to design a layout that responds to most of its related requirements, architects spend much time and effort on studying specific layouts and all existing relationships within rooms and among the interior and exterior spaces. Besides the artistic aspect of an architectural design, there is a substantial logical process behind the layout design phase. Architects cannot avoid having a large number of trial and error to reach that step. The combinatorial complexity of most architectural layout design problems also makes it practically impossible to obtain a systematic knowledge for all possible solutions.

Architectural layout design is an initial phase of a design process during which the architect takes the specification of spatial objects and generates numerous feasible drafts. It is the most critical phase which influents the final designed decision. This architectural layout design can be interpreted as solving a combinatorial problem. By which, solution methodologies for architectural layout design present the most comprehensive challenges in the area of architectural design computation due to the arrangement of all possible connections of *n* connected rooms. This combinatorial problem is known to be NP-hard (Michalek et al., 2002, Russell et al., 1999), see figure. 1.1.

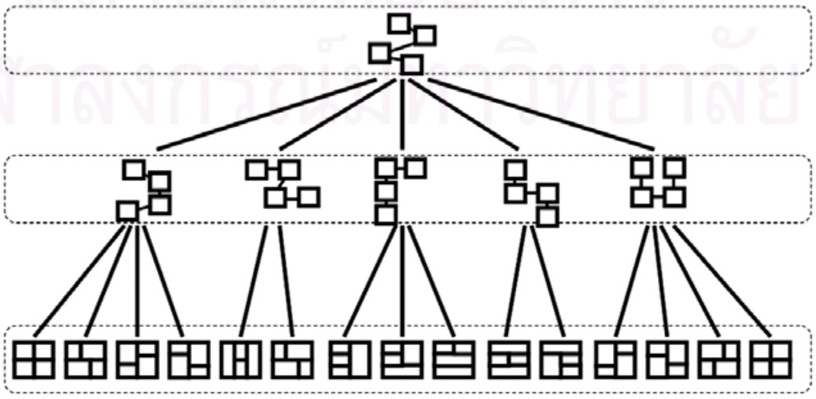


Figure 1.1: Conceptually, a huge possible placements can adjusted in a variety of ways.

As noted by Yoon (1992), the architectural layout design problem tends to be ill-defined and over-constrained. Simon (1973) identified the ill-defined behavior of the architectural layout design problem as the incomplete formulation to be used to solve in the initial try. Resolving ill-defined problem is a process of searching for and refining a set of design constraints. Moreover, the over-constrained problem is due to many possible solutions and repeated constraints (Balachandran and Gero, 1987). Hence, an architectural layout design problem needs a method of providing an optimal solution from a large set of possible solutions, and a method of allowing architects to modify the set of design constraints to continually refine the problem definition (Arvin and House, 2000). However, Tsang (1995) showed that there is no a universal best algorithm that certain algorithms may be preferred under certain circumstances.

Many researchers seek to automate the process of architectural layout design problem using several representations and solution techniques. Nevertheless, architectural layout design problem is not easily dealt with. The Interactive Layout Design Optimization (Michalek and Papalambros, 2002) reported a couple of days to solve the problem of ten rooms which is impractical to incorporate in the CAD system.

From the past decades, many previous attempts have been used to deal with this problem such as the wall representation (Flemming, 1978, Simon, 1973), non-linear programming (Imam et al, 1989, Medjdoub et al., Tang et al., 2000) and the evolutionary method (Damski et al., 1997, Michalek et al., 2002, Gero et al., 1998). There are various difficulties with each approach. The wall representation uses the special data structure to generate the linear programming subproblem which requires a special algorithm. The nonlinear programming approach guarantees only local optimal (Cagan el at., 1998, Michalek el at., 2002). The evolutionary method can only guarantee the convergence with a long running time (Jo and Gero, 1998).

The structural representation (Bloch et al., 1978, Gero, 1990, Honda et al.,1995, Schwarz et al., 1994) of a spatial requirement is needed to form the basic component of a physical design problem to be automatically solved by computer. One representation used a grid system, see figure 1.2(a). This representation is inherently discrete and multi-modal. Due to the combinatorial configurations, it cannot be solved exhaustively for reasonable-sized layout. The grid allocation approach is a successful approach for allocating a predefined space into rooms or activities. This approach can be used to redistribute activities in an office building during a reorganization. Liggett and Mitchell (1981) used a constructive placement strategy on the grid system where a room space is allocated one at a time. Then the iterative improvement based on the objective function has been used to improve the current solution.

Another structural representation is that of the Flemming wall (Flemming, 1978, Simon, 1973) identified the location of walls in the space to partition a layout into rectangular components, see figure 1.2(b). This structural representation has an advantage over the grid-based layout by limiting nonrectangular shapes of space patterns which help reduce the computational time.

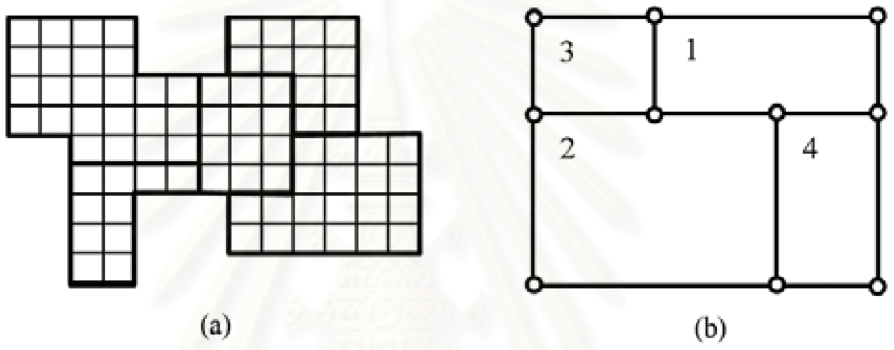


Figure 1.2: (a) Grid system and (b) Dissections based on wall-representation.

The primary structural representation used in this thesis is based on a mathematical programming similar to the work from Bloch (1978), et. al. using a coordinated system. Michalek, Choudhary and Papalambros (2002) constructed an optimization model of the quantifiable aspects that determines the best location and size of a group of interrelated rectangular spaces using a middle coordinate (x, y) of each room. This allows an optimization algorithm to alter a position of a room independently to achieve the optimal cost satisfying all architectural design requirements.

In this thesis, we develop the Mixed Integer Programming (MIP) model (Grorge, 1988, Linderoth et al., 1999, Russell et al., 1999) to determine the optimal multiobjective architectural layout design called AL-MIP. The advantage of MIP model presents an easy adaptability for other architectural requirements. This AL-MIP has been formulated to reduce the search space. Also, we narrowed the search space by allowing architect to specify additional reduction constraints such as the fixed room location, the unused grid cells, the fixed border location and the favorable choice of the nearest room to the top left corner. These formulations allow architects to design a layout beyond the rectangular boundary (Scott et al., 1999). To deal with a medium-sized problem (5-10 rooms), we resolve the problem by adding two valid inequalities based on the mathematical programming technique called AL-MIP+. These two valid inequalities consist of a non-circular connectivity constraint and an advised configuration constraint. The non-circular connectivity constraints utilize two binary variables p_{ij} and q_{ij} from the AL-MIP which causes the reduction of the feasible region while the advised configuration constraints utilize a mathematical inequalities based on the architect's preference to suggest a room configuration in North, South, East and West directions. AL-MIP+ abandons alternative solutions while maintain the final objective value by incorporating the choice of the first room to be placed near the top-left corner in the objective function. These two inequalities significantly present the reduction of computational iterations and time. In order to tackle the medium-sized problem efficiently, the machine learning has been adopted to learn a Special Order Set (SOS) in the branch and bound algorithm, called AL-MIP+GA. The robustness learning methodology Genetic Algorithm (GA) is applied to SOS variables of the branch and bound algorithm for finding the better candidate solution which will be stored into a computer as a preprocess of the branching node in the search tree. Therefore, the computational iterations and time are drastically reduced for the medium-sized problem of 10 rooms that can be solved within a few minutes.

To practically apply AL-MIP, AL-MIP+ and AL-MIP+GA model, this thesis has been developed the software named ALDO (Architectural Layout Design Optimization) to help an architect identifying the layout requirements graphically (Keatruangkamala and Sinapiromsaran, 2005), see figure 1.3. This software utilizes the graphic user interface (GUI) running on the Windows operating system and automatically solving the architectural layout instance. Furthermore, architect can request a drawing presentation of the global optimal solution or save it as a DXF format file to use with other CAD software.

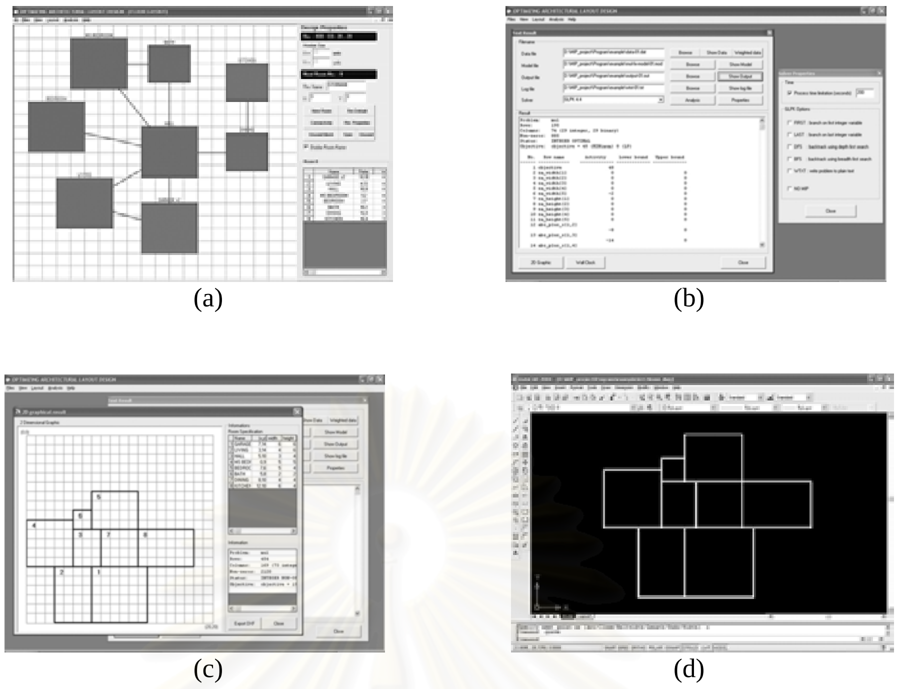


Figure 1.3: The initial software interface (a) for diagram input, (b) and (c) for the text and graphical output and (d) the export DXF file from CAD software

1.2 Contributions

The following results are expected from this research:

- To reduce the computational time using the valid inequalities.
- To reduce the computational time of the branch and bound using the Special Order Set (SOS) variables in improving the MIP solution time.

The results will contribute to the derivation of proposes of improving the computational iterations and time for the architectural layout design optimization problem.

1.3 Dissertation Organization

The organization of this dissertation is as follows. Chapter II reviews the theoretical backgrounds and related works. Chapter III presents an overview of the proposed model based on the MIP methodology. Chapter IV clarifies the concepts behind the proposed learning SOS variables using the GA. An experiment is provided in chapter V. Finally, some concluding remarks and suggestions are summarized in chapter VI.

CHAPTER II
Theoretical Background

This chapter provides summary of important theoretical backgrounds that are required in this thesis. It contains two main sections, the mathematical programming model and the machine learning algorithm.

First, we introduce the Mixed Integer Programming (MIP) model based on the linear programming (LP) model to solve an architectural layout design problem. Moreover, we also introduce the valid inequality constraints that can be used to reduce the feasible area of the problem.

Second, we describe the machine learning algorithm using Genetic Algorithm (GA) which helps to reduce the computational iterations and time.

2.1 Fundamentals of a Mathematical Programming

2.1.1 Linear Programming

LP model is concerned with the optimization (minimization or maximization) of linear function while satisfying a set of linear equality or inequality constraints or restrictions. The LP model solved by Simplex algorithm was first conceived by George B. Dantzig around 1947 while he was working as a mathematical advisor to the United States Air Force Controller on developing a mechanized planning tool for a time-staged deployment, training, and logistical supply program. Although the soviet mathematician and economist L. V. Kantorovich formulated and solved a problem of this type dealing with organization and planning in 1939, his work remained unknown until 1959. Hence concept of the general class of LP model solved by Simplex algorithm is usually credited to Dantzig.

Nevertheless, LP model is widely utilized with the use or allocation of limited resources as a labor, a material and a capital in the best possible manner so that cost is minimized/maximized. An LP model is an optimization problem in which the objective function and constraints are expressed as linear function based on the canonical form.

- **The Canonical Form**

There are various forms to represent an LP model. In this thesis, we consider the canonical form with *m* constraints and *n* nonnegative constraints

$$\begin{array}{lll} \text{maximize} & c_1x_1 + c_2x_2 + \ldots + c_nx_n & \\ \text{subject to} & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \le b_1 & \\ & a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \le b_2 & \\ & \vdots & \\ & a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \le b_m & \\ & x_1, x_2, \ldots, x_n \ge 0 & \end{array} \tag{2.1}$$

where a_{ij}, for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$ are the coefficients of the constraints. c_1, c_2, …, c_n are the coefficients of the objective function for nonnegative unknown (decision) variables, x_1, x_2, … , x_n, respectively. b_1, b_2, … , b_m are the right-side constraints.

In matrix-vector notation, the above canonical form can be written in compact form as:

$$\begin{array}{ll} \text{maximize} & \boldsymbol{c}^T\boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} \le \mathbf{b} \\ & \boldsymbol{x} \ge \mathbf{0} \end{array} \tag{2.2}$$

where $\boldsymbol{A}$ is an $m \times n$ matrix called the coefficient matrix, $\boldsymbol{c}$ is an $n \times 1$ column vector called the cost vector, $\boldsymbol{x}$ is an $n \times 1$ column vector called the decision vector and $\boldsymbol{b}$ is an $m \times 1$ column vector called the right-side vector.

In general, we can convert any LP to the canonical form. Note that the canonical form requires maximizing the objective function. For the minimized optimization direction, we multiply the objective function by -1 to reverse its direction, changing the minimizing problem to the maximizing problem. The optimal solutions of both the maximization problem and the minimization problem are the same, while their optimal values will differ by a negative sign.

$$-\,maximize(-\boldsymbol{c}^T\boldsymbol{x}) = minimize(\boldsymbol{c}^T\boldsymbol{x}) \tag{2.3}$$

In the next part, we will introduce some terminologies for finding the solution of the LP model.

- **Feasible Region, Optimal Solution and Extreme Point**

For any linear programming model, we are interested in determining the optimal value for the objective function.

Definition 1: (Feasible Region)

Given an LP in its canonical form (2.2), the feasible region F is the set of all solutions that satisfy all the constraints of the LP.

$$F = \{ \boldsymbol{x} \in R^n \mid \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq 0 \}. \tag{2.4}$$

A solution in the feasible region of the LP is said to be the feasible solution. Suppose that there are feasible solutions, the goal of the LP is to find the optimal feasible solution, as measured by the value of the objective function.

Definition 2: (Optimal Solution)

Consider an LP model if the feasible region is not empty, an optimal solution is a feasible solution that has the largest value of the objective function for the maximization problem. Let x^* be an optimal solution to the LP model.

$$\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}^* \geq \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}\ , \forall \boldsymbol{x} \in F \tag{2.5}$$

The value of the objective function corresponding to an optimal solution is called the optimal value.

Definition 3: (Extreme Point)

A point $\boldsymbol{x}$ in a convex set S is called an extreme point of S, if $\boldsymbol{x}$ cannot be represented as strict convex combination of two district points in S. In order words, if $\boldsymbol{x} = \lambda\, \boldsymbol{x}_1 + (1 - \lambda)\boldsymbol{x}_2$ with $\lambda \in (0,1)$ and $\boldsymbol{x}_1, \boldsymbol{x}_2 \in S$, then $\boldsymbol{x} = \boldsymbol{x}_1 = \boldsymbol{x}_2$.

Any LP in its canonical form (2.2) must be in one of the following four cases:

1. LP has the unique optimal solution.

   This unique optimal solution must be an extreme point.

2. LP has alternative optimal solutions.

   If there are two extreme points $\boldsymbol{x}^*_1$ and $\boldsymbol{x}^*_2$, then a convex combination of $\boldsymbol{x}^*_1$ and $\boldsymbol{x}^*_2$ is also optimal.

3. LP is unbounded.

   For a maximization problem, the feasible region is unbounded and the plane $\boldsymbol{c}^T\boldsymbol{x} = \boldsymbol{z}$ can be increased along the unbounded direction of the feasible region. In this case, the objective value is unbounded and no optimal solution exists.

4. LP has an empty feasible region.

   In this case, the system of equations and/or inequalities defining the feasible region is inconsistent. This means there is no point satisfying all constraints of the LP.

The following example illustrates the two dimensional LP problem solved by simplex method. All extreme points are illustrated in figure 2.1.

Example 2.1. Consider the following LP problem:

$$\begin{array}{lrl} \text{maximize} & 40x_1 + 36x_2 & \\ \text{subject to} & x_1 & \leq 8 \\ & x_2 & \leq 10 \\ & 5x_1 + 3x_2 & \leq 45 \\ & x_1 & \geq 0 \\ & x_2 & \geq 0 \end{array}$$

The intersection of five half spaces gives the feasible region as follow:

$$F = \left\{ \boldsymbol{x} \in R^2 | \; x_1 \leq 8, x_2 \leq 10, 5x_1 + 3x_2 \leq 45, x_1 \geq 0, x_2 \geq 0 \right\}$$

Clearly the set is a convex set and its extreme points are given as:

$$\boldsymbol{a} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \; \boldsymbol{b} = \begin{bmatrix} 8 \\ 0 \end{bmatrix}, \; \boldsymbol{c} = \begin{bmatrix} 8 \\ 5/3 \end{bmatrix}, \; \boldsymbol{d} = \begin{bmatrix} 3 \\ 10 \end{bmatrix}, \text{ and } \boldsymbol{e} = \begin{bmatrix} 0 \\ 10 \end{bmatrix}$$

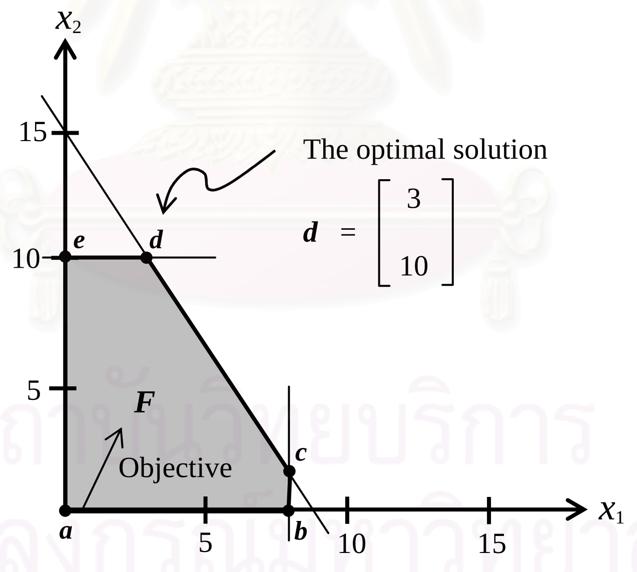


Figure 2.1: Feasible region (*F*) and extreme points (***a***, ***b***, ***c***, ***d*** and ***e***).

After solving this LP problem, we get the unique optimal solution at $x_1 = 3$, $x_2 = 10$ and the extreme point is ***d***. In the next section, we will describe the MIP which is the linear programming problem with integrality constraints.

2.1.2 Mixed Integer Programming

From the previous section, an LP model deals with a linear objective function subject to a set of linear constraints. However, in numerous applications, it may be necessary to specify that certain variables assume to have integral values. These problems can be solved using the branch and bound algorithm which will be described in details later.

An MIP is an optimization problem where some or all variables are restricted to take only integral values. General integer and MIP is NP-hard, even today's state of the art commercial LP solvers have difficulties solving MIP formulations representing engineering or business optimization models containing more than a few hundred integer variables.

Typically, the integer LP model (George and Laurence, 1988) is simply a linear program (LP) in which all variables are restricted to integral values. Nevertheless, we will refer to this problem simply as an LP model because the term linear is seldom used except to contrast a problem with an integer nonlinear programming problem. If all variables must assume only integral values, it is called a pure integer programming problem. While some variables are restricted to integral values and others remain continuous, then it refers as an MIP problem.

$$
\begin{array}{llll}
\text{(IP)} & \text{maximize} & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\
 & \text{subject to:} & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\
 & & \boldsymbol{x} \geq \mathbf{0}, \\
 & & \boldsymbol{x} \text{ is integer}
\end{array}
\qquad
\begin{array}{llll}
\text{(MIP)} & \text{maximize} & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{d}^{\mathrm{T}}\boldsymbol{y} \\
 & \text{subject to:} & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{y} \leq \boldsymbol{b} \\
 & & \boldsymbol{x} \geq \mathbf{0}, \\
 & & \boldsymbol{x} \text{ is integer} \\
 & & \boldsymbol{y} \geq \mathbf{0}
\end{array}
$$

where ***x*** and ***y*** are vectors of design variables, ***A*** and ***D*** are matrices.

Generally, an MIP model is an optimization model that can be stated mathematically as follows:

$$\begin{aligned} &\text{maximize} && z_{MIP} = \sum_{j\in I} c_j x_j + \sum_{j\in C} c_j x_j \\ &\text{subject to} && \sum_{j\in I} a_{ij} x_j + \sum_{j\in C} a_{ij} x_j \le b_i \quad , i = 1, \ldots, m \qquad (2.6) \\ & && l_j \le x_j \le u_j \quad , j \in N \\ & && x_j \in Z \quad , j \in I \end{aligned}$$

where I is the set of integer variables, C is the set of continuous variables, and $N = I \cup C$. The lower and upper bounds l_j and u_j may take on the values of plus or minus infinity. Thus, an MIP model is an LP model plus some integrality restriction on some or all variables.

2.1.3 Valid Inequality Constraints

Related to our research of MIP model, we aim to reduce the computational iterations and time. This section, we utilize a valid inequality for the constraint set of AL-MIP model. The construction of families of valid inequalities is more of an art than a formal methodology (George and Laurence, 1988). To describe an idea of valid inequalities, the following statements have been shown in details based on a given formulation *P*.

The valid inequalities definition:

The inequality denoted by (π, π_0) is called a valid inequality for *P* if $\boldsymbol{\pi x} \le \pi_0$, $\forall \boldsymbol{x} \in P$.

Note that, if (π, π_0) is a valid inequality. Then, the formulation *P* lies in the half-space $\{\boldsymbol{x} \in R^n : \boldsymbol{\pi x} \le \pi_0\}$ and $\max\{\boldsymbol{\pi x} : \boldsymbol{x} \in P\} \le \pi_0$.
or we can describe another definition with the face of *P* as follow.

If (π, π_0) is a valid inequality for the formulation *P* and $F = \{\boldsymbol{x} \in P : \boldsymbol{\pi x} = \pi_0\}$,
F is called a face of *P* and we say that (π, π_0) represents or defines *F*.

Note that, a face is said to be proper if $F \ne \emptyset$, and $F \ne$ P. Then, the face represented by (π, π_0) is nonempty and $\max\{\boldsymbol{\pi x} : \boldsymbol{x} \in P\} = \pi_0$. and if the face *F* is nonempty, we say it supports *P*. The set of optimal solutions to an LP is always a face of the feasible region.

In order to describe an idea of valid inequalities, the following figure illustrates a smaller feasible region from the valid inequalities.

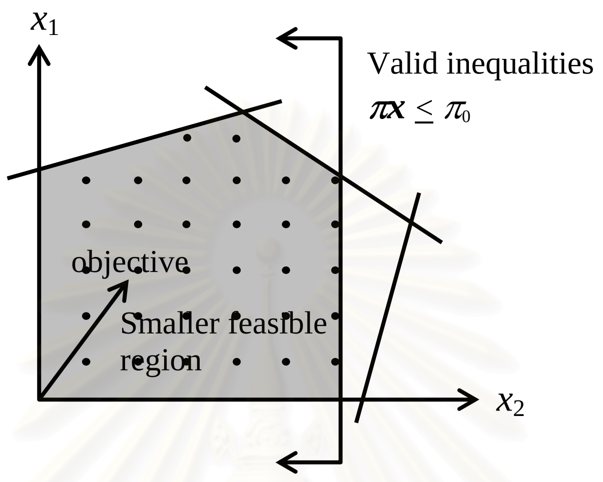


Figure 2.2: Valid inequalities cut off the non-integral feasible region.

2.1.4 Branch and Bound Algorithm

In this section, the classical and the most widely used approach for solving MIP model is the branch and bound algorithm which employs an LP model based relaxations of the MIP for exploring the solution spaces. The implementation of the branch and bound algorithm can be viewed as a tree search, where the problem at the root node of the tree is the original MIP. The new nodes are formed by branching on an existing node for which the optimal solution of the relaxation is fractional.

- **Theory of Branch and Bound Algorithm**

Branch and bound algorithm is a method guaranteed to find a global optimal solution to the MIP problem (Jeremy F S, 1979). The basic idea of the branch and bound algorithm is to partition a given problem into a number of subproblems. This process of partitioning is usually called branching and its purpose is to establish subproblems that are easier to solve than the original problem. Branching is generally represented in terms of a tree structure, as in figure 2.3 where each node *i* of the search tree represents a subproblem P_i while *c* is an integral value and x_i is a branching order called a Special Order Set (SOS). The searching tree may have many levels, with the nodes at the bottom of the branching being referred to as pendant nodes. The solution process involved a systematic evaluation of the pendant nodes of the search tree, the evaluation process consists of three key components: branching, computing bounds and fathoming. To derive the optimal solution to a given problem P_0, the set of subproblems of P_0 must represent all of P_0. For simplicity, let $\{P_i\}$ be the set of feasible integer solution to a problem P_i. Then, if P_0 is partitioned into P_1, P_2, …., P_n, it must be true that

$$\{P_0\} = \{P_1\} \cup \{P_2\} \cup \ldots \cup \{P_n\}$$

Also, it is generally more efficient to also choose subproblems P_1, P_2, … P_n such that $\{P_i\} \cap \{P_j\} = \phi$ for all $i \neq j$ where c_i is an integral value and x_i is an order variable from i to n. This is especially true when it is necessary to enumerate all solutions to the problem, because some solutions would be enumerated multiple times if the feasible regions of some subproblems have a nonempty intersection.

To help understand the branching process, consider an integer program P_0 with n variables, and suppose that a particular variable, say x_k must take on the integral value c_1, c_2, c_3, … , c_n that the subproblems are created, each of which corresponds to fixing the variables x_k at one of its possible values. Because x_k is now fixed in value, each of the subproblems involves only n - 1 variables, see figure 2.3.

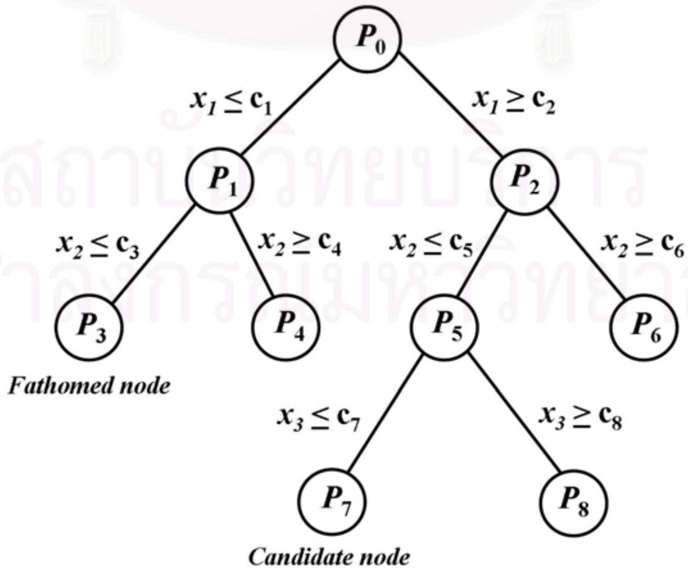


Figure 2.3: Subproblems and branching strategies.

Note, further, that during the branching process, we are essentially adding restrictions to a particular problem to form the resulting subproblems. Consequently, the feasible region of a subproblem is a subset of the feasible region of the parent problem. Thus, in the case of a maximization problem, the optimal objective value associated with a subproblem is always less than or equal to the optimal objective value associated with the parent problem. Therefore, as we descent in the search tree, the optimal objective values associated with each subproblem decrease for a maximization problem. In order to describe the branch and bound algorithm, the three key components of evaluation process will be described as follows.

- **Computing Bound**

Suppose that we know a feasible integer solution to a particular maximization integer problem. Then the objective value provided by this solution is a lower bound for the optimal objective value of the MIP. We assure of obtaining an optimal objective which design the lower bound by z_L. If several feasible integer solutions are known, z_L will correspond to the largest known objective value. That z_L is the lower bound and the integer solution corresponding to this value is called the incumbent solution, because it is the best known integer solution.

The purpose of computing upper bounds (in a maximization problem) determines the optimal solution at a node without actually solving the integer program at the node. This is usually done by solving the LP relaxation. Consider an integer subproblem P_i associated with the pendant node i. Let z denote the optimal objective value associated with subproblem P_i. That is, z corresponds to an optimal integer solution of P_i. To determine, we are interested in finding an upper bound for z that can be readily computed. Consider solving the LP relaxation of subproblem P_i, and let $\bar{z}$ denote the optimal objective value of the LP relaxation. Clearly, $\bar{z} \geq z$ the feasible region of the integer program is a subset of the feasible region.

Suppose that $\bar{z} \leq z_L$. Then, $z \leq \bar{z} \leq z_L$ and subproblem P_i does not need to be considered further because it will never yield a solution any better than the current best integer solution. This process of eliminating a subproblem P_i from further considerations is referred to as fathoming. However, if $\bar{z} \geq z_L$, then a conclusion can not be reached and further branching is needed.

- **Fathoming**

During the branch and bound process, an attempt is made to resolve each of the subproblems corresponding to the pendant nodes of the search tree. Once all of the subproblems associated with the pendant nodes are solved, then the problem is solved. A subproblem can be eliminated from further consideration in one of the following three ways:

1) The subproblems yield an optimal integer solution. In this case, we update Z_L and the incumbent solution if necessary and continue the node-selection process.
2) It can be shown that the optimal solution value of the subproblem is no better than the best integer solution found thus far. This is usually done by computing a bound on the optimal integer objective value by solving the LP relaxation. This bound is then compared with the objective value of the incumbent solution.
3) The subproblem is infeasible.

However, this is not possible to fathom a given pendant node, the subproblem associated with that node is again partitioned into a smaller subproblem by branching in some prescribed manner. The process is then repeated until all pendant nodes have been fathomed.

- **Search Strategies**

Branching also involves choosing the next subproblem (pendant node) to examine. There are several branching strategies for choosing the next pendant node, with the most common being depth-first search and best-bound search. In each of these strategies, the pendant nodes are placed in a list according to measure of importance. If the current node under examination is fathomed, then the next node in the list is selected. If the examination of the current node is complete and it can not be fathomed, the current subproblem is partitioned into additional subproblems that are then added to the list according to the branching strategy being used. A new node is then selected and the process is repeated until the list of available pendant nodes is empty.

The solution of the LP relaxation at each node generates a bound on the optimal integer solution that can be derived from that node. In the best bound search, the next subproblem chosen is simply the one with the best bound. That is, for a maximization problem, we would branch next on the node with the largest upper bound. The rationale for branching in this way is to attempt to generate an integer solution early in the branching process. This incumbent solution then could be used to fathom nodes with smaller upper bounds.

Depth-first search is also called last-in-first-out (LIFO). Last-in-first-out refers to the strategy for placing nodes in and selecting nodes from the list of pendant nodes. Using the depth-first strategy, we always choose the subproblem (node) that was placed in the list most recently. We essentially work down one side of the search tree first and the backtrack once a node is fathomed, because we can be used more efficiently. This is a result of subproblems being created by adding restrictions to the parent problem.

2.2 Fundamentals of the Genetic Algorithm

2.2.1 Theory of Genetic Algorithm

This section describes the conceptual model of the Genetic Algorithms (GA) used in this thesis. It starts with a basic form of GA along with its implementation. Then, the fundamental theory of GA and its operators are discussed.

GA is classified as one of the evolutionary computation algorithms which refer to a method that uses some forms of evolution as a major part of the process. Original GA was introduced in 1975 by John Holland (1992) but evolution-based computation approaches have been studied earlier than that period.

- **Genetic Algorithms**

GA was first proposed by John Holland at the University of Michigan in 1973. He and his students investigated and proved that GA is a significant contribution for scientific and engineering application. Since then, the outputs of researches in this field have grown rapidly. GA is not a technique that requires the use of derivatives. The obtained optima are evolved from generation to generation without a mathematical formulation such as the traditional gradient type of optimizing procedure. Gradient descent which calculates the slope of error surface at the current position works well when the error surface is relatively smooth, with few local minima. Nevertheless, most real world data has the distorted error surface by noise. The error surface would prove difficult for gradient descent because of the local minima. GA is less sensitive to local minima because it constitutes a parallel search of the solution space, as opposed to a point by point search.

Therefore, GA is usually applied to optimization problems that are difficult to solve or cannot be solved by a mathematical formulation. It is also used to resolved NP-hard and NP-complete such as traveling saleman problem (TSP), scheduling and design problems. It performs searching throughout the solution space to find the near optimal answer.

- **Genetic Algorithm Background**

GA is a technique imitating biological process of natural selection (Darwin's rule) by which only good or fit being survive (Tsang et al., 1996). The theory of Charles Dawin may be summarized as follows. (a) The individuals of a species show variation. (b) In general, more offsprings are produced than needed to replace their parents. (c) Populations cannot expand indefinitely and, on average, population sizes remain stable. (d) There must be competition for survival and (e) therefore, the best adapted variants (the fitness) survive.

GA uses a direct similarity of natural behavior following Darwin's theory. Above all, the problem to be solved by GA must be first encoded into gene. There is no uniform encoding scheme for every problem. The encoding scheme varies from one problem to another problem. The appropriate encoding for the problem has to be devised. The structure of GA is illustrated in figure 2.4.

```
Standard Genetic Algorithm()
1.  $t = 0$
2.  Generate initial population (valid genes)
3.  Calculate fitness values of each gene.
4.  While the conditions are not satisfied and $t < N$ do
    (a)  $t = t + 1$
    (b)  Select parents by random.
    (c)  Recombine the population by crossover
         and mutation operations.
    (d)  Calculate fitness values of valid child genes.
    (e)  Select the new population from the old
         population and the child population.
```

Figure 2.4: Structure of Genetic Algorithms.

At the beginning, a set of the first generation of gene population is randomly produced. We also evaluate their fitness as different beings possess unequal capability to survive. After that, the genes in the set are randomly selected to produce the next generation genes with the high fitness cost. The genes with low fitness cost are eliminated. The producing process continues until the number of generation reaches the specified value or there is no new gene to be produced. The set of new genes is generated by three main gene operations, which are mutation, crossover and inversion. Moreover, the conditions in the while loop depend on the problems to be solved. For example, the condition for the traveling saleman problem is the minimum total traveling distance. Variable *t* counts the number of generations whose maximum value is denoted by a constant *N*.

2.2.2 Principal Factors of Genetic Algorithm

The performance of the GA is controlled by the following factors.

- **Encoding Scheme**

Encoding scheme is referred to as genes of a chromosome which can be commonly structured by various ways such as string, binary string, gray code and floating point. Generally, the binary scheme is traditionally used in GA but not appreciated in some examples such as the problem concerning many variables with large domain. Another scheme is the gray code which is slightly modified from the binary coding. Note that the gray coding has the property that any two points next to each other in the problem space differ by one bit only. By analogy with genetics, the values of the variables are called the phenotype and the coding is called the genotype. Four different coding have been implemented as follows.

1) A binary coding: each variable is coded in a substring of bits whose number is related to the number of alleles that the variable could take.

Table 2.1: Example of binary coding: construction of a chromosome (4 design variables).

Variables	Types of variables	X_i (variation domain of X_i)	Number of Alleles	Substring size
x_1	Continuous	[0,10]	1024 (2^{10})	10
x_2	Continuous	[0,10]	1024 (2^{10})	10
x_3	Discrete	{10; 12.5; 15; 17.5}	4 (2^2)	2
x_4	Integer	{0;1}	4 (2^1)	1

The chromosome of an individual is then constructed by concatenating the substrings S_i corresponding to each variable x_i, see figure 2.5.

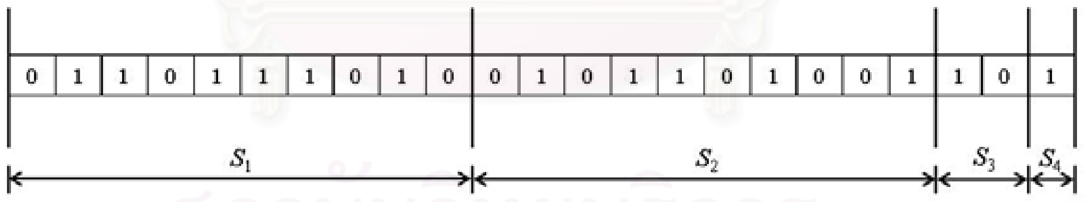


Figure 2.5: Example of chromosome for a four-variable individual with binary coding.

2) A Gray binary coding: the binary representation as described above is widely used in the GA community, but it has some drawbacks. Indeed, it is commonly accepted that a coding should reflect as closely as possible the behavior of the variables. For example, a small change in the value of the variable should lead to a small modification of the genotype. This is not systematically the case in binary coding, where subsequent alleles may have completely different chromosomes. Therefore, the

Gray coding has been introduced, and is built in such a way that two subsequent alleles differ only from one bit, see figure 2.6 for a 3-bit variable.

Alleles	Binary coding	Gray coding
1	000	000
2	001	001
3	010	011
4	011	010
5	100	110
6	101	111
7	110	101
8	111	100

Figure 2.6: Binary and Gray coding for a 3-bit variable.

3. A fixed-point representation: this coding is based on a decimal representation. Each division of the chromosome corresponds to one figure, and the place of the decimal point is fixed. This is illustrated in Fig. 2.7 for a 2-variable individual.

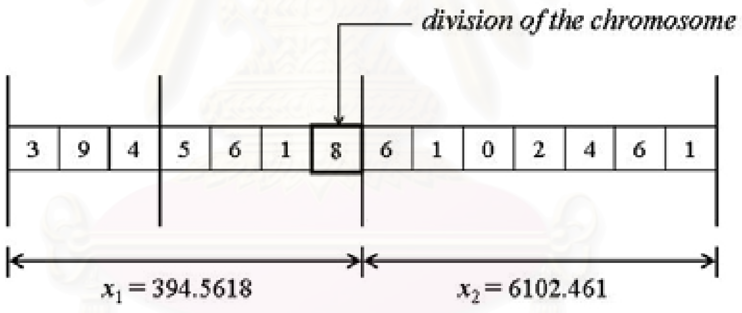


Figure 2.7: Example of chromosome for a four-variable individual with binary coding.

4) A real coding: when there are only continuous variables, a real coding is often preferred, because it is very close to the real search space. In this representation, each individual is thus coded as a vector of real values.

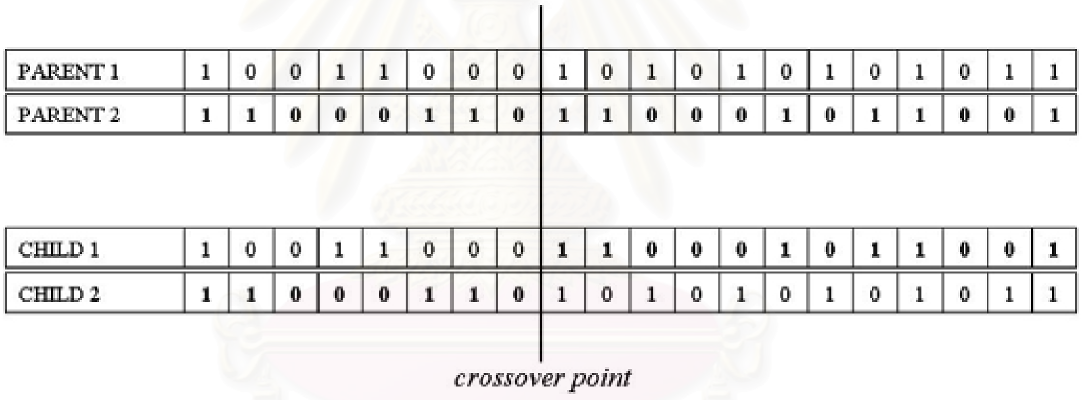
- **Fitness Function**

Fitness function is the link between the GA and the problem to be solved. It is one of the most significant elements to assess the GA performance. The value of the fitness function is calculated for an individual of population and fitness value is settled on its basis. The interaction between a chromosome and a fitness function provides a measure of its fitness that is used when carrying out reproduction. Its fitness is supposed to be proportional to the utility or ability of the individual which that chromosome represents.

- **Crossover**

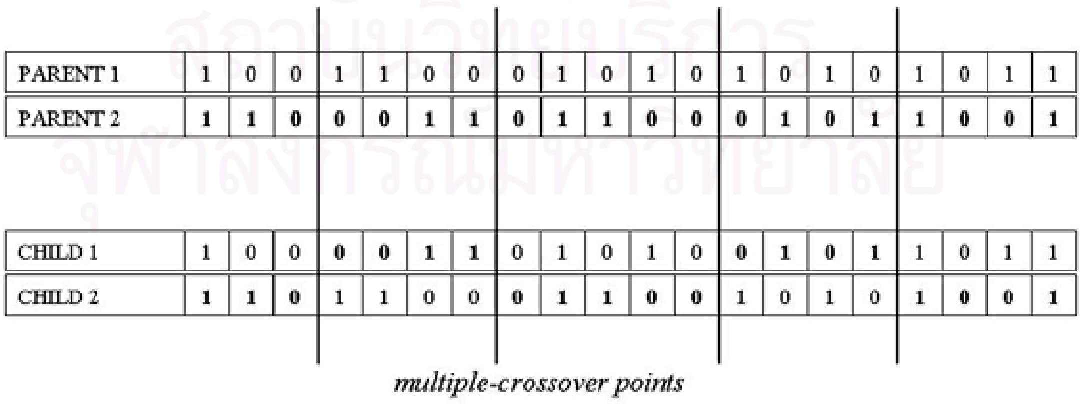
In nature, crossover occurs when two parents exchange parts of their corresponding chromosome. In GA, the crossover recombines the genetic material in two parent chromosomes to make two children. This is called by John Holland “one-point crossover”. For the one-point crossover, two children are constructed by inverting the genes of their parents from the (randomly determined) crossover site, see figure 2.8(a).

In some situation, using one-point crossover is inefficient. A multipoint crossover can be used to overcome this problem. An example is demonstrated in figure 2.8(b) where multiple crossover points are randomly selected.



PARENT 1	1	0	0	1	1	0	0	0	1	0	1	0	1	0	1	0	1	0	1	1
PARENT 2	1	1	0	0	0	1	1	0	1	1	0	0	0	1	0	1	1	0	0	1
CHILD 1	1	0	0	1	1	0	0	0	1	1	0	0	0	1	0	1	1	0	0	1
CHILD 2	1	1	0	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	1

(a)



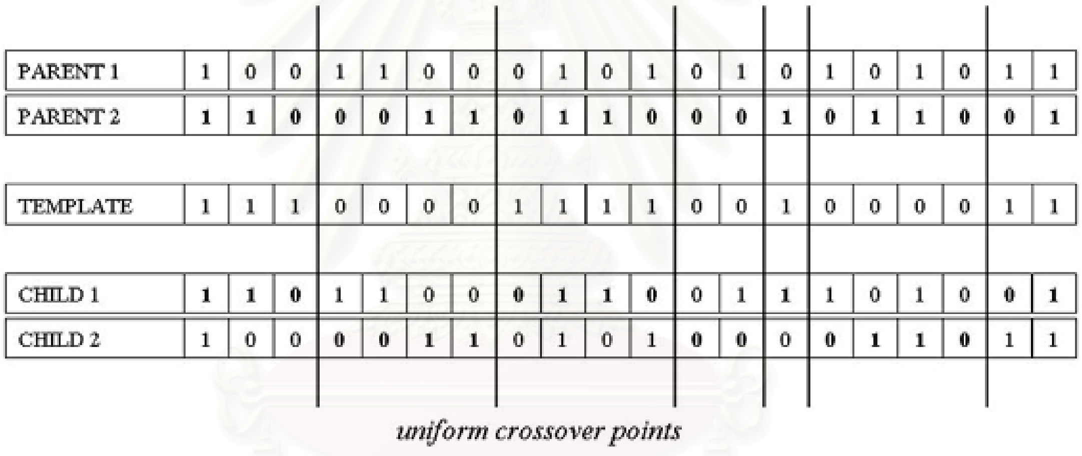
PARENT 1	1	0	0	1	1	0	0	0	1	0	1	0	1	0	1	0	1	0	1	1
PARENT 2	1	1	0	0	0	1	1	0	1	1	0	0	0	1	0	1	1	0	0	1
CHILD 1	1	0	0	0	0	1	1	0	1	0	1	0	0	1	0	1	1	0	1	1
CHILD 2	1	1	0	1	1	0	0	0	1	1	0	0	1	0	1	0	1	0	0	1

(b)

Figure 2.8: (a) Example of one-point crossover and (b) example of multiple-point crossover.

Another operator is called "uniform crossover" which is similar to multipoint crossover. But it needs a randomly generated crossover template which is the pattern of crossover point. There is an example in figure 2.9. The length of string 0-1 in the template is equal to the length of chromosome. Therefore, at 0 in the template, the gene of child 1 is placed by the gene of parent 1 and the gene of child 2 is placed by the gene of parent 2. At 1 in the template, the gene of child 1 is placed by the gene of parent 2 and the gene of child 2 is placed by the gene of parent 1.

Due to the uniform crossover exchanges bits rather than segments, it can combine features regardless of their locations. This ability may outweight the disadvantage of destroying building blocks and make uniform crossover a superior operator for some problems.



PARENT 1	1	0	0	1	1	0	0	0	1	0	1	0	1	0	1	0	1	0	1	1
PARENT 2	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**1**
TEMPLATE	1	1	1	0	0	0	0	1	1	1	1	0	0	1	0	0	0	0	1	1
CHILD 1	**1**	**1**	**0**	1	1	0	0	**0**	**1**	**1**	**0**	0	1	**1**	1	0	1	0	**0**	**1**
CHILD 2	1	0	0	**0**	**0**	**1**	**1**	0	1	0	1	**0**	**0**	0	**0**	**1**	**1**	**0**	1	1

Figure 2.9: Uniform crossover-child 1, a value of 1 in the random string corresponds to a bit from parent 1, and 0 corresponds to a bit from parent 2, and vice versa for child 2.

- **Mutation**

Mutation is the process applied to each offspring individually after the crossover. In GA, this operator creates new individuals by a small change in a single individual by a random selection. When mutation is applied to a bit string, it sweeps down the list of bits and replaces each by a randomly selected bit if the probability of test passes. It is called "Bit Mutation" as illustrated in figure 2.10. In addition, it has an associated parameter probability that is typically quite low.

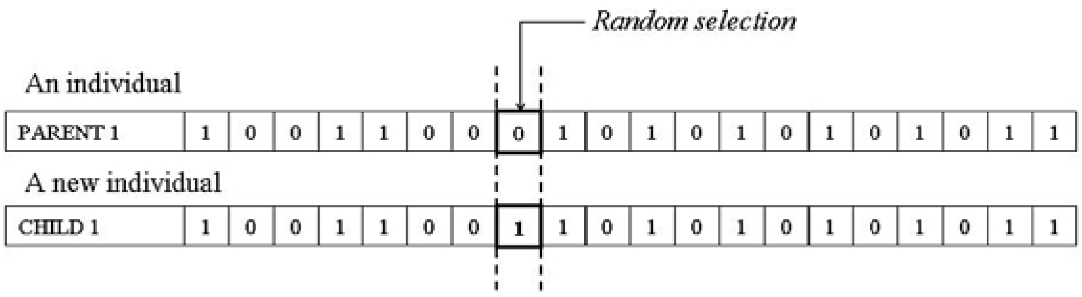


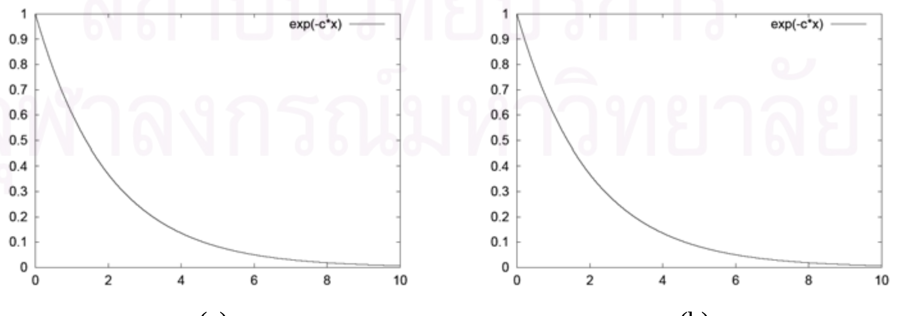
Figure 2.10: An example of mutation.

Moreover, there are several mutation types that are not the binary case where only one or two bits are flipped. The following describes the other techniques of mutations.

1) A Random mutation: for each variable that is going to be mutated, choose a random value within its range and assign this value to the variable. So, every value is possible.

2) Gauß mutation: this mutation is similar to the previous one, the only difference being that mutation step Δx_i is calculated according to Gauß' distribution $N(0,1)$: smaller mutation steps are much more probable then large mutation steps. The probability distribution of a standard mutation is shown in figure 2.11(a).

3) EXP mutation: this mutation type comes from the idea that the role of mutation at the beginning is to make large jumps whereas later on, as the search progresses it should be used more for fine–tuning so small jumps are more desirable. Exponential distribution is presented in Figure 2.11(b). Here c is the constant that depends on the generation number.



(a) (b)

Figure 2.11: Probability distributions for (a) Gauß mutation and (b) EXP mutation.

In this section, we describe the basic idea of GA from both theoretical background and their characteristics which started by giving the overview of GA and its principal operators. In general, GA is a class of search algorithms inspired by evolution from nature. For more details about GA, the reader is suggested to read the standard book such as GA in Search Optimization, and Machine Learning (Goldberg, 1989).

As far the GA robustness has been discussed with details. Figure 2.12 illustrates the entire process GA that will be applied to use in our algorithm.

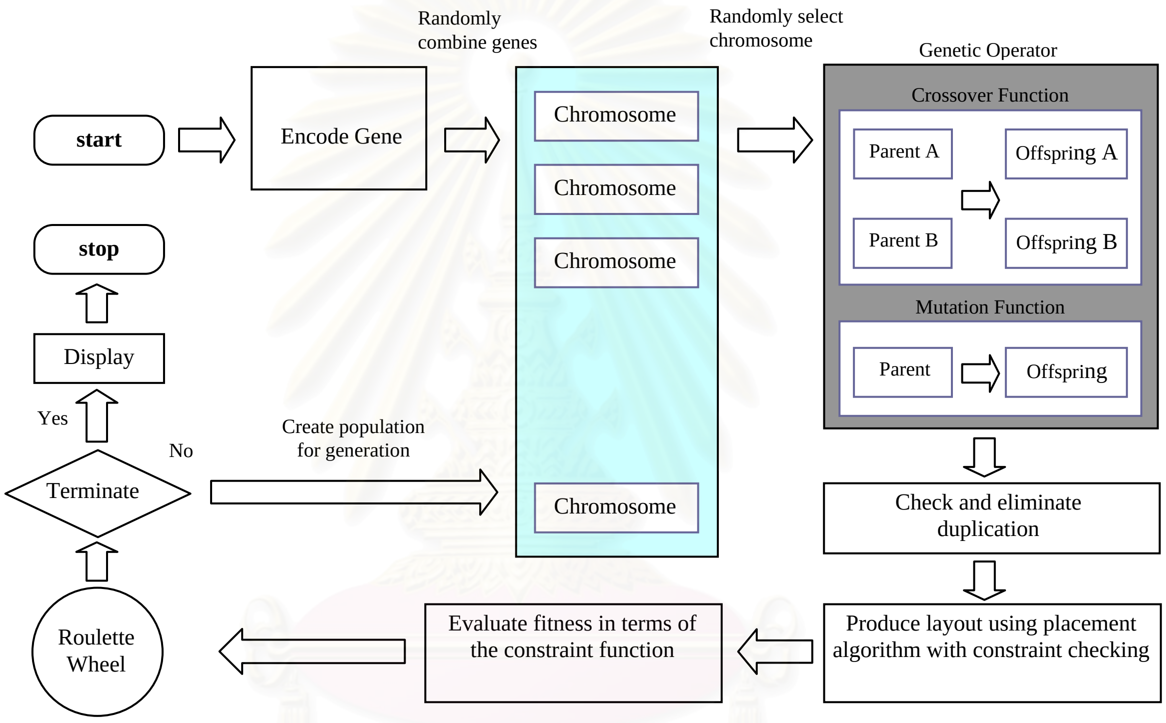


Figure 2.12: The general structure diagram of the GA is applied to solve the architectural layout design problem.

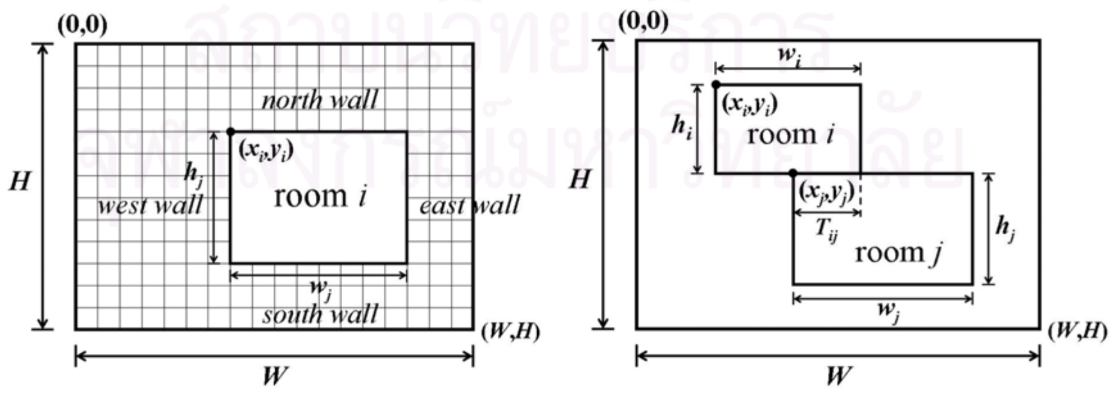
CHAPTER III
Problem Methodologies

In this thesis, we propose three methodologies to attack the architectural layout design optimization. First, we formulate the mixed integer optimization model. Our original formulation called AL-MIP guarantees the optimal design based on a multiobjective function. Second, we propose the valid inequality constraints called AL-MIP+ to reduce the AL-MIP iterations and time. These valid inequality constraints consist of non-circular connectivity constraints and advised configuration constraints that help reduce the feasible search space. Finally, the third methodology based on the machine learning algorithm utilizes an idea of Genetic Algorithm (GA) called AL-MIP+GA to learn a Special Order Set (SOS).

3.1 Architectural Layout Design Optimization Model

3.1.1 Design Variables and Parameters

The architectural layout design problem is posed as a process of finding the best location and size of a group of interrelated rectangular rooms. In this thesis, we define the room as a rectangular space to represent a specific architectural function such as living spaces, storage spaces, and facility spaces. Given a set of rooms {1, 2, …, n}, figure 3.1(a) shows the room ith representation using a point at the top left corner (x_i, y_i) with its height h_i and width w_i. The following figure also shows four walls represented by the north, the south, the east and the west.



(a) (b)

Figure 3.1: (a) Model variables and parameters based on the coordinated system and (b) model relationships between two connected rooms.

In design variables and parameters, our model is formulated based on the coordinated system of a meter unit scale using the top left corner of the boundary area as the reference origin (0, 0). The positive value of x corresponds to x units to the right of the origin while the positive value of y corresponds to y units below the origin. Coordinates and dimensions are used as design variables, see figure 3.1(a).

x_i = X coordinate of the top left corner of the room i.
y_i = Y coordinate of the top left corner of the room i.
w_i = the horizontal width of the room i.
h_i = the vertical height of the room i.

Two boundary parameters are layout width and layout height which are represented by W and H, respectively. Moreover, there are specific parameters for each room, the lower and upper limits of the room width and the room height, $w_{\min,i}$, $w_{\max,i}$, $h_{\min,i}$, $h_{\max,i}$ where $w_{\min,i}$ is the minimal width of room i, $w_{\max,i}$ is the maximal width of room i, $h_{\min,i}$ is the minimal height of room i, $h_{\max,i}$ is the maximal height of room i. In addition, T_{ij} is a minimal contact length parameter between room i and room j, see figure 3.1(b).

This thesis also concerns with the reduction of the computational iterations by limiting the variable numbers. For the connectivity sets from $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$ where n is the number of room, we can reduce the numbers of variables by fixing i less than j ($i < j$) due to the equivalent of the connectivity between i,j and j,i. Thus, we can only use the connectivity i,j where $i < j$. This help reduces the number of variables more than a half.

- **The decision binary variables p_{ij} and q_{ij}**

The possible configurations of the two room connectivities between room i and room j can be represented using the four directions of the north (top), the south (bottom), the east (right) and the west (left) direction. To capture these idea, we utilize the two decision binary variables p_{ij} and q_{ij} to represent these connectivity directions. By which, the four possible connectivities are described using four pairs of (p_{ij}, q_{ij}) as (0,0), (0,1), (1,0) and (1,1).

Ideally, these two decision binary variables, p_{ij} and q_{ij} have been used to satisfy a constraint between room i^{th} and room j^{th}. Four distinct patterns of p_{ij} and q_{ij} can be described below. First case, (p_{ij}, q_{ij}) sets to (0,0) which forces the room *i* to the left of room *j*. Second cases (p_{ij}, q_{ij}) sets to (0, 1) which forces the room *i* to the top of the room *j*. Third case, (p_{ij}, q_{ij}) sets to (1, 0) which forces the room *i* to the right of the room *j*. Fourth case, (p_{ij}, q_{ij}) sets (1, 1) which forces the room *i* to the bottom of the room *j*. In the other words, the decision to assign values p_{ij} and q_{ij} will place these two rooms in the required orientation, see figure 3.2.

To utilize an idea of these two decision binary variables, only one from four patterns of (p_{ij}, q_{ij}) will be satisfied selected to satisfy a constraint among each constraint group of AL-MIP and AL-MIP+ model. Moreover, the decision variables p_{ij} and q_{ij} can be applied to speed up the computational time which will be described at the end of this chapter.

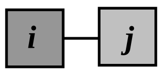
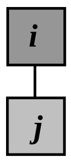
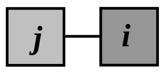
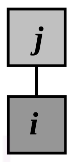
Values		Connectivity directions	
p_{ij}	q_{ij}		
0	0	room *i* connects to the left of room *j*	
0	1	room *i* connects to the top of room *j*	
1	0	room *i* connects to the right of room *j*	
1	1	room *i* connects to the bottom of room *j*	

Figure 3.2: The four connectivity directions between room *i* and room *j*.

3.1.2 Multiobjective optimization

Multiobjective optimization known as multi-criteria or multi-attribute optimization, is the process of simultaneously optimizing two or more conflicting objectives subject to certain constraints. The most widely used method for multiple optimization is the weighted-sum approach. The objective function is formulated as a weighted summation as follows.

$$\text{Minimize} \qquad \sum_{i=1}^{k} u_i f_i(x) \tag{3.1}$$

$$\text{subject to :} \qquad x \in S$$

$$\text{where} \qquad u \in \mathrm{R}^k, \quad u_i > 0$$

By choosing the different weights u_i, for the different objectives, the preference of the decision-maker is taken into account. As the objective functions are generally of different magnitudes, they might have to be normalized first. Although the formulation is simple, the method requires a special treatment, as there is not clearly the relation between the weights and the obtained solution. To determine the weights from the decision-maker’s preferences is a specific purpose procedure.

- **Architectural layout design multiobjective optimization**

With the architectural layout design, many researches usually concentrated on a single objective function. Fleming in 1978 presented a singular objective layout via the representation and generation of rectangular dissections that minimized room space. In 2000, the work of Li, Frazer and Tang dealt with maximizing the area in a given floor layout. In contrast, new researches are more interested in multiobjective preferences. In this thesis, we are interested in maximizing room areas and minimizing distance between rooms. To cope with these multiobjective preferences, we combine two objective functions into a summation of weighted components. These weights can be adjusted according to architect’s favor. In our experiment, we use equal weights to measure performance of our AL-MIP model. At optimal, there always exist alternative solutions with the same objective value due to the layout rotation. In order to eliminate alternative solutions, we randomly select one of available rooms to be placed near the top left corner. In our experiment, the first room has been selected. For selected i_o^{th} room,

$$\text{Minimize} \quad u_1 \times (x_{i\,o} + y_{i\,o}) + u_2 \times (\text{absolute distance}) - u_3 \times (\text{maximizing room area})$$

or

$$\text{Minimize} \quad u_1 (x_{i\,o} + y_{i\,o}) + u_2 \sum_{i<j} (zx_{i,j} + zy_{i,j}) - u_3 \sum_{i=1}^{n} z_i \tag{3.2}$$

where	$x_{i\,o}$, $y_{i\,o}$	are X and Y coordinate of the i_o^{th} room,	$\forall\ i_o = 1, \dots, n$,
	$zx_{i,j}$, $zy_{i,j}$	are absolute distance of room i and j,	$\forall\ i < j = 1, \dots, n$,
	z_i	is the maximized value between w_i and h_i,	$\forall\ i = 1, \dots, n$,
	u_1, u_2, u_3	are the weight values.	

Objective 3.2 denotes the minimization of the multiobjective optimization where the u_1 is the weight of the i_o^{th} room positioning to the nearest top left corner, u_2 is the weight of the total absolute distance and u_3 is the weight of the maximizing approximated room area. If an architect prefers larger room area then the weighted sum of u_3 is set to be greater than u_2. If an architect prefers a short total distance between rooms then u_2 is set to be greater than u_3. Hence, architect can generate alternative solutions by selecting different i_o^{th} room to be placed near the top left corner or reassign the desired objective weights. Moreover, x_{i_o} and y_{i_o} represent the X and Y coordinate of the i_o^{th} room while $zx_{i,j}$ and $zy_{i,j}$ represent the absolute distance in the X and Y coordinate respectively. The z_i represents the maximized value between w_i and h_i that we can use to approximate the maximized area.

- **Placing a room position near the origin**

The combinatorial nature of the alternative optimal solutions having the same objective values could affect the total solution time. To allow the AL-MIP algorithm to prune other alternative solutions, architects can force the i_o^{th} room position to the nearest origin of the boundary area. By selecting different room, architects could obtain another optimal solution.

Minimize $(x_{i_o} + y_{i_o})$ (3.3)

where x_{i_o} is the X coordinate of the i_oth room, $\forall\ i_o = 1, \dots, n,$

y_{i_o} is the Y coordinate of the i_oth room, $\forall\ i_o = 1, \dots, n.$

- **Minimizing the absolute room distance**

One interesting criterion of an architect preference deals with a short distance among rooms. Calculating the distance as a linear function is not easily achievable. In this thesis, we apply the absolute distance function called Manhattan distance, instead of the normal Euclidean distance, see figure 3.3(a). This distance function is preferred over the Euclidean distance function due to two reasons. The first reason, it maintains the unit during the comparison. There is no need to take a root of the sum square distance as in Euclidean distance. The second reason is the walking distance from room to room could not join diagonally across the room to reach the target room. Architect could only walk along the boundary to the available room. The Manhattan distance computes as the summation of an absolute difference on the X coordinate and Y coordinate between two points, see figure 3.3(a).

Minimize $\sum_{i<j} \left(|x_i - x_j| + |y_i - y_j| \right)$

or equivalently,

Minimize $\sum_{i<j} \left(zx_{i,j} + zy_{i,j} \right)$ (3.4)

subject to: $x_j - x_i \le zx_{i,j}$

$y_j - y_i \le zy_{i,j}$

where $zx_{i,j}$ is the absolute distance of room i and j on the X coordinate, $\forall i, j = 1, \dots, n,\ i < j,$

$zy_{i,j}$ is the absolute distance of room i and j on the Y coordinate $\forall i, j = 1, \dots, n,\ i < j,$

x_i, x_j are the X coordinate of the room i and j, $\forall i,j = 1, \dots, n,$

y_i, y_j are the Y coordinate of the room i and j, $\forall i,j = 1, \dots, n.$

Objective 3.4 denotes the summation of the absolute distance between room *i* and room *j* where $zx_{i,j}$ is the absolute distance on X coordinate and and $zy_{i,j}$ is the absolute distance on Y coordinate. While x_i and x_j represent the coordinate on the X coordinates of room *i* and *j*, y_i and y_j are the Y coordinates of room *i* and *j*. Figure 3.3(b) illustrates the absolute distance and its equivalent linear model.

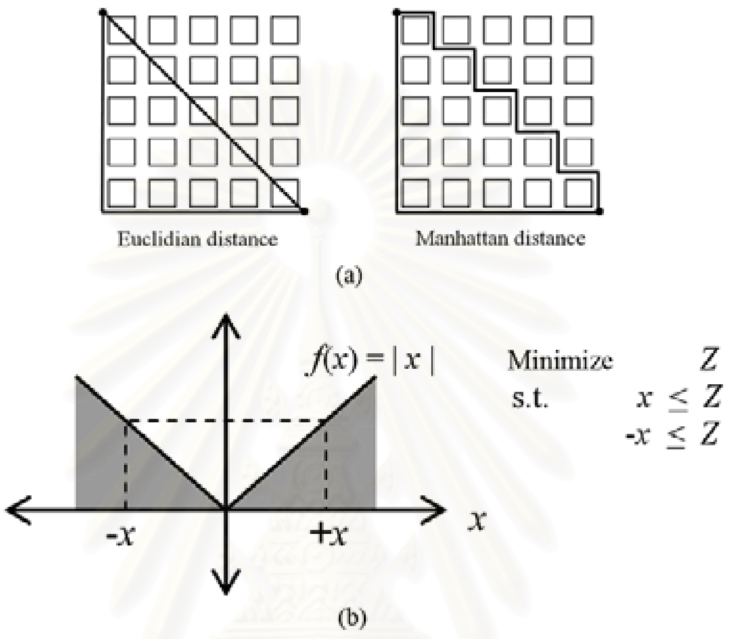


Figure 3.3: (a) Comparison between Euclidian distance function and Manhattan distance function and (b) an absolute function is formulated using a linear model.

- **Maximize approximating room area**

Another important architect's preference is the spacious room space. In practice, architects wish to design largest possible rooms within the available space. A rectangular area can be computed by multiplying two sides as a non-linear function. However, the MIP model only deals with linear functions and constraints. Therefore, we decide to maximize the room sizes which have the direct effect to the approximated room areas that the larger the room sizes are, the greater the area will be, see figure 3.4.

$$\text{Maximize} \qquad \sum_{i=1}^{n} max\ \{\ w_i, h_i\ \}$$

or equivalently,

$$\text{Maximize} \quad z_i \tag{3.5}$$

$$\text{subject to:} \quad z_i \leq w_i$$

$$z_i \leq h_i$$

where z_i is the maximized value between width and height of room i, $\forall\, i = 1, \dots, n$.

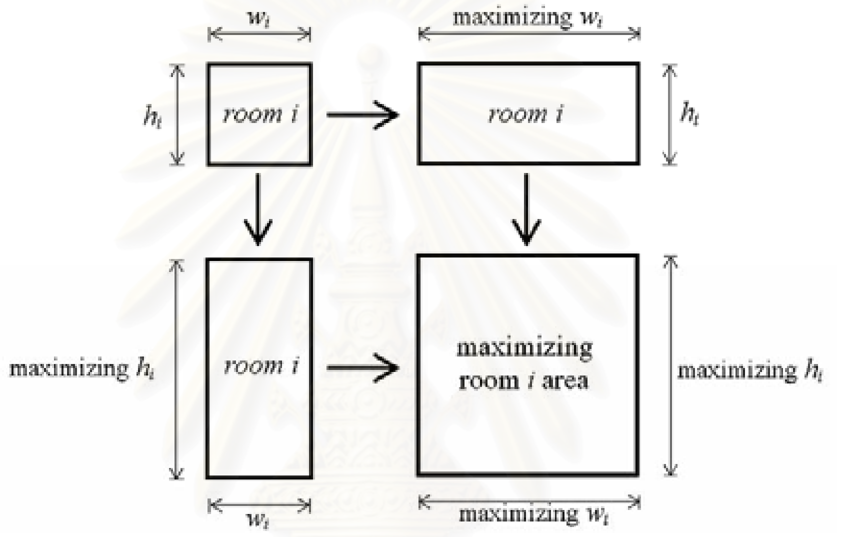


Figure 3.4: Maximizing room area is constructed by maximizing each room side.

3.1.3 Architectural constraint Formulations

In this thesis, all architectural layout design requirements can be captured using a linear function which will be described as follows.

- *Location constraint* explains the relationship between distinct rooms that ensures the location of rooms. To formulate this constraint, we use two decision binary variables p_{ij} and q_{ij}, see figure. 3.5.

$$x_i + w_i \leq x_j + W \times (p_{ij} + q_{ij}) \tag{3.6}$$

$$y_j + h_j \leq y_i + H \times (1 + p_{ij} - q_{ij}) \tag{3.7}$$

$$x_j + w_j \leq x_i + W \times (1 - p_{ij} + q_{ij}) \tag{3.8}$$

$$y_i + h_i \leq y_j + H \times (2 - p_{ij} - q_{ij}) \tag{3.9}$$

where x_i, x_j	are the X coordinate of the room i and j,	$\forall i < j = 1, \dots, n$,
y_i, y_j	are the Y coordinate of the room i and j,	$\forall i < j = 1, \dots, n$,
w_i, w_j	are the width of the room i and j,	$\forall i < j = 1, \dots, n$,
h_i, h_j	are the height of the room i and j,	$\forall i < j = 1, \dots, n$,
W, H	are the boundary width and height.	

From the location constraint, the decision variables p_{ij} and q_{ij} force the room i to the left, the bottom, the right and the top of room j corresponding to constraint 3.6, 3.7, 3.8 and 3.9. Four possible cases of (p_{ij}, q_{ij}) are (0, 0), (0, 1), (1, 0) and (1, 1) have been used to force the room connectivities that will be explained below. First case, (p_{ij}, q_{ij}) sets (0, 0). The solution must satisfy $x_i + w_i \leq x_j$ for the constraint 3.6 which implies that the j^{th} room must be placed on the right of the i^{th} room. At the same time, constraint 3.7 becomes $y_j + h_j \leq y_i + H$. Due to the large value of H, the right-hand side becomes a large positive value. Hence, any smaller positive $y_j + h_j$ will satisfy the constraint 3.7. Similarly, constraint 3.8 becomes $x_j + w_j \leq x_j + W$ so that any positive $x_j + w_j$ will be smaller than $x_i + W$. Moreover, constraint 3.9 becomes $y_i + h_i \leq y_j + 2H$. This also guarantees that any smaller positive $y_i + h_i$ will satisfy the constraint 3.9. Hence, we will say that the setting of (p_{ij}, q_{ij}) = (0, 0) forces the placement of the i^{th} room to the left of the j^{th} room. Second case, (p_{ij}, q_{ij}) sets (0, 1). The solution must satisfy $y_j + h_j \leq y_i$ for the constraint 3.7 which implies that j^{th} room must be placed on the top of the i^{th} room. Simultaneously, constraint 3.6 becomes $x_i + w_i \leq x_j + W$. The large value of W in the right-hand side becomes a large positive value. So that any smaller positive $x_i + w_i$ satisfies the constraint 3.6. Constraint 3.8 becomes $x_j + w_j \leq x_i + 2W$ that the positive $x_j + w_j$ is smaller than $x_i + 2W$ while constraint 3.9 become $y_i + h_i \leq y_j + H$ that $y_i + h_i$ is a smaller positive value. Thus, the setting of (p_{ij}, q_{ij}) = (0, 1) forces the placement of the i^{th} room to bottom of the j^{th} room. Third case, (p_{ij}, q_{ij}) sets (1, 0). The solution must satisfy $x_j + w_j \leq x_i$ for the constraint 3.8 which implies that j^{th} room must be placed on the left of the i^{th} room. Simultaneously, constraint 3.6 becomes $x_i + w_i \leq x_j + W$. The large value of W in the right-hand side becomes a large positive value. For any smaller positive $x_i + w_i$ satisfy the constraint 3.6. Similarly, constraint 3.7 and 3.9 becomes $y_j + h_j \leq y_i + 2H$ and $y_i + h_i \leq y_j + H$. The positive values of $y_j +$

h_j and $y_i + h_i$ are smaller than $y_i + 2H$ and $y_i + H$ which satisfy constraint 3.7 and 3.9 respectively. The setting of $(p_{ij}, q_{ij}) = (1,0)$ forces the placement of the i^{th} room to right of the j^{th} room. The last case, (p_{ij}, q_{ij}) sets (1, 1). The solution must satisfy $y_i + h_i \le y_j$ for the constraint 3.9 which implies that j^{th} room must be placed at the bottom of the i^{th} room. At the same time, constraint 3.6 becomes $x_i + w_i \le x_j + 2W$. The large value of W in the right-hand side becomes a large positive value. Hence, any positive $x_i + w_i$ satisfies the constraint 3.6. Similarly, constraint 3.7 and 3.8 become $y_j + h_j \le y_i + H$ and $x_j + w_j \le x_i + W$. The smaller positive values of $y_j + h_j$ and $x_j + w_j$ are smaller than $y_i + H$ and $x_i + W$ respectively which satisfy constraint 3.7 and 3.8. The setting of $(p_{ij}, q_{ij}) = (1, 1)$ forces the placement of the i^{th} room to top of the j^{th} room. The following figures present the i^{th} room placement on the left, the bottom, the right and the top for different values of p_{ij} and q_{ij}.

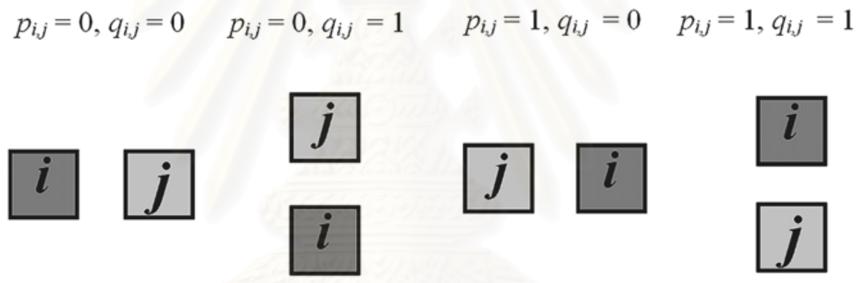

$p_{i,j} = 0, q_{i,j} = 0$
$p_{i,j} = 0, q_{i,j} = 1$
$p_{i,j} = 1, q_{i,j} = 0$
$p_{i,j} = 1, q_{i,j} = 1$
i
j
j
i
j
i
i
j

Figure 3.5: p_{ij} and q_{ij} represent the location of room i and room j.

- *Fixed position constraint* determines the room positioning in a boundary area. In a practical design, this constraint helps an architect to secure the room location in the design. For example, a high-rise building is fixed the lift core position in every levels.

$$x_i = X_i \tag{3.10}$$

$$y_i = Y_i \tag{3.11}$$

where x_i, y_i are the X and Y coordinate of room i, $\forall i = 1, \dots, n$,

X_i, Y_i are the fixed X and Y coordinate of room i, $\forall i = 1, \dots, n$.

Constraint 3.10 denotes the x_i of the room i fixed to the X coordinate while constraint 3.11 denotes the y_i of the room i fixed to the Y coordinate.

- *Unused unit cell constraint* determines the unusable area. This constraint helps an architect design various orthogonal boundary shapes. We use two binary variables (s_{ik}, t_{ik}) to identify the location of unused unit cell, kth, see figure 3.6.

$$x_i \geq x^u_k + 1 - W \times (s_{ik} + t_{ik}) \qquad (3.12)$$

$$x^u_k \geq x_i + w_i - W \times (1 + s_{ik} - t_{ik}) \qquad (3.13)$$

$$y_i \geq y^u_k + 1 - H \times (1 - s_{ik} + t_{ik}) \qquad (3.14)$$

$$y^u_k \geq y_i + h_i - H \times (2 - s_{ik} - t_{ik}) \qquad (3.15)$$

where x^u_k, y^u_k	are unused positions in X and Y coordinate of the unused kth unit, $\forall k = 1, \ldots, n$,	
s_{ik}, t_{ik}	are the decision binary variables of room i and k,	$\forall i < k = 1, \ldots, n$,
x_i, y_i	are the X and Y coordinate of room i,	$\forall i = 1, \ldots, n$,
w_i, h_i	are the width and height of the room i,	$\forall i = 1, \ldots, n$,
W, H	are the boundary width and height.	

Similar to four possible cases of connectivity constraint, the decision variables s_{ik} and t_{ik} force the room i to avoid the use of a unit cell k. The four possible cases of (s_{ik}, t_{ik}) are (0, 0), (0, 1), (1, 0) and (1, 1). The first case, (s_{ik}, t_{ik}) sets (0,0). The solution must satisfy $x_i \geq x^u_k + 1$ for constraint 3.12 which implies that the unit cell k will shift to the left of the ith room without covering it. Other constraints will satisfy unconditionally due to the large values of H and W, similar to situations in location constraints. The second case, (s_{ik}, t_{ik}) sets (0, 1). The solution must satisfy $x^u_k \geq x_i + w_i$ for constraint 3.13 which implies that the unit cell k will shift to the right of the ith room without covering it while other constraints will always satisfy. The third case, (s_{ik}, t_{ik}) sets (1, 0). The solution must satisfy $y_i \geq y^u_k + 1$ for constraint 3.14 which can implies that the unit cell k will float on the top of the ith room while other constraints will be satisfied. The last case, (s_{ik}, t_{ik}) sets (1,1). The solution must satisfy $y^u_k \geq y_i + h_i$ for constraint 3.15 which can implies that the unit cell k will fall under the ith room and other constraints will always satisfy, see figure 3.6 for illustrations.

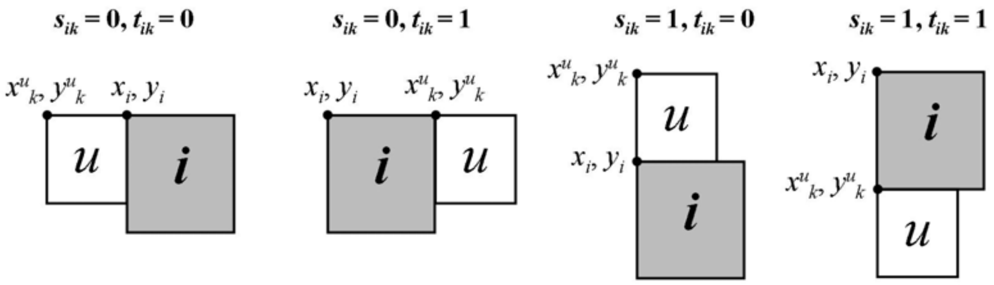


Figure 3.6: s_{ik} and t_{ik} represent connection of unused cell and room *i*.

- *Boundary constraint* forces a room to be inside a boundary.

$$x_i + w_i \leq W \tag{3.16}$$

$$y_i + h_i \leq H \tag{3.17}$$

where	x_i, y_i	are the X and Y coordinate of room *i*,	$\forall i = 1, \dots, n$
	w_i, h_i	are the width and height of the room *i* and *j*,	$\forall i = 1, \dots, n$
	W, H	are the boundary width and height.	

Constraint 3.16 denotes the room *i* within the horizontal boundary while constraint 3.17 denotes the room *i* within the vertical boundary.

- *Fixed border constraint* addresses the absolute placement of the room. This constraint is divided into four types: the north(top), the south(bottom), the east(right) and the west(left). For example, a room is positioned to the north if its touch the top border, see figure 3.7.

$$y_i = 0 \tag{3.18}$$

$$y_i + h_i = H \tag{3.19}$$

$$x_i + w_i = W \tag{3.20}$$

$$x_i = 0 \tag{3.21}$$

where	x_i, y_i	are the X and Y coordinate of room *i*,	$\forall i = 1, \dots, n,$
	w_i, h_i	are the width and height of the room *i* and room *j*,	$\forall i = 1, \dots, n,$
	W, H	are the boundary width and height.	

For the fixed border constraint, the decision variables p_{ij} and q_{ij} force the room i touching the four side of boundary as follows. Constraint 3.18 denotes the y_i of room i touch the top boundary. Constraint 3.19 denotes the $y_i + h_i$ of room i touch the bottom boundary. Constraint 3.20 denotes the $x_i + w_i$ of room i touch the right boundary while constraint 3.21 denotes the x_i of room i touch the left boundary, see figure 3.7.

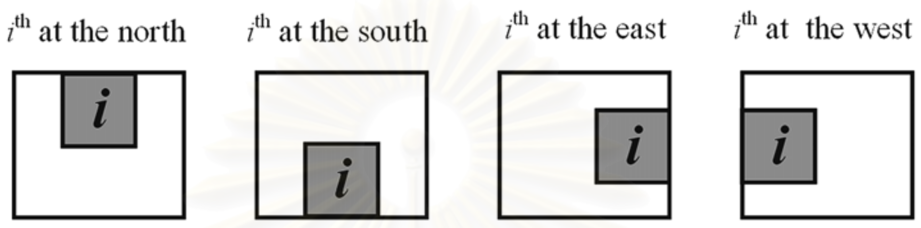


Figure 3.7: The fixed room presents at each layout boundary.

- *Connectivity constraint* forces two connecting rooms to be placed next to one another. We use the same two binary variables p_{ij} and q_{ij} with different set of constraints, see Figure 3.7.

$$x_i + w_i \geq x_j - W \times (p_{ij} + q_{ij}) \tag{3.22}$$

$$y_j + h_j \geq y_i - H \times (1 + p_{ij} - q_{ij}) \tag{3.23}$$

$$x_j + w_j \geq x_i - W \times (1 - p_{ij} + q_{ij}) \tag{3.24}$$

$$y_i + h_i \geq y_j - H \times (2 - p_{ij} - q_{ij}) \tag{3.25}$$

where	x_i, x_j	are the X coordinate of the room i and j,	$\forall i < j = 1, \dots, n$,
	y_i, y_j	are the Y coordinate of the room i and j,	$\forall i < j = 1, \dots, n$,
	w_i, w_j	are the width of the room i and j,	$\forall i < j = 1, \dots, n$,
	h_i, h_j	are the height of the room i and j,	$\forall i < j = 1, \dots, n$,
	W, H	are the boundary width and height.	

Applying the location constraints with connectivity constraints, the room i^{th} is forced to contact room j^{th} at the right, the top, the left and the bottom corresponding the four possible cases of (0, 0), (0, 1), (1, 0) and (1, 1). The first case (p_{ij}, q_{ij}) sets (0, 0). The solution must satisfy $x_i + w_i \geq x_j$ for constraint 3.22 which implies that the i^{th} room will contact at the right of j^{th} room. Other constraints will satisfy unconditionally due to the large value of H and W, similar to scenarios in connectivity constraints. The second case (p_{ij}, q_{ij}) sets (0, 1). The solution must satisfy $y_j + h_j \geq y_i$ for constraint 3.23 which implies that the the ith room will contact at the top of jth room. Other constraints will always satisfy. The third case, (p_{ij}, q_{ij}) sets (1, 0). The solution must satisfy $x_j + w_j \geq x_i$ for constraint 3.24 which can implies that the ith room will contact at the left of jth room. The last case, (p_{ij}, q_{ij}) sets (1, 0). The solution must satisfy $y_i + h_i \geq y_j$ for constraint 3.25 which can implies that the ith room will contact at the bottom of jth room and other constraints will always satisfy, see figure 3.8 for illustration.

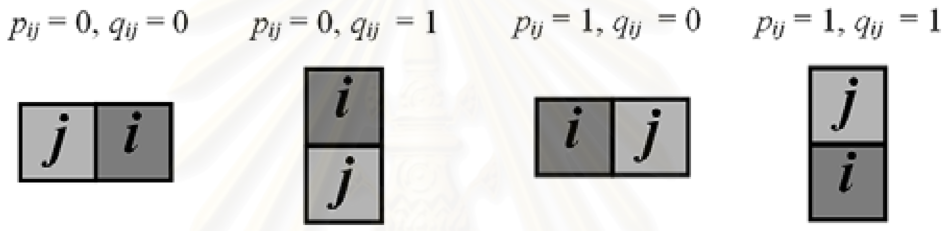


Figure 3.8: p_{ij} and q_{ij} are reused to formulate the connectivity relation between room i and room j.

- *Access-way constraint* forces the minimal contact length between two connected rooms. Two rooms are touching each other with the minimal contact length defined by the value (T_{ij}). For example, the junction between room i and room j must be wide enough to accommodate an access way, the same binary variables q_{ij} have been reused. Only q_{ij} has been used due to the fact that the vertical contact is allowed to be placed on the left (p_{ij} = 0) or on the right of the room j (p_{ij} = 1). This also true for the horizontal contact which ignores the placement of the room i above (p_{ij} = 0) and below (p_{ij} = 1) the room j.

$$H \times (q_{ij}) \geq y_i + T_{ij} - y_j - h_j \qquad (3.26)$$

$$H \times (q_{ij}) \geq y_j + T_{ij} - y_i - h_i \qquad (3.27)$$

$$W \times (1 - q_{ij}) \geq x_i + T_{ij} - x_j - w_j \qquad (3.28)$$

$$W \times (1 - q_{ij}) \geq x_j + T_{ij} - x_i - w_i \qquad (3.29)$$

where	*Tij*	is the contact length of the access-way between room *i* and *j*, $\forall i < j = 1, \dots, n$,	
	x_i, x_j	are the X coordinate of the room *i* and *j*,	$\forall i < j = 1, \dots, n$,
	y_i, y_j	are the Y coordinate of the room *i* and *j*,	$\forall i < j = 1, \dots, n$,
	w_i, w_j	are the width of the room *i* and *j*,	$\forall i < j = 1, \dots, n$,
	h_i, h_j	are the height of the room *i* and *j*,	$\forall i < j = 1, \dots, n$,
	W, H	are the boundary width and height.	

Two possible cases of q_{ij} are 0 and 1. The first case, q_{ij} is set to 0 for the vertical contact. The solution must satisfy $0 \geq y_i + T_{ij} - y_j - h_j$ for constraint 3.26. This overlapping region of room *i* and room *j* will appear to the upper corner of room *i*. For $0 \geq y_j + T_{ij} - y_i - h_i$ of constraint 3.27, the overlapping region of room *i* and room *j* will appear to the lower corner of room *i*. Constraints, 3.30 and 3.31 will satisfy unconditionally due to the large value of *W*. The second case, q_{ij} is set to 1 for the horizontal contact. The solution must satisfy $0 \geq x_i + T_{ij} - x_j - w_j$ for constraint 3.28. This overlapping region of room *i* and room *j* will appear to the left corner of room *i*. For $0 \geq x_j + T_{ij} - x_i - w_i$ of constraint 3.29, the overlapping region of room *i* and room *j* will appear to the right corner of room *i*. Constraints, 3.26 and 3.27 will also satisfy unconditionally due to the large value of *H*, see figure 3.9 for illustration.

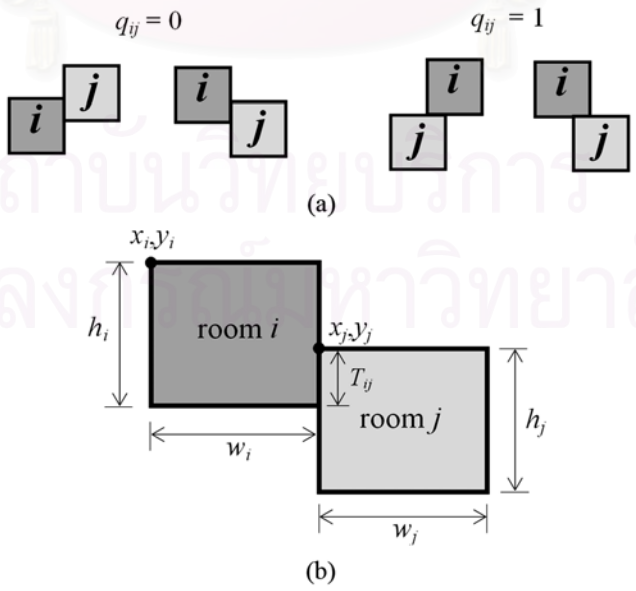


Figure 3.9: (a) q_{ij} is reused to formulate the overlapping region between room *i* and room *j* and (b) T_{ij} represents a minimal contact length between room *i* and room *j*.

- *Length constraint* determines minimal and maximal lengths of the bounded size of each room. A certain room is adjusted to appropriate dimensions between the horizontal range of $w_{\min,i}$, $w_{\max,i}$ and the vertical range of $h_{\min,i}$, $h_{\max,i}$ respectively.

$$w_{\min,i} \leq w_i \leq w_{\max,i} \tag{3.30}$$

$$h_{\min,i} \leq h_i \leq h_{\max,i} \tag{3.31}$$

where	$w_{min,i}$, $w_{\max,i}$	are the minimal and maximal width of room *i*,	$\forall i = 1, \dots, n$,
	$h_{min,i}$, $h_{\max,i}$	are the minimal and maximal height of room *i*,	$\forall i = 1, \dots, n$,
	w_i, h_i	are the width and height of room *i*,	$\forall i = 1, \dots, n$.

Constraint 3.30 denotes the width of room *i* within the minimal and maximal length while constraint 3.31 denotes the height of room *i* within the minimal and maximal length of room *i*.

3.2 Valid Inequalities Optimization Model

Due to the work of Keatruangkamala and Sinapiromsaran (2003), the solution time to solve the architectural layout design problem as the multiobjective MIP model is prohibitive for a medium to large problem size. In this thesis, we propose two modeling techniques to handle this problem. The first technique is to add valid inequalities of non-circular connectivity of the three consecutive rooms to the AL-MIP model, called non-circular AL-MIP+. The second is to apply a room location based on architect's preferences called advised AL-MIP+ that help eliminate some alternative solutions. These two techniques will be presented as follow.

3.2.1 Non-Circular Connectivity Constraints

The first technique, we use an idea of valid inequality (George and Laurence, 1988) which based on a smaller feasible region. By which, the LP relaxation region has been cut off while all integral points are maintained. The remaining LP relaxation region is strictly smaller than the LP relaxation of the original one with the corner extreme points are forced to be integral. The notion of the valid inequality can be formulated as follows.

Given the IP (Integer programming problem) as

(IP) $\max\{ \boldsymbol{c}^{T}\boldsymbol{x} : \boldsymbol{x} \in X \}$

$X = \{ \boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} , \boldsymbol{x} \in \mathbb{Z}^{+} \}$

The inequality $\boldsymbol{\pi}^{\mathbf{T}}\boldsymbol{x} \leq \pi_0$ is called a valid inequality for X if $\boldsymbol{\pi}^{\mathbf{T}}\boldsymbol{x} \leq \pi_0$ for all $\boldsymbol{x} \in X$, see figure 3.10.

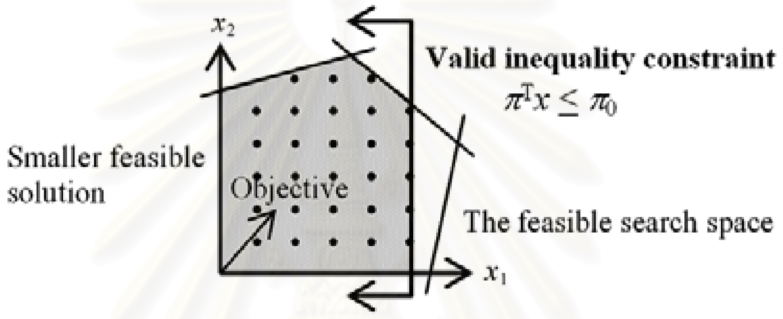

x_2
Valid inequality constraint
$\pi^T x \leq \pi_0$
Smaller feasible solution
Objective
The feasible search space
x_1

Figure 3.10: Valid inequality constraint cut off the non-integral feasible region.

The valid inequality for the architectural layout design problem is generated using the concept of the non-circular connectivity constraints. These inequality constraints are defined among three consecutive rooms i, j and k, connected in this order. The binary variables p_{ij}, p_{ik}, p_{jk}, q_{ij}, q_{ik} and q_{jk} from the AL-MIP model are used to present room connectivity. The consecutive connectivity of room i and j prohibits the placement of room i between room j and k, see figure 3.11. Therefore, the valid inequalities force the non-circular connection of the room i and k which eliminates configuration formed by four different directions, top, left, right and bottom of the i^{th} room and the j^{th} room. The non-circular connectivity constraints for each direction have been illustrated as follows.

$$p_{ik} - q_{ik} \leq W \times (p_{ij} + q_{ij}) \tag{3.32}$$

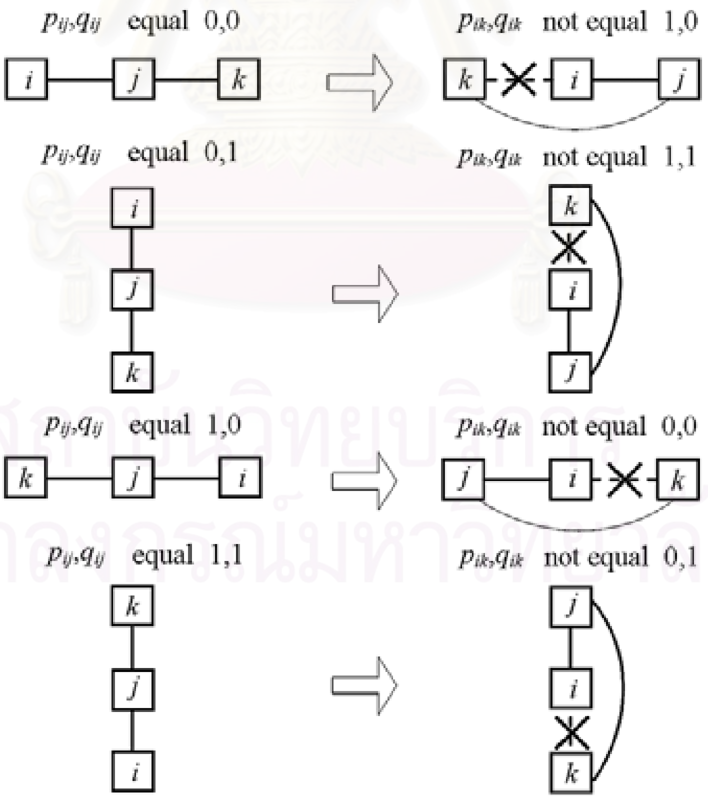
$$p_{jk} + q_{jk} - 1 \leq H \times (1 + p_{ij} - q_{ij}) \tag{3.33}$$

$$1 - p_{ik} - q_{ik} \leq W \times (1 - p_{ij} + q_{ij}) \tag{3.34}$$

$$q_{ik} - p_{ik} \leq H \times (2 - p_{ij} - q_{ij}) \tag{3.35}$$

where	$p_{ik}, p_{jk}, q_{ik}, q_{jk}$	are the decision binary variables,	$\forall i < j < k = 1, \ldots, n,$
	p_{ij}, q_{ij}	are the decision binary variables,	$\forall i < j = 1, \ldots, n,$
	W, H	are the boundary width and height.	

For the non-circular connectivity constraint, the decision variables p_{ij} and q_{ij} prohibit the room k connect to the left, the top, the bottom and the right of room i. corresponding to constraint 3.32, 3.33, 3.34 and 3.35. Four possible cases of (p_{ij}, q_{ij}) are (0,0), (0,1), (1,0) and (1,1). The first case, constraint 3.32 (p_{ij}, q_{ij}) sets (0,0). It forces the room j connect to right side of room i and prohibits room k to the left of room i. The second case, constraint 3.33 (p_{ij}, q_{ij}) sets (0,1). It forces the room j connect to the bottom of room i and prohibits room k in the above of room i. The third case, constraint 3.34 (p_{ij}, q_{ij}) sets (1,0). It forces the room j connect to left side of room i and prohibits room k to the right side of room i. And The fourth case, constraint 3.35 (p_{ij}, q_{ij}) sets (1,1). It forces the room j connect to above of room i and prohibits room k to the bottom of room i.



p_{ij},q_{ij} equal 0,0
p_{ik},q_{ik} not equal 1,0
p_{ij},q_{ij} equal 0,1
p_{ik},q_{ik} not equal 1,1
p_{ij},q_{ij} equal 1,0
p_{ik},q_{ik} not equal 0,0
p_{ij},q_{ij} equal 1,1
p_{ik},q_{ik} not equal 0,1
i
j
k

Figure 3.11: (Left) four possible connected scenarios defined by consecutive rooms *i, j* and *k* and (right) four corresponding scenarios that are eliminated from consideration.

3.2.2 Advised Configuration Constraints

To decrease the computational time, we proposed another inequality constraints based on the architect's preferences. Traditionally, some room positions in an architectural layout design was often placed extremely to the north, the south, the east and the west directions, for example, architects fix a bedroom on the north or the east direction to avoid the sunlight corresponding to the benign Feng Shui (Chinese belief). According to this traditional belief, we define constraints to allocate the bedroom on a required direction. By which, we proposed the advised configuration constraints to allocate the room positioning based on an architect's preference. These constraints can be used to eliminate an infeasible solution immensely. The following constraints present the allocation of the advised room i' for all $j \in \{1, 2, \dots, 3\}$, see figure 3.12.

$$y_{i'} \leq y_j \tag{3.36}$$

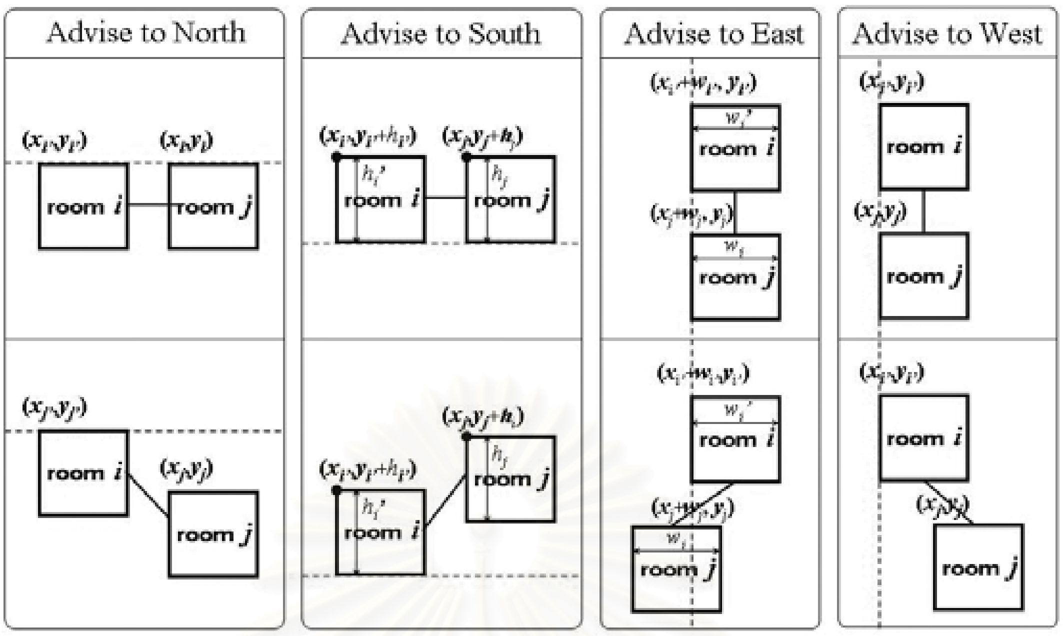
$$x_{i'} \leq x_j \tag{3.37}$$

$$x_j + w_j \leq x_{i'} + w_{i'} \tag{3.38}$$

$$y_j + h_j \leq y_{i'} + h_{i'} \tag{3.39}$$

where	$x_{i'}, y_{i'}$	are the X and Y coordinates of advised room i' in a required direction, $\forall i' = 1, \dots, n$,	
	x_j, y_j	are the X and Y coordinates of room j,	$\forall j = 1, \dots, n$,
	$w_{i'}, h_{i'}$	are the width and height of advised room i',	$\forall i' = 1, \dots, n$,
	w_j, h_j	are the width and height of room j,	$\forall j = 1, \dots, n$.

For the four possible cases of advise configuration constraint, the first case, constraint 3.36 denotes an advised room i' to the above direction of room j. The second case, constraint 3.37 denotes an advised room i' to the left direction of room j. The third case, constraint 3.38 denotes an advised room i' to the right direction of room j while the fourth case, constraint 3.39 denotes an advised room i' to the below direction of room j.



(a) (b) (c) (d)

Figure 3.12: The advised configuration constraints, (a) advised room *i'* at the north(top) of room *j*, (b) advised room *i'* at the south(bottom) of room *j*, (c) advised room *i'* at the east(right) of room *j* and (d) advised room *i'* at the west(left) of room *j*.

The following figure presents the conceptual solution spaces that these two sets of valid inequality constraints of non-circular AL-MIP+ and advised AL-MIP+ have been applied to cut off an infeasible solution on search space.

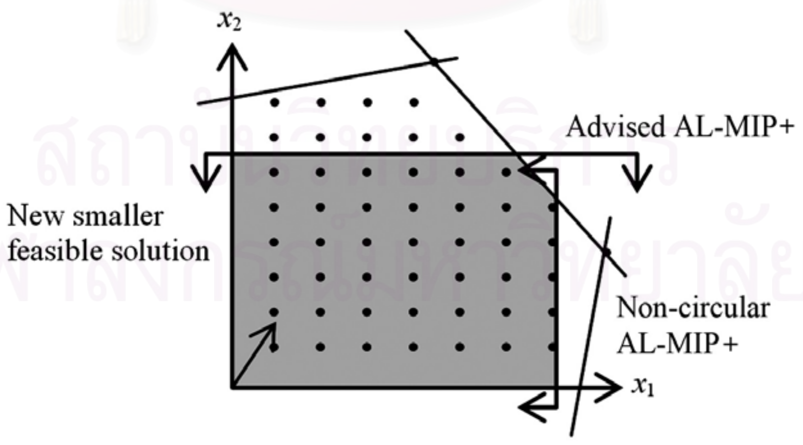


Figure 3.13: The new smaller feasible area is cut off by the valid inequality constraints.

CHAPTER IV
Machine Learning Using Genetic Algorithms

4.1 Learning Special Order Set

From the previous section, we have formulated the AL-MIP model fit to the architectural layout design that can deal with a small-sized problem. The two valid inequality constraints called AL-MIP+ have been used to reduce the computational MIP iterations and time. In order to accelerate the computational speed, the machine learning has been adopted. The robustness learning methodology Genetic Algorithms (GA) utilized an idea of the Special Order Set (SOS) based on the branching in a branch and bound algorithm.

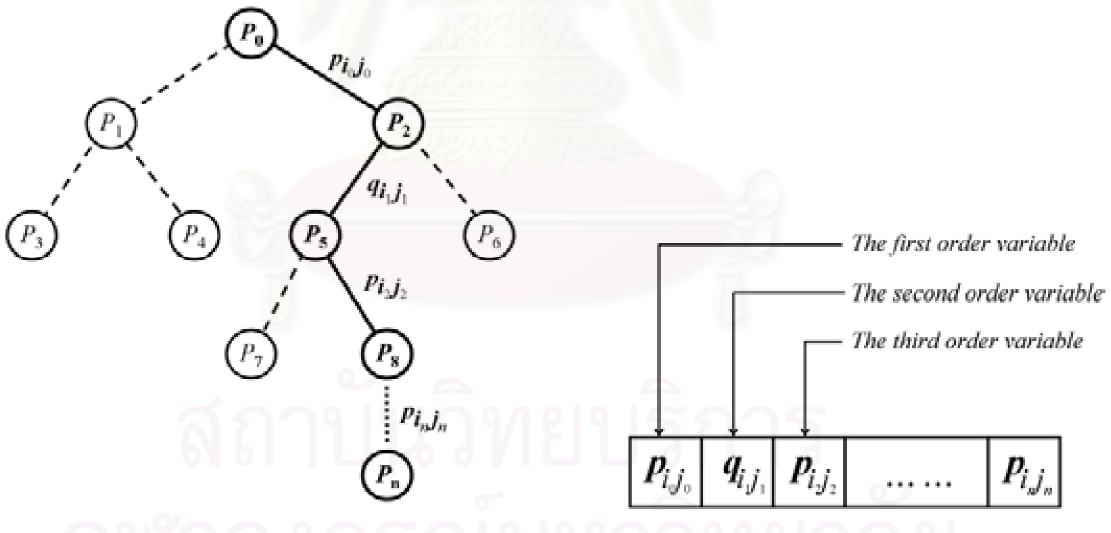
In details, the branch and bound algorithm is equipped with a best-first search strategy. After branching at a certain level, a node is selected based on the current best node and branched, then, another node is selected at the new level and branched, and so on until the last level is reached and a complete solution is obtained. The complete solution is marked as "best node", and the algorithm tracks back and eliminates nodes with a lower bound worse than the "best so far" solution. Otherwise, the algorithm branches another node and proceeds forward.

The learning algorithm based on the robustness GA has been adopted as the unsupervised learning to the branch and bound search tree. The MIP solver using the branch and bound algorithm utilize the learning algorithm GA to find an appropriate sequences of branching variables. The SOS variable is used to guide the sequences of the branching strategy which searches throughout the problem space with the variables p_{ij} and q_{ij} (see the previous section 3.1.1). After complete the learning process, the stronger gene from GA represents the appropriated SOS variables with a good path of branching in the search tree. By which the appropriated order variables of the problem constraints help reduce the search space by identifying the better candidate solution used to prune the search tree.

4.2 Genetic Algorithms Methods

4.2.1 Chromosomes

To encode a SOS into a chromosome, an idea of GA from Traveling Saleman Problem (TSP) has been adopted in this thesis. Figure 4.1(a) illustrates an idea of the branching branch and bound algorithm which are guided by the candidate SOS variables p_{ij} and q_{ij}. In the figure, the given problem P_0 is traced to the subproblems P_2, P_5, P_8, …. , P_n using the branching variables $p_{i_0j_0}$, $q_{i_1j_1}$, $p_{i_2j_2}$, …. , $p_{i_nj_n}$ respectively. Similarly, figure 4.1(b) illustrates the chromosome which corresponded to an idea of branching in figure 4.1(a). Each chromosome is constructed by designing the first-branch order $p_{i_0j_0}$ is placed at the extreme left of the chromosome, the second-branch order $q_{i_1j_1}$ and the third-branch order $p_{i_2j_2}$ are placed at the second and the third orders from the extreme left and the last order $p_{i_nj_n}$ is placed at the extreme right of the chromosome.



(a) (b)

Figure 4.1: (a) the sequential orders p_{ij} and q_{ij} in branch and bound algorithm start from the top(root) to the bottom node and (b) The corresponded orders p_{ij} and q_{ij} based on GA structural representation is constructed from the extreme left to the extreme right.

- **String Representation**

In order to encode a sequential order of the SOS variables p_{ij} and q_{ij}, a binary string representation has been applied to capture an idea of a SOS variables p_{ij} and q_{ij}. The binary string is flexible for the GA of reproduction, crossover and mutation to create a new population. Nevertheless, the SOS variables p_{ij} and q_{ij} have more information to fit with a one dimension (1D) binary string. This thesis utilizes two dimension (2D) binary string to capture entire information of the SOS variables p_{ij} and q_{ij}. The space of 2D binary string is $m \times n$, where the m presents the numbers of variables and the n presents the sequential order of variables p_{ij} and q_{ij}. The space of 2D binary string is depended on the numbers of variables p_{ij} and q_{ij} that has been used in the problem.

According to the numbers of connectivity variables between room i and room j, the length of binary string of each column is designed to cover, covering all possible cases of variables p_{ij} and q_{ij}. Also, the length of a binary string of each row will cover all sequential orders of SOS variables in branch and bound algorithm. The string length in each row can generate using the idea of the combination, see the next section for details.

- **Encoding Schema**

In genetics, the whole information of an individual structure is stored in a chromosome as genetic codes. The genome string is composed of a finite set of genes and their values. In the artificial world, a gene can be considered as an instruction in a recipe and is represented as a particular character or a set of characters in an encoding string.

To encode the SOS, 2D binary strings have been utilized to represent all information. For example, an instance of 5 rooms is used to describe the sequential SOS schema. With the 5 rooms, we have 20 variables of p_{ij} and q_{ij} that used in the SOS. Therefore, these variables need five bits to represent all possible cases of variables p_{ij} and q_{ij}. However, a five bit string can represent 32 different patterns which is larger than the number of the variables p_{ij} and q_{ij}. The remaining patterns will not be used to represent the variable p_{ij} and q_{ij}. Thus, for example, 00001 represents a variable p_{12}, 00010 represents a variable p_{13}, 00011 represents a variable p_{14}, 01011 represents a variable q_{12}, 01100 represents a variable q_{13} and 10101 represents a variable q_{45} which all numbers of zero (00000) are not used for representing the branching order, see the figure 4.2.

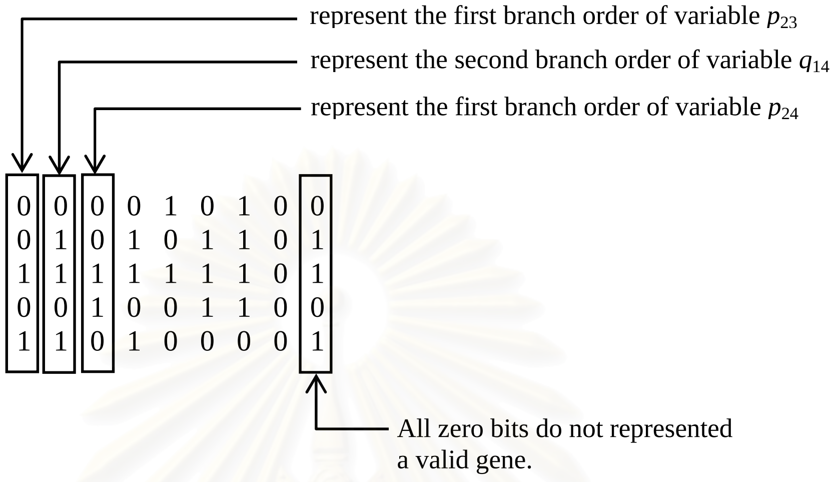

represent the first branch order of variable p_{23}
represent the second branch order of variable q_{14}
represent the first branch order of variable p_{24}
0 0 0 0 1 0 1 0 0
0 1 0 1 0 1 1 0 1
1 1 1 1 1 1 1 0 1
0 0 1 0 0 1 1 0 0
1 1 0 1 0 0 0 0 1
All zero bits do not represented a valid gene.

Figure 4.2: The 2D binary strings of 5 rooms represent the SOS variables.

Nevertheless, if the current pattern is not represented by any SOS variable, the algorithm will ignore and proceed with the next variable, and the index of this variable is not stored into a candidate SOS. This method ensures that only feasible SOS is created and will be used in the chromosome.

4.2.2 Mechanism for Creating Generations

- **Creating Generations**

The roulette wheel is used to create a new generation. The fitness of a particular chromosome determines the size of its segment on the roulette wheel. The roulette wheel is then spun repeatedly to produce a new population of the same size as the initial population. The algorithm will complete when the required number of generations has been reached. It displays the candidate SOS associated with the chromosome with the highest fitness.

- **Crossover**

At the initial population gene, the individual genes are randomly selected to produce a population of chromosomes (candidate solutions). This process is repeated to produce a population of the specific size. The Chormosomes are randomly selected for crossover and mutation operations with the probability settings based on the paper (Al-hakim, 2000). The order crossover using two parents and two crossover sites are selected randomly and the elements between the two selecting points in one of the parent are directly inherited by the offspring.

- **Mutation**

Mutation is the process applied to each offspring individually after the crossover. This operator creates new individual chromosome by a small change in a single individual chromosome by a random selection. In this thesis, the encoded SOS using the 2D binary string that the mutation is applied to a bit string. It sweeps down the bits and replace each by randomly selected bit if the probability of test passes.

4.2.3 Fitness Function

In order to describe the details of evaluate fitness function. This thesis uses the MIP optimization solver called CPLEX to evaluate the fitness value of GA. The setting of the CPLEX solver will be described below.

- **The optimization CPLEX solver**

CPLEX is an optimization software package. It is named for the simplex method and the C programming language. It was originally developed by Robert E. Bixby and distributed via CPLEX Optimization Inc. CPLEX can solve MIP problem and very large LP problems. Moreover, it has a modeling layer and is also available with several modeling systems like AIMMS, AMPL, GAMS IDE and OPL Development Studio. In this thesis, we develop a modeling language based on GAMS IDE and solve the model on CPLEX version 9.0.

- **Fitness evaluation**

As far as GA is concerned, it’s better to have a higher fitness value to provide more opportunities to be chosen in breeding new chromosomes. In this thesis, the CPLEX solver has been used to solve the MIP using the SOS variables from GA which determines the largest score from the number of iterations.

At each transition, the value of computational iterations from AL-MIP+GA (*fitness_score*) is subtracted from a standard fitness score (*standard_fitness_score*) which is obtained from the computational iterations of the AL-MIP+. The *fitness_score* higher than the *standard_fitness_score* presents a better candidate of SOS (a strong gene) which will be stored into a text file. The GA fitness is measured from the subtraction of the computational iterations of AL-MIP+ and the current computational iteration of AL-MIP+GA. We can describe the GA fitness with an equation as follow.

$$\text{Evaluate Fitness} = \mathit{Standard\_fitness\_score} - \mathit{fitness\_score} \tag{4.1}$$

Figure 4.3 presents an entire idea of GA using the flow diagram that is adopted for AL-MIP+GA. To operate the GA, we input the populations, the generations and the *standard_fitness_score* from the AL-MIP+ into the AL-MIP+GA. The learning process will terminate by the current generations (*Gen*) in each run over the required generation (*MaxGenerations*) and the GA statistics will be summarized into a text file.

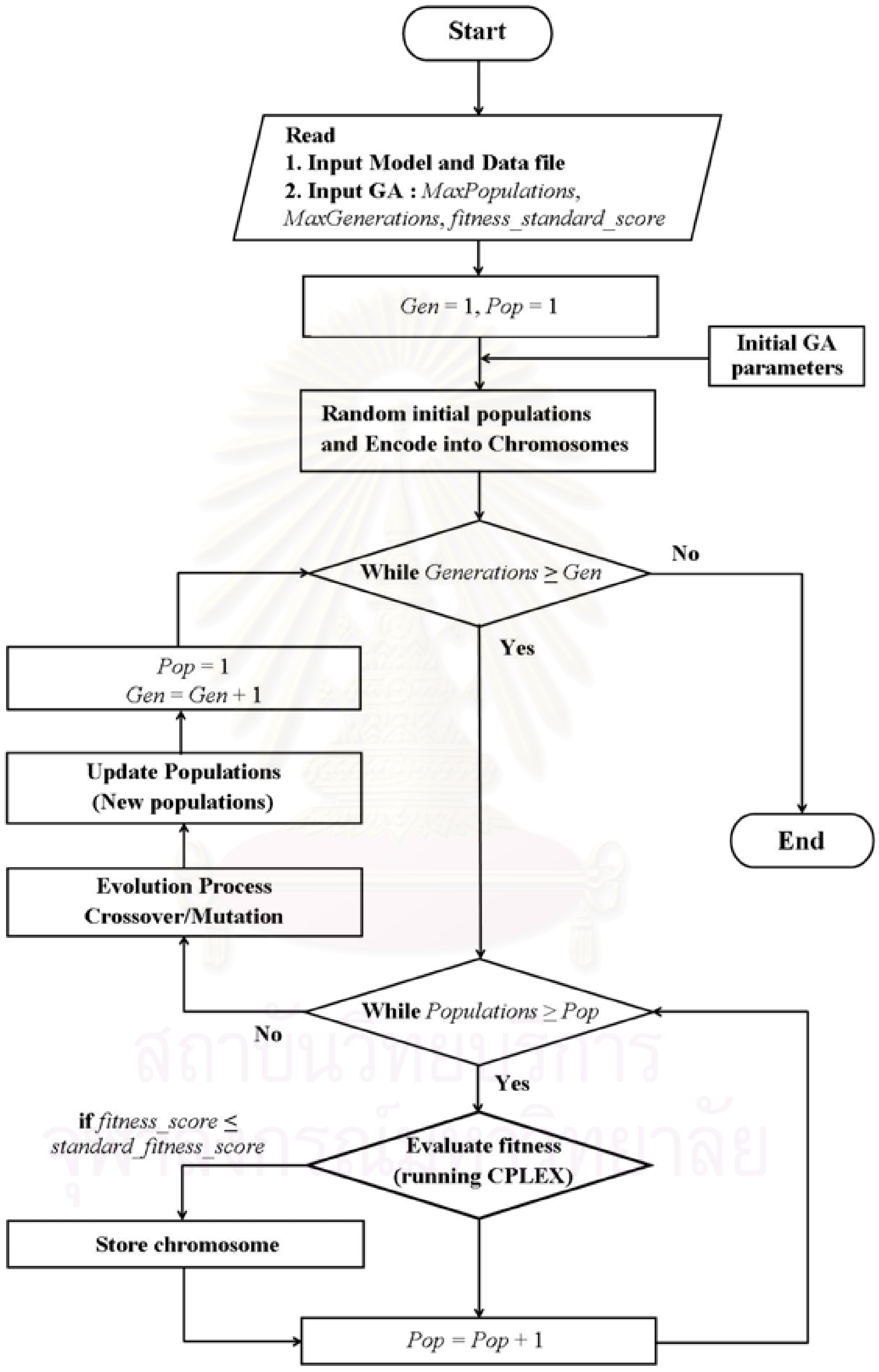

Start
Read
1. Input Model and Data file
2. Input GA : MaxPopulations, MaxGenerations, fitness_standard_score
Gen = 1, Pop = 1
Initial GA parameters
Random initial populations and Encode into Chromosomes
While Generations ≥ Gen
No
Yes
End
Pop = 1
Gen = Gen + 1
Update Populations (New populations)
Evolution Process Crossover/Mutation
While Populations ≥ Pop
No
Yes
if fitness_score ≤ standard_fitness_score
Evaluate fitness (running CPLEX)
Store chromosome
Pop = Pop + 1

Figure 4.3: Flow diagram of AL-MIP+GA based on an idea of GA.

4.2.4 Genetic Algorithms Application

The flow diagram from the previous section is used as a programming framework. Our AL-MIP+GA algorithm is developed based on the GNU C++ programming language. The GA library designed by Matthew Wall (MIT) has been used as the GA computational class which will be described in details as follows.

- **Genetic Algorithms C++ Library**

The GA library designed by Matthew Wall (1996) contains four flavors of GA. The first is the standard simple GA described by Goldberg. This algorithm uses non-overlapping populations. For each generation, the algorithm creates an entirely new population of individuals. The second is a steady-state GA that uses overlapping populations. In this variation, users can specify how much of the population should be replaced in each generation. The third variation is the incremental GA, in which each generation consists of only one or two additional children. The incremental GA allows custom replacement methods to define how the new generation should be integrated into the population. For example, a newly generated child can replace the parents, replace a random individual in the population. This GA library evolves the multiple populations in parallel using a steady-state algorithm. The algorithm migrates some of the individuals from each population to one of the other populations.

This GA library has been designed to report the statistics, replacement strategy, and parameters for running the algorithm. The population object, a container for genomes, also contains some statistics as well as selection and scaling operators. This library has built in functions for specifying when the algorithm should terminate. These include termination upon generation using a specified certain number of generations. The stopping criteria can be designed by the terminated function that is built as a library module. Moreover, this GA library keeps track of both the number of genome evaluations and population evaluations and stores into a text file.

- **Pseudo code**

We develop a computer program using GA C++ lib. Our GA learning SOS application can be described as a pseudo code based C++ programming language format as follows.

```
chromosome SOS[];  //Special Order Set variables
int fitness_score[];  //Chromosome fitness score
main learning_AL-MIP()
{   Input file: Model file and Data file;
    Input GA parameters: int MaxPopulations, MaxGenerations, Standard_fitness_score;
    int Gen = 1, Pop = 1;
    Initial GA parameters;
    SOS[] = Random initial populations and encode into chromosomes;
    While MaxGenerations ≥ Gen
    {       While MaxPopulations ≥ Pop
            {       fitness_score[Pop] = Evaluate_Fitness(SOS[Pop]);
                    If  fitness_score[Pop]  ≤  Standard_fiteness_score then
                        Store SOS[Pop] and fitness_score[Pop] as a good chromosome;
                    End if
                    Pop = Pop + 1;
            }//end while
            //Evolution process and Update old chromosome
            SOS[] = Evolutionary_Process(SOS[],  fitness_score[]);
            Pop = 1; Gen = Gen + 1;
    }//end while
}
int Evaluate_Fitness(SOS)
{   return Run CPLEX solver with this SOS;
}
chromosome Evolutionary_Process(SOS[],  fitness_score[])
{   Crossover(SOS[],  fitness_score[]);
    Mutation(SOS[],  fitness_score[]);
    return SOS[];
}
```

CHAPTER V
Experimental Results

5.1 Experimental Design

The experiments presented in this thesis have been carried out on a PC computer using Pentium Core 2 Duo and 4.0 GB of memory using GAMS CPLEX 9.0. In order to measure the performance, we simulate architectural layout design instances with 4, 5, 6, 7, 8, 9 and 10 rooms based on four distinct configurations, which are

1) a linear configuration

2) a rail configuration

3) a connected wheel configuration

4) a nested wheel configuration

See figure 5.1 for the graphical representations of these four distinct patterns.

Each configuration composes of five instances and the average measurements have been recorded. Total of 140 experimental runs have been tested and gathered. Our experiment unit scale corresponds to a meter scale. The boundary area is set on 100×100 square meters and defined the minimum and maximum room width and height between 5 to 10 meters. Moreover, the weighted-sum of u_1, u_2 and u_3 are equivalently set to 1.

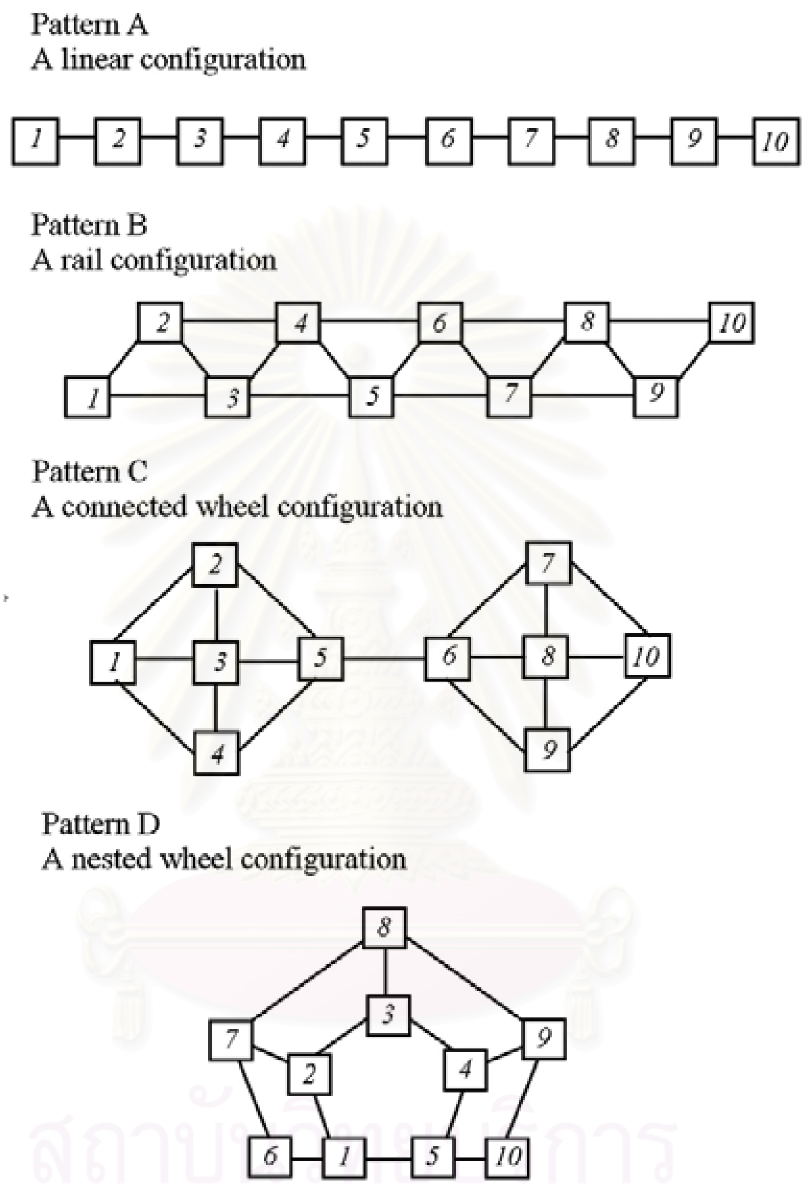


Figure 5.1: The distinct pattern A, B, C and D of 10 room configurations.

5.2 Genetic Algorithm Parameters and Design

- **Parameters Setting in GA**

GA requires parameter tunings in order to achieve the desirable solutions and performance. Three common GA parameters are population size, crossover probability, and mutation probability. The population size parameter is a major factor in determining the quality of the solutions. Setting small population size will cause the GA to converge to suboptimal solutions. On the other hand setting large population size will cause the GA to waste unnecessary computational resources. The generic crossover probability parameter is set between 0 and 1 that is enough to determine the amount of gene swapping between the parent solutions. Crossover operator is important because it ensures good mixing of candidate solutions. The higher crossover probability, the more promising solutions are mixed. This also increases the disruption of good solutions. The generic mutation probability parameter is set between 0 and 1 to determine the amount of mutation on a solution. Mutation operator is important because it enables diversity in the population. With a high mutation probability, it will behave similar to an intelligent hill-climbing strategy, in the neighborhood of a particular solution, but it may also destroy already found good solutions. The task of tuning these GA parameters has been proven to be far from trivial due to the complex interactions among the parameters and their proper settings. Several researchers have been trying to understand the interdependencies of GA parameters. One of the first empirical studies to understand the complex interactions and interdependencies of GA parameters was investigated by De Jong (1975). Based on his studies, De Jong introduced a good set of parameter settings that have been adopted widely and sometimes referred to as “standard” settings: population size of 50 to 100, crossover probability of 0.9, and mutation probability of 0.001. However, these “standard” settings have been proven problematic by later studies, which suggest that the optimal settings of GAs’ parameters are critically dependent on the nature of the function being evaluated (Goldberg, 1985; Hart & Belew, 1991; Deb, 1999a) and the encoding of decision variables.

In the real world problem, several researchers (Pelikan et al., 2000) have spent much effort on trying to design a GA parameter model and checks their models against some real-world problems. Lobo (2000) suggests using an appropriated GA parameter that determines the parameters through trial and error on the real world problem. In this thesis, the various characteristics and parameters of the steady-state GA, are determined by preliminary experiments from a small instance of 5 room configurations.

Furthermore, the distinct patterns A, B, C and D using the fixed GA parameter settings may lead to slow convergence and sub-optimal solutions, especially when large search spaces are to be explored in solving the optimization problems. To remedy this problem, the distinct patterns A, B, C and D are experimented for appropriated GA parameters that fit to each one. The appropriated GA parameters in populations, generations, crossover and mutation will be described in details.

Population size: the population size has to be considered carefully. If the population size is too small, the population will soon suffer from premature convergence because the diversity in the population is too low. In the other word, if the size is too large the convergence towards the optimum is slow and the memory requirements to run the genetic algorithm increase enormously.

In this thesis, we find out an appropriated GA population size using an experiment on a 5 room configuration of the distinct patterns A, B, C and D. By which, more than a thousand instances is experimented on the common population sizes of 10, 20, 30 and 50 (Lobo 2000, Pelikan et al., 2000).

Crossover and mutation probability: Recombination and mutation is performed with a certain fixed or variable probability. Again the setting of these parameters is the subject of deliberation. Whereas the more classical genetic algorithm theorists fervently advocate the use of a high crossover probability in the range [0.8, 1] (Goldberg 1989, Holland 1975) and a low mutation probability in the range [0, 0.01] (Goldberg, 1989; Holland, 1975).

Based on the three important results on the mutation operator, the practical result is that the lower bound for the mutation probability p_m is $p_m = 1/l$ with l the length of the chromosome (Muhlenbein 1992). This provides a more mathematical background for the mutation parameter. The probability that a chromosome with length l is not modified by mutation is:

$$P_s = (1 - P_m)^l \tag{5.1}$$

where P_m represents the bit mutation probability. If there is no crossover operator the probability of survival P_s should be no less than the inverse of the expected number of offspring.

In the past, several researchers use mutation with a low probability but the empirical and theoretical investigations demonstrated the benefits of the role of mutation as a search operator. The high levels of mutation are the most disruptive and also achieve the lowest levels of construction. This means that by using high levels of mutation the chance that new candidate gene are found decreases.

In this thesis, the experiments of the distinct crossover and the distinct mutation are experimented based on the 5 room configurations of patterns A, B, C and D. The crossover probability of 0.5, 0.8, 0.9 and 1.0, the mutation probability of 0.0005, 0.001, 0.005, 0.01 and 0.05 are experimented on thousand instances.

String length: a length of the candidate SOS is represented by a combination from the room connectivity of $i = 1, 2, 3, \ldots, n$ and $j = 1, 2, 3, \ldots, n$ where n is the numbers of room. All available SOS variables are consecutively filled in a chromosome with an index given by the ordering variables p_{ij} and q_{ij}. To find the SOS variable lengths in the chromosome, the combination has been adopted to determine the maximum numbers of the length. Due to the AL-MIP+GA, two decision binary variable p_{ij} and q_{ij} identify the connectivity between each room i and room j. The total results of an SOS variable consists of both variable p_{ij} and q_{ij}. The equation 5.2 presents the combination equation that is used to determine an SOS variable length in a candidate SOS.

$$\mathrm{C}(n,r) = \frac{n!}{r! \times (n-r)!} \tag{5.2}$$

where n is the numbers of room in the problem and r is the numbers of SOS variables used in the problem. In this thesis, the value of r is 2. Since, the binary variable of p_{ij} and q_{ij} are used in each problem.

Generations and stopping criterion: for a generic GA, there are three main ways to stop the loop as 1) allele convergence, 2) a predefined number of generations and 3) when the optimum is reached. In our thesis, we stop the GA operation using a predefined number of generations. The stopping criterion using the numbers of generations is tested on the 50, 100 and 500 generations.

- **Statistical approaches**

The appropriated GA parameters are summarized based on a statistics which is proposed by Fonseca and Fleming (1996). In the real world problem, if GA is run for several times, the search space can be divided into three categories.

1) The first part of the search space that is always dominated by all runs.
2) The second part of the search space that is dominated by some runs.
3) The third part of the search space that is never dominated by any runs.

Based on the Pareto-optimal fronts, the first part of the search space is dominated by all runs. This presents the covering 80% of all runs. The second part of the search space reaches some Pareto-fronts but not all of them. This presents the covering 50% of all runs. And the third part of the search space is never dominated by any runs. This presents less than the covering 20% of all runs. By which, the completed set of the experiments will be used for the initial comparative GA study.

- **The GA parameter experiments**

To test the effectiveness of the GA, a series of the experiments of a 5 rooms of pattern A, B, C and D are performed based on a PC computer using Pentium Core 2 Duo and 2.0 GB of memory. Each experiment, a fixed-length binary string is used as a 2D binary string with length l = 32. Each string column is represented by a 5 bit that can represent all possible cases of variable p_{ij} and q_{ij}. A simple GA with crossover probability of 0.5, 0.8, 0.9 and 1.0, the mutation probability of 0.0005, 0.001, 0.005, 0.01 and 0.05 are implemented based on the various population sizes of 10, 20, 30 and 50. The stopping criteria for the experiments are 50, 100 and 500 generations. All experiments of patterns A, B, C and D are illustrated as follows.

The experiments of pattern A:

Crossover 0.5, mutation = 0.0005						
			Populations			
			10	20	30	50
Generations	50	Fitness	580	620	1240	1596
		Avg. fitness	-10340.9	-9566.8	-8796.3	-7240.8
		Time (sec)	48	169	203	287
	100	Fitness	980	1118	2868	2112
		Avg. fitness	-3209.6	-3724.4	-5066.8	-4866.3
		Time (sec)	84	238	296	483
	500	Fitness	2458	2922	2879	3064
		Avg. fitness	1238.3	407.6	-332.4	-5132.3
		Time (sec)	325	754	1041	2282

Crossover 0.8, mutation = 0.0005						
			Populations			
			10	20	30	50
Generations	50	Fitness	1418	843	266	2303
		Avg. fitness	-2176.5	-4350.3	-7490.8	-10287.0
		Time (sec)	71	149	200	353
	100	Fitness	2417	964	2829	2536
		Avg. fitness	-2592.4	-3516.6	-5222.3	-4120.0
		Time (sec)	92	229	324	542
	500	Fitness	1294	1055	3367	1323
		Avg. fitness	-901.8	-919.0	-1140.4	-7518.8
		Time (sec)	371	782	1051	2596

Crossover 0.5, mutation = 0.001						
			Populations			
			10	20	30	50
Generations	50	Fitness	982	1268	1428	2977
		Avg. fitness	-9595.7	-8554.5	-7093.6	-6778.5
		Time (sec)	87	203	253	350
	100	Fitness	717	-1004	3720	3360
		Avg. fitness	-3009.4	-2393.5	-2309.5	-9723.3
		Time (sec)	100	254	306	732
	500	Fitness	3582	1571	3634	3624
		Avg. fitness	65.5	-2373.5	-5339.9	-4533.4
		Time (sec)	382	794	1650	2592

Crossover 0.8, mutation = 0.001						
			Populations			
			10	20	30	50
Generations	50	Fitness	1382	1228	1705	1511
		Avg. fitness	-1772.9	-3738.6	-6535.1	-7890.2
		Time (sec)	71	209	309	532
	100	Fitness	1381	1218	2362	2344
		Avg. fitness	-2517.8	-4159.2	-6724.9	-13469.0
		Time (sec)	164	386	538	1166
	500	Fitness	3607	3590	3488	1530
		Avg. fitness	1006.7	-911.8	-3032.8	-6749.9
		Time (sec)	627	1586	2256	4453

Crossover 0.5, mutation = 0.005						
			Populations			
			10	20	30	50
Generations	50	Fitness	2836	1129	907	1604
		Avg. fitness	-2604.1	-4600.1	-7618.4	-10170.0
		Time (sec)	195	556	695	1268
	100	Fitness	2964	2845	2645	2675
		Avg. fitness	-4129.4	-3781.9	-4274.9	-7704.4
		Time (sec)	399	786	1179	2317
	500	Fitness	3950	3556	3556	3633
		Avg. fitness	-602.4	-2481.4	-4911.8	-5899.9
		Time (sec)	1645	3458	6040	10699

Crossover 0.8, mutation = 0.005						
			Populations			
			10	20	30	50
Generations	50	Fitness	1512	1655	1888	1423
		Avg. fitness	-4751.9	-7787.5	-9941.4	-7573.7
		Time (sec)	199	475	918	1133
	100	Fitness	2212	2430	2445	2770
		Avg. fitness	-2923.0	-4685.9	-5918.3	-6245.1
		Time (sec)	319	578	1127	1920
	500	Fitness	3264	3113	3429	3669
		Avg. fitness	106.2	-3102.1	-5959.2	-6088.3
		Time (sec)	1207	2569	5216	11086

Crossover 0.5, mutation = 0.01						
			Populations			
			10	20	30	50
Generations	50	Fitness	3210	2254	2588	3292
		Avg. fitness	-2913.9	-4620.7	-7354.3	-8744.0
		Time (sec)	321	1216	1625	2096
	100	Fitness	2367	2855	3167	3295
		Avg. fitness	-2759.4	-4930.8	-8198.0	-6962.8
		Time (sec)	653	1409	2478	3721
	500	Fitness	3915	3855	3625	3210
		Avg. fitness	-1842.3	-3190.2	-5247.0	-6575.5
		Time (sec)	3122	6304	10887	19005

Crossover 0.8, mutation = 0.01						
			Populations			
			10	20	30	50
Generations	50	Fitness	3204	2430	3259	2251
		Avg. fitness	-2947.5	-5115.0	-6703.4	-7939.6
		Time (sec)	305	593	1091	1917
	100	Fitness	2321	2798	3012	3498
		Avg. fitness	-3396.2	-5590.5	-7151.9	-7207.5
		Time (sec)	582	1050	2044	3377
	500	Fitness	2221	2446	3822	3478
		Avg. fitness	-2424.4	-4297.8	-5684.6	-7305.4
		Time (sec)	2491	4475	8696	15734

Crossover 0.5, mutation = 0.05						
			Populations			
			10	20	30	50
Generations	50	Fitness	2767	2675	2934	2751
		Avg. fitness	-5126.9	-6001.0	-8208.6	-9081.8
		Time (sec)	483	1522	1852	2555
	100	Fitness	2710	2808	3098	3071
		Avg. fitness	-5048.7	-5755.1	-7740.5	-8682.4
		Time (sec)	763	1745	2849	5193
	500	Fitness	3291	3345	3442	3621
		Avg. fitness	-5087.1	-5820.6	-7847.5	-8721.0
		Time (sec)	3776	9207	15240	23660

Crossover 0.8, mutation = 0.05						
			Populations			
			10	20	30	50
Generations	50	Fitness	2107	2665	2850	3180
		Avg. fitness	-5119.1	-7206.4	-8477.9	-9034.0
		Time (sec)	358.4	666	1208.2	1978.2
	100	Fitness	2957	3105	2854	3001
		Avg. fitness	-5663.2	-7461.9	-8415.8	-8880.7
		Time (sec)	676	1203	2331	3667
	500	Fitness	3219	3455	3709	3512
		Avg. fitness	-5124.7	-6891.3	-7877.7	-8902.8
		Time (sec)	3101	5181	9851	16982

Crossover 0.9, mutation = 0.0005						
			Populations			
			10	20	30	50
Generations	50	Fitness	1117	1194	668	1721
		Avg. fitness	-2695.9	-4007.7	-6210.0	-10021.3
		Time (sec)	71	214	318	571
	100	Fitness	1476	826	2723	765
		Avg. fitness	-2110.9	-2350.5	-3112.5	-8548.3
		Time (sec)	146	310	419	1060
	500	Fitness	1169	1442	3258	3350
		Avg. fitness	-1355.5	-1007.9	-884.3	-3696.5
		Time (sec)	608	1258	1679	3422

Crossover 1.0, mutation = 0.0005						
			Populations			
			10	20	30	50
Generations	50	Fitness	1929	1055	2814	3278
		Avg. fitness	-5885.5	-6389.9	-8314.2	-9000.7
		Time (sec)	175	381.78	673	1091
	100	Fitness	2321	2380	3415	3207
		Avg. fitness	-3361.8	-2334.9	-1826.9	-8873.0
		Time (sec)	321	514	820	1968
	500	Fitness	2129	2576	3597	2047
		Avg. fitness	624.2	315.4	76.8	-2667.9
		Time (sec)	960	1820.34	3085	6514

Crossover 0.9, mutation = 0.001						
			Populations			
			10	20	30	50
Generations	50	Fitness	964	889	1801	2408
		Avg. fitness	-7945.9	-6585.3	-6688.0	-8629.4
		Time (sec)	128	303	423	704
	100	Fitness	1435	1422	1329	1842
		Avg. fitness	-2149.6	-4186.4	-7153.5	-5331.2
		Time (sec)	219	412	773	1211
	500	Fitness	2795	2450	2933	3212
		Avg. fitness	339.0	-1310.5	-3251.1	-3434.2
		Time (sec)	744	2179	3217	5271

Crossover 1.0, mutation = 0.001						
			Populations			
			10	20	30	50
Generations	50	Fitness	1324	2022	2120	2085
		Avg. fitness	-3726.9	-5566.2	-8642.5	-8305.6
		Time (sec)	188	409	720	1203
	100	Fitness	1767	2450	3028	2567
		Avg. fitness	-3128.7	-3349.0	-4313.6	-7300.0
		Time (sec)	356	695	1189	2414
	500	Fitness	3182	2568	2526	3494
		Avg. fitness	1396.6	-134.5	-1695.4	-2388.4
		Time (sec)	1400	2966	5191	8906

Crossover 0.9, mutation = 0.005						
			Populations			
			10	20	30	50
Generations	50	Fitness	3095	2426	1437	2682
		Avg. fitness	-1339.8	-4520.4	-8176.8	-10158.6
		Time (sec)	186	403	822	1402
	100	Fitness	3166	3265	2815	3080
		Avg. fitness	-1281.5	-4118.0	-7388.1	-8569.7
		Time (sec)	379	789	1523	2732
	500	Fitness	3892	3534	3477	3625
		Avg. fitness	577.0	-1740.5	-4241.1	-6973.4
		Time (sec)	1691	3452	6628	12150

Crossover 1.0, mutation = 0.005						
			Populations			
			10	20	30	50
Generations	50	Fitness	2366	2556	2275	2391
		Avg. fitness	-1587.0	-5837.6	-8477.8	-8542.0
		Time (sec)	249	502	960	1628
	100	Fitness	2626	2811	2516	3049
		Avg. fitness	-1739.2	-3947.8	-6317.4	-7883.7
		Time (sec)	489	922	1732	3134
	500	Fitness	3130	2855	3492	3522
		Avg. fitness	-1195.5	-2923.0	-4769.8	-6661.7
		Time (sec)	2320	4393	8266	14577

Crossover 0.9, mutation = 0.01						
			Populations			
			10	20	30	50
Generations	50	Fitness	2249	2560	2302	2591
		Avg. fitness	-2536.9	-4916.0	-7812.5	-8790.4
		Time (sec)	305	587	1163	2037
	100	Fitness	3585	3245	2708	3135
		Avg. fitness	-2053.0	-4697.2	-7835.9	-8136.7
		Time (sec)	550	1117	2143	3592
	500	Fitness	3319	3860	4176	3734
		Avg. fitness	-1031.5	-3303.8	-5923.8	-7664.5
		Time (sec)	2240	4574	8782	14963

Crossover 1.0, mutation = 0.01						
			Populations			
			10	20	30	50
Generations	50	Fitness	2399	2800	2982	2592
		Avg. fitness	-4088.9	-6953.3	-7899.5	-8748.1
		Time (sec)	333	622	1166	1973
	100	Fitness	2684	2540	3491	2815
		Avg. fitness	-2364.3	-4571.1	-6964.6	-8957.0
		Time (sec)	553	1080	2849	5371
	500	Fitness	2957	3250	3146	3583
		Avg. fitness	-1582.6	-4124.4	-6834.5	-7461.6
		Time (sec)	2389	4852	9302	16938

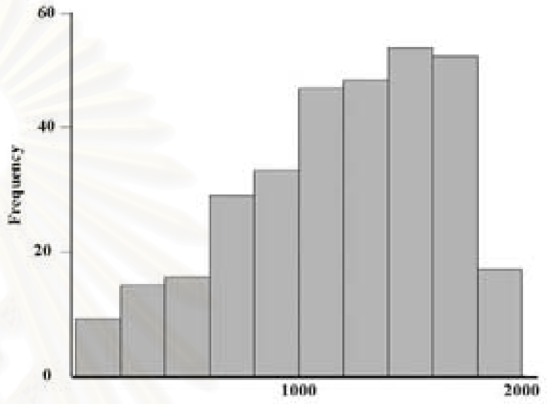
Crossover 0.9, mutation = 0.05						
			Populations			
			10	20	30	50
Generations	50	Fitness	2304	2811	2913	3044
		Avg. fitness	-6871.2	-7256.0	-8404.5	-8549.0
		Time (sec)	396	683	1212.4	1970
	100	Fitness	3050	3240	2936	2418
		Avg. fitness	-5247.3	-6510.9	-8459.9	-9022.3
		Time (sec)	679	1196	2204	3745
	500	Fitness	3511	3460	3338	3085
		Avg. fitness	-5423.0	-6533.8	-8332.4	-9067.9
		Time (sec)	3082	5433	11003	18927

Crossover 1.0, mutation = 0.05						
			Populations			
			10	20	30	50
Generations	50	Fitness	2496	2324	2465	2746
		Avg. fitness	-7346.2	-9704.5	-9385.7	-8924.4
		Time (sec)	410	686	1243	2003
	100	Fitness	2987	3105	3273	2859
		Avg. fitness	-6276.0	-7170.5	-8357.7	-8936.4
		Time (sec)	693	1207	8215	6705
	500	Fitness	3222	3565	2863	3146
		Avg. fitness	-5166.2	-6639.9	-8384.6	-9123.6
		Time (sec)	3536	6906	13990	21151

From the experiments, the 240 instances of pattern A are tested based on the various characteristics of GA parameters. The outputs of all runs are normalized using the respective minimal and maximal values of the fitness in the Pareto-fronts where more than a half of experiments present the high fitness value. By which, the data are not normally distributed. The histogram has been used to present the frequency from the various characteristics of GA parameters that can be illustrated below.

Pattern A: Statistical data of the fitness value	
Mean	1184.67
Mean	1184.67
Std Dev	462.85
Std Error Mean	29.87
Upper 95% Mean	1243.52
Upper 95% Mean	1243.52
Numbers	240

(a)



(b)

Figure 5.2: (a) the statistical data of pattern A and
(b) the histogram of the fitness values tested by the various GA parameters.

As a result, the various probabilities of crossovers and mutations found on each run are not significantly influent the fitness value. On the other hand the high populations and high generations significantly influent the high fitness value. Using the Pareto-optimal fronts, the dominated GA parameters covered 80% instances of populations and generations can be illustrated with the circle using the following table.

		Populations			
		10	20	30	50
Generations	50				
	100			●	●
	500	●	●	●	●

The experiments of pattern B:

Crossover 0.5, mutation = 0.0005						
			Populations			
			10	20	30	50
Generations	50	Fitness	125	687	444	832
		Avg. fitness	-1478.0	-4513.7	-3297.1	-5953.7
		Time (sec)	23	82	130	213
	100	Fitness	654	409	637	914
		Avg. fitness	-145.8	-3660.6	-3231.8	-2863.8
		Time (sec)	52	152	220	392
	500	Fitness	1442	1380	1779	1647
		Avg. fitness	682.9	62.0	-790.9	-2822.3
		Time (sec)	268	573	790	1751

Crossover 0.8, mutation = 0.0005						
			Populations			
			10	20	30	50
Generations	50	Fitness	384	843	337	90
		Avg. fitness	-2519.9	-3847.4	-3150.7	-5858.4
		Time (sec)	55	94	156	275
	100	Fitness	826	964	387	304
		Avg. fitness	-151.6	-1071.4	-3009.5	-2749.4
		Time (sec)	74	148	261	414
	500	Fitness	1545	1055	1451	1206
		Avg. fitness	60.7	-1208.2	-391.1	-1332.6
		Time (sec)	396	919	1183	2089

Crossover 0.5, mutation = 0.001						
			Populations			
			10	20	30	50
Generations	50	Fitness	309	504	1120	1169
		Avg. fitness	-2736.2	-3842.8	-3141.7	-5623.2
		Time (sec)	52	116	169	302
	100	Fitness	467	776	1497	645
		Avg. fitness	-775.2	-2400.6	-3039.1	-3641.7
		Time (sec)	97	204	324	556
	500	Fitness	1242	1571	974	1284
		Avg. fitness	-458.4	-110.7	-2559.6	-4009.0
		Time (sec)	450	857	1508	2760

Crossover 0.8, mutation = 0.001						
			Populations			
			10	20	30	50
Generations	50	Fitness	49	308	766	1002
		Avg. fitness	-2523.8	-2099.7	-4674.3	-3043.4
		Time (sec)	65	120	230.4	340
	100	Fitness	180	228	857	888
		Avg. fitness	-2834.9	-3794.2	-2174.5	-4414.5
		Time (sec)	138	245	327	684
	500	Fitness	1640	1218	1560	840
		Avg. fitness	613.2	-2043.3	-948.8	-2313.7
		Time (sec)	413.25	1018	1425	2697

Crossover 0.5, mutation = 0.005						
			Populations			
			10	20	30	50
Generations	50	Fitness	119	372	868	1149
		Avg. fitness	-3305.5	-3761.7	-2932.8	-3664.7
		Time (sec)	145	348	467	798
	100	Fitness	1259	1172	1418	1372
		Avg. fitness	-1264.0	-3123.8	-2310.4	-3090.5
		Time (sec)	279	599	937	1529
	500	Fitness	1761	1404	1667	1805
		Avg. fitness	-749.4	-2143.0	-2485.6	-2981.4
		Time (sec)	1348	2996	4367	7874

Crossover 0.8, mutation = 0.005						
			Populations			
			10	20	30	50
Generations	50	Fitness	117	496	1091	929
		Avg. fitness	-3095.4	-4551.6	-4860.2	-4106.1
		Time (sec)	160	358	560	856
	100	Fitness	1058	1172	1138	1374
		Avg. fitness	-2242.1	-3153.2	-3738.8	-4591.0
		Time (sec)	295	605	993	1663
	500	Fitness	1061	1457	1454	1779
		Avg. fitness	-1080.0	-2276.8	-2712.7	-3547.7
		Time (sec)	1231	2746	4187.6	7848

Crossover 0.5, mutation = 0.01						
			Populations			
			10	20	30	50
Generations	50	Fitness	555	1440	1356	1595
		Avg. fitness	-3087.6	-2472.1	-3688.1	1595.0
		Time (sec)	263	505	820	1430
	100	Fitness	1730	1287	1656	1375
		Avg. fitness	-612.3	-2953.0	-3099.0	-3805.2
		Time (sec)	421	1006	1576	2932
	500	Fitness	1831	1842	1990	1465
		Avg. fitness	-859.2	-2037.2	-2418.5	-3400.2
		Time (sec)	2271	4920	7989	14244

Crossover 0.8, mutation = 0.01						
			Populations			
			10	20	30	50
Generations	50	Fitness	705	1585	1111	1105
		Avg. fitness	-1222.5	-2078.6	-3525.2	-4351.8
		Time (sec)	238	514	859.2	1475
	100	Fitness	1291	1617	1846	1477
		Avg. fitness	-1486.4	-2523.5	-2966.0	-3753.0
		Time (sec)	450	986	1522.5	3650
	500	Fitness	1834	1763	1388	1335
		Avg. fitness	-902.5	-2196.5	-2966.1	-3646.3
		Time (sec)	2040	4499	7238.4	12957

Crossover 0.5, mutation = 0.05						
			Populations			
			10	20	30	50
Generations	50	Fitness	633	798	1016	878
		Avg. fitness	-3357.8	-3876.7	-4101.4	-4624.2
		Time (sec)	279	586	612	1498
	100	Fitness	1277	1505	1612	1466
		Avg. fitness	-3083.9	-3921.2	-4210.0	-4494.3
		Time (sec)	579	1163	1759	2959
	500	Fitness	1790	1561	1682	1505
		Avg. fitness	-2511.3	-3418.5	-4058.2	-4556.6
		Time (sec)	2599	5635	9085	12069

Crossover 0.8, mutation = 0.05						
			Populations			
			10	20	30	50
Generations	50	Fitness	1010	1098	1111	1069
		Avg. fitness	-3145.6	-3723.2	-4366.6	-4392.8
		Time (sec)	284	590	913	1491
	100	Fitness	1203	1245	1218	1335
		Avg. fitness	-2895.7	-3801.8	-4364.6	-4600.9
		Time (sec)	532	1100	1695	2841
	500	Fitness	1640	1493	1633	1690
		Avg. fitness	-2363.8	-3542.3	-3895.7	-4460.5
		Time (sec)	2752	5049	7666	1327

Crossover 0.9, mutation = 0.0005						
			Populations			
			10	20	30	50
Generations	50	Fitness	752	529	552	1000
		Avg. fitness	-2917.7	-5388.7	-4517.9	-4046.4
		Time (sec)	76	165	251	414
	100	Fitness	311	1118	927	948
		Avg. fitness	-2146.7	-4190.5	-1957.1	-4372.1
		Time (sec)	118	291	372	748
	500	Fitness	1468	1536	1625	1536
		Avg. fitness	751.2	-450.5	-1314.1	-1314.7
		Time (sec)	501	1168	1744	3012.

Crossover 1.0, mutation = 0.0005						
			Populations			
			10	20	30	50
Generations	50	Fitness	259	1194	281	1138
		Avg. fitness	-2323.2	-882.2	-4424.0	-4294.8
		Time (sec)	135	274	490	833
	100	Fitness	698	922	1643	1795
		Avg. fitness	-1018.6	-2314.6	-1180.2	-3554.8
		Time (sec)	254	537	873	1522
	500	Fitness	1829	1083	1744	1536
		Avg. fitness	785.2	-366.6	755.9	-1369.5
		Time (sec)	1128	2513	3331	6550

Crossover 0.9, mutation = 0.001						
			Populations			
			10	20	30	50
Generations	50	Fitness	534	678	649	866
		Avg. fitness	-1979.7	-3156.8	-3969.1	-4834.7
		Time (sec)	85	170	291	499
	100	Fitness	-88	1590	991	1153
		Avg. fitness	-2016.2	-2283.2	-2825.5	-3938.0
		Time (sec)	157	329	516	921
	500	Fitness	1447	1763	1970	1883
		Avg. fitness	-55.0	-771.3	-907.4	-3460.8
		Time (sec)	736	1447	2254	2783

Crossover 1.0, mutation = 0.001						
			Populations			
			10	20	30	50
Generations	50	Fitness	701	1321	1302	865
		Avg. fitness	-967.1	-1988.8	-3464.9	-3978.0
		Time (sec)	125	312	514	818
	100	Fitness	473	579	1415	904
		Avg. fitness	-1503.6	-3320.4	-1246.7	-3926.0
		Time (sec)	262	623	794	847
	500	Fitness	1500	1729	1664	1587
		Avg. fitness	137.9	167.6	-185.3	-1634.2
		Time (sec)	1086	2468	3983	7279

Crossover 0.9, mutation = 0.005						
			Populations			
			10	20	30	50
Generations	50	Fitness	455	1287	616	1184
		Avg. fitness	-2685.6	-2498.8	-3136.2	-4964.8
		Time (sec)	175	361	568	1126
	100	Fitness	1451	1731	1059	1294
		Avg. fitness	-1421.5	-2570.8	-2878.3	-4095.1
		Time (sec)	333	686	1019	1869
	500	Fitness	1801	1287	1356	1754
		Avg. fitness	-198.1	-2498.8	-2205.7	-3640.7
		Time (sec)	1663	3290	5173	9347

Crossover 1.0, mutation = 0.005						
			Populations			
			10	20	30	50
Generations	50	Fitness	628	1447	1170	878
		Avg. fitness	-2384.7	-4498.6	-2902.0	-4394.9
		Time (sec)	217	467	608	1166
	100	Fitness	999	1785	1257	1175
		Avg. fitness	-1419.2	-1811.6	-3396.5	-3911.2
		Time (sec)	386	818	1361	2209
	500	Fitness	1897	1785	1480	1578
		Avg. fitness	-215.3	-1811.6	-2501.1	-2993.8
		Time (sec)	1898	4370	6984	12137

Crossover 0.9, mutation = 0.01						
			Populations			
			10	20	30	50
Generations	50	Fitness	685	1009	1219	1421
		Avg. fitness	-2549.4	-3702.5	-3984.2	-4128.1
		Time (sec)	251	591	897	1517
	100	Fitness	1217	1089	1106	1369
		Avg. fitness	-2172.9	-2778.8	-3157.1	-3243.7
		Time (sec)	462	824	1693	2805
	500	Fitness	1787	1887	1732	1525
		Avg. fitness	-651.9	-2216.0	-2899.8	-3926.6
		Time (sec)	2413	5265	8282	14289

Crossover 1.0, mutation = 0.01						
			Populations			
			10	20	30	50
Generations	50	Fitness	1272	1459	949	1349
		Avg. fitness	-1850.9	-3129.0	-4530.7	-4304.7
		Time (sec)	246	547	932.8	1512
	100	Fitness	1496	1464	1181	1565
		Avg. fitness	-1557.3	-2744.3	-3202.9	-4306.0
		Time (sec)	501	1086	1701	2966
	500	Fitness	1755	1850	1706	1794
		Avg. fitness	-955.4	-2763.2	-3547.8	-4034.5
		Time (sec)	2656	6211	9828	1691

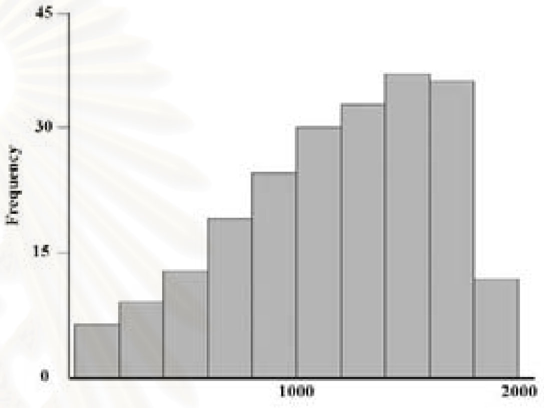
Crossover 0.9, mutation = 0.05						
			Populations			
			10	20	30	50
Generations	50	Fitness	788	668	711	870
		Avg. fitness	-3420.7	-3965.2	-4342.3	-4652.8
		Time (sec)	291.2	597	910.4	1540.8
	100	Fitness	1189	704	1041	1303
		Avg. fitness	-3040.7	-4111.1	-4063.3	-4587.1
		Time (sec)	557.7	1256	1752.3	3205.95
	500	Fitness	1669	1257	1336	1658
		Avg. fitness	-2366.0	-3715.2	-3971.4	-4448.6
		Time (sec)	2988	6665	10639.8	17712

Crossover 1.0, mutation = 0.05						
			Populations			
			10	20	30	50
Generations	50	Fitness	1347	780	944	775
		Avg. fitness	-3019.5	-3782.9	-4580.6	-4452.4
		Time (sec)	270	542	947	1553
	100	Fitness	1718	1208	1026	902
		Avg. fitness	-2459.1	-3733.3	-4215.7	-4495.5
		Time (sec)	625	1375	2047	3576
	500	Fitness	1573	1624	1652	1686
		Avg. fitness	-2664.9	-3592.5	-4222.7	-4630.5
		Time (sec)	3252	6760	10681	18422

From the experiments of pattern B, the 240 instances are tested based on the various characteristics of GA parameters. Also, the outputs of all runs are normalized using the respective minimal and maximal values of the fitness in the Pareto-fronts where more than a half of experiments present the high fitness value. By which, the data are not normally distributed. The histogram has been used to present the fitness frequency from the various characteristics of GA parameters that can be illustrated below.

Pattern B: Statistical data of the fitness value	
Mean	1184.41
Medium	1243.50
Std Dev	462.35
Std Error Mean	29.84
Upper 95% Mean	1243.20
Upper 95% Mean	1243.62
Numbers	240

(a)



(b)

Figure 5.3: (a) the statistical data of pattern B and
(b) the histogram of the fitness values tested by the various GA parameters.

Similar to the pattern A, the various probabilities of crossovers and mutations of pattern B found on each run are not significantly influent the fitness value. Whereas the high populations and high generations influent the high fitness value significantly. Using the Pareto-optimal fronts, the dominated GA parameters covered 80% instances of populations and generations can be illustrated with the circle using the following table.

		Populations			
		10	20	30	50
Generations	50				
	100		●	●	●
	500	●	●	●	●

The experiments of pattern C:

Crossover 0.5, mutation = 0.0005						
			Populations			
			10	20	30	50
Generations	50	Fitness	124	754	606	912
		Avg. fitness	-500.9	-1383.8	-1877.8	-1857.5
		Time (sec)	30	79	115	202
	100	Fitness	2471	2476	2036	2683
		Avg. fitness	-1744.9	-256.0	-1456.5	-1562.3
		Time (sec)	83	127	207	368
	500	Fitness	2607	3594	3427	3622
		Avg. fitness	1713.3	2695.3	2813.4	-237.4
		Time (sec)	272	489	788	1616

Crossover 0.8, mutation = 0.0005						
			Populations			
			10	20	30	50
Generations	50	Fitness	688	259	714	871
		Avg. fitness	1271.4	-1469.2	-1060.3	-928.0
		Time (sec)	46	380	157	386
	100	Fitness	1623	2575	2744	3079
		Avg. fitness	317.6	-760.5	-2833.4	-1098.6
		Time (sec)	80	182	262	450
	500	Fitness	1435	3663	3363	3208
		Avg. fitness	407.7	2455.4	695.5	-188.3
		Time (sec)	80	182	262	450

Crossover 0.5, mutation = 0.001						
			Populations			
			10	20	30	50
Generations	50	Fitness	556	307	116	1104
		Avg. fitness	-736.3	-842.9	-1141.8	-1623.5
		Time (sec)	55	108	265	531
	100	Fitness	1840	2396	2760	1558
		Avg. fitness	-775.2	-400.6	-271.4	-1631.2
		Time (sec)	92	223	279	579
	500	Fitness	3627	3618	3466	3676
		Avg. fitness	2189.1	2542.9	2137.6	1261.3
		Time (sec)	395	782	1153	2234

Crossover 0.8, mutation = 0.001						
			Populations			
			10	20	30	50
Generations	50	Fitness	926	1621	1211	1184
		Avg. fitness	1784.8	744.7	-456.4	-341.8
		Time (sec)	68	125	179	295
	100	Fitness	2961	2834	2872	3146
		Avg. fitness	-319.9	218.8	-182.9	-975.0
		Time (sec)	100	249	362	647
	500	Fitness	3743	3523	3842	3743
		Avg. fitness	996.1	1278.1	3010.2	603.1
		Time (sec)	100	249	362	647

Crossover 0.5, mutation = 0.005						
			Populations			
			10	20	30	50
Generations	50	Fitness	533	434	1458	1258
		Avg. fitness	-709.6	-745.6	-611.6	-1562.3
		Time (sec)	152	312	436	791
	100	Fitness	2951	2433	2974	3404
		Avg. fitness	1254.1	-655.4	-544.2	-1078.8
		Time (sec)	209	235	255	1554
	500	Fitness	3324	3345	3711	3804
		Avg. fitness	766.7	1024.6	-91.5	-98.6
		Time (sec)	1417	2675	4380	6410

Crossover 0.8, mutation = 0.005						
			Populations			
			10	20	30	50
Generations	50	Fitness	265	675	1401	1554
		Avg. fitness	-1282.2	-1124.5	-1481.0	-676.2
		Time (sec)	163	347	490	365
	100	Fitness	2678	2975	3162	3236
		Avg. fitness	2129.3	-654.2	-593.9	-1741.7
		Time (sec)	243	688	897	1664
	500	Fitness	3524	3936	3935	3818
		Avg. fitness	1752.7	1244.3	-161.7	-195.7
		Time (sec)	243	688	897	1664

Crossover 0.5, mutation = 0.01						
			Populations			
			10	20	30	50
Generations	50	Fitness	1221	1168	983	1009
		Avg. fitness	-308.6	-472.1	-688.1	-1595.3
		Time (sec)	216	443	670	892
	100	Fitness	2806	2831	3643	3025
		Avg. fitness	-612.3	-953.0	-1099.0	-3805.2
		Time (sec)	344	659	1138	2232
	500	Fitness	3028	3152	3378	3223
		Avg. fitness	859.2	-2037.2	-1218.5	-3400.2
		Time (sec)	1774	3954	5990	7743

Crossover 0.8, mutation = 0.01						
			Populations			
			10	20	30	50
Generations	50	Fitness	1580	328	1259	1321
		Avg. fitness	-222.5	-278.6	-1642.6	-451.9
		Time (sec)	221	445	826	887
	100	Fitness	2711	3395	3684	3102
		Avg. fitness	-1486.4	-1523.5	-1114.5	-375.3
		Time (sec)	498	1344	1626	2524
	500	Fitness	3451	3700	3215	3504
		Avg. fitness	902.5	-1196.5	-966.2	366.3
		Time (sec)	498	1344	1626	2524

Crossover 0.5, mutation = 0.05						
			Populations			
			10	20	30	50
Generations	50	Fitness	1367	1164	1696	1218
		Avg. fitness	-744.5	-970.3	-1784.6	-1423.6
		Time (sec)	260	528	826	1365
	100	Fitness	3348	3259	3482	2876
		Avg. fitness	53.2	-855.9	-1088.9	-1507.2
		Time (sec)	499	1043	1448	2772
	500	Fitness	3323	3621	3540	3486
		Avg. fitness	-97.0	-646.1	-1052.3	-1500.7
		Time (sec)	2532	5182	7730	8185

Crossover 0.8, mutation = 0.05						
			Populations			
			10	20	30	50
Generations	50	Fitness	1187	2094	2731	2923
		Avg. fitness	-259.7	-1047.0	-1266.1	-1330.4
		Time (sec)	252	503	791	1335
	100	Fitness	2943	3530	3347	3419
		Avg. fitness	-511.1	-871.8	-1220.1	-1473.3
		Time (sec)	643	1030	3637	4786
	500	Fitness	3788	3384	3592	3627
		Avg. fitness	137.2	-868.9	-1087.8	-1457.5
		Time (sec)	643	1030	3637	4786

Crossover 0.9, mutation = 0.0005						
			Populations			
			10	20	30	50
Generations	50	Fitness	792	215	432	1056
		Avg. fitness	-511.7	-799.3	-1392.5	-1239.6
		Time (sec)	57	142	357	474
	100	Fitness	1913	3786	1941	2865
		Avg. fitness	-747.8	2129.2	-1507.4	-372.1
		Time (sec)	153	198	442	1485
	500	Fitness	3445	3092	3659	3426
		Avg. fitness	2925.0	1632.7	2456.9	-477.5
		Time (sec)	471	1049	1430	3197

Crossover 1.0, mutation = 0.0005						
			Populations			
			10	20	30	50
Generations	50	Fitness	670	794	695	1286
		Avg. fitness	254.0	1432.0	-1005.1	-1880.1
		Time (sec)	116	199	452	702
	100	Fitness	2690	2355	2726	2868
		Avg. fitness	478.0	-1303.0	-898.2	-1477.8
		Time (sec)	230	543	869	1485
	500	Fitness	3296	3683	3384	3716
		Avg. fitness	2882.1	2468.6	2821.9	2711.0
		Time (sec)	1011	2110	3000	5128

Crossover 0.9, mutation = 0.001						
			Populations			
			10	20	30	50
Generations	50	Fitness	281	627	1557	1078
		Avg. fitness	1979.4	-771.3	969.2	-834.8
		Time (sec)	83	244	749	1046
	100	Fitness	2112	2816	2378	2767
		Avg. fitness	1016.2	-315.7	-1825.5	-393.8
		Time (sec)	224	475	655	1543
	500	Fitness	3472	3231	3728	3519
		Avg. fitness	1549.7	1288.3	907.4	-346.1
		Time (sec)	845	1572	3154	5543

Crossover 1.0, mutation = 0.001						
			Populations			
			10	20	30	50
Generations	50	Fitness	361	677	1543	2057
		Avg. fitness	-1170.2	-2187.7	-3811.4	-4375.8
		Time (sec)	157	286	847	1169
	100	Fitness	2851	2042	2547	1627
		Avg. fitness	-1653.9	-3652.4	-1371.3	-4318.6
		Time (sec)	288	657	1105	1642
	500	Fitness	2700	3112	3795	3856
		Avg. fitness	151.7	184.3	-203.9	-1797.7
		Time (sec)	1239	2936	4626	8862

Crossover 0.9, mutation = 0.005						
			Populations			
			10	20	30	50
Generations	50	Fitness	848	770	901	1112
		Avg. fitness	1148.2	-736.3	-1121.9	-1567.6
		Time (sec)	137	337	883	1196
	100	Fitness	2086	3403	3471	2947
		Avg. fitness	1600.6	-279.2	261.5	-1381.5
		Time (sec)	273	573	960	1768
	500	Fitness	3845	3618	3618	3319
		Avg. fitness	2302.9	-122.1	-122.1	-1145.6
		Time (sec)	1354	3533	4926	7919

Crossover 1.0, mutation = 0.005						
			Populations			
			10	20	30	50
Generations	50	Fitness	1792	1154	2264	2161
		Avg. fitness	213.3	-630.3	-77.2	-1637.8
		Time (sec)	297	391	993	1225
	100	Fitness	2650	2834	3694	2947
		Avg. fitness	1013.7	632.6	-793.1	-1381.5
		Time (sec)	334	712	1174	1768
	500	Fitness	3859	3670	3555	3549
		Avg. fitness	2618.1	1159.8	213.0	-557.0
		Time (sec)	1507	3649	5572	10050

Crossover 0.9, mutation = 0.01						
			Populations			
			10	20	30	50
Generations	50	Fitness	1439	1119	1660	1284
		Avg. fitness	-542.9	-702.5	849.2	-842.6
		Time (sec)	233	427	911	1221
	100	Fitness	2556	3287	3223	3150
		Avg. fitness	273.0	-779.9	-735.6	-937.3
		Time (sec)	412	879	1328	2435
	500	Fitness	3452	3462	3637	3202
		Avg. fitness	561.2	1817.5	-988.8	-1260.8
		Time (sec)	1766	4326	6533	9023

Crossover 1.0, mutation = 0.01						
			Populations			
			10	20	30	50
Generations	50	Fitness	2609	2990	2945	2765
		Avg. fitness	890.5	1129.0	543.8	-347.6
		Time (sec)	332	455	1034	1303
	100	Fitness	3067	3000	3421	3208
		Avg. fitness	-553.7	-7554.4	-323.0	-1537.3
		Time (sec)	442	893	1336	2435
	500	Fitness	3597	3393	3497	3674
		Avg. fitness	-1355.4	2123.3	-754.4	-1191.8
		Time (sec)	1922	3649	4134	11525

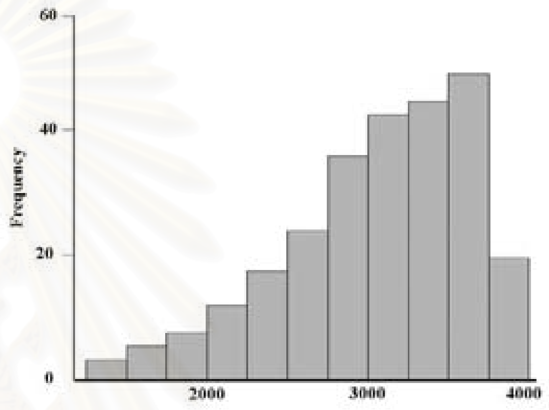
Crossover 0.9, mutation = 0.05						
			Populations			
			10	20	30	50
Generations	50	Fitness	1309	1043	2985	3247
		Avg. fitness	-673.4	-1271.6	-1324.9	-1560.1
		Time (sec)	250	492	975	1275
	100	Fitness	2766	3000	3630	3631
		Avg. fitness	-263.3	-1079.3	-1344.3	-1394.9
		Time (sec)	488	996	1527	2761
	500	Fitness	3751	3456	3363	3681
		Avg. fitness	-57.6	-738.5	-1322.2	-1485.8
		Time (sec)	2395	4914	7575	11724

Crossover 1.0, mutation = 0.05						
			Populations			
			10	20	30	50
Generations	50	Fitness	1883	2683	3129	2834
		Avg. fitness	-1005.6	-1271.6	-1465.6	-1685.6
		Time (sec)	458	506	1193	1329
	100	Fitness	2074	3430	2982	3004
		Avg. fitness	-430.3	-1095.4	-1444.8	-1554.9
		Time (sec)	503	1033	1574	2793
	500	Fitness	3533	3576	3625	3412
		Avg. fitness	-228.4	-1027.6	-1454.1	-1589.3
		Time (sec)	2491	5349	7886	13898

From the experiments, the 240 instances of pattern C are tested based on the various characteristics of GA parameters. Similarly, the outputs of all runs are normalized using the respective minimal and maximal values of the fitness in the Pareto-fronts where more than a half of experiments present the high fitness value. By which, the data are not normally distributed. The histogram has been used to present the fitness frequency from the various characteristics of GA parameters that can be illustrated below.

Pattern C: Statistical data of the fitness value	
Mean	3055.26
Medium	3157.5
Std Dev	585.99
Std Error Mean	37.82
Upper 95% Mean	3129.78
Upper 95% Mean	2980.75
Numbers	240

(a)



(b)

Figure 5.4: (a) the statistical data of pattern C and
(b) the histogram of the fitness values tested by the various GA parameters.

As a result, the various probabilities of crossovers and mutations of pattern C found on each run are not significantly influent the fitness value. Whereas the high populations and high generations influent the high fitness value significantly. Using the Pareto-optimal fronts, the dominated GA parameters covered 80% instances of populations and generations can be illustrated with the circle using the following table.

		Populations			
		10	20	30	50
Generations	50				
	100		●	●	●
	500	●	●	●	●

The experiments of pattern D:

Crossover 0.5, mutation = 0.0005						
			Populations			
			10	20	30	50
Generations	50	Fitness	224	262	289	295
		Avg. fitness	-1092.7	-3009.4	-4006.9	-4380.8
		Time (sec)	49	64	104	160
	100	Fitness	779	672	59	912
		Avg. fitness	-2826.7	-3712.3	-2341.7	-3905.9
		Time (sec)	84	103	147	287
	500	Fitness	1245	1021	1487	1220
		Avg. fitness	-1358.4	-1056.0	-2233.4	-1910.4
		Time (sec)	187	395	625	1094

Crossover 0.8, mutation = 0.0005						
			Populations			
			10	20	30	50
Generations	50	Fitness	157	346	545	583
		Avg. fitness	-2246.7	-3256.9	-3546.4	-6219.3
		Time (sec)	56	84	106	211
	100	Fitness	893	1532	1793	1026
		Avg. fitness	-1820.3	-3747.6	-4208.0	-5897.9
		Time (sec)	128	147	216	412
	500	Fitness	1273	1644	1988	1623
		Avg. fitness	-484.0	-1593.6	-1501.7	-3933.8
		Time (sec)	269	567	860	2163

Crossover 0.5, mutation = 0.001						
			Populations			
			10	20	30	50
Generations	50	Fitness	458	533	765	544
		Avg. fitness	-876.3	-1255.8	-2369.0	-855.5
		Time (sec)	52	98	185	320
	100	Fitness	1130	256	1111	548
		Avg. fitness	-970.2	-3663.0	-2423.4	-4532.4
		Time (sec)	101	158	223	404
	500	Fitness	1317	1764	2038	1737
		Avg. fitness	-406.5	-1465.8	-3835.2	-3299.3
		Time (sec)	259	633	1100	1751

Crossover 0.8, mutation = 0.001						
			Populations			
			10	20	30	50
Generations	50	Fitness	168	610	723	871
		Avg. fitness	-3219.0	-4847.4	-3951.7	-3159.1
		Time (sec)	56	103	137	284
	100	Fitness	531	1486	2046	1863
		Avg. fitness	-3202.0	-4102.1	-2114.8	-5337.6
		Time (sec)	94	180	262	550
	500	Fitness	1239	1544	2098	2320
		Avg. fitness	1238.2	-836.3	-1431.9	-1665.6
		Time (sec)	315	852	2230	4455

Crossover 0.5, mutation = 0.005						
			Populations			
			10	20	30	50
Generations	50	Fitness	233	442	411	315
		Avg. fitness	-1445.6	-1233.7	-5609.7	-6052.5
		Time (sec)	102	255	392	620
	100	Fitness	893	985	1320	1590
		Avg. fitness	543.3	-1489.4	-2330.9	-5895.1
		Time (sec)	239	676	981	1270
	500	Fitness	1998	2152	2295	2003
		Avg. fitness	-1009.8	-3542.8	-3595.4	-4671.8
		Time (sec)	872	2110	3256	5791

Crossover 0.8, mutation = 0.005						
			Populations			
			10	20	30	50
Generations	50	Fitness	543	445	895	940
		Avg. fitness	544.7	-2557.0	-3800.3	-5359.1
		Time (sec)	133	298	351	706
	100	Fitness	233	887	1349	1850
		Avg. fitness	-1190.6	-23456.0	-3904.0	-4819.1
		Time (sec)	308	723	690	1246
	500	Fitness	887	1355	2420	2235
		Avg. fitness	-1290.4	-1448.6	-2911.8	-2356.9
		Time (sec)	945	2323	3240	6506

Crossover 0.5, mutation = 0.01						
			Populations			
			10	20	30	50
Generations	50	Fitness	553	322	675	445
		Avg. fitness	-3140.2	-3573.6	-2786.2	-3481.4
		Time (sec)	178	386	688	870
	100	Fitness	566	1220	1100	1235
		Avg. fitness	-1200.8	-2967.6	-2194.9	-2936.0
		Time (sec)	360	845	1550	1875
	500	Fitness	1366	1842	2113	2115
		Avg. fitness	-711.9	-2035.8	-2361.3	-2832.3
		Time (sec)	1558	3050	4998	7756

Crossover 0.8, mutation = 0.01						
			Populations			
			10	20	30	50
Generations	50	Fitness	445	567	765	863
		Avg. fitness	-1130.8	-1922.7	-3260.8	-4025.4
		Time (sec)	187	360	488	966
	100	Fitness	566	1050	1912	1290
		Avg. fitness	-1374.9	-2334.2	-2743.5	-3471.5
		Time (sec)	396	955	1046	1855
	500	Fitness	1120	1265	1456	1766
		Avg. fitness	-834.8	-2031.7	-2743.6	-3372.8
		Time (sec)	1661	3373	5339	8530

Crossover 0.5, mutation = 0.05						
			Populations			
			10	20	30	50
Generations	50	Fitness	1256	1513	1508	1158
		Avg. fitness	-5255.3	-5383.5	-5505.7	-6127.2
		Time (sec)	219	430	843	1088
	100	Fitness	1318	1683	1346	1396
		Avg. fitness	-4101.1	-4998.4	-5714.6	-6159.0
		Time (sec)	403	832	1289	2179
	500	Fitness	1827	1845	1979	2058
		Avg. fitness	-3596.0	-4870.2	-5460.9	-5916.8
		Time (sec)	2020	3988	6536	10017

Crossover 0.8, mutation = 0.05						
			Populations			
			10	20	30	50
Generations	50	Fitness	985	1158	1572	1178
		Avg. fitness	-4682.3	-5304.5	-5449.7	-6138.2
		Time (sec)	239	438	656	1118
	100	Fitness	1288	1608	1430	1366
		Avg. fitness	-4307.2	-4829.5	-5623.3	-6130.6
		Time (sec)	423	1265	1301	2173
	500	Fitness	1979	1934	1861	1811
		Avg. fitness	-3463.1	-4954.3	-5440.0	-5853.8
		Time (sec)	1995	4208	6441	9937

Crossover 0.9, mutation = 0.0005						
			Populations			
			10	20	30	50
Generations	50	Fitness	477	1414	1273	969
		Avg. fitness	-3035.4	-3548.7	-5017.4	-4489.4
		Time (sec)	58	112	170	300
	100	Fitness	988	882	1502	977
		Avg. fitness	-1009.8	-3927.8	-1383.0	-5328.2
		Time (sec)	152	216	271	287
	500	Fitness	1844	2037	2086	1837
		Avg. fitness	-1143.7	-73.2	217.9	-2117.4
		Time (sec)	287	395	1180	2239

Crossover 1.0, mutation = 0.0005						
			Populations			
			10	20	30	50
Generations	50	Fitness	557	1154	931	1788
		Avg. fitness	-2310.0	-4476.0	-5065.0	-5652.3
		Time (sec)	93	226	367	619
	100	Fitness	549	928	479	1107
		Avg. fitness	-1517.1	-2974.6	-5065.7	-5704.2
		Time (sec)	199	404	659	1156
	500	Fitness	1244	1881	2226	1784
		Avg. fitness	-179.9	-321.4	-909.5	-2281.6
		Time (sec)	780	1596	2511	4488

Crossover 0.9, mutation = 0.001						
			Populations			
			10	20	30	50
Generations	50	Fitness	445	567	776	998
		Avg. fitness	-233.6	-292.5	-1724.7	-2474.5
		Time (sec)	132	151	275	508
	100	Fitness	1230	2032	1549	1153
		Avg. fitness	-1560.7	-1599.7	-1170.2	-3394.1
		Time (sec)	183	357	395	662
	500	Fitness	1255	2302	2355	1998
		Avg. fitness	-1947.7	-1256.3	-1014.5	-2384.6
		Time (sec)	495	1324	2886	5012

Crossover 1.0, mutation = 0.001						
			Populations			
			10	20	30	50
Generations	50	Fitness	511	1118	728	1417
		Avg. fitness	253.1	-134.3	-134.3	-1260.2
		Time (sec)	123	182	325	723
	100	Fitness	1209	1827	2779	1750
		Avg. fitness	1263.0	-809.9	-1234.1	-1724.3
		Time (sec)	236	429	723	1155
	500	Fitness	1771	2091	2580	2448
		Avg. fitness	1760.6	-307.1	287.6	-1519.6
		Time (sec)	922	1665	3533	5322

Crossover 0.9, mutation = 0.005						
			Populations			
			10	20	30	50
Generations	50	Fitness	658	875	996	1243
		Avg. fitness	-2524.4	-2348.9	-2948.0	-4666.9
		Time (sec)	147	303	489	766
	100	Fitness	1430	1445	1236	1345
		Avg. fitness	-1336.2	-2416.6	-2705.6	-3849.4
		Time (sec)	352	805	705	1446
	500	Fitness	1560	1856	2332	2200
		Avg. fitness	-186.2	-2348.9	-2073.4	-3422.3
		Time (sec)	1259	2653	3362	7988

Crossover 1.0, mutation = 0.005						
			Populations			
			10	20	30	50
Generations	50	Fitness	812	1263	818	686
		Avg. fitness	-3390.7	-4966.0	-4080.1	-3072.1
		Time (sec)	156	320	529	858
	100	Fitness	1796	1359	2166	1511
		Avg. fitness	-1415.2	-4369.4	-3419.8	-5915.1
		Time (sec)	269	680	863	1692
	500	Fitness	1940	1904	1964	2138
		Avg. fitness	-948.5	-2543.1	-4364.3	-2686.5
		Time (sec)	1216	2777	4647	7943

Crossover 0.5, mutation = 0.01						
			Populations			
			10	20	30	50
Generations	50	Fitness	766	1088	1290	970
		Avg. fitness	-3849.8	-3582.1	-4495.7	-7117.0
		Time (sec)	189	398	544	1021
	100	Fitness	860	1144	2239	2198
		Avg. fitness	-2037.6	-3685.3	-4126.0	-5870.4
		Time (sec)	402	935	1123	1873
	500	Fitness	1776	2009	2120	2030
		Avg. fitness	-283.9	-3582.1	-3161.9	-5219.0
		Time (sec)	1728	3557	5789	9402

Crossover 1.0, mutation = 0.01						
			Populations			
			10	20	30	50
Generations	50	Fitness	542	445	763	877
		Avg. fitness	-1910.5	-5898.6	-4616.7	-4985.4
		Time (sec)	211	405	612	989
	100	Fitness	1120	1237	2014	1228
		Avg. fitness	-1280.4	-3433.1	-5891.9	-6326.8
		Time (sec)	420	929	1288	1830
	500	Fitness	2344	2554	2276	2098
		Avg. fitness	-1260.7	2889.5	-2929.4	-3433.6
		Time (sec)	1893	4223	6128	9533

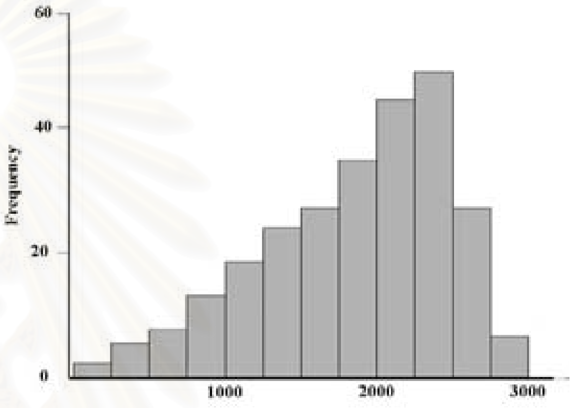
Crossover 0.5, mutation = 0.05						
			Populations			
			10	20	30	50
Generations	50	Fitness	898	1159	1524	1534
		Avg. fitness	-3971.7	-4845.9	-5674.7	-6082.2
		Time (sec)	207	424	656	1101
	100	Fitness	1679	1277	1294	1486
		Avg. fitness	-3994.6	-4840.3	-5620.3	-6016.7
		Time (sec)	413	988	1306	2163
	500	Fitness	1899	1941	1970	1895
		Avg. fitness	-3584.1	-5084.5	-5469.9	-5984.0
		Time (sec)	2006	4218	6411	10067

Crossover 1.0, mutation = 0.05						
			Populations			
			10	20	30	50
Generations	50	Fitness	1177	1270	1695	1482
		Avg. fitness	-4388.9	-5361.4	-5757.3	-6176.3
		Time (sec)	212	441	665	1102
	100	Fitness	1550	1868	1644	1553
		Avg. fitness	-4515.5	-5449.6	-5931.4	-6315.0
		Time (sec)	427	974	1318	2199
	500	Fitness	1958	1845	1760	1805
		Avg. fitness	-3941.5	-5051.5	-5643.3	-6106.2
		Time (sec)	2037	4192	6371	10131

From the experiments of pattern D, the 240 instances are tested based on the various characteristics of GA parameters. Similarly, the outputs of all runs are normalized using the respective minimal and maximal values of the fitness in the Pareto-fronts where more than a half of experiments present the high fitness value. By which, the data are not normally distributed. The histogram has been used to present the fitness frequency from the various characteristics of GA parameters that can be illustrated below.

Pattern D: Statistical data of the fitness value	
Mean	1856.43
Medium	1908.00
Std Dev	629.02
Std Error Mean	40.60
Upper 95% Mean	1936.42
Upper 95% Mean	1776.45
Numbers	240

(a)



(b)

Figure 5.5: (a) the statistical data of pattern D and
(b) the histogram of the fitness values tested by the various GA parameters.

Similar to all previous patterns, the various probabilities of crossovers and mutations of pattern D found on each run are not significantly influent the fitness value. Whereas the high populations and high generations significantly influent the high fitness value. Using the Pareto-optimal fronts, the dominated GA parameters covered 80% instances of populations and generations can be illustrated with the circle using the following table.

		Populations			
		10	20	30	50
Generations	50				
	100			●	●
	500	●	●	●	●

As the results, the experiments from patterns A, B, C and D present the high fitness value when the populations and generations increase. The high levels of population of 30, 50 and the generations of 100, 500 present achievements of the high levels of the fitness value included the average fitness values in all patterns. While, the high levels of crossover and mutation are the most disruptive and also achieve the lowest levels of construction of gene. This means that by using high levels of crossover and mutation, the chance that new candidate gene are found decreases. The performance of GA is not so much influenced by these operators than the population sizes and generations. Therefore, this thesis uses the common crossover of 0.9 and mutation 0.001 suggested by De jong (1975) and Goldberg (1985).

The appropriated GA parameters selected here are sufficiently to solve the architectural layout design problem. The Pareto-fronts have been adopted to guarantee these GA parameters which dominate 80% of the fitness values. Nevertheless, the characteristics of GA parameters used in this thesis are selected by concerning the trade-off between the high fitness value in the Pareto-fronts and the minimal running time. The appropriated GA parameters well suited for pattern A, B, C and D can be described as the following statements.

- Population size of 30
- Generation of 100
- Crossover probability of 0.9
- Mutation probability of 0.001
- Selection is a roulette wheel.

5.3 MIP, Valid Inequalities and Learning Methodology Results

In this section, the medium-sized instances (4-10 rooms) are experimented and illustrated. To measure each methodology performance, the objective values, the number of iterations, the computational iteration percentages and the computational time, are illustrated on table 5.1, table 5.2 and table 5.3, respectively.

In the table 5.1, each configuration illustrates the objective values and the numbers of iterations among AL-MIP, AL-MIP+ and AL-MIP+GA.

In the table 5.2, each configuration illustrates the two parts of the computational iteration percentages from AL-MIP, AL-MIP+ and AL-MIP+GA. First, the computational iteration percentages of AL-MIP+ illustrate the comparison between AL-MIP+ and AL-MIP. Second, the computational iteration percentages of AL-MIP+GA illustrate the comparison among AL-MIP and AL-MIP+.

In the table 5.3, each configuration illustrates the computational time and the computational time percentages of AL-MIP, AL-MIP+ and AL-MIP+GA among the four distinct patterns A, B, C and D. This table illustrates the two parts of the computational time and the computational time percentage gains.

Indeed, the GA parameters used in this thesis are experimented on a medium-sized problem of 4, 5, 6, 7, 8, 9 and 10 room configurations among the distinct pattern A, B, C and D. Each experiment has been performed with population sizes of 30, generation iterations of 100, crossover probability of 0.9 and mutation probability of 0.001, see table 5.3.

Table 5.1: Iteration comparisons of AL-MIP, AL-MIP+ and AL-MIP+GA.

			AL-MIP	AL_MIP+		AL-MIP+GA
Room No.	**Patterns**	**Objective Value**		non-circular AL-MIP	non-circular and advised AL-MIP	
4	**A**	15	1.10E+03	8.49E+02	6.15E+02	4.53E+02
	B	17	1.19E+03	1.04E+03	6.17E+02	4.55E+02
	C	18	1.39E+03	1.21E+03	6.55E+02	5.29E+02
	D	15	1.07E+03	1.01E+03	5.51E+02	4.81E+02
5	**A**	50	1.31E+04	1.06E+04	6.99E+03	3.69E+03
	B	50	8.88E+03	7.62E+03	3.65E+03	2.41E+03
	C	55	1.11E+04	8.66E+03	5.04E+03	1.82E+03
	D	50	1.52E+04	1.13E+04	5.56E+03	3.81E+03
6	**A**	90	1.84E+05	1.16E+05	4.38E+04	2.78E+04
	B	93	4.38E+04	2.59E+04	1.62E+04	1.06E+04
	C	104	6.11E+04	2.62E+04	2.55E+04	5.40E+03
	D	94	2.27E+05	1.26E+05	2.71E+04	2.39E+04
7	**A**	150	5.25E+06	3.23E+06	2.51E+05	1.83E+05
	B	166	1.76E+05	8.54E+04	5.65E+04	4.08E+04
	C	165	2.01E+05	1.36E+05	9.22E+04	3.23E+04
	D	151	1.11E+06	1.01E+06	1.35E+05	1.04E+05
8	**A**	225	1.65E+08	1.04E+08	3.00E+06	1.98E+06
	B	237	7.80E+05	6.27E+05	2.79E+05	1.64E+05
	C	255	8.69E+05	6.82E+05	5.92E+05	1.96E+05
	D	240	9.07E+06	4.86E+06	1.75E+06	6.89E+05
9	**A**	310	7.55E+08	5.75E+08	1.71E+07	1.15E+07
	B	350	2.90E+06	1.77E+06	1.53E+06	7.71E+05
	C	370	3.57E+06	2.85E+06	8.55E+05	4.75E+05
	D	329	1.41E+08	7.55E+07	3.77E+06	2.46E+06
10	**A**	425	2.18E+09	1.15E+09	4.15E+07	2.65E+07
	B	482	1.41E+07	8.04E+06	3.63E+06	1.70E+06
	C	526	1.08E+07	5.65E+06	4.21E+06	1.60E+06
	D	449	6.24E+08	2.94E+08	2.61E+07	1.78E+07

Table 5.2: Iteration percentage comparisons of AL-MIP, AL-MIP+ and AL-MIP+GA.

Room No.	Patterns	**Valid Inequalities**			**Learning Methodology**		
		Compare non-circular with AL-MIP	Compare non-Circular and advised AL-MIP with AL-MIP	Compare non-Circular and advised AL-MIP with non-circular AL-MIP	**Compare AL-MIP+GA with**		
					AL-MIP	non-circular AL-MIP	non-circular and advised AL-MIP
4	A	22.61	43.94	27.56	58.71	46.64	26.34
	B	12.68	48.19	40.67	61.80	56.25	26.26
	C	12.81	52.88	45.96	61.94	56.35	19.24
	D	6.07	48.55	45.23	55.09	52.19	12.70
5	A	19.38	46.84	34.06	71.93	65.18	47.19
	B	14.25	58.91	52.09	72.86	68.35	33.95
	C	22.01	54.63	41.82	83.65	79.04	63.97
	D	25.77	63.42	50.71	74.88	66.16	31.34
6	A	37.28	76.25	62.12	84.93	75.97	36.56
	B	40.76	62.98	37.50	75.72	59.01	34.41
	C	57.08	58.27	2.77	91.17	79.43	78.84
	D	44.32	88.06	78.55	89.47	81.09	11.82
7	A	38.56	95.22	92.22	96.52	94.34	27.23
	B	51.37	67.82	33.83	76.73	52.15	27.69
	C	32.65	54.17	31.95	83.96	76.18	65.00
	D	8.68	87.87	86.72	90.63	89.74	22.79
8	A	37.21	98.18	97.10	98.80	98.09	34.04
	B	19.60	64.28	55.57	78.92	73.78	40.98
	C	21.46	31.83	13.20	77.42	71.25	66.88
	D	46.37	80.74	64.09	92.40	85.82	60.52
9	A	23.88	97.74	97.03	98.48	98.00	32.59
	B	38.93	47.10	13.39	73.37	56.39	49.65
	C	20.23	76.05	69.98	86.70	83.33	44.48
	D	46.44	97.33	95.01	98.26	96.74	34.74
10	A	47.01	98.09	96.40	98.78	97.71	36.24
	B	43.20	74.36	54.86	87.96	78.80	53.04
	C	47.48	60.85	25.45	85.09	71.61	61.91
	D	52.90	95.81	91.10	97.15	93.96	32.08

Table 5.3: Time and Time percentage comparisons of AL-MIP, AL-MIP+ and AL-MIP+GA.

Room No.	**Patterns**	**Computational time (sec)**				**Percentage gains (%)**		
		AL-MIP	**AL-MIP+**		**AL-MIP+GA**	**Compare AL-MIP+GA with**		
			non-circular AL-MIP	non-circular and advised AL-MIP		AL-MIP	non-circular AL-MIP	non-circular and advised AL-MIP
4	A	0.06	0.05	0.04	0.03	50.00	40.00	25.00
	B	0.09	0.08	0.05	0.04	55.56	50.00	20.00
	C	0.08	0.07	0.05	0.04	50.00	42.86	20.00
	D	0.05	0.05	0.04	0.03	40.00	40.00	25.00
5	A	0.36	0.31	0.23	0.18	50.00	41.94	21.74
	B	0.31	0.25	0.14	0.11	64.52	56.00	21.43
	C	0.31	0.24	0.16	0.11	64.52	54.17	31.25
	D	0.41	0.3	0.17	0.12	70.73	60.00	29.41
6	A	5.93	3.62	1.35	1.17	80.27	67.68	13.33
	B	1.15	0.77	0.54	0.43	62.61	44.16	20.37
	C	1.86	0.79	0.77	0.60	67.74	24.05	22.08
	D	7.24	4.07	0.87	0.74	89.78	81.82	14.94
7	A	320.97	179.71	9.42	6.13	98.09	96.59	34.93
	B	5.32	2.82	2.02	1.46	72.56	48.23	27.72
	C	6.55	4.23	3.55	1.81	72.37	57.21	38.44
	D	44.81	40.42	4.32	2.54	94.33	93.72	41.20
8	A	11444.71	7897.16	114.9	67.73	99.41	99.14	35.43
	B	26.86	21.94	11.82	7.77	71.07	64.59	34.26
	C	32.45	24.7	20.35	11.19	62.43	50.65	45.01
	D	595.06	316.8	80.16	42.09	92.93	86.71	47.49
9	A	38685.49	17362.18	918.03	537.83	98.61	96.90	41.41
	B	114.97	71.86	59.93	41.78	58.44	33.51	30.29
	C	172.07	134.88	36.98	24.47	85.78	81.86	33.83
	D	12699.56	5979.77	290.96	161.98	98.72	97.29	44.33
10	A	188851.03	83266.98	8327.53	4946.26	97.38	94.06	40.60
	B	829.99	469.99	259.79	178.60	76.07	57.74	31.25
	C	1142.34	841.63	325.59	119.05	89.58	85.85	63.44
	D	48987.75	26561.01	2452.82	1187.83	97.58	95.53	51.57

Table 5.1 shows the objective value, the number of iterations, the computational time in seconds and the percentage gains among AL-MIP, AL-MIP+ and AL-MIP+GA of four architectural patterns vary from 4 to 10 rooms. The column of the objective value is used to compare the optimal solutions from all methodologies. All experiments have the same objective values even though they are different solutions. These results confirm with the theory of mathematical optimization.

According to the table 5.1, the distinct patterns A, B, C and D of 5-10 room configurations illustrate the various computational iterations. A linear configuration (pattern A) uses higher computational iterations. A nested wheel configuration (pattern D) uses less computational iterations than a linear configuration while a rail (pattern B) and a connected configuration (pattern C) use a small numbers in computational iterations. This due to the structural connectivity composes of a large number of repeated patterns of a circular connection that utilizes the non-circular AL-MIP which reduces a feasible region more than a linear and a nested wheel configuration (pattern A and D). For a small room number (4-5 rooms), two computational iterations of a rail and a connected wheel configuration have similar of computational iterations. Both show the similar exponential growth. For a medium room number (6 – 10 rooms), a connected wheel configuration presents a quite different computational iteration between a rail and a connected wheel configuration that differentiate more than 3 times for a 10 rooms. This illustrates that the connected circular constraints as a wheel configuration is suitable to use with a wheel configuration.

Table 5.2 shows the performance among AL-MIP, AL-MIP+ and AL-MIP+GA. The percentage gain is computed by subtracting a measure (computational iterations) of the AL-MIP and AL-MIP+ from the AL-MIP+GA. The larger the positive value is, the better the gain will be. Note that different patterns have different percentage gains. To measure the performance gain of AL-MIP+GA, the final column presents the percentages AL-MIP+GA comparison with our previous methodology AL-MIP+. For a linear configuration (Pattern A), the minimum and maximum iteration percentages are 26.34 and 47.19 for 4 and 6 rooms. For a rail configuration (Pattern B), the minimum and maximum iteration percentages are 26.26 and 53.04 for 4 and 10 rooms. For a connected wheel configuration (Pattern C), the minimum and maximum iteration percentages are 19.24 and 78.84 for 4 and 6 rooms. For a nested wheel configuration (Pattern D), the minimum and maximum iteration percentages are 11.82 and 60.52 of 6 and 8 rooms respectively.

An average computational iteration from 4-10 rooms of a linear configuration (Pattern A) is 38.00, a rail configuration (Pattern B) is 38.00, a connected wheel is 57.19 and a nested wheel is 29.43 respectively.

According to table 5.2, at the forth and fifth columns, the computational iteration percentage gains of valid inequalities present higher reduction of the advised AL-MIP than non-circular AL-MIP. In a medium room number (6-10 rooms), an advised AL-MIP presents a high reduction in a linear configuration. This presents an advised AL-MIP+, is highly suitable for a linear configuration. Moreover, at the final column, the learning methodology using GA presents the higher reduction of computational iterations in a rail and a connected wheel configuration. This presents a room configuration with a higher connection degree among each room is suitable for the AL-MIP+GA.

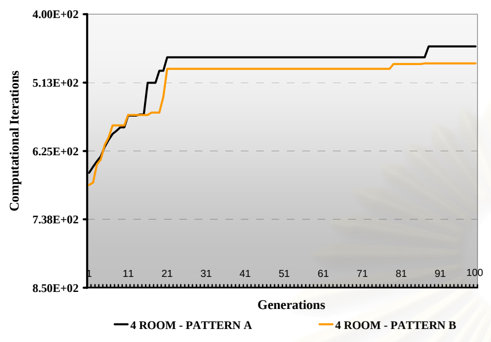
Table 5.3 illustrates the computational time percentages among AL-MIP, AL-MIP+ and AL-MIP+GA of four architectural patterns varying from 4 to 10 rooms. For a linear configuration (Pattern A), the percentage gains of the minimum and the maximum are 13.33 and 41.05 of 6 and 8 rooms. For a rail configuration (Pattern B), the percentage gains of the minimum and the maximum are 20.00 and 27.72 of 4 and 7 rooms. For a connected wheel configuration (Pattern A), the percentage gains of the minimum and the maximum are 20.00 and 63.44 of 4 and 10 rooms. For a nested wheel configuration (Pattern D), the percentage gains of the minimum and the maximum are 14.49 and 51.57 of 6 and 10 rooms.

An average computational time from 4-10 rooms of a linear configuration (Pattern A) is 27.82, a rail configuration (Pattern B) is 22.03, a connected wheel is 35.59 and a nested wheel is 48.48, respectively.

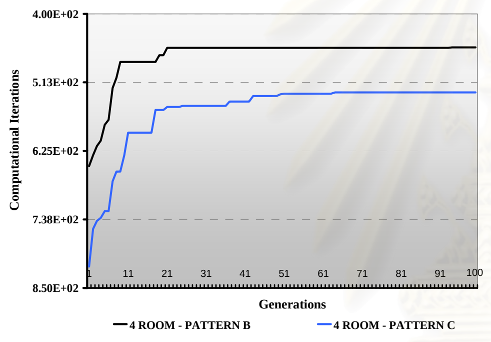
To summarize all patterns, AL-MIP+GA achieves the iterations and time more than 44% and 25% for 5 rooms while AL-MIP+GA achieve the iterations and time more than 45% and 43% for 10 rooms respectively. These results illustrate, the larger the problem is the larger the percentage gain will be. For a linear configuration (pattern B) and a rail configuration (pattern C), we can achieve more than average 40% of the iteration improvement over 6 rooms. This due to the structural connectivity composes of a large number of repeated patterns of circular connections. The memory usages also improve for a larger problem sizes due to the small number of iterations.

- **Fitness and the Candidate Special Order Set Results**

As far as GA is concerned, the fitness curve and the candidate SOS vary from 4-10 rooms of patterns A, B, C and D. These can illustrate using the figure 5.6 to figure 5.12, respectively.



(a)



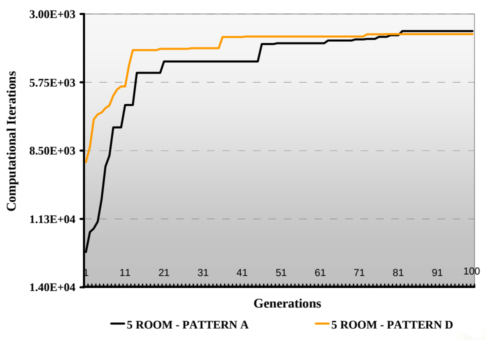
(b)

4 Rooms: the candidate SOS

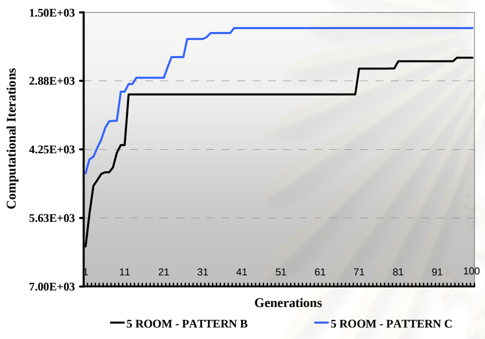
Branching Orders	Pattern A	Pattern B	Pattern C	Pattern D
1	q_{14}	q_{34}	q_{23}	p_{34}
2	q_{24}	p_{24}	q_{14}	q_{12}
3	q_{24}	q_{34}	p_{34}	q_{24}
4	p_{14}	q_{24}	p_{13}	p_{34}
5	q_{14}	p_{23}	p_{24}	p_{34}
6	p_{24}	q_{12}	q_{12}	q_{13}
7	q_{13}	p_{14}	p_{13}	p_{12}
8	p_{13}	p_{34}	q_{13}	p_{13}
9	q_{13}	q_{12}	q_{24}	p_{34}
10	p_{24}	q_{23}	q_{13}	p_{34}
11	p_{14}	p_{14}	p_{24}	p_{12}
12	q_{24}	q_{14}		q_{34}
13		p_{12}		q_{34}
14		p_{23}		
15				

(c)

Figure 5.6: The 4 rooms fitness of computational iterations
(a) between room A and room D, (b) between room B and room C and
(c) illustrates 4 rooms candidate SOS variable p_{ij} and q_{ij} of pattern A, B, D and D.



(a)

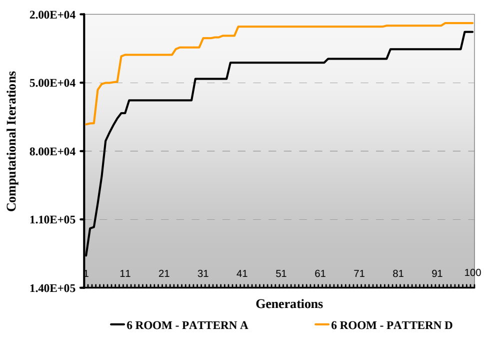


(b)

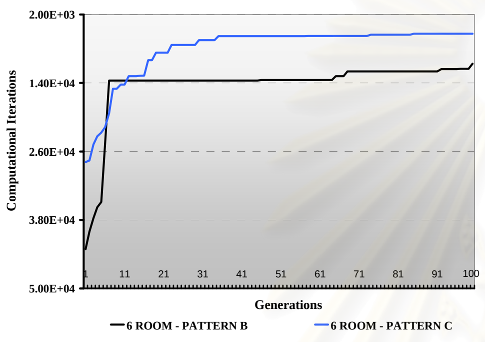
5 Rooms: the candidate SOS				
Branching Orders	Pattern A	Pattern B	Pattern C	Pattern D
1	$q_{1,2}$	$p_{1,5}$	$q_{1,5}$	$q_{1,3}$
2	$q_{1,3}$	$p_{1,4}$	$p_{2,5}$	$p_{2,5}$
3	$p_{2,4}$	$p_{1,4}$	$q_{4,5}$	$p_{2,4}$
4	$q_{2,5}$	$q_{3,5}$	$p_{2,5}$	$q_{2,4}$
5	$p_{1,3}$	$q_{4,5}$	$q_{2,5}$	$p_{2,3}$
6	$q_{2,5}$	$p_{2,5}$	$p_{3,5}$	$q_{2,3}$
7	$q_{2,4}$	$p_{2,5}$	$q_{2,4}$	$q_{1,3}$
8	$p_{3,5}$	$q_{2,5}$	$p_{1,3}$	$q_{3,5}$
9	$q_{3,5}$	$q_{1,4}$	$p_{2,5}$	$p_{1,3}$
10	$p_{1,5}$	$q_{3,5}$	$p_{1,5}$	$p_{1,4}$
11	$p_{1,4}$	$p_{1,3}$	$p_{2,5}$	$p_{1,3}$
12	$q_{1,4}$	$q_{3,5}$	$p_{2,5}$	$p_{2,4}$
13	$q_{1,5}$	$p_{1,5}$	$q_{4,5}$	$q_{2,4}$
14	$p_{1,3}$	$p_{2,5}$	$q_{3,5}$	$p_{1,4}$
15	$p_{2,5}$	$q_{1,3}$	$q_{2,5}$	$q_{2,4}$
16	$q_{2,5}$	$p_{2,5}$	$p_{4,5}$	$p_{3,5}$
17		$q_{2,5}$	$p_{2,5}$	$q_{1,3}$
18		$p_{2,5}$	$q_{1,2}$	$q_{3,5}$
19		$q_{1,5}$		$p_{2,5}$
20		$q_{4,5}$		$q_{1,2}$
21				$q_{2,5}$
22				
23				
24				
25				

(c)

Figure 5.7: The 5 rooms fitness of computational iterations (a) between room A and room D, (b) between room B and room C and (c) illustrates 5 rooms candidate SOS variable p_{ij} and q_{ij} of pattern A, B, D and D.



(a)

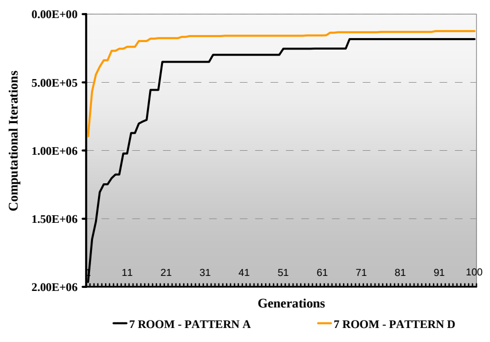


(b)

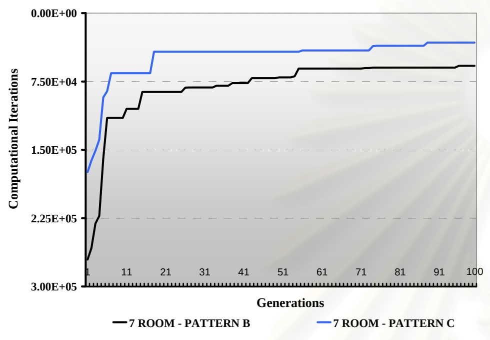
6 rooms: the candidate SOS				
Branching Orders	Pattern A	Pattern B	Pattern C	Pattern D
1	$q_{2,4}$	$p_{2,5}$	$p_{5,6}$	$q_{3,6}$
2	$q_{1,5}$	$p_{1,3}$	$p_{2,5}$	$q_{2,4}$
3	$q_{1,6}$	$p_{1,5}$	$q_{5,6}$	$q_{2,5}$
4	$p_{1,2}$	$p_{1,4}$	$p_{2,6}$	$p_{2,4}$
5	$p_{1,2}$	$p_{3,5}$	$p_{2,5}$	$p_{1,4}$
6	$p_{1,6}$	$q_{1,5}$	$p_{1,5}$	$p_{2,6}$
7	$q_{2,6}$	$q_{2,5}$	$p_{3,5}$	$p_{2,4}$
8	$q_{3,6}$	$p_{1,5}$	$p_{2,6}$	$q_{2,6}$
9	$p_{3,6}$	$p_{1,5}$	$p_{2,6}$	$p_{1,4}$
10	$q_{1,4}$	$q_{1,3}$	$q_{1,5}$	$p_{1,4}$
11	$q_{1,5}$	$p_{1,4}$	$p_{4,6}$	$p_{2,5}$
12	$q_{1,5}$	$q_{1,2}$	$q_{2,4}$	$p_{1,5}$
13	$q_{2,5}$	$p_{5,6}$	$p_{5,6}$	$q_{2,6}$
14	$q_{1,3}$	$q_{3,6}$	$q_{4,5}$	$q_{1,3}$
15	$q_{1,3}$	$q_{1,3}$	$q_{1,6}$	$p_{2,6}$
16	$q_{1,4}$	$p_{1,4}$	$p_{2,6}$	$q_{1,3}$
17	$q_{1,6}$	$q_{2,6}$	$q_{5,6}$	$q_{1,3}$
18	$q_{1,6}$	$p_{5,6}$	$p_{4,5}$	$p_{4,6}$
19	$q_{3,6}$	$p_{3,6}$	$p_{1,5}$	$q_{1,3}$
20	$q_{1,5}$	$p_{2,6}$	$q_{2,5}$	$p_{1,2}$
21	$q_{2,6}$	$p_{3,6}$	$p_{3,6}$	$p_{4,6}$
22	$p_{1,4}$	$q_{4,6}$	$q_{2,6}$	$q_{4,6}$
23	$p_{3,6}$	$p_{1,4}$	$q_{4,6}$	$q_{3,6}$
24	$q_{2,4}$	$p_{1,6}$	$p_{1,6}$	$q_{1,5}$
25		$p_{1,4}$	$q_{2,6}$	$p_{3,6}$
26		$p_{2,5}$	$q_{1,6}$	$q_{1,3}$
27		$q_{1,6}$	$q_{3,6}$	$p_{4,6}$
28		$p_{1,3}$	$p_{2,6}$	$q_{1,5}$
29		$p_{1,4}$		$p_{3,6}$
30		$q_{4,6}$		$p_{1,2}$
31		$q_{1,5}$		
32		$q_{3,6}$		
33				
34				
35				

(c)

Figure 5.8: The 6 rooms fitness of computational iterations
(b) between room A and room D, (b) between room B and room C and
(c) illustrates 6 rooms candidate SOS variable p_{ij} and q_{ij} of pattern A, B, D and D.



(a)

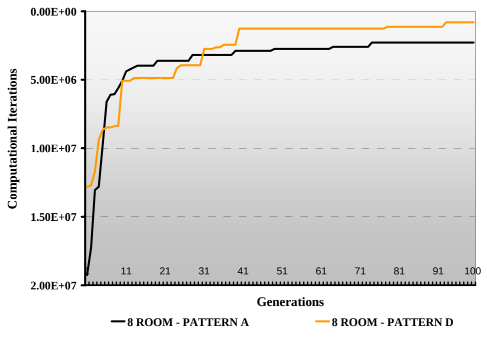


(b)

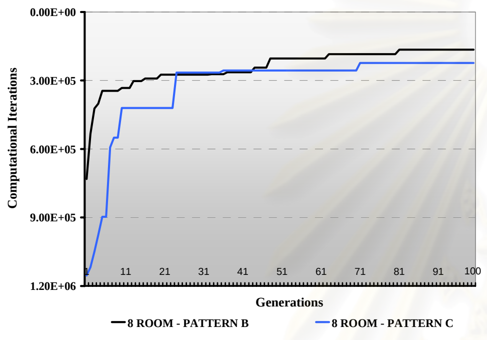
7 rooms: the candidate SOS				
Branching Orders	Pattern A	Pattern B	Pattern C	Pattern D
1	$p_{5,6}$	$q_{1,5}$	$p_{2,5}$	$p_{3,4}$
2	$q_{3,7}$	$q_{1,3}$	$p_{6,7}$	$p_{3,7}$
3	$p_{2,5}$	$q_{2,6}$	$p_{2,5}$	$q_{3,7}$
4	$p_{1,4}$	$p_{2,5}$	$q_{2,7}$	$q_{5,7}$
5	$q_{1,4}$	$p_{1,3}$	$q_{1,7}$	$q_{1,4}$
6	$q_{2,7}$	$p_{1,3}$	$p_{2,5}$	$p_{2,5}$
7	$p_{4,7}$	$p_{5,6}$	$q_{3,6}$	$p_{4,7}$
8	$p_{2,5}$	$q_{1,6}$	$q_{1,6}$	$p_{5,7}$
9	$q_{1,3}$	$p_{1,7}$	$p_{1,7}$	$q_{2,5}$
10	$q_{4,7}$	$q_{1,6}$	$q_{4,7}$	$q_{2,4}$
11	$p_{1,5}$	$p_{5,6}$	$p_{2,5}$	$q_{2,4}$
12	$p_{1,4}$	$q_{1,3}$	$p_{2,7}$	$p_{2,5}$
13	$q_{5,7}$	$p_{6,7}$	$p_{4,7}$	$q_{5,6}$
14	$p_{1,6}$	$p_{1,4}$	$p_{1,4}$	$q_{3,7}$
15	$p_{1,4}$	$p_{1,6}$	$q_{2,7}$	$p_{4,6}$
16	$q_{3,7}$	$q_{2,5}$	$q_{4,5}$	$q_{1,2}$
17	$p_{2,5}$	$q_{1,7}$	$p_{6,7}$	$p_{2,5}$
18	$p_{2,5}$	$p_{1,7}$	$p_{4,5}$	$q_{4,6}$
19	$p_{1,7}$	$q_{1,7}$	$q_{1,6}$	$q_{4,7}$
20	$p_{2,5}$	$p_{2,7}$	$p_{1,6}$	$q_{2,4}$
21	$q_{2,4}$	$p_{2,7}$	$q_{1,5}$	$q_{5,7}$
22	$p_{3,7}$	$p_{4,7}$	$q_{3,7}$	$q_{3,6}$
23	$q_{3,7}$	$q_{1,2}$	$q_{1,6}$	$p_{5,6}$
24	$q_{2,4}$	$p_{6,7}$	$q_{6,7}$	$q_{1,4}$
25	$q_{3,5}$	$p_{4,6}$	$p_{1,7}$	$q_{1,2}$
26	$q_{1,4}$	$p_{1,5}$	$q_{1,2}$	$p_{5,7}$
27		$q_{1,5}$	$p_{3,7}$	
28			$q_{2,6}$	
29				
30				

(c)

Figure 5.9: The 7 rooms fitness of computational iterations
(c) between room A and room D, (b) between room B and room C and
(c) illustrates 7 rooms candidate SOS variable p_{ij} and q_{ij} of pattern A, B, D and D.



(a)



(b)

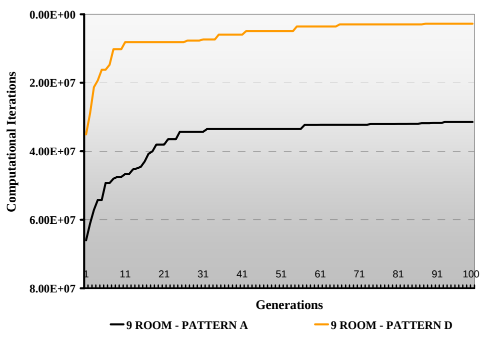
8 rooms: the candidate SOS				
Branching Orders	Pattern A	Pattern B	Pattern C	Pattern D
1	$p_{5,8}$	$q_{2,8}$	$q_{2,8}$	$q_{4,7}$
2	$q_{2,5}$	$q_{2,7}$	$q_{2,6}$	$p_{1,3}$
3	$q_{1,4}$	$q_{1,4}$	$q_{1,5}$	$q_{4,8}$
4	$p_{1,8}$	$p_{1,8}$	$p_{1,8}$	$p_{1,4}$
5	$q_{2,6}$	$q_{2,6}$	$q_{2,5}$	$q_{3,6}$
6	$p_{3,8}$	$q_{2,5}$	$p_{2,8}$	$p_{3,5}$
7	$q_{5,7}$	$p_{3,8}$	$q_{1,6}$	$q_{4,6}$
8	$p_{4,8}$	$p_{2,8}$	$p_{2,6}$	$p_{2,6}$
9	$q_{1,4}$	$q_{2,7}$	$p_{2,5}$	$p_{4,7}$
10	$q_{1,3}$	$q_{4,7}$	$q_{4,8}$	$q_{1,3}$
11	$p_{3,8}$	$p_{4,8}$	$q_{4,6}$	$p_{4,8}$
12	$p_{4,7}$	$p_{1,7}$	$q_{5,7}$	$p_{4,6}$
13	$q_{5,8}$	$p_{2,8}$	$p_{5,8}$	$q_{1,4}$
14	$q_{1,6}$	$p_{2,5}$	$p_{4,8}$	$p_{3,6}$
15	$p_{2,8}$	$q_{3,8}$	$p_{1,6}$	$q_{1,3}$
16	$p_{1,4}$	$q_{3,7}$	$p_{4,6}$	$p_{2,4}$
17	$q_{1,6}$	$p_{1,6}$	$q_{3,8}$	$p_{1,4}$
18	$q_{4,8}$	$q_{3,6}$	$q_{3,7}$	$p_{1,3}$
19	$p_{3,7}$	$p_{3,8}$	$p_{1,5}$	$p_{3,6}$
20	$p_{1,8}$	$q_{1,7}$	$q_{3,6}$	$q_{2,6}$
21	$q_{1,4}$	$p_{3,7}$	$q_{1,8}$	$q_{2,4}$
22	$p_{3,6}$	$p_{3,6}$	$p_{3,8}$	$q_{3,5}$
23	$q_{4,8}$	$q_{5,8}$	$p_{3,7}$	$p_{2,6}$
24	$q_{1,8}$	$q_{4,8}$	$p_{5,7}$	$q_{1,7}$
25	$p_{2,4}$	$p_{4,7}$	$p_{3,6}$	$p_{3,6}$
26	$p_{1,5}$	$q_{3,6}$	$q_{1,7}$	$q_{1,4}$
27	$q_{2,7}$	$q_{1,8}$	$q_{1,6}$	$q_{4,6}$
28	$p_{2,5}$	$q_{1,6}$	$q_{5,8}$	$p_{1,7}$
29	$p_{3,8}$	$p_{5,8}$	$p_{5,7}$	$p_{1,4}$
30	$q_{3,8}$	$q_{1,5}$	$q_{1,8}$	$q_{5,8}$
31	$q_{3,5}$	$q_{4,7}$	$p_{2,5}$	$q_{3,6}$
32		$p_{1,5}$	$q_{1,7}$	$p_{5,8}$
33		$p_{1,4}$	$p_{1,7}$	$p_{2,4}$
34				$q_{1,4}$
35				$p_{1,4}$
36				
37				

(c)

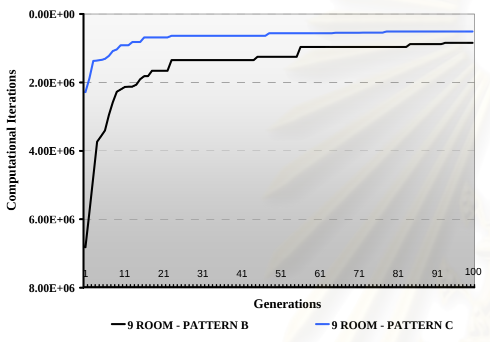
Figure 5.10: The 8 rooms fitness of computational iterations

(d) between room A and room D, (b) between room B and room C and

(c) illustrates 8 rooms candidate SOS variable p_{ij} and q_{ij} of pattern A, B, D and D.



(a)



(b)

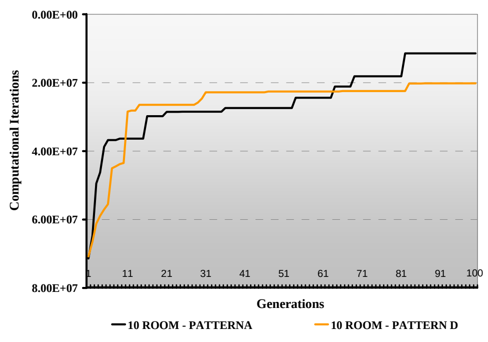
9 rooms : the candidate SOS				
Branching Orders	Pattern A	Pattern B	Pattern C	Pattern D
1	$p_{5,8}$	$q_{2,9}$	$q_{2,9}$	$q_{5,6}$
2	$p_{3,6}$	$p_{3,7}$	$p_{4,6}$	$p_{1,7}$
3	$p_{1,9}$	$q_{2,8}$	$q_{3,9}$	$q_{5,9}$
4	$q_{2,7}$	$q_{4,7}$	$p_{1,6}$	$p_{2,6}$
5	$p_{5,9}$	$p_{4,9}$	$q_{3,8}$	$q_{5,3}$
6	$q_{1,5}$	$p_{1,7}$	$q_{2,8}$	$q_{3,7}$
7	$q_{4,7}$	$p_{4,8}$	$q_{4,6}$	$q_{1,3}$
8	$p_{5,6}$	$q_{6,9}$	$p_{4,9}$	$q_{3,9}$
9	$q_{1,4}$	$q_{2,7}$	$q_{2,7}$	$p_{5,6}$
10	$q_{5,9}$	$q_{2,6}$	$p_{3,6}$	$q_{6,8}$
11	$p_{2,5}$	$q_{2,5}$	$q_{2,6}$	$p_{5,3}$
12	$p_{2,9}$	$q_{3,8}$	$p_{2,9}$	$q_{1,7}$
13	$q_{1,8}$	$q_{1,5}$	$q_{1,6}$	$q_{3,9}$
14	$q_{3,9}$	$q_{1,4}$	$q_{4,8}$	$p_{5,6}$
15	$p_{2,5}$	$q_{5,8}$	$q_{4,7}$	$q_{1,3}$
16	$q_{1,3}$	$p_{5,9}$	$q_{1,5}$	$p_{7,9}$
17	$p_{1,8}$	$p_{1,9}$	$p_{1,9}$	$p_{1,7}$
18	$q_{1,4}$	$p_{1,8}$	$p_{2,8}$	$p_{1,3}$
19	$q_{3,8}$	$q_{4,8}$	$q_{2,6}$	$p_{6,8}$
20	$q_{5,9}$	$p_{2,9}$	$q_{4,9}$	$q_{2,8}$
21	$p_{7,9}$	$q_{2,7}$	$p_{3,6}$	$q_{7,9}$
22	$p_{2,8}$	$p_{1,6}$	$q_{5,9}$	$p_{2,8}$
23	$p_{6,9}$	$p_{2,8}$	$p_{4,8}$	$p_{2,6}$
24	$q_{1,9}$	$p_{2,5}$	$p_{4,7}$	$q_{1,4}$
25	$p_{1,4}$	$q_{3,9}$	$q_{3,7}$	$p_{3,9}$
26	$q_{2,8}$	$q_{3,7}$	$p_{3,8}$	$p_{1,4}$
27	$p_{3,8}$	$p_{5,8}$	$q_{5,8}$	$q_{7,9}$
28	$p_{5,,8}$	$q_{4,9}$	$q_{1,9}$	$q_{5,,3}$
29	$q_{1,6}$	$p_{3,8}$	$p_{5,,8}$	$p_{7,9}$
30	$p_{4,7}$	$p_{3,6}$	$p_{5,7}$	$q_{3,7}$
31	$q_{7,9}$	$p_{6,9}$	$q_{1,8}$	$p_{5,9}$
32	$q_{3,7}$	$q_{5,9}$	$q_{1,7}$	$q_{3,9}$
33	$q_{1,7}$	$p_{4,7}$	$p_{2,8}$	$p_{3,7}$
34	$p_{3,6}$	$q_{1,9}$	$p_{1,8}$	$q_{2,6}$
35	$q_{6,9}$	$q_{1,7}$	$q_{5,7}$	$p_{3,9}$
36	$p_{2,8}$	$q_{3,6}$	$p_{5,9}$	$q_{1,4}$
37		$p_{3,9}$	$p_{1,7}$	$q_{6,8}$
38		$q_{1,6}$	$p_{1,6}$	$p_{1,4}$
39		$p_{1,5}$	$p_{1,5}$	$p_{5,9}$
40		$p_{1,4}$		$q_{2,8}$
41				$q_{2,6}$
42				

(c)

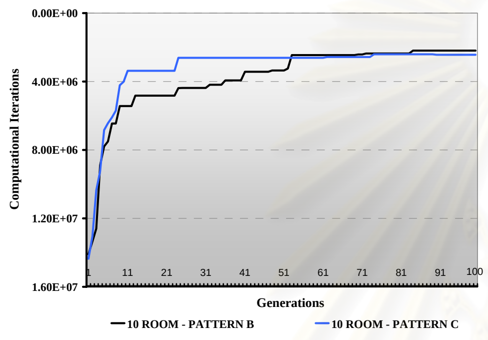
Figure 5.11: The 9 rooms fitness of computational iterations

(e) between room A and room D, (b) between room B and room C and

(c) illustrates 9 rooms candidate SOS variable p_{ij} and q_{ij} of pattern A, B, D and D.



(a)



(b)

10 rooms: the candidate SOS				
Branching Orders	Pattern A	Pattern B	Pattern C	Pattern D
1	$p_{4,8}$	$q_{2,10}$	$q_{2,10}$	$q_{5,9}$
2	$p_{1,5}$	$q_{2,9}$	$p_{4,10}$	$p_{2,10}$
3	$q_{3,6}$	$p_{4,7}$	$q_{2,9}$	$q_{1,9}$
4	$q_{2,5}$	$q_{1,8}$	$p_{3,8}$	$q_{3,10}$
5	$p_{4,7}$	$q_{2,8}$	$q_{1,8}$	$p_{6,9}$
6	$q_{2,7}$	$q_{4,7}$	$p_{3,7}$	$q_{5,8}$
7	$p_{5,9}$	$q_{2,7}$	$q_{2,8}$	$q_{1,10}$
8	$p_{2,9}$	$q_{3,9}$	$p_{1,9}$	$q_{3,7}$
9	$p_{1,10}$	$q_{2,6}$	$q_{2,6}$	$p_{1,9}$
10	$q_{1,5}$	$q_{1,4}$	$q_{5,10}$	$p_{5,9}$
11	$q_{4,10}$	$p_{1,10}$	$p_{2,9}$	$p_{5,8}$
12	$p_{2,9}$	$q_{2,5}$	$q_{1,6}$	$q_{6,9}$
13	$p_{2,5}$	$p_{5,10}$	$p_{2,8}$	$q_{3,10}$
14	$q_{4,8}$	$p_{2,10}$	$q_{1,5}$	$p_{5,3}$
15	$q_{1,5}$	$q_{1,10}$	$q_{5,10}$	$q_{3,9}$
16	$p_{2,8}$	$q_{1,9}$	$p_{1,10}$	$p_{6,10}$
17	$p_{3,10}$	$p_{2,9}$	$q_{4,10}$	$p_{3,7}$
18	$q_{1,4}$	$q_{1,8}$	$p_{2,7}$	$p_{5,2}$
19	$p_{2,8}$	$p_{2,8}$	$q_{4,9}$	$p_{1,7}$
20	$q_{1,10}$	$q_{3,7}$	$p_{3,6}$	$q_{4,6}$
21	$p_{2,5}$	$q_{4,10}$	$q_{4,8}$	$q_{6,8}$
22	$p_{1,9}$	$q_{3,6}$	$p_{1,10}$	$q_{3,10}$
23	$q_{2,6}$	$q_{2,7}$	$p_{3,9}$	$p_{6,8}$
24	$q_{4,7}$	$q_{1,5}$	$q_{4,6}$	$q_{2,8}$
25	$p_{5,9}$	$p_{2,8}$	$P_{3,10}$	$p_{4,7}$
26	$q_{3,9}$	$p_{2,5}$	$q_{1,9}$	$q_{2,6}$
27	$p_{4,6}$	$q_{3,10}$	$q_{4,7}$	$p_{3,10}$
28	$p_{5,8}$	$q_{1,6}$	$q_{1,8}$	$p_{2,6}$
29	$q_{1,6}$	$p_{1,9}$	$p_{4,9}$	$q_{1,4}$
30	$p_{1,10}$	$p_{3,7}$	$q_{3,6}$	$q_{3,6}$
31	$p_{4,9}$	$q_{1,10}$	$p_{4,8}$	$q_{7,9}$
32	$q_{1,5}$	$p_{3,6}$	$q_{5,6}$	$q_{4,7}$
33	$q_{6,8}$	$q_{6,10}$	$p_{4,7}$	$p_{7,9}$
34	$p_{7,9}$	$p_{3,9}$	$q_{3,10}$	$q_{3,7}$
35	$q_{6,10}$	$p_{5,9}$	$p_{4,6}$	$p_{1,10}$
36	$p_{5,9}$	$p_{5,8}$	$q_{3,9}$	$q_{3,9}$
37	$q_{1,10}$	$q_{6,9}$	$p_{5,9}$	$p_{1,8}$
38	$p_{3,9}$	$p_{6,10}$	$q_{1,10}$	$p_{3,7}$
39	$p_{3,6}$	$q_{5,9}$	$p_{5,8}$	$p_{3,9}$
40	$q_{1,9}$	$q_{3,8}$	$q_{3,8}$	$q_{1,4}$
41	$p_{3,10}$	$p_{3,10}$	$p_{1,9}$	$q_{2,8}$
42	$q_{5,8}$	$q_{5,8}$	$q_{3,7}$	$p_{3,6}$
43	$p_{6,9}$	$p_{5,8}$	$q_{5,8}$	$q_{2,6}$
44	$p_{1,9}$	$p_{4,8}$	$p_{2,10}$	$q_{1,9}$
45	$p_{3,6}$	$p_{6,9}$	$p_{5,7}$	$p_{2,8}$
46	$q_{4,8}$	$q_{5,10}$	$q_{1,7}$	$p_{6,9}$
47	$q_{5,7}$	$q_{1,7}$	$q_{5,7}$	$p_{2,6}$
48	$p_{2,9}$	$p_{1,8}$	$p_{5,10}$	$q_{1,8}$
49	$q_{3,9}$	$q_{4,8}$	$p_{1,8}$	$p_{4,6}$
50		$p_{4,9}$	$p_{1,7}$	$p_{1,7}$
51		$p_{1,7}$	$q_{2,7}$	$p_{3,9}$
52		$p_{1,6}$	$p_{1,6}$	$p_{1,4}$
53		$q_{4,8}$	$p_{1,5}$	$p_{1,3}$
54		$p_{1,5}$		
55		$p_{1,4}$		

(c)

Figure 5.12: The 10 rooms fitness of computational iterations
(f) between room A and room D, (b) between room B and room C and
(c) illustrates 10 rooms candidate SOS variable p_{ij} and q_{ij} of pattern A, B, D and D.

The GA design process is executed with a random initial population. In order to show the improvement of the AL-MIP+GA, the populations are made up of 30, the numbers of generations of the genetic search process are set at 100, the crossover probability is set at 0.01 and the mutation probability is set at 0.001. Our AL-MIP+GA normally terminate after the repeated process of generations reaching the required generations. The strong gene present the candidate SOS that will appropriately use to speed up computational solution time of AL-MIP+.

Figure 5.6 to 5.12 illustrate computational iterations of a typical run. The vertical axis of 4-10 room configurations represents the computational iteration scales where the upward direction corresponds to the improvement of computational iterations. The horizontal axis represents the change of generations.

From the fitness results, each figure illustrates the improvement behaviors of the computational iterations as the generations increase. In order to understand this behavior, figure 5.6 to figure 5.12 illustrate the fitness curve comparisons. Due to the different of the computational scales between pattern A, D and pattern B, C, we use two fitness figures to illustrate the computational iterations for each case. At the beginning of a period, the fitness curve presents the higher growth between generations 1 to 40. This presents our AL-MIP+GA corresponding to a general learning rate GA (Chen et al., 1993 and Goldberg, 1989). The candidate SOS variables are found after 40 generations which the mutation will be adopted to increase the better fitness value of the candidate SOS.

Moreover, we illustrate the candidate SOS among pattern A, B, C and D in each case on figure 5.6(c) to figure 5.12(c). Each case, patterns A, B, C and D present a nonequivalent length of candidate SOS from a nonequivalent connectivity degree. The length of candidate SOS will increase corresponded to the increase of room numbers.

Finally, the AL-MIP+GA model described here illustrate the potential uses in the MIP branch and bound algorithm. The candidate SOS used a robustness GA can reduce an average computational iteration and time more than a thirty percent compared to the AL-MIP+ model.

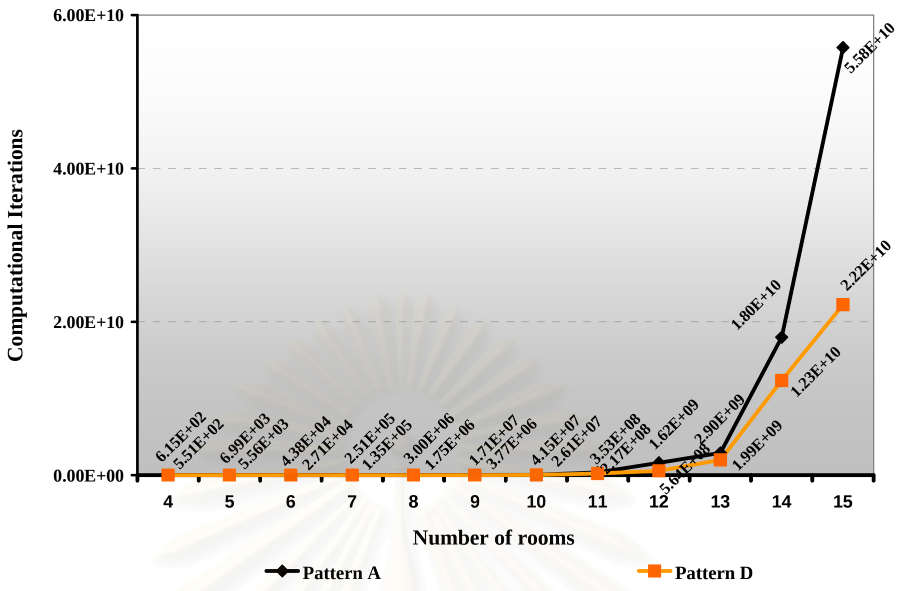
- **The efficiency of AL-MIP+ model**

To verify an efficiency of AL-MIP+ model we extent the numbers of room from 11 to 15 room configurations of patterns A, B, C and D. These experiments are tested on the similar environments from the previous section. The computational iterations and time from 4 to 15 room configurations can be illustrated as follows.

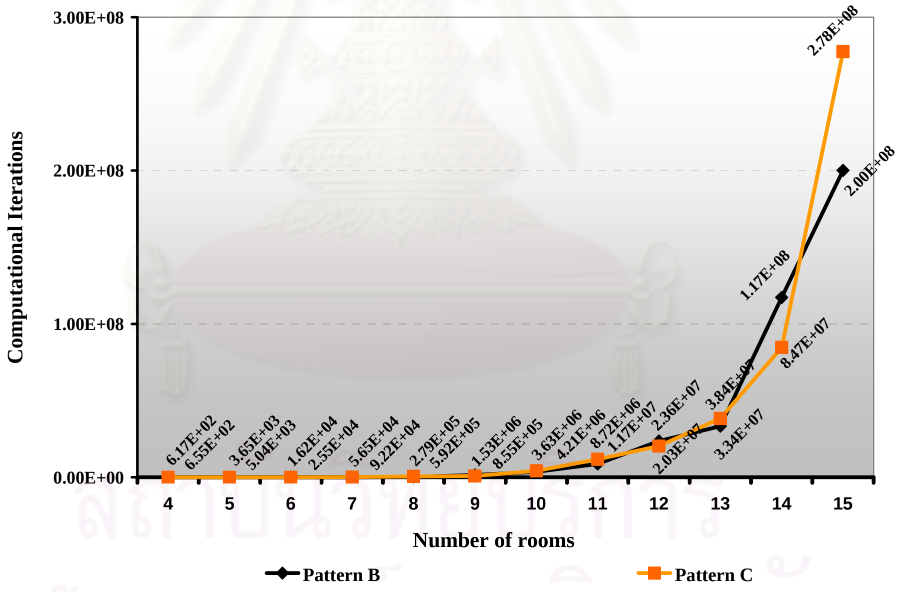
Table 5.4: computational iteration and time comparisons of 4 to 15 rooms.

Room numbers	Pattern A		Pattern B		Pattern C		Pattern D	
	Computations		Computations		Computations		Computations	
	Iterations	Time (sec)	Iterations	Time (sec)	Iterations	Time (sec)	Iterations	Time (sec)
4	6.15E+02	0.03	6.17E+02	0.04	6.55E+02	0.04	5.51E+02	0.03
5	6.99E+03	0.18	3.65E+03	0.11	5.04E+03	0.11	5.56E+03	0.12
6	4.38E+04	1.17	1.62E+04	0.43	2.55E+04	0.60	2.71E+04	0.74
7	2.51E+05	6.13	5.65E+04	1.46	9.22E+04	1.81	1.35E+05	2.54
8	3.00E+06	67.73	2.79E+05	7.77	5.92E+05	11.19	1.75E+06	42.09
9	1.71E+07	537.83	1.53E+06	41.78	8.55E+05	24.47	3.77E+06	161.98
10	4.15E+07	4946.26	3.63E+06	178.6	4.21E+06	119.05	2.61E+07	687.83
11	3.53E+08	31656.06	8.72E+06	723.09	1.17E+07	465.26	2.17E+08	3858.99
12	1.62E+09	101299.41	2.36E+07	2045.1	2.03E+07	1521.23	5.64E+08	10633.37
13	2.90E+09	227923.66	3.34E+07	2079.12	3.84E+07	5443.72	1.99E+09	37485.80
14	1.80E+10	501432.05	1.17E+08	15610.25	8.47E+07	34185.90	1.23E+10	121011.97
15	5.58E+10	902577.69	2.00E+08	37337.12	2.78E+08	67981.47	2.22E+10	509821.54

As the results, the AL-MIP+ model presents the effectiveness to solve the larger scale problem from 4 to 15 room configurations which has the exponential growth of the computational iterations and time. Particularly, pattern A and D present the large increases of computational iterations comparing to pattern B and C, see figure 5.13. Moreover the running time of pattern A and D present more than a week to achieve the solution for the 15 room configurations. With the results of the growth function confirm that an architectural layout design is an NP hard problem. Therefore it is not easily solved by using a conventional technique.



(a)



(b)

Figure 5.13: The computational iteration comparisons from 4 to 15 rooms (a) between room A and room D and (b) between room B and room C.

CHAPTER VI
Conclusions and Suggestions

6.1 Conclusions and Suggestions

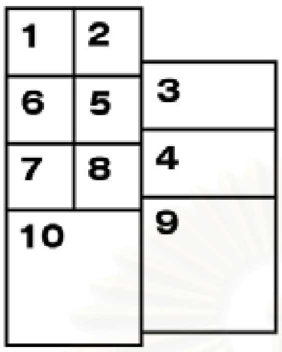
We propose the feasibility of the AL-MIP, the AL-MIP+ and the AL-MIP+GA to solve an architectural layout design optimization. Dealing with a medium-sized problem (5-10 rooms), the AL-MIP+ helps reduce the computational iterations and time considerably. The experiments show the feasibility of using AL-MIP model included two valid inequalities. The average computational time for 10 room configurations of pattern B, C and D, can be solved in a few minute with the global optimal. More than one third can be reduced the computational iterations and time from AL-MIP due to a smaller feasible region.

The AL-MIP+GA based on the learning methodology using GA is adopted to reduce the computational iterations and time. This GA identifies the current best candidate of a Special Order Set (a strong gene) which achieves an average of 30 to 70 percentage gain reductions compare to the AL-MIP+ while the computational iterations and time illustrate an average more than 90 percent reduction gains comparing to AL-MIP. Indeed, the graphical results of 10 room configurations, see figure 6.1 presents an achievement of the global optimal solution.

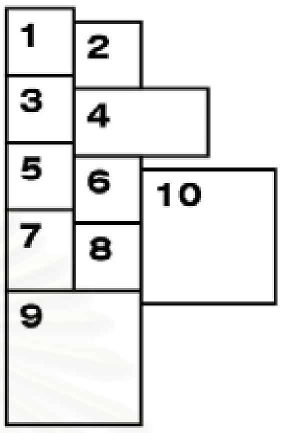
The AL-MIP included valid inequalities and learning methodology reveal a significant potential for computational optimization algorithms. The consistency between mathematical formulation and machine learning creates a distinct MIP as an optimization methodology.

- **Applying AL-MIP+GA**

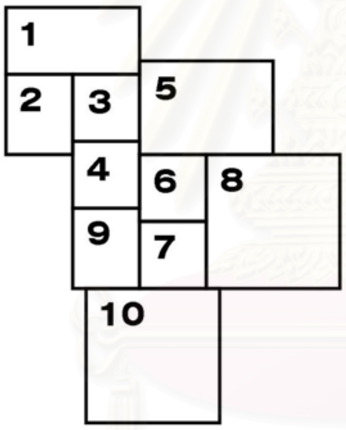
In order to apply the AL-MIP+GA for the architectural layout design, the candidate SOS can be used to reduce the computational iterations and time by adding to the problem as a preprocess data. Indeed, some architectural layout design patterns might be often used in a layout design. For example a linear pattern that is used as a pattern of circulation of museum design and a pattern of circulation of factory design. Therefore, a regularly architectural layout design pattern can be swiftly solved using the candidate SOS which we have stored from the AL-MIP+GA. In the other words, we can save the computational iterations and time by learning the candidate SOS that has been regularly used and store as a preprocess data.



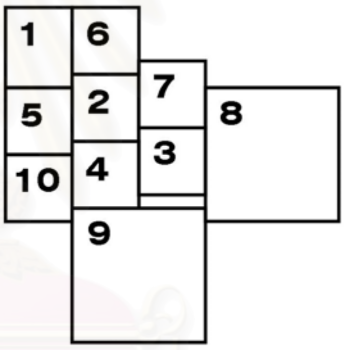
Pattern A



Pattern B



Pattern C



Pattern D

Figure 6.1: The graphical results of 10 room configurations of patterns A, B, C and D.

- **Conclusions**

In this thesis, several results of architectural layout design problem using the AL-MIP, the AL-MIP+ and the AL-MIP+GA can be concluded as follows.

1. The AL-MIP, the AL-MIP+ and the AL-MIP+GA can be utilized with architectural layout design problem that helps architects solve the medium-sized problem within a reasonable time.
2. The valid inequalities can be used to reduce the search space while still maintain the integer optimal solutions.
3. The candidate SOS can reduce the search space from the MIP branch and bound Algorithm.
4. GA is the robustness learning methodology for the MIP branch and bound algorithm.
5. This thesis presents the distinct MIP methods that consist of valid inequalities and learning methodology.

6.2 Suggestions

Due to the nature of design problems, the fitness function of the quality of the solutions during the genetic process can be computationally very demanding. Great efforts have been made towards reducing the number of evaluations needed before the final solution is reached.

Moreover, our approach can be further developed a possible perspective direction for improvement an architectural layout design problem.

- **Improve Architectural Layout Design Constraints**

New constraints and objectives can be added to the model to improve optimization behavior, better represent architectural criteria, and improve the quality of layouts

- **Multiple Floors**

The ability to apply an architectural layout design for a multi-level floor layout is an important area since the modern high-rise building is comprised of a multi floor design.

- **Complex Shapes**

A more generalized unit component that can represent non-rectangular and non-orthogonal shapes would be necessary to generalize this idea to handle a practical architectural layout design problem.

- **Parallel Computing**

For the perspective views, over a medium-sized room (10-20 rooms), our approach with the domain expert and the parallel computing should be adopted to reduce the computational iterations and time.

References

Balachandran M and Gero JS. Dimensioning of architectural floor plans under conflicting objectives. Environment and Planning 1987;14:29-37.

Battle D.L., and Vose M.D. Isomorphisms of genetic algorithms. Paper presented at the Foundations of Genetic Algorithms; 1990.

Baykan C and Fox M. Constraint satisfaction techniques for spatial planning. Intelligent CAD Systems III, Practical Experience and Evaluation 1991.

Bloch CJ and Krishnamurti R. The counting of rectangular dissections. Environment and Planning 1978;2:207-214.

Cagan J, Degentesh D and Yin S. A simulated annealing based algorithm using hierarchical models for general three-dimentional component layout. Computer-Adied Design 1998;30(10):781-790.

Chen YH and Wang YZ. Genetic algorithms for optimized retriangulation in the context of reverse engineering. Computer-Aided Design 1993;31(4):261-271.

Damski JC and Gero JS. An evolutionary approach to generating constraint-based space layout topologies. In: Junge R (ed.). CAAD Future 1997;855–874.

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

Deb K. Genetic Algorithm in Search and Optimization: The Technique and Applications. Kanpur: Indian Institute of Technology; 1999.

Deb K and Gene AS. A robust optimal design technique for mechanicalcomponent design. Evolutionary algorithms in engineering applications. Berlin: Springer 1997;497–514.

Frederick S H and Gerald J L. Introduction to mathematical programming. McGraw-Hill Publishing Company; 1990.

Flemming U. Representation and generation of rectangular dissections. Annual ACM IEEE Design Automation Conference 1978;15:138-144.

Flemming U. A generative expert system for the design of building layouts. Artificial intelligence in engineering: design. New York: Elsevier; 1988.

Frazer J. Creative design and the generative evolutionary paradigm. In: Bentley PJ, Corne DW, editors. Creative evolutionary systems. New York: Academic Press: 2002;253-274.

George LN and Laurence AW. Integer and combinatorial optimization. New York: A Wiley-Interscience Publication; 1988.

Gero JS and Kazakov VA. Evolving design genes in space layout planning problems. Arificial Intelligence in Engineering 1998;12(3):163–176.

Gero JS. Design prototypes: a knowledge representation schema for design. AI Magazine 1990;11(4):26-36.

Goldberg DE. Genetic algorithms in search, optimization and machine learning. Massachusetts: Addison-Wesley Pubblishing Company, Reading; 1989.

Hart W.E. and Belew R.K. Optimizing an arbitrary function is hard for the genetic algorithms. Paper presented at the Proceedings of the Fourth International Conference on Genetic Algorithms; 1991.

Hesser J. and Männer R. Towards an optimal mutation probability for genetic algorithms. Paper presented at the Parallel Problem Solving from Nature-Proceedings of the first workshop, PPSN1, Dortmund, Germany: Springer-Verlag, Berlin, Germany. 1991.

Homayouni H. A Survey of Computational Approaches to Space Layout Planning (1965-2000). Computational Approaches to Space Layout Planning; 2006.

Honda K and Mizoguchi F. Constraint-based approach for automatic spatial layout planning. 11th Conference on Artificial Intelligence for Applications 1995;38.

Jeremy F S. Mathematical Programming Structures and Algorithms. A Wiley-Interscience Publication 1979.

Jo JH and Gero JS. Space layout planning using an evolutionary approach. Artificial Intelligence in Engineering 1998;12:146-162.

Keatruangkamala K and Sinapiromsaran K. Optimizing Architectural Design via Mixed Integer Programming. Proceeding in CAADFutures 2005;11:175-184.

Keatruangkamala K and Sinapiromsaran, K. Heuristic cut for identifying the solution of the architectural layout design optimization. Proceeding in the Second Graduate Congress of Mathematics and Physical Science; 2006.

Koide T and Wakabayashi S. A timing-driven floorplanning algorithm with the Elmore delay model for building block layout. Integration, the VLSI journal 1999;27:57-76.

Levin PH. Use of graphs to decide the optimum layout of building. Architect 1964;14:809-815.

Li SP, Frazer JH and Tang MX. A constraint based generative system for floor layouts. 2000;10:441-450.

Liggett RS and Mitchell WJ. Optimal space planning in practice. Computer Aided Design 1981;13(5):277–288.

Liggett RS. Designer-automated algorithm partnership: an interactivegraphic approach to facility layout. Evaluating and predicting design performance, New York: Wiley Interscience 1992;101–123.

Linderoth and Savelsbergh MWP. A computational study of search strategies for mixed integer programming. INFORMS J. on Computing 1999;11:173-187.

Lobo F. The Parameter-Less Genetic Algorithm: Rational and Automated Parameter Selection for Simplified Genetic Algorithm Operation. Unpublished doctoral dissertation, Universidade de Lisboa; 2000.

Medjdoub B and Yannou B. Topological enumeration heuristics in constraint-based space layout planning. AI in Design'98, 1998;271–290.

Medjdoub B and Yannou B. Seperating topology and geometry in space planning. Computer-Aided Design. 2000; 32:39-61.

Michalek J and Papalambros PY. Interactive layout design optimization. Engineering Optimization 2002;34(5):461-484.

Michalek, J, Choudhary R and Papalambros PY. Architectural layout design optimization. Engineering Optimization 2002; 34(5):485–501.

Mitchell WJ, Steadman JP and Liggett RS. Synthesis and optimization of small rectangular floor plans. Environment and Planning B 1976; 3(1):37–70.

O'Sullivan B. Constraint-aided conceptual design. PhD thesis, Dept of Computer Science, University College Cork, Ireland; 1999.

Pefferkorn GE. A Heuristic problem solving design system for equipment or furniture layouts. Communications of the ACM 1975.

Pelikan M., Goldberg D. E. and Cant u-Paz E. Bayesian Optimization Algorithm, Population Sizing and Convergence. Ilinois Genetic Algorithms Laboratory; 2001.

Romualdas B and Ina P. Optimization of architectural layout by the improved genetic algorithm. Journal of Civil Engineering and Management 2005;11(1)13-21.

Schwarz A, et al. On the use of the automated building design system. Computer Aided Design 1994;26:747–762.

Schwarz A, Berry DM and Shaviv E. Representing and solving the automated building design problem. Computer-Aided Design 1994;26(9):689-698.

Scott A and Donald H. Making Design Come Alive: Using Physically Based Modelling Techniques in Space Layout Planning. CAADFutures 1999:245-262.

Simon HA. The structure of ill-structured problems. Artificial Intelligence 1973;4: 181-201.

Steadman P. Graph-theoretic representation of architectural arrangement. In The Architecture of Form. L. London, New York, Melbourne: Cambridge Univ Press 1976;94-115.

Tsang K S, Man K F and Kwong S. Genetic Algorithms: Concepts and Applications. IEEE Transaction on Industrial Electronic, 1996.

Willoughby T, Paterson W and Drummond G. Computer aided architectural planning. Operational Research Quarterly 1970;21:91-98.

Yin S, Cagan J. An extended pattern search algorithm for three-dimensional component layout. Transactions of the ASME 1997;122:102-108.

Appendix

Appendix A

Practical study

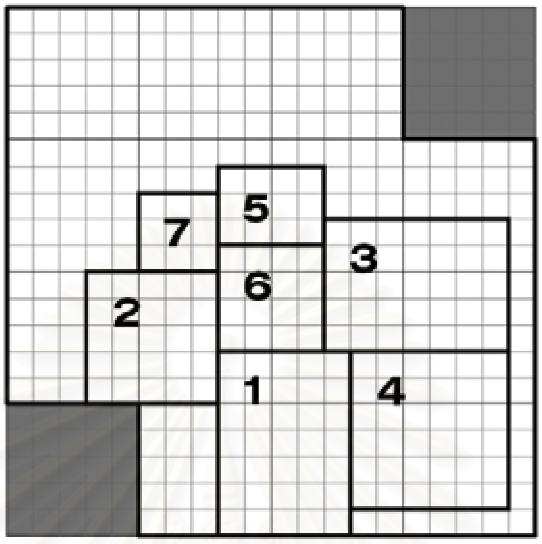
To verify the robustness of algorithm, we experiment based on the studying of two stories house. This study, is solved using the fixed position constraint, fixed border constraint and unoccupied unit constraints for a non-circular AL-MIP+ model. This study allows us to construct non-rectangular boundary shape which is motivated by the staircase area. To exhibit the flexibility of these three constraints, a realistic two stories house allocated on asymmetric boundary, has been solved using the non-circular AL-MIP+ model. The initial specification of the requirements is shown in the following information.

Two stories house study: room specifications

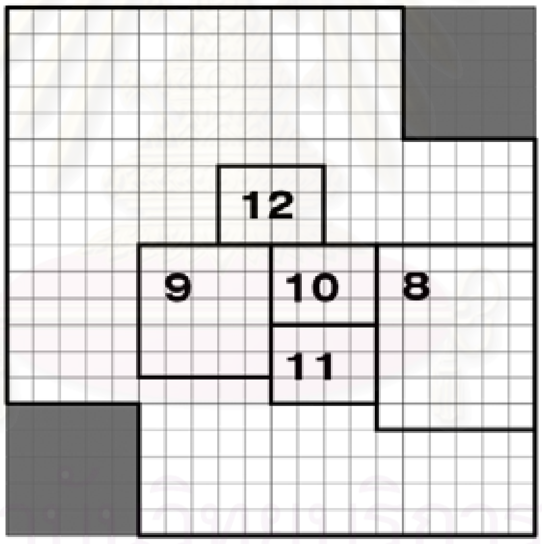
No.	Room	Width (m.)		Height (m.)		Connect
		min	max	min	max	
	1st Floor					
1.	Garage x 2 cars	5	6	5	7	2, 4, South
2.	Living Rm.	5	8	5	8	1, 6
3.	Dining Rm.	5	7	5	7	4, 6
4.	Kitchen	5	6	5	7	1, 3
5.	Staircase	4	4	3	3	6
6.	Hall 1	4	6	4	6	2, 3, 5, 7
7.	Bath	3	4	3	4	6
	2nd Floor					
8.	Master bedroom	6	7	6	7	10, 11, East
9.	Bedroom 2	5	7	5	7	10, 11
10.	Hall 2	3	5	3	5	8, 9, 11, 12
11.	Bath	3	4	3	4	8, 9, 10
12.	Staircase	4	4	3	3	10

Remark: unit scale in meter.

Based on our non-circular AL-MIP+ model, the total computational time of these two stories house are 54.468 seconds. The number of iterations and computational time of the first floor are 179282 and 41.625 seconds while the iterations and computational time of the second floor are 48369 and 12.843 seconds, respectively. The optimal layout design is shown in the figure A.1.



(a)



(b)

Figure A.1: The realistic of two stories house solved by the non-circular AL-MIP cooperate with three adjustable constraints and the gray region presents unoccupied unit spaces
(a) the computational time of 1st floor plan is 41.625 seconds and
(b) the computational time of 2nd floor plan is 12.843 seconds.

Appendix B

GAMS IDE model for AL-MIP+

This appendix section presents the GAMS IDE model for AL-MIP+ methodology.

```
$ontext
-----------------------------------------------------------------------------------------------------------------------
            GAMS IDE model for AL-MIP+
            Developed by Kamol Keatruangkamala
-----------------------------------------------------------------------------------------------------------------------

$Offtext

set         ROOM;
            ALIAS(ROOM,i);
            ALIAS(ROOM,j);
            ALIAS(ROOM,k);
set         LINK(i,j)
            CONNECT(i,j)
            CONNECT3(i,j,k)
            fixABOVE(i,j)
            fixLEFT(i,j)
            fixRIGHT(a,b)
            fixBOTTOM(a,b);

PARAMETERS
            DELTA
            Panel_Width
            Panel_Height
            Wmin(i)
            Wmax(i)
            Hmin(i)
            Hmax(i);


PARAMETER                   WeightLeftCorner(i);
PARAMETERS                  WeightMinDistance
                            WeightMaxArea;

VARIABLE        z;
POSITIVE VARIABLES
                            zx(i,j)
                            zy(i,j)
                            za(i);

POSITIVE VARIABLES
                            x(i)
                            y(i)
                            w(i)
                            h(i);
BINARY VARIABLES
                            p(i,j)
                            q(i,j)
                            r(i);

    w.lo(i) = Wmin(i);
    w.up(i) = Wmax(i);
    h.lo(i) = Hmin(i);
    h.up(i) = Hmax(i);

EQUATIONS
```

```
        obj_Min
        za_width(i)
        za_height(i)
        abs_plus_x(i,j)
        abs_minus_x(i,j)
        abs_plus_y(i,j)
        abs_minus_y(i,j)
        widthsize(i)
        heightsize(i)
        force_ij_left(i,j)
        force_ij_bottom(i,j)
        force_ij_right(i,j)
        force_ij_top(i,j)
        join_ij_left(i,j)
        join_ij_bottom(i,j)
        join_ij_right(i,j)
        join_ij_top(i,j)
        overlap_Up(i,j)
        overlap_Down(i,j)
        overlap_Left(i,j)
        overlap_Right(i,j)
        not10_a(i,j,k)
        not11_a(i,j,k)
        not00_a(i,j,k)
        not01_a(i,j,k)
        fixedABOVE(i,j)
        fixedLEFT(i,j)
        fixedRIGHT(a,b)
        fixedBOTTOM(a,b);

        obj_Min..                z        =e=      sum(i, WeightLeftCorner(i)*(x(i)+y(i)))
                                                   + WeightMinDistance*sum(LINK(i,j), zx(i,j) + zy(i,j))
                                                   - WeightMaxArea*sum(i, za(i));
        za_width(i)..                              za(i)  =l=  w(i);
        za_height(i)..                             za(i)  =l=  h(i);
        abs_plus_x(LINK(i,j))..                    x(i) - x(j) =l= zx(i,j);
        abs_minus_x(LINK(i,j))..                   x(j) - x(i) =l= zx(i,j);
        abs_plus_y(LINK(i,j))..                    y(i) - y(j) =l= zy(i,j);
        abs_minus_y(LINK(i,j))..                   y(j) - y(i) =l= zy(i,j);
        widthsize(i)..                             x(i) + w(i) =l= Panel_Width;
        heightsize(i)..                            y(i) + h(i) =l= Panel_Height;
        force_ij_left(LINK(i,j))..                 x(i) + w(i) =l= x(j) + Panel_Width*(p(i,j) +q(i,j));
        force_ij_bottom(LINK(i,j))..               y(j) + h(j) =l= y(i) + Panel_Height*(1 +p(i,j) -q(i,j));
        force_ij_right(LINK(i,j))..                x(j) + w(j) =l= x(i) + Panel_Width*(1 -p(i,j) +q(i,j));
        force_ij_top(LINK(i,j))..                  y(i) + h(i) =l= y(j) + Panel_Height*(2 -p(i,j) -q(i,j));
        join_ij_left(CONNECT(i,j))..               x(i) + w(i) =g= x(j) - Panel_Width*(p(i,j) -q(i,j));
        join_ij_bottom(CONNECT(i,j))..             y(j) + h(j) =g= y(i) - Panel_Height*(1 +p(i,j) -q(i,j));
        join_ij_right(CONNECT(i,j))..              x(j) + w(j) =g= x(i) - Panel_Width*(1 -p(i,j) +q(i,j));
        join_ij_top(CONNECT(i,j))..                y(i) + h(i) =g= y(j) - Panel_Height*(2 -p(i,j) -q(i,j));
        overlap_Up(CONNECT(i,j))..                 0 =g= y(i) + DELTA - y(j) - h(j) - Panel_Height*(q(i,j));
        overlap_Down(CONNECT(i,j))..               0 =g= y(j) + DELTA - y(i) - h(i) - Panel_Height*(q(i,j));
        overlap_Left(CONNECT(i,j))..               0 =g= x(i) + DELTA - x(j) - w(j) - Panel_Width*(1 -q(i,j));
        overlap_Right(CONNECT(i,j))..              0 =g= x(j) + DELTA - x(i) - w(i) - Panel_Width*(1 -q(i,j));
        not10_a(CONNECT3(i,j,k))..                 p(i,k) -q(i,k)    =l= Panel_width*(p(i,j) +q(i,j));
        not11_a(CONNECT3(i,j,k))..                 p(i,k) +q(i,k)-1  =l= Panel_Width*(1 +p(i,j) -q(i,j));
        not00_a(CONNECT3(i,j,k))..                 1 -p(i,k) -q(i,k) =l= Panel_width*(1 -p(i,j) +q(i,j));
        not01_a(CONNECT3(i,j,k))..                 q(i,k) -p(i,k)    =l= Panel_Width*(2 -p(i,j) -q(i,j));
        fixedABOVE(fixABOVE(i,j))..                y(i) =l= y(j);
        fixedLEFT(fixLEFT(i,j))..                  x(i) =l= x(j);
        fixedRight(fixRIGHT(a,b))..                x(a) + w(a) =g= x(b) + w(b);
        fixedBOTTOM(fixBOTTOM(a,b))..              y(a) + h(a) =g= y(b) + h(b);

MODEL          ALDO   / ALL /;
SOLVE          ALDO   USING MIP MINIMIZING z;
DISPLAY  x.l, y.l, w.l, h.l, p.l, q.l, Wmin, Wmax, Hmin, Hmax;
```

Biography

Name: Kamol Keatruangkamala

Email: kamolkeat@hotmail.com

Educations

- 2007 Ph.d.,candidate (Computer Science), Faculty of Science, Chulalongkorn University, Bangkok, Thailand
- 2003 Master of Economic. (M.Econ), Faculty of Economic, National Institute of Development Administration, Bangkok, Thailand
- 2000 Master of Archtecture. (M.Arch), Faculty of Architecture, Chulalongkorn University, Bangkok, Thailand
- 1997 Bachelor of Architecture. (B.Arch), Faculty of Architecture, King Mongkut's Institute of Technology Lardkrabang, Bangkok, Thailand

Awards and Recognitions

- 2003 International conference committee of the 8th CAADRIA'2003 (CAADRIA: Computer Aided-Architectural Design Research in Asia)
- 2002 Member in council of Thai architects
- 2001 Program committee in Computer Aided-Architectural Design, Faculty of architecture, Rangsit University, Thailand
- 1992 Best military student Award from Territorial Defend (AE 1992).

Publications:

- Keatruangkamala K, Sinapiromsaran K. Optimizing Architectural Design via Mixed Integer Programming. Proceeding in CAADFutures 2005;11:175-184.
- Keatruangkamala K, Sinapiromsaran, K. Heuristic cut for identifying the solution of the architectural layout design optimization. Proceeding in the Second Graduate Congress of Mathematics and Physical Science; 2006.
- Keatruangkamala K, Sinapiromsaran, K. Strong Valid Inequality Constraints for Architectural Layout Design. Proceeding in the Second Graduate Congress of Mathematics and Physical Science; 2007.