Component Grouping
for
Printed Circuit Board Assembly

BY

SZE MAN TING

A Thesis Submitted to The Hong Kong Polytechnic University for the

Degree of Master of Philosophy

DEPARTMENT OF MANUFACTURING ENGINEERING

THE HONG KONG POLYTECHNIC UNIVERSITY

1999

Abstract of thesis entitled

"Component Grouping for Printed Circuit Board Assembly"

submitted by Sze Man Ting

for the degree of Master of Philosophy

at The Hong Kong Polytechnic University in September 1998

ABSTRACT

In a Printed Circuit Board (PCB) assembly company, an assembly line is designed to have several placement machines (may or may not be the same types) to deal with various types of components since the production volume is large and the placement operation is a bottleneck of the line. This research project studies the component grouping problem in order to assign components to proper placement machines so that the optimal cycle time and throughput of the line can be obtained.

In addition to an extensive literature review on PCB assembly being carried out, the component grouping problem was identified in the project. Several mathematical models were formulated to cope with different objectives for both single-sided and double-sided board assembly cases. Furthermore, a new linear programming algorithm was proposed to solve the component grouping problem with the objective of minimizing cycle time, based on the revised simplex method. Moreover, the Genetic Algorithm (GA) technique was applied to the component grouping problem, and the results from both the linear programming method and the GA technique were compared with the solution

from the integer programming model, obtained by the commercial package, CPLEX. The conclusion is that the result generated by the linear programming method with rounding off is acceptable in engineering, whereas the solution improvement rate of the genetic algorithm is very good at the beginning, but it decreases quickly in an exponential rate as the solution gets closer to its optimal.

ACKNOWLEDGMENTS

First of all, the author would like to express her acknowledgements to the Department of Manufacturing Engineering, The Hong Kong Polytechnic University in providing funding to this research.

The author would like to take this opportunity to express her sincere gratitude to her chief supervisor, Dr. P. Ji. Without his strong guidance, invaluable advice and continuous encouragement, this thesis is impossible to be completed. The author would also like to thank her co-supervisor, Prof. W. B. Lee, for his suggestions and comments. Thanks also go to Madam C. F. Yeung, for her on-going assistance and advice.

The author would also like to thank her colleagues in the Department of Manufacturing Engineering for their thoughtful discussions and suggestions. Furthermore, dedicated thanks are extended to Curt Cheng who has been providing moral support and wholehearted encouragement.

Last, but not least, special thanks must go to her mother, for her continuous encouragement and patience during the past two years.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter One

INTRODUCTION

1.1 PCB Assembly

1.1.1 PCB assembly process

Printed circuit board (PCB) assembly is a process-oriented industry, with a large number of components to be mounted on a board. Two main electronic assembly technologies have been developed in PCB, that is, Plated-Through Hole (PTH), and Surface Mount Technology (SMT). A board normally uses a combination of both technologies. A typical PCB assembly line has six operations: (1) Applying solder paste: solder paste is typically applied in a screen print fashion. To avoid solder bridging during reflow, solder paste must be deposited to the board accurately. (2) Component placement: components are placed to boards by automatic pick and place or turret machines. (3) Solder reflow: solder paste must be heated to the melting point to form the connection between Surface Mounted Devices (SMD) and the circuit. (4) Cleaning: cleaning is a necessary process after reflow. (5) Testing: in-circuit test and function-level test are required to verify the package configurations, to find opens or shorts on the board, and to check if the board operates according to the customer's design specification. Finally, (6) Inspection: the board should be inspected and verified to ensure to have a superior quality. Fig. 1.1 shows a typical SMT line in Tescon Co., Ltd. [Muk92].

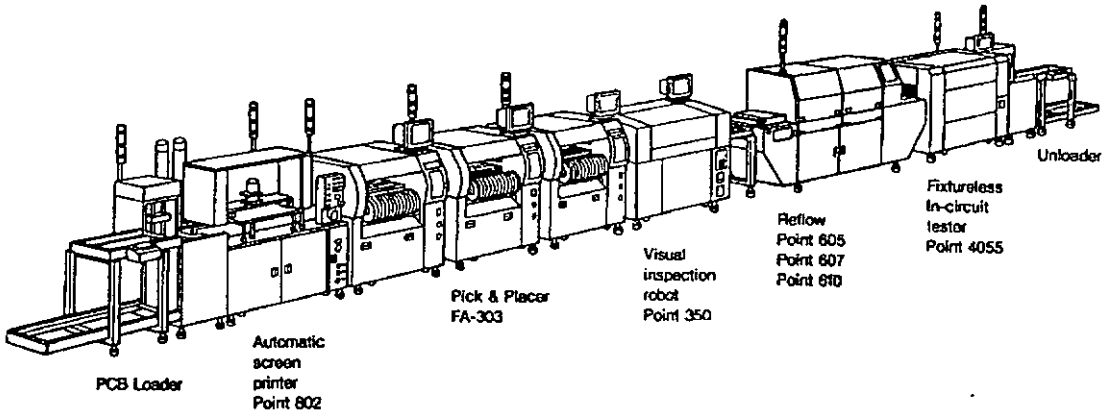


Figure 1.1: A typical PCB assembly line

Some PCBs need surface mounting on both sides of the boards and others need surface mount on one side only. An assembly line will be designed to have one placement station if one side mounting is required. For double side mounting, an additional placement station is needed, so there are two placement stations in the line. In the first station, solder paste is applied to the top side, then components are onserted in a series of placement machines, followed by the soldering process in an infrared (IR) oven, so the component onsertion for the top side is finished. The board is then inverted and passes through the second station, under the same working process, components are onserted to the bottom side.

Among the assembly operations, the interest here is the component placement (pick and place) process. The optimization of the component placement process can, no doubt, increase a company's competitiveness because

- The automatic placement (pick and place) machines are cost intensive, and frequently are a bottleneck due to their limited capacities. Consequently, they should be run in a high utilization.
- Due to the large production volumes, a minor reduction in the cycle time will save a significant production time and cost. For example, to produce 50,000

boards, a reduction of 6 seconds in the cycle time will save 5,000 minutes, that is, over 80 working hours.

- Consequently, the production cost is reduced by the increment of the production output from the reduction of the cycle time.

1.1.2 Production planning in PCB assembly

The production planning for a large PCB assembly company is different from that for a small firm. In such a company, the production volume is very large for a board, several assembly lines have to be installed. A line is equipped with several placement machines, which may or may not be of the same type. Appendix I is a line layout for an electronic company, which has five production lines, and the line configurations are presented at Appendix II.

After receiving the customer orders, all the PCB assembly manufacturers would like to complete the orders with minimum time and cost. Therefore, the optimizations on both product arrangement (which line assembles which board) and line performance (minimum cycle time or delay) are necessary. This production planning process can be summarized in Fig. 1.2.

The production planning system operates in two successive tasks. Firstly, the optimal cycle times for all assembly lines should be determined. The cycle time depends on how the components are grouped to machines. After that, the result obtained from Task 1 is an input to Task 2 to decide which product to be assembled on which line and its quantity to be assembled on the line. Actually, products/boards are expected to be allocated to the most economical line, so it can be completed before the due date. This project will study the first task,

whereas the second task has been investigated by Ji, *et al.* [Jip94], who formulated the product arrangement problem as a generalized transportation model. A special algorithm was presented to obtain the optimal solution from the dual of the model [Jip94].

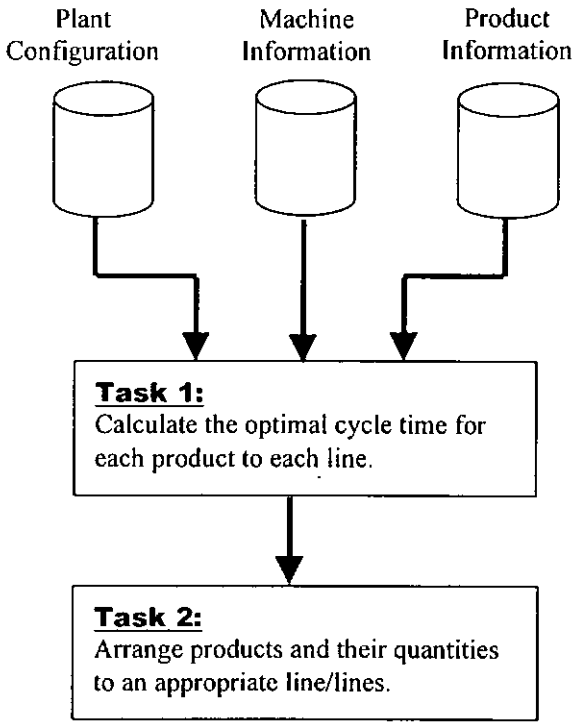


Figure 1.2: Architecture of the production planning process

In order to optimize the production planning system, the following input data should be available:

- Line configuration, including the number of assembly lines in the company, machine type on a line, and also the available production time for each line.
- Machine information, that is, the types of machines used and the relevant placement times for different component types.
- Product information, including the component type and the component number on each product, the product demands and the unit production cost for each product to be assembled on a line.

1.1.3 Assembly line configuration

The approach of assembly line design is di0fferent from one company to others, depending on the production environments. For low volume and high mix manufacturing, it is common to have one or few placement machines on an assembly line. However, as it is mentioned before, a large PCB assembly company normally has several production lines to cope with different kinds of boards. Fig. 1.3 shows some possible configurations for assembly lines. Lines 1 and 2 are configured for surface mounting on single side of the boards while Line 3 is designed for surface mounting on both sides, so there are two placement stations in the line. In each of the component placement stations, the placement machines may not be of the same type. The design of different machines in a line enhances the throughput and productivity of the line due to its unique configuration.

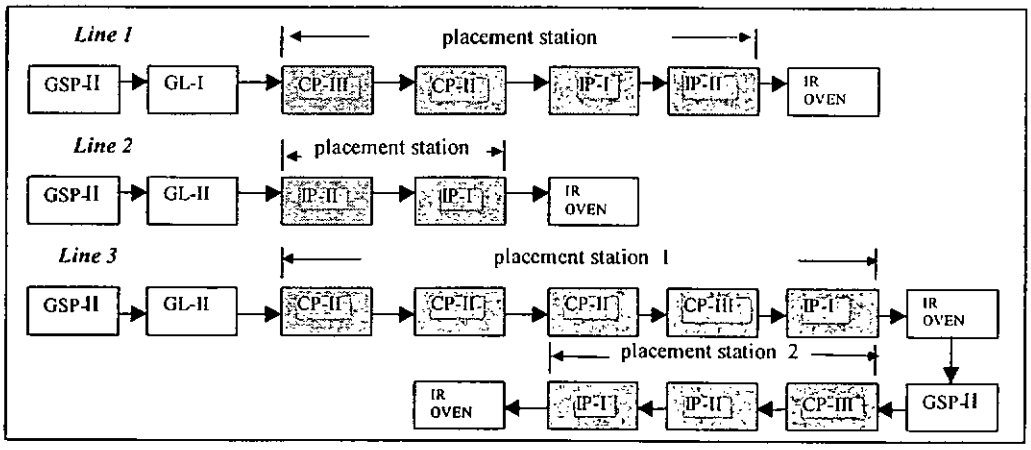


Figure 1.3: Typical PCB assembly line configuration

A board normally contains a number of component types and the component placement time varies for different machines and component types. For example, CP-II is a type of placement machine that assembles chip components very fast but it takes an extremely long time on integrated circuit

(IC) components. On the other hand, IP-II (another type of placement machine) has a better performance on IC components. Appendix III lists the placement times for different kinds of machines.

Why does a line have different types of machines? Suppose a PCB assembly company has three production lines with the same number of machines, for example, two placement machines on each line. Line (a) has two CP-II machines, line (b) has two IP-II machines, whereas line (c) has one CP-II plus one IP-II machine. If these three lines are assigned to produce a PCB with two component types only: 100 resistors and 50 PLCCs, and their unit component placement times are given at Table 1.1, then there is a question concerned about which line has the best performance on this board.

Component type	Quantities per board	Machine type	
		CP-II	IP-II
Resistor	100	0.3 sec/comp.	0.7 sec/comp.
PLCC	50	3.5 sec/comp.	1.7 sec/comp.

Table 1.1: Placement times for resistor, PLCC on CP-II and IP-II

In order to balance the workload among the machines, components will be evenly assigned, each CP-II at line (a) or each IP-II at line (b) processes 50 resistors and 25 PLCCs. For line (c), CP-II processes all resistors plus 10 PLCCs and the rest PLCCs will be processed at IP-II. The cycle times under these groupings can be easily calculated, shown at Table 1.2. Certainly, line (c) has the highest production rate. Compared with line (b), 9.5 seconds (77.5 – 68) can be reduced per board, and thus, 132 working hours will be saved if the order quantity is 50,000 (9.5 seconds × 50,000 = 475,000 minutes).

Placement machines	Cycle time	Remarks:
line (a): 2 CP-II	102.5 sec/board	$(100 \times 0.3 + 50 \times 3.5)/2$
line (b): 2 IP-II	77.5 sec/board	$(100 \times 0.7 + 50 \times 1.7)/2$
line (c): 1 CP-II + 1 IP-II	68 sec/board	$\max\{1.7 \times 40, 100 \times 0.3 + 10 \times 3.5\}$; 40 PLCCs $\Rightarrow$ IP-II 100 resistors + 10 PLCCs $\Rightarrow$ CP-II

Table 1.2: Comparison of three different line configurations

Obviously, if a board is assigned to a production line with only one type of placement machine, the production efficiency is very low and a lot of time will be wasted, especially when the production volume is high.

1.1.4 Component grouping

Certainly, a real board contains a number of different types of components, so the component assignment (or called component grouping) is not so simple as the previous example. In the real situation, a board is characterized by hundreds of electronic components in different shapes, sizes and patterns, different placement machines are designed to meet the component requirements. A decision regarding the component grouping to individual machines must be made. Components will be assigned in a way such that each component is assembled by only one machine. When the board passes through all the machines, all components have to be onserted.

Fig. 1.4 shows a board with five components (C1 to C5) to be assembled on a production line with three placement machines (CP-I, CP-II, and IP-I). Machine CP-I will firstly assemble some components, followed by machine CP-

II and finally machine IP-I. The goal of this grouping to be achieved is to ensure that the assembly line is working under a good performance.

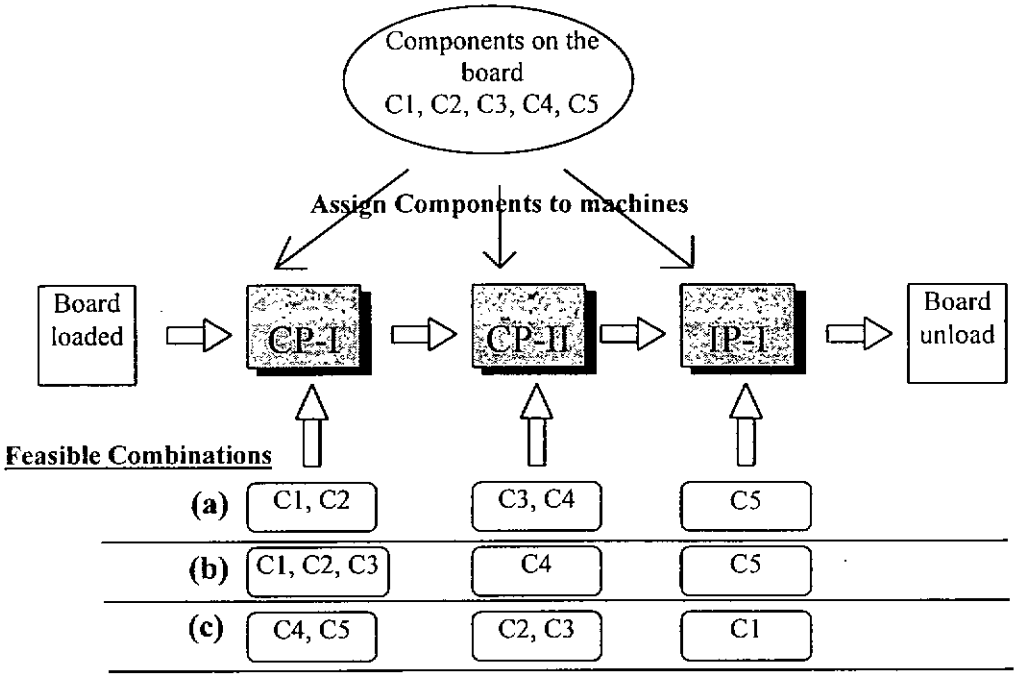


Figure 1.4: A simple example of a board to be assembled on a line

Of course, the line performance depends on how the components are grouped to machines. Fig. 1.4 also shows three grouping solutions. Production planners can solve this problem manually and their experience can guide them to assign components to machines. If a grouping result is in low efficiency, they may rearrange the components in the hope of achieving better results. The ideal way to solve this problem is to determine all feasible combinations of grouping, and then select a combination with the best result. Obviously, it is not practical to do so due to the time required in generating all feasible combinations. There are 243 (3^5) possible grouping combinations for the simple case in Fig. 1.4 with only 5 components and 3 machines. A real board contains hundreds of components, so it is impossible to check all combinations. The best grouping method is to formulate the problem in a mathematical model so that an optimal or

near optimal grouping can be obtained by a mathematical or heuristic approach. And this is what this project is going to do. Few researchers have studied this component grouping problem in PCB assembly, particularly when the placement times for the same type of components are different for non-identical machines. A detailed review on this problem and other related PCB assembly problems is provided in Chapter 2.

1.2 Objectives

Certainly, the objective of this project is to increase the productivity or the throughput of an assembly line by grouping proper components to suitable machines. In order to achieve this objective, the component grouping problem will be formulated into a mathematical model. Furthermore, some approaches will be investigated, so a mathematical procedure and a heuristic algorithm will be proposed. The results will be compared with an exact optimal solution approach.

1.3 Research Scope

The production planning for PCB assembly contains a lot of problems, and each of the problem has its unique objective. For example, board grouping aims to reduce the setup time, whereas component grouping will reduce the assembly time. Since the production volume is very large when the size of a PCB assembly company is big, the reduction of assembly time is far more important than the set up time. Therefore, the grouping of components to a series of machines is studied in order to obtain the best performance of a line.

A literature review is conducted in Chapter 2. This review focuses on three areas: 1) the production planning problems in PCB assembly, 2) related optimization problems in Operations Research, and 3) the mathematical and heuristic techniques developed for those optimization problems. In Chapter 3 several mathematical models are formulated and studied for component grouping in single-sided and double-sided PCB assembly. Models with objectives of minimizing cycle time and balance delay are also discussed. In order to develop a good algorithm to solve the models developed in Chapter 3, Chapter 4 compares the models with some special problems, such as the assignment problem, the line balancing problem. Based on the idea of the revised simplex method, a new algorithm is also proposed in this chapter to cope with the mathematical model in Chapter 3. Chapter 5 provides a heuristic solution procedure to the same problem. This solution procedure is revised from the idea of Genetic Algorithms. The performance of this heuristic approach is also discussed in this chapter. The last chapter concludes this project and also points out some possible work in future.

Chapter Two

LITERATURE REVIEW

2.1 Introduction

This chapter is organized in three sections. Section 2.2 provides a general review on PCB assembly. This section will survey what problems have been identified on PCB assembly by previous researchers, and how they attempted to solve these problems. It is noted that the production planning on PCB assembly actually can be broken down into different kinds of optimization problems, which belong to certain kinds of problems in Operations Research. As there are so many types of optimization problems in PCB assembly, only those related to the component grouping problem will be reviewed in Section 2.3. Finally, the last section will briefly survey the optimization techniques, on both mathematical and heuristic.

2.2 Review on PCB Assembly Problems

The needs of the smaller, denser boards with greater functionality and reliability drive the assembly technologies from manual operations to fully automated ones. The expensive automated assembly equipment forces manufacturers to search for a planning system that maximizes output. Feldmann, *et al.* [Fel94a] pointed out that in order to improve the efficiency of a PCB assembly line, a number of related problems have to be solved:

1. Which order has to be assembled on which line?
2. Which order sequence guarantees the fastest throughput time?

3. When does the setup of components have to be changed?
4. Which component has to be assembled on which placement machine?
5. Where is the best feeder location for every single feeder?
6. In which sequence has the components to be placed?
7. Which boundary conditions have to be observed?

The purposes of these questions are to optimize the production rate either by reducing setup, changeover time or the placement cycle time. The hierarchical relationship between these questions was described at Fig. 2.1 [Amm97]. In the planning system, the top level is to group machines and PCBs, associated with the assembly line design and the assignment of PCB families to machine groups. After the board families are assigned to machine groups or assembly lines, the next level is to decide the sequence of PCBs for individual family, and to group and then assign components to machines (component grouping) when a machine group contains multiple non-identical machines. The arrangement and sequencing problem is at the lowest level, which includes the arrangement of components to feeders, and the sequencing of component placement in a machine.

Each of these problems is rather complex, so it is very difficult to solve them all simultaneously. Another factor is the production volume. A great deal of literatures specified PCB assembly in a low-volume, high-mix manufacturing [Sad93, Mai93, Sau94], then the researchers addressed the reduction of setup and change-over time as the most important objectives. However, as the production volume increases and the board variety decreases, how to reduce the cycle time for individual board becomes more and more important.

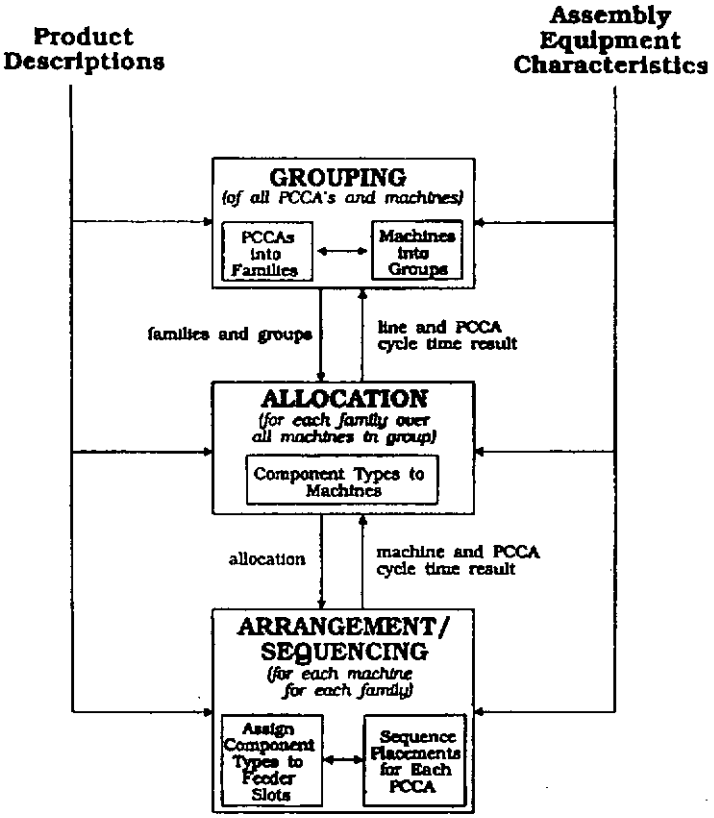


Figure 2.1: The relationship among PCB assembly problems [Amm97]

2.2.1 Machine and PCB grouping

In order to obtain a good production rate (throughput), the first step is to design an efficient assembly line. Dessouky, *et al.* [Des95] made an effort on this problem and presented a new approach for grouping machines into workstations so that the throughput was maximized while the work-in-process (WIP) inventory was minimized. Santos, *et al.* studied how to integrate different software packages in order to find the best PCB assembly line configuration [San95]. Since sequencers and feeders are also the resources required in PCB assembly, Randhawa, *et al.* presented an integer programming model to determine the minimum number of sequencers and their sizes for PCB assembly to meet the

production requirements with the objective of minimizing the sequencing cost [Ran85].

Kung, *et al.* [Kun91] implemented a simulation model by using WITNESS for an existing PCB assembly line, and evaluated the impact of some parameters, such as the number of different PCB types, the sequencing rules used, the batch size and the number of kanbans by the just-in-time management approach. One software tool, called Automated Generating and Simulation of Insertion Sequences (AGASIS), was presented by Feldmann [Fel93, Fel94a, Fel94b, Fel94c] for optimization, simulation and NC-programming of the PCB assembly.

Many researches addressed the PCB grouping problem. Maimon and Shtub defined the PCB grouping problem as a Mixed Integer Program (MIP) and presented a heuristic solution [Mai91]. The concept of the heuristic is to adopt a control variable (threshold) R to minimize the number of the board split. The number of loading and unloading for each PCB could be minimized, and so does the setup time. However, they assumed the setup time is independent of component type or PCB type, which may not always be true in reality. In 1992 they proposed another grouping approach that is based on cluster analysis to measure the similarity between PCBs [Sht92]. The PCB-component relationship is represented by a PCB-component incidence matrix with the objective of minimizing total set-up time of PCBs and components. Luzzatto, *et al.* designed a procedure to group PCBs into a number of cells with the objectives of minimizing the setup time and maximizing the workload balance for a number of inserting machines at the assembly stage [Luz93]. Cunningham, *et al.* presented

a heuristic system, written in LISP, for job scheduling in PCB assembly [Cun86]. Jobs are firstly divided into groups, then the branch and bound searching technique should be used to find a good schedule for each group.

Davis, *et al.* applied a "greedy board" heuristic in assigning board families to manufacturing cells and found that after the use of the heuristic, the line efficiency was increased [Dav90]. Garetti, *et al.* designed a scheduling system for SMT electronic boards assembly [Gar96]. The goal of the system is to minimize the makespan (the time a board spent on the line) by reducing setup times and idle times of the machine. The scheduling system has two stages. Stage 1 is the generation of board mix (group or family) and Stage 2 is the sequencing of each board mix. In order to reduce setup times, board mixes should be formed so that they could be produced without any need of setup. Once board mixes are generated, either the machines will have less idle time or the number of setups will be reduced. After that, the sequence of individual board in the board mix should be determined, so that either the maximum operating time of a machine or the system setup time would be minimized.

Askin, *et al.* studied both the PCB grouping and components grouping problems in an open-shop assembly cell [Ask94]. They studied the situation where there was a set of different board types to be assembled on an assembly system with multiple, identical automatic placement machines. The boards were grouped into production families. For a given PCB family the assignment of components to the machines and the scheduling of the PCBs were formulated into a mathematical model. According to three different requirements, e.g.,

work-load balancing, the shortest total processing time, and board similarity, three heuristics were presented and compared.

Peter and Subramanian considered the production of PCB assembly in a system with multiple identical placement machines operating in parallel [Pet96]. Four strategies (unique, sequence dependent, minimum set up, and trade-off set up) were developed and tested. These four strategies focused on different objectives. The unique setup strategy minimizes the individual board processing time, sequencing dependent and tradeoff strategies minimize the makespan, while the last strategy, the minimum set up attempts to minimize the changeover time. They concluded that the best methodology for solving the operational planning problem is to try both the sequence dependent and tradeoff setup procedures and to choose the better one between the two.

2.2.2 PCB sequencing and components allocation

Ignall and Schrage studied the general sequencing problem for a set of jobs to be processed on a number of machines [Ign65]. They found an optimal sequence by the branch and bound integer programming technique with the objective of minimizing the makespan. Later, McCormick, *et al.* proposed a heuristic approach which is based on an equivalent maximum flow and the critical path techniques to solve the problem [Mcc89]. Only a few researchers have addressed the problem of PCB sequencing. Sadiq, *et al.* developed a rule-based approach to find a near optimal solution for sequencing a group of PCBs on a single placement machine [Sad93]. By using the database with a historical record about component usage, boards are firstly sequenced on the machine and

then feeders are arranged to these boards to minimize the number of component changes. The sequence with the minimum number of changes is selected as the starting point for the reassignment process. Once this production sequence is selected, the algorithm will check if the reassignment of components inside a contiguous group would result in a reduction of total production time. If that is the case, it suggests the sequence and also which job may require reassignment of components.

Applying Group Technology (GT) to PCB assembly gives an advantage of saving set up time and maximizing machine utilization, as there is no component setup required when changing from one PCB type to another and the boards share common components. Carmon, *et al.* presented a group set up (GSU) algorithm to reduce the setup time for PCB assembly [Car89]. With their proposed GSU, boards are divided into groups, each of which is assembled in two stages. The first stage is for the common-component assembly and the second stage is to assemble the residual components on each board. The machine is firstly set up for all the common components in the group and assembles these common components to all boards. Then setting up for the residual components on each product should be made. The traditional and GSU production methods were compared, the result showed that GSU performs better in terms of both labor time and throughput. Maimon, *et al.* extended the study by comparing three scheduling methods – traditional, sequence dependent method (SDS) and GSU [Mai93]. The schematic presentation of GSU and SDS is shown in Fig. 2.2. The marked spaces represent the set up time saved. Under certain assumptions, GSU performs better than SDS in terms of throughput, whereas SDS performs

better than GSU in terms of the average WIP level, while the performance of the traditional method is the worst.

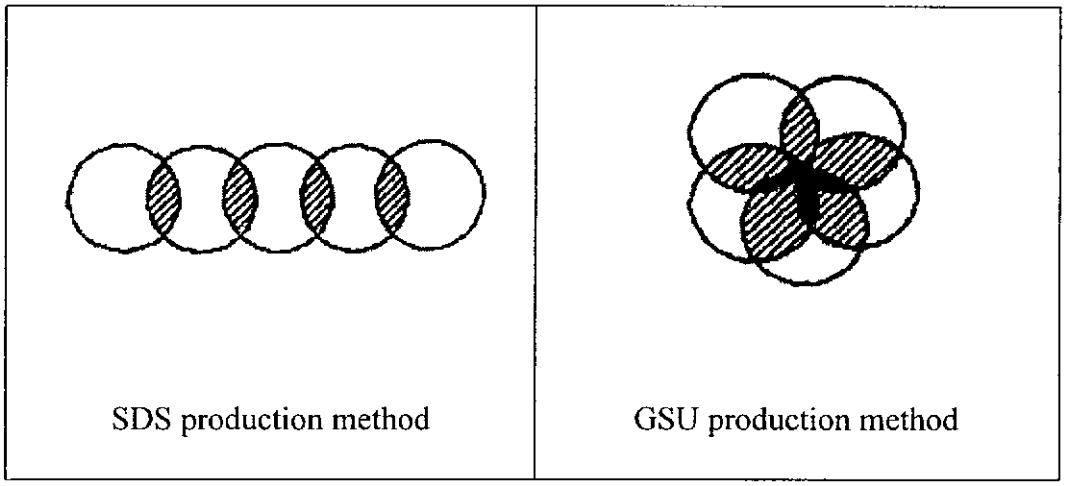


Figure 2.2: A schematic presentation for SDS and GSU scheduling methods

Ben-Arieh and Dror studied the problem of assigning components to a number of insertion (placement) machines, so that the minimum cycle time can be obtained [Ben90], assuming that there is no setup time, inserting time is one unit for each component, and regardless of the circuit board type and components' location on the board. Two algorithms for the same problem but different assignment constraints were presented. They showed that the production rate depends on how the components are assigned to machines but does not depend on the sequence of the circuit boards. Günther, *et al.* concerned the allocation of components among various identical placement machines for component kitting in semi-automated PCB assembly [Gün96]. The production time and component magazine capacity were taken into account. The problem was formulated as a mixed integer linear optimization model. Besides, they also presented a rule-based heuristic with the objective of minimizing the number of assembly station taken into operations. Watkins, *et al.* presented a heuristic to select and move components from bottleneck to non-bottleneck machines, such

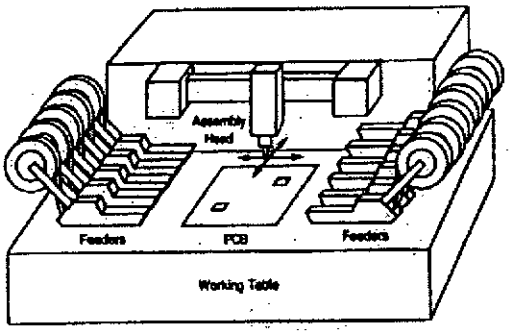
that the line was balanced and the component relocation costs were reduced [Wat95].

Ammons, *et al.* discussed the component allocation for an electronic assembly system with multiple, non-identical machines, such that the work load of PCB assembly of each machine for all boards can be balanced [Amm97]. The problem of allocating components to machines was formulated into a large scale Integer Linear Programming (ILP) model and two methods were presented. The first method involves in using a list-processing-based heuristic. This method assumes that every machine has the ability to place any component with identical processing times, but unfortunately this is not true as each placement machine has its own configuration. The second method is to use the mathematical programming technique. A computer software package, called MINTO, is introduced. MINTO is the software designed by using conventional Integer Linear Programming-based branch and bound method. Due to the large scale integer linear programming model, MINTO is not an effective software package to solve the problem.

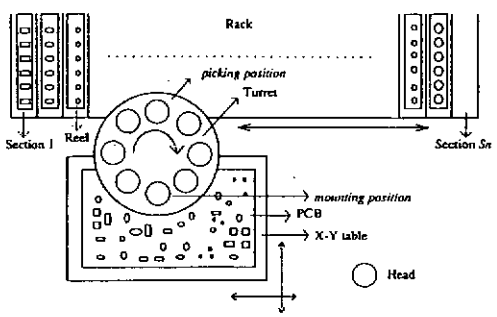
2.2.3 Feeder arrangement and component sequencing

To minimize the machine operation time to assemble a PCB, Operations Research techniques have helped to (1) determine the sequence of component placement and (2) arrange component types to feeder locations. These two problems are normally formulated as the Traveling Salesman Problem (TSP), which is well known as NP-hard and difficult to solve. For different machine types, the constructions of the machine heads and feeders are different (Fig. 2.3

shows two types of machines), thus the placement actions, as well as the algorithms for solving the components sequencing problem will not be the same.



A pick and pace machine [Won93]



A turret multi-head machine [Soh96]

Figure 2.3: The schematic diagram for two types of placement machines

The component sequencing problem for a pick and place (placement) machine was firstly modeled as the rural postman problem by Ball and Magazine [Bal88]. Under the constraint that the gripe movement is rectilinear, the placement process was treated as a directed network, and a closed path with the minimum cost would be found. A balance and connect heuristic was presented to solve this NP-hard problem. Later, Ji, *et al.* formulated the same problem as the linear assignment problem and studied the effect of arranging the components to a feeder [Jiz91, Jiz93]. They stated that the linear assignment problem can be solved efficiently by using an existing algorithm. It takes an advantage over the rural postman problem which has to be solved heuristically. In the PCB assembly planning process, with the consideration of two essential elements, that is, the construction of cost matrix and the merge operation, the assignment model becomes more flexible and an optimal placement sequence can be found. Chan and Mercier formulated the component insertion problem as a traveling salesman

problem [Cha89]. They considered the cost as a function of time spent or distance traveled or a combination of both, and pointed out that the traveling salesman approach should be more acceptable in the manufacturing environment because of faster computers and more efficient algorithms. Nelson, *et al.* compared the three heuristic methods to find a component sequence that minimizes the total cycle time [Nel95].

Leu, *et al.* developed a genetic algorithm for solving the component sequence and feeder assignment problems [Leu93]. To deal with different constructions of placement machines, three types of PCB assembly planning problems, i.e., the traveling salesperson problem, the pick and place problem, and the moving board with time delay problem, were studied.

Bard, *et al.* presented the component placement sequence in a turret machine with movable feeder as the traveling salesman problem [Bar94]. The problem was formulated as a quadratic integer programming by the use of a fixed nonnegative multiplier. They decomposed the problem into two sub-problems, solved them separately and then combined the results in order to use the branch and bound technique to obtain the optimality. Sohn and Park simplified the component sequencing problem at a machine by considering the multi-heads as one head, and presented a heuristic algorithm [Soh96]. The algorithm consists of two phases. Firstly, for a given assignment of the component reels to the rack, an initial sequence of components on the board is found. Next, the solution is improved by revising the mounting sequence repeatedly with the pair-wise exchange of reels.

Huang and Srihari applied an expert system in PROLOG to identify a 'near optimal' solution to find the feeder location and placement sequence for the SMD assembly [Hua93]. For the same problem, Mettala and Egbelu [Met89] presented a rule base approach to sequence robot moves, whereas Souza and Lijun [Sou95] introduced a knowledge-based Component Placement System (CPS) to solve the optimization problem on a multi-head concurrent operation PCB assembly machine.

2.3 Optimization Models

From the above literature review, it was noticed that optimization techniques are widely used in PCB assembly. In general, optimization is to find the best solution to a problem. If a model can be constructed to a problem and the model is one of the standard mathematical formulations, a solution is usually attainable by using available algorithms. Many mathematical models have been presented and studied, such as linear programming, dynamic programming, nonlinear programming, goal programming and so on. Due to their mathematical properties, different techniques have been developed and a short review on some techniques will be provided in the next section. The optimization problem in PCB assembly is to find a good or the best solution for a number of variables so that an objective can be minimized (e.g., cost, set up time, total assembly time, the number of set ups) or maximized (e.g., throughput, profit) under certain constraints, that is, limited resources (e.g., a number of sequencers or placement machines, available working hours). In the case of the component grouping problem to be studied in this project, it can be constructed in the types of min-

max programming, integer programming, and linear integer programming (these models will be formulated in Chapter 3), so this section will concentrate on these three models only.

2.3.1 Linear programming

A model is defined as a linear programming when the objective function, the constraint functions are all linear, and the variables are continuous. The general linear programming problem can be simply described as follows: Given a set of m linear inequalities or equations in n variables, that is, the constraints, the linear programming (LP) is to find non-negative values of these variables which will satisfy all the constraints and maximize or minimize some linear function of the variables. The linear programming has the general form [Bea88]:

$$\min\ (\max)\ \sum_{j=1}^{n} c_j x_j + c_0$$

Subject to:

$$\sum_{j=1}^{n} a_{ij} x_j (\leq, =, \geq) b_i \qquad i = 1, \ldots\ldots, m \qquad \text{(LP2-1)}$$

where c_0, c_j and a_{ij} are constants or known data while x_j are non-negative variables.

2.3.2 Min-max programming

If the objective function in LP2-1 is to minimize the maximum value of several functions $f(x)$, LP2-1 is no longer a linear programming due to the non-linear objective function. However, if $f(x)$ is piecewise linear and convex, it was

proved the problem can be transformed into an LP form [Mur83, Mur95a]. The general mathematical model for this min-max type programming is as follows:

$$\text{minimize } \{z(x) = \max (\sum_{j=1}^{n} c_j^1 x_j + c_0^1 , \sum_{j=1}^{n} c_j^2 x_j + c_0^2 , \ldots, \sum_{j=1}^{n} c_j^k x_j + c_0^k)\}$$

subject to the constraints in LP2-1.

To transform the above model into an LP, a new variable x_{n+1} and the following constraints should be introduced:

$$x_{n+1} - \sum_{j=1}^{n} c_j^1 x_j + c_0^1 \geq 0$$

$$\vdots$$

$$x_{n+1} - \sum_{j=1}^{n} c_j^k x_j + c_0^k \geq 0$$

Then the new model becomes to minimize $z(x) = x_{n+1}$, subject to all the above constraints plus the constraints in LP2-1. As the new objective function is linear and has only one variable, hence, the new model is an LP formulation.

2.3.3 Integer programming

If the variables (x_j) in model LP2-1 are constricted to integers, the model becomes integer linear programming (ILP). An integer linear programming is more difficult than the linear programming, and neither the theory nor the computational aspects of the integer programming are as well developed as they are for the linear programming. The theory of the linear programming serves as a guide for developing results for the integer programming. In many real life situations an LP model may be used to replace the ILP model, such as the job

shop scheduling problem, the flow shop sequencing problem, the capital and cost budgeting problem [Naz95], etc. The Branch and Bound (B&B) and the cutting plane methods are two common algorithms for the integer programming. Some other heuristic algorithms have been developed, such as Lagrangian relaxation [Kar87] and neural network method [Lin92], but none of these methods are reliable from the computational standpoint, particularly when the number of integer variables is large.

2.3.4 Dual linear programming

If a linear programming model is described as follows:

$$\max \sum_{j=1}^{n} c_j x_j$$

Subject to:

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i \qquad i = 1, ..., m \qquad \text{(LP2-2)}$$

where, c_j, a_{ij} are constants and x_j are variables. Then another linear programming model can be formed from LP2-2 as follows:

$$\min \sum_{i=1}^{m} b_i y_i$$

Subject to:

$$\sum_{i=1}^{m} a_{ij} y_i \ge c_j \qquad j = 1, ..., n \qquad \text{(LP2-3)}$$

where y_i are variables. This model is called the duel of LP2-2 while LP2-2 is defined as the primal. It should be noted that the dual of the dual is the primal.

Model LP2-2 has an optimal solution if and only if model LP2-3 has an optimal solution. The dual and the primal problems are very closely related to the extent that the optimum simplex solution of either problem automatically yields the optimal solution to the other problem. The dual to a primal problem with n inequality constraints in m variables is a problem with m constraints in n nonnegative variables. Thus it may be easier to solve the dual if n is large in comparison with m [Had62, Bea88]. The dual simplex method, aside from its use to solve a special class of LP problems, is useful in carrying out post-optimization in sensitivity analysis and parametric programming.

2.4 Algorithms

The simplex method is the general algorithm for solving linear programming whereas the branch and bound method is the most popular algorithm used for solving integer linear programming. Gillet presented a FORTRAN computer program for the mentioned algorithms [Gil76]. The disadvantage of the simplex and the B&B method is that theoretically the computational time will grow exponentially in the number of iterations needed to reach the optimal solution. In the real situation many problems have their own special structures. In order to take this advantage, some special-purpose algorithms have been developed to improve the performance of problem solving. Sometimes, particularly in the case of an integer programming model, the mathematical model may be so complex that it is very expensive to reach its optimal value. In this case, it may be necessary to abandon the search for the optimal solution and simply to seek a good solution using heuristics. The

advantages of a heuristic over an exact optimization algorithm is that it is usually much faster to execute and has a higher flexibility to deal with many complex problems.

2.4.1 Algorithms for linear programming

2.4.1.1 Graphical solution

The graphical method is the basis for the development of the general solution method, the simplex method, for linear programming. For the LP with only two variables, the optimal solution can be found by the graphical method. Nevertheless, if there are three or more variables, the graphical method is impractical. After the problem is formulated mathematically, the first step is to plot the feasible solution space that satisfies all the constraints simultaneously, i.e., all the feasible solutions. This space is called the feasible set. A straight line which represents the objective function is then plotted. In a maximization problem, such a line is called an iso-profit line whereas in a minimization problem, it is an iso-cost line. To find the optimal solution, this line "uphill" or "downhill" to the point where any further increment or decrement would render an infeasible solution. The detail of the algorithm can be referred to [Tah97].

2.4.1.2 The simplex algorithm

The simplex method was designed by Dantzig [Dan51]. It is the general method for linear programming. The idea of the simplex method is to make a trip on the polyhedron underlying a linear programming, from vertex to vertex along edges, until an optimal vertex is reached. Before applying the simplex method on

an LP, all the constraints must be transformed into equality constraints. If a constraint of an LP is a "≤" type, it can be converted into an equality constraint by adding a slack variable. If it is a "≥" type, a surplus variable should be subtracted from the original constraint to become an equality constraint. The formal iterative steps of the simplex method involve [Tah97]:

Step 1: Determine a starting feasible solution from the standard form.

Step 2: Select an entering variable from among the current non-basic variables using the optimality condition. Stop if there is no entering variable.

Step 3: Select a leaving variable from the current basic variables by using the feasibility condition.

Step 4: Determine the values of the new basic variables by making the entering variable basic and the leaving variable non-basic. Go to Step 2.

In order to get a starting solution, two techniques have been proposed: (1) the M-method and (2) the two-phase method [Tah97] and some artificial variables are required to deal with "≥" and "=" types of constraints.

Unlike the simplex method, which starts from a feasible solution but non-optimal, the dual simplex method starts from a non-feasible but optimal solution. The successive iterations are designed to move toward feasibility without violating optimality. At the iteration when feasibility is restored, the algorithm ends. The general ideas used in both the simplex and the dual simplex methods are the same. Another variation of the simplex method is the revised simplex method which can enhance the computational efficiency. The details and the

examples of these simplex methods can be easily found out from, such as, [Tah97, Had62].

2.4.1.3 The Dantig-Wolfe decomposition algorithm

The idea of this algorithm is to decompose a linear programming problem into smaller sub-problems and then solve the sub-problems almost independently. It only generates individual extreme points as needed for a sub-problem and does not normally generate the entire set of extreme points [Las70]. This takes an advantage to solve a large problem that may be computationally infeasible. Yet, this algorithm can only be applied on a special structure of certain large linear problems since the constraints have to be divided into groups [Las70, Tah97, Nas96].

2.4.1.4 The interior point algorithm

Karmarkar proposed this polynomial-time interior point algorithm [Kar84], which cuts across the interior of the solution space. The effectiveness of the algorithm appears to be in the solution of extremely large LP problems. Compared with the simplex method, the interior-point algorithm is likely to perform well if certain conditions are met such as that no good initial solution is available or the problem is degenerate [Nas96].

2.4.1.5 The ellipsoid method

The ellipsoid method approximates the optimal solution to an LP problem by examining the interior of the feasible region and creating a sequence of

ellipsoids to approach the optimal solution [Kha79]. The ellipsoid method is designed to find a point that strictly satisfies a system of linear inequalities. In some cases (such as, the Klee-Minty problem), the simplex method would be much worse than the ellipsoid method. However, in many other problems, the simplex method is much better than the ellipsoid method. The details of the ellipsoid method are presented by [Sch86] and [Nas96].

2.4.2 Algorithms for integer linear programming

The Integer Linear Programming (ILP) algorithms are based on exploiting the tremendous computational success of LP. Thus, before applying an ILP algorithm, the integer restriction on the problem should be released first to form a regular LP. Starting from the continuous optimal point obtained from the LP model, integer constraints are added iteratively to modify the LP solution space in a manner that will eventually render the optimum extreme point that satisfies the integer requirements.

2.4.2.1 The branch-and-bound method

The branch and bound (B&B) technique is a well-structured systematic search of the space of all feasible solutions in a constrained optimization problem that has a finite number of feasible solutions. It originates from Land and Doig [Lan60]. The B&B technique involves branching from one problem to a set of sub-problems and establishing bounds on the new sub-problems. For the ILP with a large number of integer points in the feasible region, the B&B can be a very efficient method for eliminating non-optimal points. As the method has to

solve a complete linear programming at each node, it could be very time consuming for the problem with many integer variables and constraints.

The implicit enumeration method is a class of the B&B, and it is designed specifically for solving binary (zero-one) integer programming. For a problem with n variables, there will be 2^n possible solutions, and most of them may be infeasible. The problem can be solved simply by enumerating all possible solutions, rejecting those infeasible and selecting the feasible solution which minimizes or maximizes the objection function [Gar72].

2.4.2.2 The cutting plane method

The cutting plan method is another common approach for ILP, and it is the first systematic technique, developed by Gomory [Gom58] for the pure integer programming problem. The algorithm is based on the generation of a sequence of linear constraints that "cut out" part of the feasible region of the corresponding LP until the optimal integer solution is found. Taha, H. A. suggested that when the choice is between the cutting plane method and the branch and bound method, the latter is generally superior to the former. The conclusion is, as stated by Taha [Tah97], "the importance of the integer problem in practice is not yet matched by efficient solution algorithms. It is unlikely that a new theoretical breakthrough will be achieved in the area of integer heuristics."

2.4.3 Heuristics

Since an exact integer solution needs a lot of computing time, many heuristic techniques such as Lagrangian relaxation, simulated annealing, tabu

search, Genetic Algorithms (GA) and artificial neural networks have been applied in the operations research field to obtain a near-optimal solution. As GA is selected as a heuristic algorithm to solve the component grouping problem in this project, it will be discussed in Chapter 5 separately.

2.4.3.1 Lagrangian relaxation

Lagrangian relaxation is a well-known technique, which is available to find lower bounds (minimization) or upper bounds (maximization). It is based upon the observation that many difficult integer programming problems can be modeled as a relatively easy problem by relaxing some constraints [Fis85], so a near optimal solution can be achieved.

2.4.3.2 Simulated annealing

Simulated Annealing (SA) is a well-known heuristic technique of local search, in which a sub-set of the feasible solutions is explored by repeatedly moving from the current solution to a neighboring solution, similar to the random descent method in that the neighborhood is sampled at random [Ree93]. In order to implement it for a particular problem, a number of decisions must be made. Firstly, there are some generic decisions, which concern the parameters of the annealing algorithm itself. The second class of decisions is problem specific. These two decisions will affect the speed of the algorithm and the quality of the solution obtained. Simulated annealing is applicable to almost any combinatorial optimization problem and easy to implement. However, a very long running time is needed to converge to the optimum.

Simulated annealing has a successful application to a wide variety of optimization problems, such as, the traveling salesman [Cer85, Gol86], graph partitioning [Joh89], quadratic assignment [Wil87], and operational forest planning [Mur95b]. Recently, Murray, *et al.* [Mur96] used the simulated annealing technique to solve the location planning problem.

2.4.3.3 Tabu search

Tabu Search (TS) is designed for getting a global optimum to a combinatorial optimization problem. It was first presented and then modified by Glover [Glo86, Glo89, Glo90]. Tabu search keeps track not only of local information, like the current value of the objective function, but also of some information related to the exploration process. The details of TS procedures can be referred to [Ree93, Aar97] or [Glo86, Glo89, Glo90].

Tabu search has been applied in the quadratic assignment problem [Kap94], the mixed integer (0, 1) multistage stochastic programming [Lφk96], a machine scheduling problem, and the clustered traveling salesman problem [Lap96]. Besides, Ashayeri, *et al.* used Tabu search to allocate PCB types to machines [Ash98].

2.4.3.4 Artificial neural networks

The philosophy of the Artificial Neural Networks (ANN) approach is to abstract some key ingredients from biology to construct simple mathematical models [Ree93]. There are two kinds of architectures in neural network modeling: feed-forward and feed-back. In feed-forward networks signals are

proceeded from a set of input units in the bottom to output units in the top, layer by layer. In feed-back networks, on the other hand, the synapses are bi-directional. As a matter of fact, ANN is a very active research area, many ANN models have been proposed. Detailed information on ANN can be found from [Zur92].

2.5 Conclusion

This chapter provided an extensive literature review on the production planning problems in PCB assembly. Besides, a very brief review on both optimization models and the techniques to solve the models was also conducted in this chapter. A detailed survey can be made on these models or techniques, however, this is not the purpose of this project. Fortunately, detailed information is available from the above-mentioned or other references.

Some observations based on the above review can be concluded as follows:

1. There are many optimization problems in PCB assembly, these problems are co-related to each other and most of them have fruitful literatures. However, only a few authors addressed the component grouping problem (in the middle level in Fig. 2.1). Furthermore, these authors assumed that the component process times are identical even for different placement machines. So, the component grouping problem has not been fully studied yet.
2. Some nonlinear programming models may be possible to be changed into linear programming. If the constraints of a problem are linear and the

objective function is to minimize the maximum value of several linear functions, then this problem can be converted to linear programming.

3. A very long computing time is required to obtain the optimal solution of an integer linear programming problem and none of the available algorithms are efficient from the computational standpoint, particularly when the problem size is large. If possible, a linear programming should be used to replace an integer linear programming.
4. Heuristic techniques provide a choice to obtain a near-optimal solution for the integer programming. It is noticed that each heuristic has its own characteristics and there is no specific one that is acknowledged to be the best. In this study, the Genetic Algorithm (GA) technique will be used as a tool to solve the component grouping problem in Chapter 5. One reason is that the GA technique has been widely applied to the field of PCB assembly [Lim97, Jai96, Leu93].

In the next chapter the component grouping problem is constructed to various types of mathematical models for both single-sided and double-sided PCB assembly with different objectives. Furthermore, these models will be compared and analyzed. The numerical examples in the next chapter will demonstrate how to formulate the models and the optimal solutions of the examples are provided from a commercial software package, CPLEX.

Chapter Three

MATHEMATICAL MODELS

3.1 Introduction

In order to solve a problem, it is very meaningful to write out the problem in mathematical terms, if possible. This mathematical description or representation of the problem is called a mathematical model. In this chapter, the component grouping problem in both single-sided and double-sided PCB assembly is constructed into mathematical models with different objectives. The objective can be either to maximize the throughput of the line from the idea of a general assembly line design, or to minimize the balance delay from the idea of line performance measurement. It is found that due to a long computing time required in the balance delay model and some other reasons, the model with the objective of maximizing the throughput, or equivalently, minimizing the cycle time, is much preferable. Furthermore, although a line designed for assembling double-sided PCB is different from that for single-sided, the mathematical model from the former can be decomposed into two models of the latter.

3.2 Single-Sided PCB Assembly

There is only one placement station if PCBs need surface mounting on one side of the board only. After the board has gone through all the machines in that station, all components will be attached to the board. From the definition of a general assembly line balancing problem, a line is balanced if the idle time on all the stations along the line is as low as possible. Baybars İ. [Bay86a] stated

that for an assembly line with a fixed station number, this can be attained by maximizing the production rate or equivalently, minimizing the cycle time. The cycle time is defined as the assembly time required at the slowest station. In a PCB assembly line, each placement machine can be considered as a station, and the cycle time can be defined as the time on a machine which takes the longest time to assemble all of the assigned components. On the other hand, as mentioned by McClain *et al.* [Mcc92], the balance delay is a formula in terms of idle time to measure the quality of an assembly line and a line is balanced when the balance delay is minimized. It can be expressed as:

$$\text{Balance Delay \% } D = \left(\frac{NC_l - \sum_{i=1}^{N} C_i}{NC_l} \right) \text{x100} \qquad (3.1)$$

where, N is the total number of machines,

C_i is the assembly time for machine i,

C_l is the line cycle time.

So, two objectives, minimizing the cycle time and minimizing the balance delay, will be used to formulate the mathematical models in the component grouping problem.

3.2.1 Minimizing cycle time

Suppose, n components (may be the same type) in a board need to be assembled on m machines in a line, where n is much greater than m. The placement time for machine i to assemble component j is t_{ij} and the set up time (loading and unloading) for machine i is s_i. A binary decision variable x_{ij} is

introduced to indicate whether component j is assigned to machine i or not. Then, the problem of maximizing the throughput or minimizing the line cycle time can be formulated as follows:

$$Minimize \left[\max \left(\sum_{j=1}^{n} t_{ij} x_{ij} + s_i \mid i = 1, 2, ..., m \right) \right] \tag{3.2}$$

Subject to:

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad \text{for } j = 1, 2, ..., n \tag{3.3}$$

$$x_{ij} = \begin{cases} 1 & \text{If component } j \text{ is assigned to machine } i \\ 0 & \text{Otherwise} \end{cases} \tag{3.4}$$

$$\text{for } i = 1, 2, ..., m; j = 1, 2, ..., n \tag{LP3-1}$$

The objective function (3.2) determines the optimal line cycle time and is to minimize the assembly time for a machine with the maximum assembly time. Constraint set (3.3) guarantees that each component is assigned to one machine only while (3.4) is self-explanatory.

Obviously, LP3-1 is not an integer linear programming (ILP) formulation, however, it can be converted into an ILP by introducing a new variable as follows:

$$Minimize\ T \tag{3.5}$$

Subject to:

$$T - \sum_{j=1}^{n} t_{ij}\, x_{ij} \geq s_i \qquad \text{for } i = 1, 2, ..., m \tag{3.6}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad \text{for } j = 1, 2, ..., n \tag{3.3}$$

$$x_{ij} = \begin{cases} 1 & \text{If component } j \text{ is assigned to machine } i \\ 0 & \text{Otherwise} \end{cases} \qquad (3.4)$$

for $i = 1, 2, \ldots, m; j = 1, 2, \ldots, n$

$$T \geq 0 \qquad (3.7)$$

(LP3-2)

The new variable T is used to represent the line cycle time. The objective function in LP3-1 is divided into two parts: the objective function (3.5) and the constraint set (3.6) in LP3-2. Mathematically, LP3-1 and LP3-2 are equivalent, but LP3-2 is an integer linear programming model, and can be solved by a general algorithm, such as, the branch and bound method.

Model LP3-2 has $nm + 1$ variables and $n + m$ constraints. In general, n is large, for example, 500 components in a board. If there are 3 machines in a line to assemble these components, LP3-2 will have 503 constraints and 1501 variables. So, the formulation is bulky and difficult to handle. However, fortunately, many components on a board are of the same type. By using n to indicate the total component type on a board instead of the total component number, c_j to denote the number of component type j (as seen in Table 3.1), the min-max type problem (LP3-1) can be simplified as follows:

$$Minimize \left[\max \left(\sum_{j=1}^{n} t_{ij} x_{ij} + s_i \mid i = 1, 2, \ldots, m \right) \right] \qquad (3.2)$$

Subject to:

$$\sum_{i=1}^{m} x_{ij} = c_j \qquad \text{for } j = 1, 2, \ldots, n \qquad (3.8)$$

$$x_{ij} \geq 0 \text{ and integer} \qquad \text{for } i = 1, 2, \ldots, m; j = 1, 2, \ldots, n \qquad (3.9)$$

(LP3-3)

Similarly, the ILP format of LP3-3 can be written as follows:

Minimize T (3.5)

Subject to:

$$T - \sum_{j=1}^{n} t_{ij}\, x_{ij} \geq s_i \qquad \text{for } i = 1, 2, \ldots, m \qquad (3.6)$$

$$\sum_{i=1}^{m} x_{ij} = c_j \qquad \text{for } j = 1, 2, \ldots, n \qquad (3.8)$$

$$T \geq 0 \qquad (3.7)$$

$$x_{ij} \geq 0 \text{ and integer} \qquad \text{for } i = 1, 2, \ldots, m; j = 1, 2, \ldots, n \qquad (3.9)$$

(LP3-4)

Here, x_{ij} is no longer a binary variable, it represents the quantity of component type j to be assembled on machine i. The objective functions for LP3-3 and LP3-4 are the same as those in model LP3-1 and LP3-2. Constraints set (3.8) states that all the components have to be assembled. Compared with LP3-1 and LP3-2, n in LP3-3 and LP3-4 is greatly reduced, and consequently, both constraints and variables are reduced, too.

		Component type (j)					Set up time (s_i)
	t_{ij}	1	2	3	.....	n	
Machine (i)	1	t_{11}	t_{12}	t_{13}	.....	t_{1n}	s_1
	2	t_{21}	t_{22}	t_{23}	.....	t_{2n}	s_2
	3	t_{31}	t_{32}	t_{33}	.....	t_{3n}	s_3
	.	.	.	.	.....	.	.
	.	.	.	.	.....	.	.
	.	.	.	.	.....	.	.
	m	t_{m1}	t_{m2}	t_{m3}	.....	t_{mn}	s_m
Number of comp. j (c_j)		c_1	c_2	c_3	.....	c_n	

Table 3.1: The general case for a line to assemble a single-sided PCB

3.2.2 Minimizing balance delay

A similar result can be obtained if the objective of the component grouping problem is considered as to minimize the balance delay of the line. From the definition of the balance delay in (3.1), the new mathematical model is described as follows:

$$Minimize \quad \frac{mT - \sum_{i=1}^{m}\left(\sum_{j=1}^{n} t_{ij}x_{ij} + s_i\right)}{mT} \tag{3.10}$$

Subject to:

$$T - \sum_{j=1}^{n} t_{ij}\, x_{ij} \geq s_i \qquad \text{for } i = 1, 2, ..., m \tag{3.6}$$

$$\sum_{i=1}^{m} x_{ij} = c_j \qquad \text{for } j = 1, 2, ..., n \tag{3.8}$$

$$T \geq 0 \tag{3.7}$$

$$x_{ij} \geq 0 \text{ and integer} \qquad \text{for } i = 1, 2, \ldots, m; j = 1, 2, \ldots, n \tag{3.9}$$

(LP3-5)

LP3-5 shares the same constraint sets in LP3-4, but a different objective function. Obviously, LP3-5 is a non-linear integer programming due to its non-linear objective function. To make the matter worse, LP3-5 can not be converted into an integer linear programming model.

3.2.3 Cycle time vs. balance delay

The initial formulation of the component grouping problem to minimize the cycle time is a type of min-max non-linear model, but it can be transferred to

an integer linear programming after a new variable T is introduced. This has the advantages of short computing time and matured development of algorithms for linear programming. By comparison, since the formula of balance delay is non-linear, model LP3-5 is a class of non-linear programming, although it has linear constraint sets. Unlike model LP3-3, it can not be converted into a linear programming, and there is no good algorithm to handle nonlinear models due to the irregular behavior of the nonlinear functions [Tah97]. Besides, there is a case that when a line is perfectly balanced, that is, the balance delay is minimized to be 0%, but the cycle time is not be minimized. Take Fig. 1.4 in Chapter 1 for example, the component placement times are shown in Table 3.2, and the cycle times and balance delays for the three feasible grouping combinations given in Fig. 1.4 are computed in Table 3.3.

Machine	Component C1	Component C2	Component C3	Component C4	Component C5
CP-I	0.2 sec/each	0.2 sec/each	0.4 sec/each	0.6 sec/each	0.6 sec/each
CP-II	0.5 sec/each	0.4 sec/each	0.8 sec/each	1.0 sec/each	0.5 sec/each
IP-I	1.2 sec /each	1.2 sec/each	0.2 sec/each	0.2 sec/each	0.7 sec/each

Table 3.2: Component placement times

By comparing the results, the line efficiency for combination (c) is perfect, there are no idle time for the production line and the balance delay is 0%. As this grouping decision satisfies all the constraint requirements in LP3-5 and gives out a minimum objective value, it is one of the optimal solutions for LP3-5. However, the production rate (throughput) under this grouping decision is not optimal, although the line is balanced. Instead, combination (b) is the best component grouping and assignment decision among the three, it has the least cycle time and gives out the highest production rate. Therefore, it can be

concluded that the balance delay model LP3-5 is not a proper formulation for the component grouping problem, and only the model with the objective of minimizing the cycle time will be further studied in this project.

Combination	Assembly Time (C_i)			Line Cycle Time	Balance Delay
	CP-I	CP-II	IP-I		
(a)	0.4	1.8	0.7	1.8	46.30%
(b)	0.8	1.0	0.7	1.0	16.70%
(c)	1.2	1.2	1.2	1.2	0%

Table 3.3: Solutions for the three combinations

3.3 Double-Sided PCB Assembly

There will be two placement stations if a PCB needs surface mounting on both sides of the board. The process of component attachment will be finished only after the board is passed through the two placement stations. The cycle time in this case is defined as the time in the machine that requires the maximum assembly time among all the machines in the two stations. Two radically different formulations of the component assignment problem with minimizing cycle time will be presented from two different views.

3.3.1 Models

Suppose a double-sided PCB is assigned to a production line. The board contains n types of components on the bottom side and q types of components on the top side, so $j = 1, 2, \ldots, n$ is for the component types in the bottom side, while $j = n + 1, n + 2, \ldots, n + q$ is for the top side. Some types of the components on the bottom side may be the same as the one in the top side, but in the formulation,

they are classified as different types. Since there are two placement stations on the production line, station one that processes components on the bottom side has m automatic placement machines ($i = 1, 2, \ldots, m$), while station two that processes components on the top side has p automatic placement machines ($i = m + 1, m + 2, \ldots, m + p$). The placement time for component type j to be assembled on machine i is still t_{ij}, and similarly, c_j denotes the number of component type j, for $j = 1, 2, \ldots, n + q$ and s_i as the setup time required for machine i, for $i = 1, 2, \ldots, m + p$. Table 3.4 is a tableau for the problem, so the min-max type model of minimizing the cycle time is as follows:

$$Mininize\left[\max\left(\sum_{j=1}^{n+q} t_{ij}x_{ij}+s_i \;\middle|\; i=1,2,...,m+p\right)\right] \tag{3.11}$$

Subject to:

$$\sum_{i=1}^{m} x_{ij} = \begin{cases} c_j & \text{for } j = 1, 2, ..., n \\ 0 & \text{for } j = n+1, n+2, ..., n+q \end{cases} \tag{3.12}$$

$$\sum_{i=m+1}^{m+p} x_{ij} = \begin{cases} 0 & \text{for } j = 1, 2, ..., n \\ c_j & \text{for } j = n+1, n+2, ..., n+q \end{cases} \tag{3.13}$$

$$x_{ij} \geq 0 \text{ and integer} \qquad \text{for } i = 1, 2, \ldots, m+p; j = 1, 2, \ldots, n+q \tag{3.14}$$

(LP3-6)

		Components on the bottom side				*Components on the top side*				*Set up*
		1	2	.....	n	n+1	n+2	.....	n+q	*Time* s_i
Machines in station one	1	t_{11}	t_{12}	.....	t_{1n}	$t_{1(n+1)}$	$t_{1(n+2)}$	.....	$t_{1(n+q)}$	s_1
	2	t_{21}	t_{22}	.....	t_{2n}	$t_{2(n+1)}$	$t_{2(n+2)}$	.....	$t_{2(n+q)}$	s_2
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	m	t_{m1}	t_{m2}	.....	t_{mn}	$t_{m(n+1)}$	$t_{m(n+1)}$	.....	$t_{m(n+q)}$	s_m
comp. no. c_j		c_1	c_2	.....	c_n	0	0	.....	0	
Machines in station two	m+1	$t_{(m+1)1}$	$t_{(m+1)2}$	.....	$t_{(m+1)n}$	$t_{(m+1)(n+1)}$	$t_{(m+1)(n+2)}$	.....	$t_{(m+1)(n+q)}$	s_{m+1}
	m+2	$t_{(m+2)1}$	$t_{(m+2)2}$	.....	$t_{(m+2)n}$	$t_{(m+2)(n+1)}$	$t_{(m+2)(n+2)}$	.....	$t_{(m+2)(n+q)}$	s_{m+2}
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	$m+p$	$t_{(m+p)1}$	$t_{(m+p)2}$	.....	$t_{(m+p)n}$	$t_{(m+p)(n+1)}$	$t_{(m+p)(n+2)}$	.....	$t_{(m+p)(n+q)}$	s_{m+p}
comp. no. c_j		0	0	.....	0	c_{n+1}	c_{n+2}	.....	c_{n+q}	

Table 3.4: A tableau for model LP3-6

Similarly, LP3-6 can be converted into an integer linear programming by introducing a variable for the cycle time, T, as follows:

Minimize T

Subject to:

$$T - \sum_{j=1}^{n+q} t_{ij} x_{ij} \geq s_i \qquad \text{for } i = 1, 2, \ldots, m+p \qquad (3.15)$$

$$\sum_{i=1}^{m} x_{ij} = \begin{cases} c_j \\ 0 \end{cases} \qquad \begin{matrix} \text{for } j = 1, 2, \ldots, n \\ \text{for } j = n+1, n+2, \ldots, n+q \end{matrix} \qquad (3.12)$$

$$\sum_{i=m+1}^{m+p} x_{ij} = \begin{cases} 0 \\ c_j \end{cases} \qquad \begin{matrix} \text{for } j = 1, 2, \ldots, n \\ \text{for } j = n+1, n+2, \ldots, n+q \end{matrix} \qquad (3.13)$$

$$T \geq 0 \qquad (3.7)$$

$$x_{ij} \geq 0 \text{ and integer} \qquad \text{for } i = 1, 2, \ldots, m+p; j = 1, 2, \ldots, n+q \qquad (3.14)$$

(LP3-7)

Another way to formulate the problem is to assign t_{ij} to ∞ and x_{ij} to zero if i is a machine in the first station (for the bottom side) and j is a component in the top side, or if i is a machine in the second station (for the top side) and j is a component in the bottom side. This is reasonable as all of the machines in the first station are unable to process any components at the top side, so the placement time can be assumed to infinity, as seen in Table 3.5. The objective function of the model is to minimize two sets of independent linear functions:

Minimize

$$\left[\max\left(\sum_{j=1}^{n} t_{ij}x_{ij} + s_i \quad | i = 1,2,...,m; \quad \sum_{j=n+1}^{n+q} t_{ij}x_{ij} + s_i \quad | i = m+1, m+2,...,m+q\right)\right] \tag{3.16}$$

Subject to:

$$\sum_{i=1}^{m} x_{ij} = c_j \qquad \text{for } j = 1, 2, \ldots, n \tag{3.8}$$

$$\sum_{i=m+1}^{m+p} x_{ij} = c_j \qquad \text{for } j = n+1, n+2, \ldots, n+q \tag{3.17}$$

$$x_{ij} \geq 0 \text{ and integer} \qquad \text{for } i = 1, 2, \ldots, m+p; j = 1, 2, \ldots, n+q \tag{3.14}$$

(LP3-8)

The linear programming format of model LP3-8 is written as:

Minimize T

Subject to:

$$T - \sum_{j=1}^{n} t_{ij}x_{ij} \geq s_i \qquad \text{for } i = 1, 2, \ldots, m \tag{3.6}$$

$$T - \sum_{j=n+1}^{n+q} t_{ij} x_{ij} \geq s_i \qquad \text{for } i = m+1, m+2, \ldots, m+p \qquad (3.18)$$

$$\sum_{i=1}^{m} x_{ij} = c_j \qquad \text{for } j = 1, 2, \ldots, n \qquad (3.8)$$

$$\sum_{i=m+1}^{m+p} x_{ij} = c_j \qquad \text{for } j = n+1, n+2, \ldots, n+q \qquad (3.17)$$

$$T \geq 0 \qquad (3.7)$$

$$x_{ij} \geq 0 \text{ and integer} \qquad \text{for } i = 1, 2, \ldots, m+p; j = 1, 2, \ldots, n+q \qquad (3.14)$$

(LP3-9)

		Components on bottom side				*Components on top side*				*Set up*
		1	2	.....	n	$n+1$	$n+2$	.....	$n+q$	*time,* s_i
Machines in station one	1	t_{11}	t_{12}	.....	t_{1n}	∞	∞	.....	∞	s_1
	2	t_{21}	t_{22}	.....	t_{2n}	∞	∞	.....	∞	s_2
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	m	t_{m1}	t_{m2}	.....	t_{mn}	∞	∞	.....	∞	s_m
Machines in station two	$m+1$	∞	∞	.....	∞	$t_{(m+1)(n+1)}$	$t_{(m+1)(n+2)}$	.....	$t_{(m+1)(n+q)}$	s_{m+1}
	$m+2$	∞	∞	.....	∞	$t_{(m+2)(n+1)}$	$t_{(m+2)(n+2)}$	.....	$t_{(m+2)(n+q)}$	s_{m+2}
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	$m+p$	∞	∞	.....	∞	$t_{(m+p)(n+1)}$	$t_{(m+p)(n+2)}$	.....	$t_{(m+p)(n+q)}$	s_{m+p}
Comp. no. c_j		c_1	c_2	.....	c_n	c_{n+1}	c_{n+2}	.....	c_{n+q}	

Table 3.5: A tableau for model LP3-7

3.3.2 Computational complexity

The solutions of model LP3-7 and LP3-9 are exactly the same due to their equivalent physical meanings. This gives rise to a question: which model has a better performance? The performance of a model can be measured by the

computing complexity. Since the simplex method is the most widely used method for linear programming, a short performance analysis of the two models will be given based on the "worst-case" of the simplex method.

For a problem with n variables and m constraints, there are ${}^{n}C_{m}$ basic solutions [Nas96], as well as the iterations, so the complexity will exponentially increase as the variables and constraints increase. In order to compare LP3-7 and LP3-9, the variable and the constraint numbers of models are listed in Table 3.6:

Models	No. of variables (slack variables included)	No. of constraints
LP3-7	$(m+p)(n+q)+1+m+p+2(n+q)$	$m+p+2(n+q)$
LP3-9	$(m+p)(n+q)+1+n+q+m+p$	$m+p+n+q$

Table 3.6: Numbers of variables and constraints in LP3-7 and LP3-9

Let

$$a = (m+p)(n+q)+1,$$

$$b = m+p+2(n+q),$$

$$c = n+q+m+p,$$

Then, the maximum number of iterations required in model LP3-7 is:

$${}^{a+b}C_b = \frac{(a+b)!}{a!b!}$$

$$= \frac{(b+1)(b+2)....(b+a)}{a!}$$

Similarly, the maximum number of iterations required in LP3-9 is

$${}^{a+c}C_c = \frac{(c+1)(c+2)....(c+a)}{a!}$$

As $b = c + n + p \geq c$, so ${}^{a+b}C_b \geq {}^{a+c}C_c$, and LP3-7 requires a longer computing time than LP3-9. In conclusion, model LP3-9 is a superior formulation for the component grouping problem in double-sided PCB assembly.

3.3.3 A decomposition approach for solving the double-sided PCB assembly

Model LP3-9 can be solved by an integer linear programming algorithm. However, as a matter of fact, the two placement stations in a double-sided PCB assembly line are independent of each other since the machines in one station can only assemble the components on one side only. So it is possible to decompose the huge problem into two sub-problems, and each of the sub-problems has the same structure as LP3-4. Then, the cycle time of each station will be determined individually and the cycle time of the line is the longer cycle time in the two stations. The procedure is listed as follows:

Step 1: Let T' be the cycle time for bottom side, and by LP3-4, the mathematical model is:

Minimize T'

Subject to:

$$T' - \sum_{j=1}^{n} t_{ij}\, x_{ij} \geq s_i \qquad \text{for } i = 1, 2, \ldots, m$$

$$\sum_{i=1}^{m} x_{ij} = c_j \qquad \text{for } j = 1, 2, \ldots, n$$

$$x_{ij} \geq 0 \text{ and integer} \qquad \text{for } i = 1, 2, \ldots, m; j = 1, 2, \ldots, n$$

Step 2: Let T'' be the cycle time for the top side, and the mathematical model is,

Minimize T''

Subject to:

$$T'' - \sum_{j=n+1}^{n+q} t_{ij}\, x_{ij} \geq s_i \qquad \text{for } i = m+1, m+2, \ldots, m+p$$

$$\sum_{i=m+1}^{m+p} x_{ij} = c_j \qquad \text{for } j = n+1, n+2, \ldots, n+q$$

$$x_{ij} \geq 0 \text{ and integer} \qquad \text{for } i = m+1, m+2, \ldots, m+p;$$

$$j = n+1, n+2, \ldots, n+q$$

Step 3: The cycle time (T) of the line is the longer time among the two stations' cycle times, that is, $T = max\{T', T''\}$.

It should be pointed out that the optimal objective value (T) determined by either Model LP3-9 or the above decomposition method are the same, but the decision variables (x_{ij}) may be different. An example in the next section will show this point. Furthermore, whether the component grouping problem for double-sided PCB assembly should be decomposed or not depends on the computing time. The coefficient matrix of LP3-9 has a size of $[(m + p) \times (n + q)]$, whereas the coefficient matrix for the two sub-problems has a size of $(m \times n)$ and $(p \times q)$. Clearly, $[mn + mq + np + pq] \geq [mn + pq]$, it is more efficient to solve the problem by the decomposition approach.

3.4 Examples

Two examples are presented to illustrate the mathematical formulations of the component grouping problem for both single-sided and double-sided PCB assembly. Example 1 shows how the component grouping problem in single-

sided board is formulated by using LP3-4, and Example 2 demonstrates how to solve a double-sided board problem by either the comprehensive model LP3-9 or the decomposition procedure. The optimal solutions of the examples are obtained from an integer linear programming package, CPLEX, from CPLEX Optimization, Inc.

3.4.1 Example 1: Single-sided PCB assembly

This example illustrates how model LP3-4 is implemented in grouping components to machines such that a line has the optimal performance. A board with 277 components is assembled in an assembly line with three machines. Most of these components are of the same type, so $m = 3$ and $n = 4$. The relative placement times and set up times are given in Table 3.7 (in 0.1 seconds). The infinity (∞) in column 3 means that machine 1 is unable to process component 3. In this case, a significant large value (say, 90000) is assigned in the mathematical model.

Machine i	Component type j				Set up time s_i
	1	2	3	4	
1	3	7	∞	12	110
2	7	18	19	24	147
3	23	26	33	34	147
No. of Component c_j	231	24	12	10	

Table 3.7: Data for example 1

The component grouping problem in the form of LP3-4 becomes:

Minimize T

Subject to:

$$T - 3x_{11} - 7x_{12} - 90000x_{13} - 12x_{14} \geq 110$$

$$T - 7x_{21} - 18x_{22} - 19x_{23} - 24x_{24} \geq 147$$

$$T - 23x_{31} - 26x_{32} - 33x_{33} - 34x_{34} \geq 147$$

$$x_{11} + x_{21} + x_{31} = 231$$

$$x_{12} + x_{22} + x_{32} = 24$$

$$x_{13} + x_{23} + x_{33} = 12$$

$$x_{14} + x_{24} + x_{34} = 10$$

$$T, x_{ij} \geq 0; \qquad i = 1, 2, ..., 3; j = 1, 2, ..., 4$$

The optimal cycle time for the above formulation is 74.6 seconds, shown in Section 3.4.3. There are only 13 variables and 7 constraints, compared with LP3-2, the variable and the constraint numbers will be 832 (i.e., 3 x 277 + 1) and 280 (i.e. 3 + 277), so LP3-4 is a preferable model.

3.4.2 Example 2: Double-sided PCB assembly

Example 1 can be extended to the case of double-sided PCB assembly if the information on the other side is added. Suppose the information described in example 1 is for the first placement station and the top side, Table 3.8 is the information for the second placement station and the bottom side (the placement times are in 0.1 seconds). The optimal component grouping can be determined by formulating the problem as LP3-9 and has the following form:

Minimize T

Subject to:

$$T - 3x_{11} - 7x_{12} - 90000x_{13} - 12x_{14} \geq 110$$

$$T - 7x_{21} - 18x_{22} - 19x_{23} - 24x_{24} \geq 147$$

$$T - 23x_{31} - 26x_{32} - 33x_{33} - 34x_{34} \geq 147$$

$T - 3x_{45} - 7x_{46} - 7x_{47} - 5x_{48} - 90000x_{49} - 90000x_{4,10} - 90000x_{4,11} \geq 110$

$T - 7x_{55} - 12x_{56} - 15x_{57} - 16x_{58} - 15x_{59} - 15x_{5,10} - 21x_{5,11} \geq 147$

$T - 23x_{65} - 38x_{66} - 35x_{67} - 35x_{68} - 27x_{69} - 33x_{6,10} - 43x_{6,11} \geq 147$

$x_{11} + x_{21} + x_{31} = 231$

$x_{12} + x_{22} + x_{32} = 24$

$x_{13} + x_{23} + x_{33} = 12$

$x_{14} + x_{24} + x_{34} = 10$

$x_{45} + x_{55} + x_{65} = 324$

$x_{46} + x_{56} + x_{66} = 37$

$x_{47} + x_{57} + x_{67} = 12$

$x_{48} + x_{58} + x_{68} = 5$

$x_{49} + x_{59} + x_{69} = 7$

$x_{4,10} + x_{5,10} + x_{6,10} = 5$

$x_{4,11} + x_{5,11} + x_{6,11} = 4$

$T, x_{ij} \geq 0$ and integer $\quad i = 1, 2,\ldots, 6;\ j = 1, 2,\ldots, 11$

The optimal cycle time for the above model is 97.1 seconds, obtained from CPLEX.

Machine i	Component Type j							*Setup Time* s_i
	5	6	7	8	9	10	11	
4	3	7	7	5	∞	∞	∞	110
5	7	12	15	16	15	15	21	147
6	23	38	35	35	27	33	43	147
No. of Component c_j	324	37	12	5	7	5	4	

Table 3.8: Data for the bottom side PCB

The procedure to decompose the above formation into two sub-problems is as follows:

Step 1: The optimal cycle time (T') for top side has been determined in example 1 and $T' = 74.6$ seconds.

Step 2: The optimal cycle time (T'') for the bottom side can be determined by the model described below, and T'' =97.1 seconds.

Minimize T''

Subject to:

$T''-3x_{45}-7x_{46}-7x_{47}-5x_{48}-90000x_{49}-90000x_{4,10}-90000x_{4,11} \geq 110$

$T''-7x_{55}-12x_{56}-15x_{57}-16x_{58}-15x_{59}-15x_{5,10}-21x_{5,11} \geq 147$

$T''-23x_{65}-38x_{66}-35x_{67}-35x_{68}-27x_{69}-33x_{6,10}-43x_{6,11} \geq 147$

$x_{45}+x_{55}+x_{65}=324$

$x_{46}+x_{56}+x_{66}=37$

$x_{47}+x_{57}+x_{67}=12$

$x_{48}+x_{58}+x_{68}=5$

$x_{49}+x_{59}+x_{69}=7$

$x_{4,10}+x_{5,10}+x_{6,10}=5$

$x_{4,11}+x_{5,11}+x_{6,11}=4$

$x_{ij} \geq 0$ and integer; $i = 4, 5, 6; j = 5, 6, 7, 8, 9, 10, 11$

Step 3: The cycle time for the line is thus $max\{74.6, 97.1\}$, i.e., 97.1 seconds.

The solutions for the two methods are presented at the section 3.4.3. Over 2000 iterations were required to reach the optimum if the problem is modeled as LP3-9, but only 1102 iterations were needed (102 iterations for the top side + 1000 iterations for the bottom side) by the decomposition approach. It is noted that although the optimal cycle time is the same, i.e., 97.1s, the component assignments determined by the two approaches are different. For example, the number of component 1 assigned to machine 1 is 231 in the comprehensive model LP3-9, but 179 in example 1.

3.4.3 CPLEX results

The following results were generated from CPLEX.

Results for Example 1: Single-sided PCB assembly

```
Presolve time =    0.05 sec.

Integer optimal solution:  Objective =
7.4600000000e+002
Solution time =    0.27 sec.    Iterations = 102
Nodes = 101


Variable Name           Solution Value
T                            746.000000
x11                          179.000000
x21                           52.000000
x12                           14.000000
x32                           10.000000
x23                           11.000000
x33                            1.000000
x24                            1.000000
x34                            9.000000
All other variables in the range 1-13 are zero.
```

Results for Example 2: Double-sided PCB assembly

```
Presolve time =    0.33 sec.

Integer optimal solution (0.0001/0):  Objective =
9.7100000000e+002
Solution time =    5.00 sec.    Iterations = 2211
Nodes = 1907


Variable Name           Solution Value
T                            971.000000
x11                          231.000000
x12                            7.000000
x22                           17.000000
x23                           12.000000
x14                            9.000000
x24                            1.000000
x45                          274.000000
x55                           49.000000
x65                            1.000000
```

```
x56                              37.000000
x47                               2.000000
x57                               1.000000
x67                               9.000000
x48                               5.000000
x69                               7.000000
x610                              5.000000
x511                              1.000000
x611                              3.000000
All other variables in the range 1-34 are zero.
```

Results for Example 2: Double-sided PCB assembly – Bottom side only

```
Presolve time =    0.05 sec.

Integer optimal solution (0.0001/0):  Objective =
9.7100000000e+002
Solution time =    2.36 sec.    Iterations = 1000
Nodes = 923

Variable Name            Solution Value
T"                            971.000000
x45                           274.000000
x55                            50.000000
x56                            37.000000
x47                             2.000000
x67                            10.000000
x48                             5.000000
x69                             7.000000
x510                            2.000000
x610                            3.000000
x611                            4.000000
All other variables in the range 1-22 are zero.
```

3.5 Conclusion

After the component grouping problem is formulated as various types of mathematical models, the following observations can be concluded:

1. The min-max integer nonlinear component grouping problem can be transferred into an ILP, and the binary ILP can be extended to a general ILP.

These changes make it possible to reduce the problem size and to use some available algorithms.

2. Due to the longer computing time required in the balance delay model and the fact that a perfect balance line may not necessarily give a good throughput of the line, the minimum cycle time is a better choice for the objective function in the mathematical model.
3. Although the optimal cycle time for double-sided PCB assembly can be obtained directly from its mathematical model, decomposing the problem to two independent sub-problems (each of the sub-problems has the same data structure as a single-sided PCB) is preferable. It is because the time required to solve a large problem is longer than the time required to solve two small problems.

Chapter 4 will compare the mathematical models developed in this chapter with other special models in order to use an available algorithm. An effective algorithm for solving the component grouping problem may exist if its data structure is the same as a special model. In addition, a special algorithm will be presented to find a near optimal solution for the component grouping problem based on the idea of the revised simplex method.

Chapter Four

A LINEAR PROGRAMMING SOLUTION

4.1 Introduction

As indicated in Chapter 3, the component grouping problem can be formulated into a min-max type (LP3-3) or an integer linear (LP3-4) programming model. It will be great if its structure is the same as any other mathematical models, so there may exist an effective algorithm to solve the model. Thus, in this chapter, the component grouping model is compared with the models that have similar special structures. Besides, an effective algorithm is developed to determine the optimal solution by releasing the integer constraints in LP3-4. The new algorithm is proposed based on the idea of the revised dual simplex method. An efficient algorithm for the linear programming model is essential for PCB assembly. First of all, the integer constraint causes the problem much more complicated and expensive to solve than the corresponding continuous linear programming problem. Secondly, its solution can be used as a lower bound for an integer programming algorithm, if the optimal integer solution is definitely required. Finally, the most important point is that the decimal solution from the LP model with rounding off is acceptable for PCB assembly in practice.

4.2 A Comparison with Other Similar Models

4.2.1 Other similar models

The Agent Bottleneck Generalized Assignment Problem (ABGAP) [Maz88] has a very similar structure to the component grouping problem in the form of the min-max programming. For the ease of comparison, these two models are listed as follows:

ABGAP	The Component Grouping Problem
$min\{max\{\sum_{j=1}^{n} c_{ij} x_{ij}\}\}$	$min\ \{max\{\sum_{j=1}^{n} t_{ij}\ x_{ij}\ + s_i\}\}$
Subject to:	*Subject to:*
$\sum_{i=1}^{m} x_{ij} = 1 \qquad j = 1, 2, \ldots, n$	$\sum_{i=1}^{m} x_{ij} = 1 \qquad j = 1, 2, \ldots, n$
$\sum_{j=1}^{n} a_{ij} x_{ij} \le b_i \quad i = 1, 2, \ldots, m$	
$x_{ij} = 0$ or 1	$x_{ij} = 0$ or 1

Table 4.1: The component grouping problem vs. ABGAP

It is noted that the component grouping problem is a special case of ABGAP as the second constraint set in ABGAP does not exist in the component grouping problem. Although there is an additional parameter s_i in the component grouping problem, s_i in the objective function is independent of variable x_{ij}, it will not affect much in the objective function. Thus, an algorithm for ABGAP should be applicable to the component grouping problem. However, the survey shows there are no special effective algorithms for ABGAP. A detailed survey on several assignment models is provided in the next section.

Another similar model is the simple assembly line balance problem II (SALB-II), identified by Baybars in 1986 [Bay86a]. Table 4.2 lists the SALB-II model and the component grouping model.

SALB-II	The Component Grouping Problem
$\min T$	$\min T$
Subject to:	*Subject to:*
$\sum_{i=1}^{m} x_{ij} = 1 \qquad j = 1, 2, \ldots, n$	$\sum_{i=1}^{m} x_{ij} = 1 \qquad j = 1, 2, \ldots, n$
$T - \sum_{j=1}^{n} t_j x_{ij} \geq 0 \qquad i = 1, 2, \ldots, m$	$T - \sum_{j=1}^{n} t_{ij} x_{ij} \geq 0 \qquad i = 1, 2, \ldots, m$
$x_{kj} - \sum_{i=1}^{k} x_{ih} \leq 0 \qquad j = 1, 2, \ldots, n,$ $k = 1, 2, \ldots, m, h \in P(j)$	
$x_{ij} = 0$ or 1	$x_{ij} = 0$ or 1

Table 4.2 The component grouping problem vs. SALBP-II

SLAB-II has a requirement on the precedence of assembly activities, so the model is more complicated. This does not mean that the algorithms for SLAB-II can be used for the component grouping problem. Actually, the assembly times (t_j) in SLAB-II are independent of work stations, but different machines have different placement times (t_{ij}) in the case of PCB assembly.

A brief review on these two relevant models is conducted in the following sections to see whether any ideas from the algorithms for the models can be used for the component grouping problem.

4.2.2 A survey on the assignment problems

Two types of the assignment problems, the generalized assignment problem and the bottleneck generalized assignment problem, are reviewed in this section. In general, the assignment problem is to allocate jobs to machines. If m and n denote the numbers of machines and jobs, respectively, the cost of processing job j on machine i is denoted as c_{ij}, and machine i takes a_{ij} time to process job j, b_i is the total available process time on machine i, then, the generalized assignment problem (GAP) can be formulated as follows:

$$\textit{Minimize} \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}$$

Subject to:

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad j = 1, 2, \ldots, n$$

$$\sum_{j=1}^{n} a_{ij}x_{ij} \le b_i \qquad i = 1, 2, \ldots, m$$

$$x_{ij} = 0 \text{ or } 1$$

G. T. Ross developed a branch and bound (B&B) algorithm to solve GAP [Ros75]. By repeatedly solving a series of binary knapsack problem, rather than using a linear programming, the lower bound for the problem can be improved and the calculation will stop when the lower bound is a feasible solution to the problem. Based on a heuristic preprocessing, a reduction phase and a branch and bound scheme, an enumerative algorithm is presented for solving GAP by Martello, *et al.* [Mat81]. Hallefjord, *et al.* proposed an approximation technique, aggregation/diaggregation, to solve a large scale GAP [Hal93]. The original

large problem was aggregated into a problem with a more moderate size, and then the aggregated model was solved. The result was diaggregated into a feasible solution to the original problem.

It is apparent that the classical assignment problem is a special case of the generalized assignment problem, where $n = m$ and $a_{ij}/b_i = 1$. A number of special algorithms for the classical assignment problem were developed [Wei91, Orl92]. Kennington, *et al.* presented a heuristic algorithm to find a near optimal integer solution for the singly constrained assignment problem [Ken94]. The method is based on the Lagrangian duality theory and involves in solving a series of the pure assignment problems. Kapov proposed a Tabu search to solve a special assignment problem, the quadratic assignment problem [Kap94]. By relaxing the given assignment problem into a series of simple network flow problem, Hung, *et al.* presented a new algorithm with a good computational result [Hun80]. The idea behind the algorithm is to relax the column constraints so that an optimal solution to the relaxed problem can be found easily. Then a violated constraint is enforced and a new optimal solution is constructed. The best algorithm for the assignment problem is acknowledged to be the auction algorithm and the shortest augmenting path algorithm. This conclusion comes from Zaki's comparison work [Zak95]. Derigs discussed the shortest augmenting path method in detail [Der85]. The auction algorithm was initially introduced by Bertsekas [Ber81, Ber88, Ber91, Ber92a] and it was further extended to the transportation problem [Ber89] and the network flow problem [Ber90, Ber94, Ber92b]. Computational studies [Bar89, Sch94, Ami94] showed that the auction algorithm is efficient.

Another generalized assignment problem is the bottleneck generalized assignment problem (BGAP). Two types of BGAP, called task BGAP (TBGAP) and agent BGAP (ABGAP), were specified by Mazzola, *et al.* [Maz88]. They also presented a procedure to search an optimal solution for TBGAP. These two problems have the same constraint sets as the generalized assignment problem but different objective functions. The objective function for ABGAP is

$$min\{max\{\sum_{j=1}^{n} c_{ij}x_{ij}\}\mid i=1,2,...m\}$$

and the objective function for TBGAP is

$$\min\{\max\{c_{ij}x_{ij}\}\mid i=1,2,\ldots,m;\ j=1,2,\ldots,n\}$$

Some other contributions to the solution of TBGAP were addressed in [Maz93, Mar95]. Mazzola, *et al.* [Maz93] obtained a lower bound and an upper bound for the problem, then an optimal solution would be searched within these two bounds. Martello, *et al.* [Mar95] introduced an approximate algorithm and an exact B&B procedure to cope with the same problem.

The bottleneck assignment problem [Gro59, Gar70] is a specific case of TBGAP, where $m = n$ and $a_{ij}/b_i = 1$ and has been studied since 1959. Garfinkel, *et al.* [Gar70] issued an algorithm for it and compared the algorithm with Gross's work. The mathematical model for ABGAP was formulated at [Maz88], but no specific algorithms were provided. The bottleneck assignment problem was solved by a "greedy" algorithm presented by Garfinkel and Rao [Gar76]. Yu and Kouvelis studied the complexity of the min-max (bottleneck) assignment problem. They concluded it is an NP-hard problem [Yug93].

4.2.3 A survey on the simple assembly line balancing problems

In a production line, the task assignment is normally based on balancing the work load among the machines. A line is balanced when the tasks are grouped so that the sum of the idle times on all the stations along the line is minimized. For a given fixed cycle time, this target can be attained by minimizing the total number of stations, and is called simple assembly line balancing type I (SALB-I). If the number of stations is given and fixed, in order to have a balanced line, the goal becomes maximizing the production rate or minimizing the cycle time, which is called simple assembly line balancing type II (SALB-II). Both problems are NP-hard, so heuristic approaches are usually applied. Baybars gave a detailed survey on both optimal [Bay86a] and heuristic [Bay86b] algorithms for these two line balancing problems.

SALB-I can be formulated as 0-1 integer programming [Whi61], or dynamic programming [Hel63, Hen86], and some techniques for these two programmings were applied to determine an exact optimal solution [Wee81, Kim95]. Many heuristic techniques [Hel61, Bar89, Sch96] were also presented to solve SALB-I. Compared with SALB-I, few researchers addressed to the SALB-II problem, and most of them are heuristic [Hac89, Mcc89, Sch96]. This also indicates that SALB-II is more difficult to solve.

The conclusion from the survey is that there is no algorithm ready to use for the component grouping problem. Since the integer solution of the model is very difficult to obtain, a linear programming solution can be used in engineering, as mentioned in Chapter 2 and pointed out by Taha [Tah97]. So a

linear programming solution and a heuristic approach will be presented in this chapter and in the next chapter.

4.3 The Dual

In order to get an efficient linear programming algorithm for the component grouping problem, the dual of the model LP3-4 should be studied. Before the dual is formulated, the integer constraint in LP3-4 should be released. Model LP3-4 can be rewritten as follows:

Minimize T

Subject to:

$$T - \sum_{j=1}^{n} t_{ij}\, x_{ij} \geq s_i \qquad i = 1, 2, \ldots, m \qquad (3.6)$$

$$\sum_{i=1}^{m} x_{ij} \geq c_j \qquad j = 1, 2, \ldots, n \qquad (4.1)$$

$$x_{ij} \geq 0 \qquad i = 1, 2, \ldots, m, j = 1, 2, \ldots, n \qquad (4.2)$$

(LP4-1)

In the above model LP4-1, constraint (4.2) is released from constraint (3.9) in LP3-4 and it states that x_{ij} can be any positive fraction value. Moreover, the constraint (3.8) in LP3-4 is rewritten as (4.1). The difference here is that the sign "=" is replaced by "≥". The minimization requirement in the objective will drive the constraint set (4.1) to the constraint set (3.6) in LP3-4, so the change will not affect the optimal solution in the model.

If model LP4-1 is considered as the primal, the constraint set (3.6) is defined as the dual variable u_i, and the constraint set (4.1) as v_j, then the dual of LP4-1 can be formulated as follows:

$$Maximize\ \tau = \sum_{j=1}^{n} c_j v_j + \sum_{i=1}^{m} s_i u_i \tag{4.3}$$

Subject to:

$$v_j - t_{ij} u_i \le 0 \qquad i = 1, 2, ..., m; j = 1, 2, ..., n \tag{4.4}$$

$$\sum_{i=1}^{m} u_i = 1 \tag{4.5}$$

$$u_i \ge 0 \qquad i = 1, 2, ..., m \tag{4.6}$$

$$v_j \ge 0 \qquad j = 1, 2, ..., n \tag{4.7}$$

(LP4-2)

There are $mn + 1$ constraints and $m + n$ variables in LP4-2, each constraint in set (4.4) represents a cell (i, j) in the original problem, while constraint (4.5) is from the objective function in the primal LP4-1.

4.3.1 Dual of the generalized transportation problem

Although the generalized transportation problem (GTP) and the component grouping problem are different from each other, they have a close relationship in their dual forms. The GTP formulation arose in different contexts [Bal64, Tho86, Jip94], and the most related application was the production scheduling of SMT by Ji, *et al.* [Jip94]. The GTP model can be described as follows:

$$Minimize \sum_{i=1}^{m}\sum_{j=1}^{n} b_{ij}x_{ij} \tag{4.8}$$

Subject to:

$$\sum_{j=1}^{n} a_{ij}x_{ij} \le p_i \qquad i = 1, 2, \ldots, m \tag{4.9}$$

$$\sum_{i=1}^{m} x_{ij} \ge d_j \qquad j = 1, 2, \ldots, n \tag{4.10}$$

$$x_{ij} \ge 0 \text{ and integer} \qquad i = 1, 2, \ldots, m; j = 1, 2, \ldots, n \tag{3.9}$$

(LP4-3)

After released integer constraint (3.9) to (4.2) and defined constraint set (4.9) as the dual variable $-u_i$, constraint set (4.10) as v_j, the dual of the GTP can be formulated as shown in Table 4.3, together with the dual of the component grouping problem:

Dual of the GTP	Dual of the component grouping problem
$\max \varphi = \sum_{j=1}^{n} d_j v_j - \sum_{i=1}^{m} p_i u_i$	$\max \tau = \sum_{j=1}^{n} c_j v_j + \sum_{i=1}^{m} s_i u_i$
Subject to:	*Subject to:*
$v_j - a_{ij}u_i \le b_{ij}$ $i = 1, 2, \ldots, m;$ $j = 1, 2, \ldots, n$	$v_j - t_{ij}u_i \le 0$ $i = 1, 2, \ldots, m;$ $j = 1, 2, \ldots, n$
	$\sum_{i=1}^{m} u_i = 1$
$u_i \ge 0$ $i = 1, 2, \ldots, m$	$u_i \ge 0$ $i = 1, 2, \ldots, m$
$v_j \ge 0$ $j = 1, 2, \ldots, n$	$v_j \ge 0$ $j = 1, 2, \ldots, n$

Table 4.3 The Dual of the component grouping problem vs. the dual of the GTP

The main difference between the above two models is, there is an additional constraint in the dual of the component grouping problem. Due to the similarity of these two models, a survey on the transportation problems is necessary.

4.3.2 A survey on the generalized transportation problem

The transportation problem (TP) is a special case of GTP when the coefficients of x_{ij} in LP4-3 are the same rather than the arbitrary number a_{ij}. The simplex method, which is suitable for any LP, can be used to solve TP, but it requires a great time if the magnitudes of m and n are large. Although TP should be formulated as an integer linear programming, the simplex method is applicable to TP because TP has the property of total unimodularity, that is, the application of any LP algorithms automatically yields an optimal integer solution [Nem88].

GTP was firstly modeled and applied in an aircraft routing problem by Fergusan and Dantzig [Fer56]. They extended a study to the case when the demand on each route is unknown. Hadley, G. [Had67] presented and compared the usage of the simplex and the stepping stone methods to both TP and GTP, the latter algorithm showed to have a much better efficiency. The loop technique of the stepping stone algorithm for the transportation problem was extended to GTP by Balas and Ivanescu [Bal64], later, a dual method and the poly-ω technique were specialized to solve the same problem [Bal66]. Balachandran and Thompson presented a series of papers on the development of an operator theory of parametric programming for the GTP [Bal75a, Bal75b, Bal75c, Bal75d]. Thompson and Sethi [Tho86] developed a pivot and probe algorithm to solve the

generalized transportation problem with side constraints, the result showed that the method was able to solve the problem very quickly with relatively few pivots and without using basis re-inversion. More practically, Eisemann, K. [Eis64] focused the usage of GTP in the machine loading problem, a generalized stepping stone method was developed to deal with many large-scale problems. Ji, *et al.* [Jip94] applied GTP to SMT production scheduling, and from the idea of the revised simplex method they developed an efficient algorithm for the dual model.

4.4 The Revised Simplex Method

4.4.1 Principles of the revised simplex method

The revised simplex method is one of the variations of the simplex method that can enhance the computational efficiency of the simplex method. It utilizes the exact same steps used in the simplex algorithm. The only difference is that in the simplex method, the next iteration is computed by row operations with data in the simplex tableau whereas the revised method is to use matrix operations. The crux of the computation is rooted in the basis matrix $\mathbf{B}$ and its inverse $\mathbf{B}^{-1}$.

The idea of the revised method is to use the current basis inverse $\mathbf{B}^{-1}$ to carry out the necessary computation for determining the entering and leaving variables. One of the greatest advantages of the revised version is that it solves a linear programming problem by the simplex algorithm, without having all the data in the model explicitly in hand at any time. It calculates the coefficients in the equations only when they are needed, and thus, leads to fewer computations.

If there is an optimization problem of: max z = $\underline{CX}$, subject to ($\underline{D}$,$\underline{I}$)$\underline{X}$ = $\underline{A}$, $\underline{X} \geq 0$, then its general simplex tableau in matrix form can be described as shown in Table 4.4, and the detailed generation has been proved by Taha [Tah97]. In the general simplex tableau, vector $\underline{X}$ is partitioned into $\underline{X}_I$ and $\underline{X}_{II}$ such that $\underline{X}_I$, $\underline{X}_{II}$ correspond to the original decision variables and the slack variables. Besides, vector $\underline{C}$ is partitioned as $\underline{C}_I$, $\underline{C}_{II}$ and is the cost of $\underline{X}_I$ and $\underline{X}_{II}$, respectively. $\underline{B}$ is the feasible basis, $\underline{X}_B$ is the basic variables, and $\underline{C}_B$ is the cost coefficient, which corresponds to $\underline{X}_B$. The simplex method starts with a feasible basis, $\underline{B}$, and then moves to a new feasible basis, $\underline{B}_{next}$, which produces a better value of the objective function until the optimal solution is eventually reached. Thus the basis $\underline{B}$ is the principal element that drives the computations in the simplex method.

Basic	$\underline{X}_I$	$\underline{X}_{II}$	Solution
$\underline{Z}$	$\underline{C}_B\underline{B}^{-1}\underline{D}-\underline{C}_I$	$\underline{C}_B\underline{B}^{-1}-\underline{C}_{II}$	$\underline{C}_B\underline{B}^{-1}\underline{A}$
$\underline{X}_B$	$\underline{B}^{-1}\underline{D}$	$\underline{B}^{-1}$	$\underline{B}^{-1}\underline{A}$

Table 4.4: A general simplex tableau

For those problems that should add both artificial and slack variables to make all constraints become equalities in simplex method, the dual simplex takes the benefit of ignoring the usage of artificial variables. In formulating the mathematical model for a simplex iteration, those "≥" inequality constraints will be firstly added a minus variable, called an artificial variable, and followed by adding a slack variable to make the constraint to be equal. But the principle of formulating model for the dual simplex is to multiple –1 to both the right-hand side and the left-hand side when there is a "≥" inequality constraint. This modification will make a "≤" inequality constraint. After that, a slack variable

can be added to make the inequality to be equal, so no artificial variables will be added. In this case, the right-hand side, $\underline{\mathbf{A}}$, probably has a negative value in Table 4.4, this causes an infeasible starting basic, $\underline{\mathbf{B}}$, in the dual simplex iteration. During the iteration, $\underline{\mathbf{B}}$ will move to a new infeasible basic, $\underline{\mathbf{B}}_{next}$, which produces a better basis solution until the basic solution is feasible and consequently, it is optimal. The basic iteration steps for both simplex and the dual simplex are the same, except the selection procedures of entering and leaving variables are different.

4.4.2 Ideas for the component grouping problem

In order to present the algorithm developed for the component grouping problem, the simplex tableau in Table 4.4 is rewritten as follows:

	$\mathbf{X_B}$	$\mathbf{Z}$
$\mathbf{X_I}$	$\mathbf{DB^{-1}}$	$\mathbf{DB^{-1}C_B - C_I}$
$\mathbf{X_{II}}$	$\mathbf{B^{-1}}$	$\mathbf{B^{-1}C_B - C_{II}}$
solution	$\mathbf{AB^{-1}}$	$\mathbf{AB^{-1}C_B}$

Table 4.5: A general revised simplex tableau.

Actually, Table 4.5 is the transpose of Table 4.4 and it is defined that $\underline{\mathbf{D}}^T = \mathbf{D}$, $\underline{\mathbf{B}}^T = \mathbf{B}$, and so on. For example, take a transpose of the matrix multiplication $\underline{\mathbf{B}}^{-1}\underline{\mathbf{D}}$ in Table 4.4, since $[\underline{\mathbf{B}}^{-1}\underline{\mathbf{D}}]^T = [\underline{\mathbf{D}}]^T[\underline{\mathbf{B}}^{-1}]^T$, so it becomes $\mathbf{DB}^{-1}$ in Table 4.5. In the format of linear programming (LP4-1) in the component grouping problem, all the constraint sets are in "≥" type, then, the use of the revised simplex algorithm will lead to a lot of artificial variables. So, the revised dual simplex method is a preferable algorithm.

By applying the revised dual simplex method to the component grouping problem, the initial simplex tableau is shown at Table 4.6. Since Table 4.4 is for

a maximizing problem, but the component grouping model is to minimize the cycle time, so the elements in the last row in Table 4.6, that is, the coefficients of the objective, are all negative.

		$\mathbf{X_B}=$	u_1	u_2	...	u_m	v_1	...	v_n		Z
	T		1	1	1	1	0	0	0		1
	x_{11}		$-t_{11}$	0	0	0	1	0	0		0
	⋮		⋮	0	0	0	0	1	0		0
	x_{1n}		$-t_{1n}$	0	0	0	0	0	1		0
	x_{21}		0	$-t_{21}$	0	0	1	0	0		0
$\mathbf{X_I}=$	⋮	$\mathbf{D}=$	0	⋮	0	0	0	1	0	$\mathbf{C_I}=$	0
	x_{2n}		0	$-t_{2n}$	0	0	0	0	1		0
	⋮		0	0	⋱	0	0	⋱	0		0
	x_{m1}		0	0	0	$-t_{m1}$	1	0	0		0
	⋮		0	0	0	⋮	0	1	0		0
	x_{mn}		0	0	0	$-t_{mn}$	0	0	1		0
	u_1		1	0	0	0	0	0	0		0
	u_2		0	1	0	⋮	⋮	⋮	0		0
	⋮		0	0	1	0	⋮	⋮	0		0
$\mathbf{X_{II}}=$	u_m	$\mathbf{B}=[b_{ij}]$	0	⋮	0	1	0	⋮	0	$\mathbf{C_{II}}=$	0
	v_1		0	⋮	⋮	0	1	0	0		0
	⋮		0	⋮	⋮	⋮	0	1	0		0
	v_m		0	0	0	0	0	0	1		0
		$\mathbf{A}=$	$-s_1$	$-s_2$	...	$-s_m$	$-c_1$	...	$-c_n$		0

Table 4.6: Initial tableau for the component grouping problem

The matrix of the constraint coefficients, **D**, is sparse, and it is a $[(mn + 1) \times (m + n)]$ matrix, but only $2mn + m$ elements are non-zeros. Similarly, in $\mathbf{C_I}$ and $\mathbf{C_{II}}$, only one has a coefficient of value 1, all others are zero. Obviously, the application of the classic simplex method to this problem will waste a lot of time. Besides, the zero values in the cost coefficients cause a difficulty in selecting a

suitable pivot column since all variables have the same coefficients, thus the computing efficiency is low. Here, a better initial basis for the component grouping problem is designed and the iteration steps are simplified. The initial basis saves some iterations, whereas the simplification of iteration steps can reduce some operations.

Traditionally, the initial basis, **B**, should be an identity matrix, **I**, with the sizes of $[(m + n)\times(m + n)]$ such that $\mathbf{B} = \mathbf{B}^{-1}$ and $\mathbf{X_B}$ starts with slack variables, say, $\mathbf{X_B} = (u_1 \ldots u_m\ v_1 \ldots v_n)$. But the special data structure of the component grouping problem allows a new initial basis **B** to be available by exchanging the first row of **B** with the first row of **D** in Table 4.6, and then multiplying –1 to the rest elements. So, instead of setting the initial basis as $\mathbf{B} = \mathbf{I}$, the initial basis is set as, $\mathbf{B} = \begin{bmatrix} 1 & \mathbf{1} & 0 \\ 0 & -\mathbf{I} & 0 \\ 0 & 0 & -\mathbf{I} \end{bmatrix}$, and **B** is still equal to its inverse (a proof will be given in the next section). Whereas, the initial setting of other elements become, $\mathbf{X_B} = (-T\ -u_2\ \cdots\ -u_m\ -v_1\ \cdots\ -v_n)$, $\Gamma = \{(m + 1, n + 1), (2, 0), \ldots, (m, 0), (0, 1), \ldots(0, n)\}$. Here, Γ is the basic cells that correspond to $\mathbf{X_B}$. From Table 4.6, $\mathbf{X_{II}}$ contains all the slack variables, these variables are divided into two parts, $\mathbf{X_{II}} = [u_i, v_j]$, and the coefficient matrix corresponding to $\mathbf{X_{II}}$ under the new initial solution becomes $\mathbf{B}^{-1}\mathbf{C_B} = \begin{bmatrix} 1 & \mathbf{1} & 0 \\ 0 & -\mathbf{I} & 0 \\ 0 & 0 & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} = [1\ \mathbf{0}\ \mathbf{0}]^T$, i.e., except u_1, all other coefficients are zero.

After the starting basis and its related information are calculated, the optimal solution can be obtained by the following procedure from the idea of the revised dual simplex method:

Step 1. The current values of the basic variables are computed as $\mathbf{X_B} = \mathbf{AB^{-1}}$. Select the leaving variable, at the kth position that having the most positive value, then the kth element in Γ is the leaving cell. If all the elements of $\mathbf{X_B}$ are negative, stop; and the current solution is optimal. Otherwise, go to step 2.

Step 2: The entering variable is dependent on the ratio of cost and constraint coefficients for all the non-basic cells.

a) Firstly, the kth column in $\mathbf{B^{-1}}$ is divided into two parts, $[b_{ik}]^T = [p_1 \cdots p_m \; q_1 \cdots q_n]^T = [p_i \; q_j]^T$.

b) Compute the cost coefficients, $\mathbf{DB^{-1}C_B} - \mathbf{C_I}$, for all the non-basic variables, i.e., $(i, j) \notin \Gamma$. As T is the basic variable from the first to the last iteration, the data of the first row at **D** will not be changed, thus, in the following calculation, $\mathbf{C_I}$ and $\mathbf{C_{II}}$ are all zeros, so $\mathbf{DB^{-1}C_B - C_I} = \mathbf{DB^{-1}C_B}$. The matrix **D** described at Table 4.6 (without the first row) can be presented as, $\mathbf{D} = \begin{bmatrix} -t_{1j} & 0 & 0 & \mathbf{I} \\ 0 & \ddots & 0 & \mathbf{I} \\ 0 & 0 & -t_{mj} & \mathbf{I} \end{bmatrix} = [-t_{ij} \; \mathbf{I}]$. As the values of $\mathbf{B^{-1}C_B}$ have been calculated at step 1 and are equal to $[u_i \; v_j]$, the cost coefficients become $[-t_{ij} \; \mathbf{I}] \begin{bmatrix} u_i \\ v_j \end{bmatrix} = [-t_{ij}u_i + v_j]$. Similarly,

the constraint coefficients for the non-basic variables in the kth column are $[-t_{ij}\ \mathbf{I}]\begin{bmatrix} p_i \\ q_j \end{bmatrix} = [-t_{ij}p_i + q_j]$.

c) The entering variable is associated with

$$w_{i_e j_e} = \min(w_{ij}),\ \text{where}\ w_{ij} = \frac{-t_{ij}u_i + v_j}{-t_{ij}p_i + q_j},\ \text{if}\ -t_{ij}p_i + q_j < 0.$$

Now, the entering variable is $x_{i_e j_e}$.

Step 3. Obtain a new basis by interchanging the row data corresponding to entering and leaving vectors using formula $\mathbf{B_{next}} = \mathbf{B}^{-1}\mathbf{E}$. Following the procedure of obtaining $\mathbf{E}$ described in [Tah97], $\mathbf{B_{next}} = [\hat{b}_{ij}]$

$$= \begin{bmatrix} b_{11} & \cdots & b_{1k} & \cdots & b_{1(m+n)} \\ \vdots & \ddots & \cdots & \cdots & \vdots \\ b_{k1} & \cdots & b_{kk} & \cdots & b_{k(m+n)} \\ \vdots & \cdots & \cdots & \ddots & \vdots \\ b_{(m+n)1} & \cdots & b_{(m+n)k} & \cdots & b_{(m+n)(m+n)} \end{bmatrix} \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \frac{t_{i_e j_e} b_{i_e 1} - b_{(m+j_e)1}}{-t_{i_e j_e} p_{i_e} + q_{j_e}} & \cdots & \frac{1}{-t_{i_e j_e} p_{i_e} + q_{j_e}} & \cdots & \frac{t_{i_e j_e} b_{i_e (m+n)} - b_{(m+j_e)(m+n)}}{-t_{i_e j_e} p_{i_e} + q_{j_e}} \\ 0 & 0 & 0 & 1 & \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} b_{11} + \frac{(t_{i_e j_e} b_{i_e 1} - b_{(m+j_e)1}) b_{1k}}{-t_{i_e j_e} p_{i_e} + q_{j_e}} & \cdots & \frac{b_{1k}}{-t_{i_e j_e} p_{i_e} + q_{j_e}} & \cdots & b_{1(m+n)} + \frac{(t_{i_e j_e} b_{i_e 1} - b_{(m+j_e)1}) b_{1k}}{-t_{i_e j_e} p_{i_e} + q_{j_e}} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ b_{k1} + \frac{(t_{i_e j_e} b_{i_e k} - b_{(m+j_e)k}) b_{kk}}{-t_{i_e j_e} p_{i_e} + q_{j_e}} & \cdots & \frac{b_{kk}}{-t_{i_e j_e} p_{i_e} + q_{j_e}} & \cdots & b_{k(m+n)} + \frac{(t_{i_e j_e} b_{i_e k} - b_{(m+j_e)k}) b_{kk}}{-t_{i_e j_e} p_{i_e} + q_{j_e}} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ b_{(m+n)1} + \frac{(t_{i_e j_e} b_{i_e m} - b_{(m+j_e)m}) b_{(m+n)k}}{-t_{i_e j_e} p_{i_e} + q_{j_e}} & \cdots & \frac{b_{(m+n)k}}{-t_{i_e j_e} p_{i_e} + q_{j_e}} & \cdots & b_{(m+n)(m+n)} + \frac{(t_{i_e j_e} b_{i_e m} - b_{(m+j_e)m}) b_{(m+n)k}}{-t_{i_e j_e} p_{i_e} + q_{j_e}} \end{bmatrix}$$

The above formulation can be simplified. It is noted that the kth column has a general form of $\hat{b}_{ik} = \frac{b_{ik}}{-t_{i_e j_e} p_{i_e} + q_{j_e}}$, and the elements for the other columns are $\hat{b}_{ij} = b_{ij} + (t_{i_e j_e} b_{i_e j} - b_{(m+j_e)j})\hat{b}_{ik}$. The new cost coefficients for $\mathbf{X_{II}}$ can be obtained by using the above formulation: $\hat{u}_i = u_i - w_{i_e j_e} p_i$,

$\hat{v}_j = v_j - w_{i_e j_e} q_j$ and the objective function can be updated as $\tau = \sum_{j=1}^{n} c_j v_j + \sum_{i=1}^{m} s_i u_i$. Set $u_i = \hat{u}_i, v_j = \hat{v}_j$, and $\mathbf{B}^{-1} = \mathbf{B}_{\mathbf{next}}^{-1}$, and go to step 1 for the next iteration.

4.5 A New Algorithm

The presentation of the simplified revised dual simplex method in the previous section shows that many operations can be eliminated due to the special data structure of the component grouping problem in PCB assembly. The following is the procedure of the algorithm for the problem, implemented by the computer language C++ (see Appendix IV).

4.5.1 Initialization

(1) Set $v_j = 0$, ($j = 1, 2, ..., n$) and $u_1 = 1$, $u_i = 0$, ($i = 2, 3, ..., m$), that is,

$$\mathbf{V} = [v_j] = 0, \qquad j = 1, 2, ..., n$$

$$\mathbf{U} = [u_i] = \begin{cases} 1 & i = 1 \\ 0 & i = 2,3,...,m \end{cases}$$

(2) Let the basic matrix $\mathbf{B} = \mathbf{B}^{-1} = [b_{ij}]$; $\qquad i, j = 1, 2, ..., m + n$

$$\text{where } b_{ij} = \begin{cases} 1 & i = 1; j = 1,2,...,m \\ -1 & i = j = 2,3,...,m+n \\ 0 & \text{otherwise} \end{cases}$$

(3) The basic cell set is $\Gamma = \{(m + 1, n + 1), (2, 0), (3, 0), ..., (m, 0), (0, 1), (0, 2), ..., (0, n)\}$, and let $\mathbf{A} = (-s_1, -s_2, ..., -s_m, -c_1, -c_2, ..., -c_n)$.

(4) Get the initial objective by computing:

$$\tau = \sum_{j=1}^{n} c_j v_j + \sum_{i=1}^{m} s_i u_i$$

The starting basis **B** can be divided into four sub-matrices as follows:

- The upper left sub-matrix is an $m \times m$ matrix with the following structure: the elements in the first row are all 1, while the elements in the first column are all 0 except the first element b_{11} is 1. Other elements in this sub-matrix form an $(m-1) \times (m-1)$ negative identity matrix.
- The upper right sub-matrix is an $m \times n$ zero matrix.
- The lower left sub-matrix is an $n \times m$ zero matrix.
- The lower right sub-matrix is an $n \times n$ negative identity matrix.

To prove that $\mathbf{B} = \mathbf{B}^{-1}$, **B** is restructured as 9 sub-matrices, $\mathbf{B} = \begin{bmatrix} 1 & \mathbf{1} & 0 \\ 0 & -\mathbf{I} & 0 \\ 0 & 0 & -\mathbf{I} \end{bmatrix}$ and

$$\mathbf{B}\times\mathbf{B} = \begin{bmatrix} 1 & \mathbf{1} & 0 \\ 0 & -\mathbf{I} & 0 \\ 0 & 0 & -\mathbf{I} \end{bmatrix} \times \begin{bmatrix} 1 & \mathbf{1} & 0 \\ 0 & -\mathbf{I} & 0 \\ 0 & 0 & -\mathbf{I} \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{1}-\mathbf{I} & 0 \\ 0 & \mathbf{I} & 0 \\ 0 & 0 & \mathbf{I} \end{bmatrix} = \mathbf{I} = \mathbf{B}\times\mathbf{B}^{-1}$$

Thus, $\mathbf{B} = \mathbf{B}^{-1}$.

4.5.2 Main procedure

Step 1 Determination of the leaving cell:

Step 1.1 Compute $\mathbf{X} = \mathbf{A}\mathbf{B}^{-1} = (x_1, x_2, \ldots, x_{m+n})$.

Step 1.2 Find the largest value x_k among the elements of **X**, i.e., $x_k = \max\{x_i\}$, $i = 1, 2, \ldots, m + n$.

Step 1.3 If $x_k \leq 0$, the solution is optimal and stop. Otherwise, the kth element in Γ is the leaving cell.

Step 2 Determination of the entering cell:

Step 2.1 Let

$$\mathbf{P} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_i \\ \vdots \\ p_m \end{bmatrix} = \begin{bmatrix} b_{1k} \\ b_{2k} \\ \vdots \\ b_{ik} \\ \vdots \\ b_{mk} \end{bmatrix} \text{ and } \mathbf{Q} = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_j \\ \vdots \\ q_n \end{bmatrix} = \begin{bmatrix} b_{(m+1)k} \\ b_{(m+2)k} \\ \vdots \\ b_{(m+j)k} \\ \vdots \\ b_{(m+n)k} \end{bmatrix}$$

Step 2.2 For all non-basic cells (i, j), i.e., $(i, j) \notin \Gamma$, if $-t_{ij}p_i + q_j < 0$, compute

$$w_{ij} = \frac{-t_{ij}u_i + v_j}{-t_{ij}p_i + q_j},$$

Step 2.3 Find the smallest value $w_{i_e j_e}$ among all w_{ij}, and the corresponding cell (i_e, j_e) is the entering cell.

Step 3 Updating:

Step 3.1 Update the basis inverse $\mathbf{B}^{-1}$.

For the elements in the kth column:

$$\hat{b}_{ik} = \frac{b_{ik}}{-t_{i_e j_e}p_{i_e} + q_{j_e}} \qquad i = 1, 2, \ldots, m + n$$

For the elements in other columns:

$$\hat{b}_{ij} = b_{ij} + (t_{i_e j_e}b_{i_e j} - b_{(m+j_e)j})\hat{b}_{ik} \qquad i, j = 1, 2, \ldots, m + n \text{ and } j \neq k$$

Step 3.2 Replace the kth cell with (i_e, j_e) in Γ.

Step 3.3 Recompute the objective:

$$\hat{u}_i = u_i - w_{i_e j_e} p_i \qquad i = 1, 2, ..., m$$

$$\hat{v}_j = v_j - w_{i_e j_e} q_j \qquad j = 1, 2, ..., n$$

$$\tau = \sum_{j=1}^{n} c_j v_j + \sum_{i=1}^{m} s_i u_i$$

Go to step 1.

4.5.3 Advantages of the new algorithm

Similar to the revised dual simplex method, the simplified algorithm for the component grouping problem starts with an initial non-feasible solution for LP4-1, but the initial solution for the dual problem LP4-2 is feasible. By adjusting the entering and the leaving cells, and updating $\mathbf{B}^{-1}$, the solution for LP4-1 becomes feasible, and the iteration stops.

Compared with the revised simplex method, the basic matrix **B** is not an identical matrix at the beginning, but is still equal to its inverse, i.e., $\mathbf{B} = \mathbf{B}^{-1}$. This better starting solution reduces iterations. Moreover, the procedure is simplified that the zero operations are reduced. Finally, it is not necessary to build the linear programming model LP4-1 or LP4-2 in this new algorithm and to introduce slack or artificial variables. The necessary information required in an iteration can be obtained directly from the original problem. In short, compared with the revised simplex algorithm, the simplified algorithm is a more convenient and faster way to obtain an optimal linear programming solution for the component grouping problem.

4.6 A Numerical Example

The simplified algorithm is applied to the example 1 in the following. The primal of example 1 was formulated in the last chapter. Let u_1, u_2, u_3 be the dual variables for the first 3 constraints and v_1, v_2, v_3, v_4 be the dual variables for the rest constraints, then the dual is:

$$max\ \tau = 110u_1 + 147u_2 + 147u_3 + 231v_1 + 24v_2 + 12v_3 + 10v_4$$

Subject to:

$$v_1 - 3u_1 \le 0; \quad v_2 - 7u_1 \le 0; \quad v_3 - 90000u_1 \le 0; \quad v_4 - 12u_1 \le 0;$$
$$v_1 - 7u_2 \le 0; \quad v_2 - 18u_2 \le 0; \quad v_3 - 19u_2 \le 0; \quad v_4 - 24u_2 \le 0;$$
$$v_1 - 23u_3 \le 0; \quad v_2 - 26u_3 \le 0; \quad v_3 - 33u_3 \le 0; \quad v_4 - 34u_3 \le 0;$$
$$u_1 + u_2 + u_3 = 1;$$
$$u_1, u_2, u_3 \ge 0 \text{ and } v_1, v_2, v_3, v_4 \ge 0.$$

4.6.1 Solution procedure

- Iteration 0 (initialization)

$$\boldsymbol{U} = [u_i] = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \qquad \boldsymbol{V} = [v_j] = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{B} = \mathbf{B}^{-1} = [b_{ij}] = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & .0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

$\Gamma = [(4, 5), (2, 0), (3, 0), (0, 1), (0, 2), (0, 3), (0, 4)]$,

$\mathbf{A} = [-110, -147, -147, -231, -24, -12, -10]$, and

$$\tau = \sum_{j=1}^{n} c_j v_j + \sum_{i=1}^{m} s_i u_i = 110.$$

- Iteration 1:

$\mathbf{X} = [x_{45}, x_{20}, x_{30}, x_{01}, x_{02}, x_{03}, x_{04}] = \mathbf{AB}^{-1} = [-110, 37, 37, 231, 24, 12, 10]$.

$x_{01} = 231$ is selected, so $k = 4$ and the 4th cell (0, 1) in Γ is the leaving cell.

$$\mathbf{P} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} b_{14} \\ b_{24} \\ b_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{and} \quad \mathbf{Q} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} b_{44} \\ b_{54} \\ b_{64} \\ b_{74} \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

For non-basic cells with $-t_i p_i + q_j < 0$, $w_{i_e j_e} = \min\{w_{11}, w_{21}, w_{31}\} = \min\left\{\frac{-3}{-1}, \frac{0}{-1}, \frac{0}{-1}\right\} = 0 = w_{21}$. So, (2, 1) is the entering cell, it will replace the basic cell (0, 1) in Γ.

The kth or the 4th column in the inverse matrix $\mathbf{B}^{-1}$ becomes:

$$\begin{bmatrix} \hat{b}_{14} \\ \hat{b}_{24} \\ \hat{b}_{34} \\ \hat{b}_{44} \\ \hat{b}_{54} \\ \hat{b}_{64} \\ \hat{b}_{74} \end{bmatrix} = \frac{1}{-t_{21}p_2 + q_1} \begin{bmatrix} \hat{b}_{14} \\ \hat{b}_{24} \\ \hat{b}_{34} \\ \hat{b}_{44} \\ \hat{b}_{54} \\ \hat{b}_{64} \\ \hat{b}_{74} \end{bmatrix} = \frac{1}{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

whereas, the elements of the inverse matrix $\mathbf{B}^{-1}$ in column 1 are:

$$\begin{bmatrix}\hat{b}_{11}\\ \hat{b}_{21}\\ \hat{b}_{31}\\ \hat{b}_{41}\\ \hat{b}_{51}\\ \hat{b}_{66}\\ \hat{b}_{71}\end{bmatrix} = \begin{bmatrix}b_{11}\\ b_{21}\\ b_{31}\\ b_{41}\\ b_{51}\\ b_{61}\\ b_{71}\end{bmatrix} + (t_{21}b_{21} - b_{41}) \begin{bmatrix}\hat{b}_{14}\\ \hat{b}_{24}\\ \hat{b}_{34}\\ \hat{b}_{44}\\ \hat{b}_{54}\\ \hat{b}_{64}\\ \hat{b}_{74}\end{bmatrix} = \begin{bmatrix}1\\ 0\\ 0\\ 0\\ 0\\ 0\\ 0\end{bmatrix} + (7 \times 0 - 0) \begin{bmatrix}0\\ 0\\ 0\\ 1\\ 0\\ 0\\ 0\end{bmatrix} = \begin{bmatrix}1\\ 0\\ 0\\ 0\\ 0\\ 0\\ 0\end{bmatrix}$$

Similarly, other elements of $\mathbf{B}^{-1}$ can be updated, and

$$\mathbf{B}^{-1} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -7 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

$\Gamma = [(4, 5)\ (2, 0)\ (3, 0)\ (2, 1)\ (0, 2)\ (0, 3)\ (0, 4)]$

Now, the solution becomes,

$$\begin{bmatrix}\hat{u}_1\\ \hat{u}_2\\ \hat{u}_3\end{bmatrix} = \begin{bmatrix}u_1\\ u_2\\ u_3\end{bmatrix} - w_{21} \begin{bmatrix}p_1\\ p_2\\ p_3\end{bmatrix} = \begin{bmatrix}1\\ 0\\ 0\end{bmatrix} - 0 \times \begin{bmatrix}0\\ 0\\ 0\end{bmatrix} = \begin{bmatrix}1\\ 0\\ 0\end{bmatrix} \quad \text{and}$$

$$\begin{bmatrix}\hat{v}_1\\ \hat{v}_2\\ \hat{v}_3\\ \hat{v}_4\end{bmatrix} = \begin{bmatrix}v_1\\ v_2\\ v_3\\ v_4\end{bmatrix} - w_{21} \begin{bmatrix}q_1\\ q_2\\ q_3\\ q_4\end{bmatrix} = \begin{bmatrix}0\\ 0\\ 0\\ 0\end{bmatrix} - 0 \times \begin{bmatrix}-1\\ 0\\ 0\\ 0\end{bmatrix} = \begin{bmatrix}0\\ 0\\ 0\\ 0\end{bmatrix}$$

$$\tau = \sum_{j=1}^{n} c_j v_j + \sum_{i=1}^{m} s_i u_i = 110$$

- Final iteration:

After 7 more iterations, the final solution of this problem is as follows:

$$\mathbf{B}^{-1} = \begin{bmatrix} 0.617 & 0 & -0.128 & 0.128 & -0.016 & 0.016 & 0 \\ 0.265 & 0 & 0.088 & -0.088 & -0.007 & 0.007 & 0 \\ 0.118 & 0 & 0.040 & -0.040 & 0.023 & -0.023 & 0 \\ 1.85 & 0 & 0.617 & 0.383 & -0.049 & 0.049 & 0 \\ 3.17 & 1 & 1.06 & -1.06 & -0.084 & 0.084 & 0 \\ 4.5 & 0 & 1.5 & -1.5 & 0.882 & 0.118 & 0 \\ 3.55 & 0 & 1.18 & -1.18 & 0.696 & -0.696 & 1 \end{bmatrix},$$

$\Gamma = [(4, 5) (2, 3) (1, 1) (2, 1) (3, 2) (1, 2) (3, 4)]$,

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 0.589 \\ 0.252 \\ 0.159 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} 1.767 \\ 4.123 \\ 4.796 \\ 5.391 \end{bmatrix}$$

$$\tau = \sum_{j=1}^{n} c_j v_j + \sum_{i=1}^{m} s_i u_i = 743.8$$

Now, $\mathbf{X} = (x_{45}, x_{23}, x_{11}, x_{21}, x_{32}, x_{12}, x_{34}) = (-743.8, -12, -178.31, -52.686, -9.877, -14.123, -10.0)$, all elements in **X** are negative, so the solution is optimal and the objective: $T = \tau = 743.8$.

4.6.2 Result analysis

The optimal solution obtained is not integer, and it is meaningless to have a fraction assignment. In order to make the grouping meaningful, the next task is to either use an integer programming technique, like B&B, to seek the optimal integer solution, or round off the fraction values to integers. If the latter is applied, the solution becomes $x_{23} = 12$, $x_{11} = 178$, $x_{21} = 53$, $x_{32} = 10$, $x_{12} = 14$, $x_{34} = 10$, and the cycle time is 747. Actually, the optimal integer solution is $x_{23} = 11$,

$x_{11} = 179$, $x_{21} = 52$, $x_{32} = 10$, $x_{12} = 14$, $x_{34} = 9$, $x_{33} = 1$, $x_{24} = 1$ with the cycle time of 746 (in Chapter three). There is a difference of 0.1 second, and the relative error is about 0.13% only, which is acceptable in the real situation for a company. By the way, it is worth to point out that, although the optimal assignment can save 0.1 second theoretically, in the real situation, it is better to combine $x_{33} = 1$ with x_{23}, and $x_{24} = 1$ with x_{34}, respectively. This is because setting up a machine for one component is impractical. In this case, the modified integer linear programming solution will be the same with the linear programming solution obtained by the new algorithm.

4.7 Conclusion

In this chapter, a survey on the assignment problems, the simple assembly line balancing problems and the generalized transportation problem was carried out in the hope of finding an existing algorithm for the component grouping problem. However, due to the unique data structure in the component grouping problem no special algorithms are available. Instead, from the idea of the revised simplex method, a new algorithm is proposed. This new algorithm has some advantages. Firstly, a new starting basis, **B**, can reduce the number of iterations required in the revised simplex method. Secondly, it erases the unnecessary computation induced by the sparse structure. Finally, it is not necessary to formulate the component grouping problem as a mathematical model.

Obviously, the results obtained by the new algorithm are not integer. To make it practical, the decimal solution can be rounded off to integers. This

solution is most likely not optimal but it is near optimal and is acceptable in engineering.

In Chapter 5, the genetic algorithm technique is selected as a heuristic technique to search a near optimal solution for the same problem. This near optimal solution will be compared with the one obtained by the linear programming approach with rounding off. The performance of the genetic algorithm in the component grouping problem will also be investigated.

Chapter Five

A GENETIC ALGORITHM APPROACH

5.1 Introduction

In Chapter 3, the component grouping problem in PCB assembly was firstly constructed as a min-max type integer programming model. Yu and Kouvelis discussed the complexity of a min-max type problem [Yug93]. They proved that the min-max assignment problem is NP-hard. Actually, the min-max type component grouping model (i.e., LP3-1 or LP3-3) is more difficult than the min-max assignment problem. First of all, in the min-max assignment problem, the decision variables are limited to be binary, that is, $x_{ij} = 0$ or 1 and the number of jobs should be equal to the number of tasks, i.e., $n = m$. However, in the component grouping model, generally, $n \neq m$, and x_{ij} can be any non-negative integer number. Furthermore, the objective function of model LP3-1 or LP3-3 is different from that of the min-max assignment problem. As a matter of fact, the component grouping problem is a general integer programming formulation. Papadimitriou [Pap81] studied the complexity of the general integer programming, and he concluded that the general integer programming is NP-complete. Consequently, the component grouping problem is NP-complete, which is the "hardest" of all problems in NP and it requires extremely long time in finding an optimal integer solution. Therefore, based on model LP3-3, a heuristic technique, Genetic Algorithms (GAs), is applied to seek for an optimal grouping. The algorithm is implemented by a computer language, Matlab (Appendix V), and illustrated by a numerical example.

5.2 The Genetic Algorithm Technique

Genetic algorithms (GAs) are a stochastic search technique based on the mechanism of natural selection of natural genetics. It starts with a set of random solutions called a population. Each member in the population, called a chromosome, encodes a solution to the problem. Each chromosome is evaluated by measuring its fitness value. A selection technique is used to pick up a chromosome from the population to generate new solutions. The fitter the chromosomes, the higher the probabilities of being selected. The operations to generate new solutions are crossover and mutation. The solutions generated by these operations, called offspring, are evaluated by their fitness value and may become one of the chromosomes in the population and the 'quality' of the population will be improved after each generation.

5.2.1 Elements of the genetic algorithms

In detail, if GAs are applied, the following elements should be defined [Cha95]:

- Technique for encoding solutions

Chromosomes are often bit strings, and they are capable of encoding a wide variety of information, and have been shown to be effective representation mechanisms in unexpected domains. Certainly, there are many other types of encoding techniques.

- Way to create an initial population of chromosomes

The initial chromosomes can be generated by any initialization techniques, such as weighted random initialization. As the critical features of the

final solution will be produced by the search and recombination mechanisms of the algorithm, rather than the initialization procedures, an initial solution generated randomly is acceptable.

➢ Fitness measurement

The objective function provides the mechanism for evaluating a chromosome. If the chromosome is a bit string, in order to measure its fitness, the bit strings should firstly be converted into real numbers, then, the real numbers are placed into the objective function in the problem. The value returned from the objective function will be the evaluation of the chromosome.

➢ Selection operation

An operator selects chromosomes in the population for doing genetic operations. Four selection techniques are commonly used, they are: Roulette, Random, Fit-Fit, and Fit-Weak [Cha95].

- Roulette selection is one of the traditional GA selection techniques. The principle of the roulette selection is a linear search through a roulette wheel with slots in the wheel that is weighted in proportional to the individual's fitness value.
- Random selection technique randomly selects a parent from the population. It is a little more disruptive then the roulette selection.
- Fit-Fit selection pairs an individual with the next fittest individual in the population by simply picking through the ordered list of individuals.
- Fit-Weak selection is maximally disruptive of genetic information. The fittest individual is paired with least fit. The next fittest is paired with the next least fit, and so on.

➢ Genetic operations, including crossover and mutation.

(a) Crossover operation:

This operator roughly mimics biological recombination between two single chromosome organisms. It operates on two chromosomes at a time and generates offspring by combining both chromosomes' features. One-Point crossover is the simplest crossover technique. This technique is to choose a random cut point and generate the offspring by combining the segment of one parent to the left of the cut point, with the segment of the other parent to the right of the cut point, as shown in Fig. 5.1.

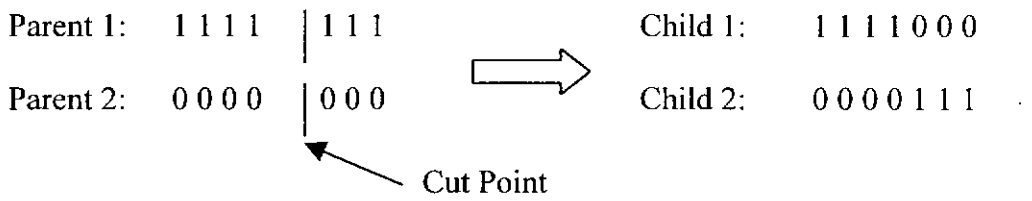


Figure 5.1: One-point crossover

Some other crossover techniques are similar to the one-point crossover, namely,

- *n*-point crossover: The idea behind *n*-point crossover is to align two chromosomes together, cut the chromosomes at identical places into $n + 1$ fragments, and create two new children by mixing the fragments.
- *n*-point shuffle crossover: *n*-point shuffle crossover is similar to *n*-point crossover. The procedure is identical, except that the genes are shuffled before crossover, and unshuffled after crosssover.
- uniform crossover: In uniform crossover, if two parent chromosomes have length l, each parent copies l/2 genes to each child. The genes are chosen independently.

- variable crossover: In the variable crossover, each chromosome is prefixed with extra bits, the extra bits determine the crossover method to be chosen.

(b) Mutation operation:

Mutation is used to increase the variability of the population. It provides a small amount of random search, each fundamental unit (bit, position or token) in a structure has a finite probability of changing. If mutation is applied to a bit string, it sweeps down the list of bits, replaces each by a randomly selected bit if a probability test is passed. Fig. 5.2 is an example of the operation of bit mutation with the probability test set at 0.009. The bit string has the length of four, so four random numbers will be generated. If the random number passes the probability test, a new bit will be created and replaces the old one.

Old chromosome	Random Numbers				New bit	New chromosome
0 0 1 0	0.852	0.564	0.256	0.001	1	0 0 1 1

Figure 5.2: An example for bit mutation

5.2.2 Operation parameters

5.2.2.1 Population size

The population size for the search is set by a GA user from the requirements of the model. Chamber, L. [Cha95] pointed out that a small population size causes the loss of genetic diversity and compromises the search. As a small population would be more likely to converge quickly on a local optimum, a comparatively fit individual in a small starting population will be selected for generating offspring, and its descendants will quickly come to dominate, further limit genetic diversity. Also, with the sparse spread of initial

points, better optima may never be visited before the search converges to a poor local optimum. Thus, in the handbook he suggested to use a large population size, although it may cause an expensive time. On the other hand, Davis, L. [Dav91] claimed that it is difficult to decide whether a user should run a smaller population several times or a single time but using a larger population size.

5.2.2.2 Crossover rate

Crossover rate is defined as the ratio of the number of offspring produced in each generation to the population size. This ratio controls the expected number of chromosomes to undergo the crossover operation. A higher crossover rate allows exploration of more solution space and reduces the chances of settling for a false optimum; but if this rate is too high, it results in the wastage of a lot of computation time in exploring unpromising regions of the solution space [Gen97].

5.2.2.3 Mutation rate

Mutation rate is defined as the percentage of the total number of genes in the population. It controls the rate at which new genes are introduced into the population for trial of mutation. If it is too low, many genes that would have been useful are never tried out. But if it is too high, there will be much random perturbation, the offspring will lose their resemblance to their parents, and the algorithm will lose the ability to learn from the history of search [Gen97].

5.2.3 Advantages of genetic algorithms

Genetic algorithms have a good performance in a large, complex, and poorly understood search space, where classical search tools are inappropriate. It is a technique that avoids many shortcomings exhibited by local search techniques on difficult search spaces. Many search techniques search from a single point, but GAs work from a population. By maintaining a population of well-adapted sample points, the probability of reaching a false peak is reduced. The detailed information on the genetic algorithms is available at [Buc92, Mit96]. In the following sections, a genetic algorithm for the component grouping problem is proposed.

5.3 A New Algorithm for the Component Grouping Problem

The general structure of the genetic algorithm for the component grouping problem is shown in Fig. 5.3. The algorithm starts with an initial population, which is generated randomly, and then the chromosomes in the population will be measured from the objective function presented at model LP3-3. Roulette selection is used to choose chromosomes for the genetic operations, i.e., crossover and mutation to generate new genes. The traditional crossover techniques like n-point or uniform crossover are not suitable for the component grouping problem. This is due to the fact that traditional chromosomes are in the format of vectors, whereas they are in a matrix format in the component grouping problem. Therefore, new crossover and mutation mechanisms are designed in this project. After an offspring are generated, its fitness will be measured and it may become a member of population provided it has a satisfactory fitness value.

Then, a new roulette wheel is performed and the iteration repeats until a predetermined number of iterations is conducted.

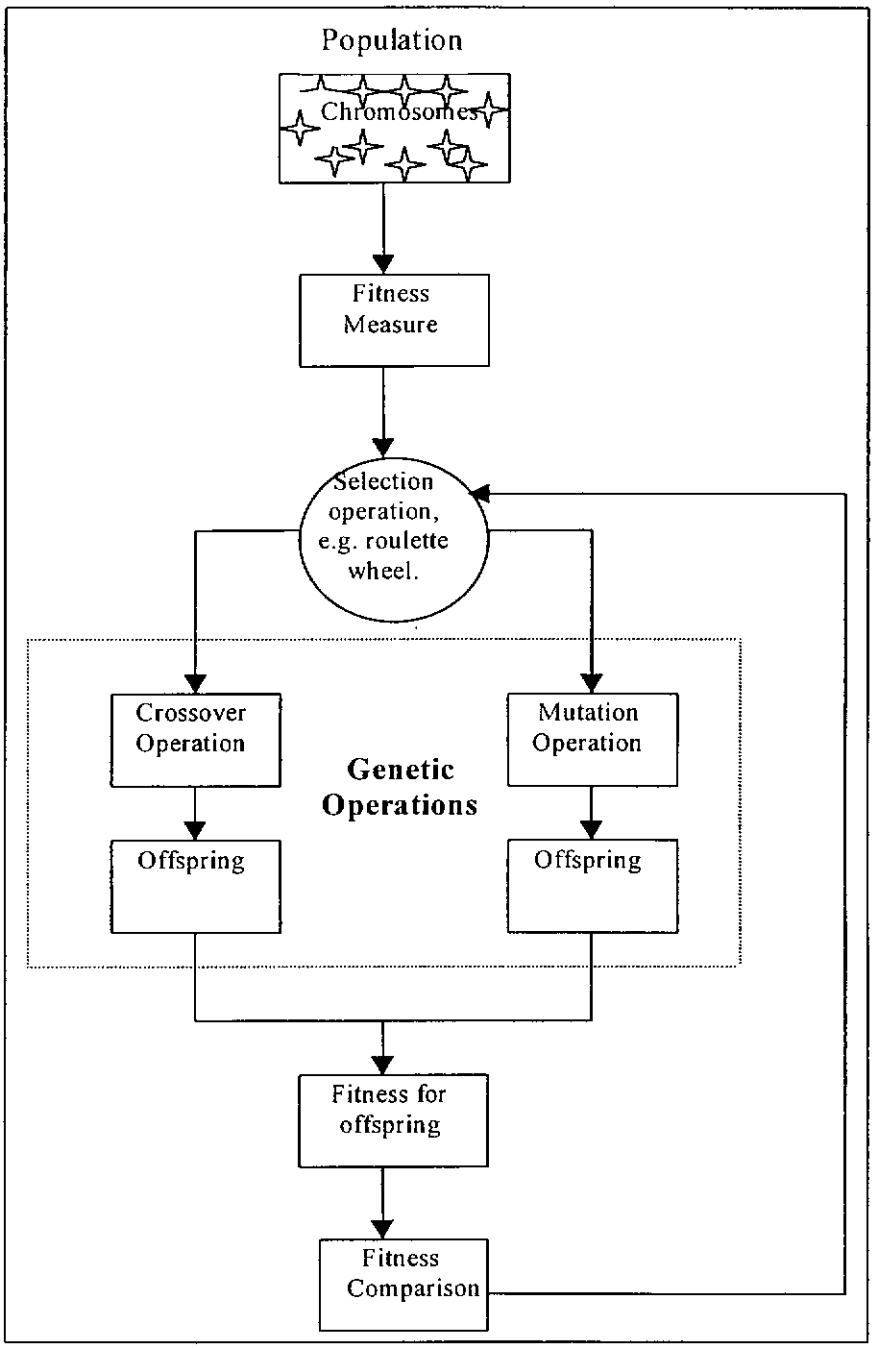


Figure 5.3: The general structure of the genetic algorithm

5.3.1 Initialization

The following initialization procedure is used to generate a feasible chromosome for the PCB component grouping problem and it will fulfill all the constraint sets stated in model LP3-3.

Step 1: Assign an integer number to b_i randomly, such that:

$$\sum_{i=1}^{m} b_i = \sum_{j=1}^{n} c_j$$

Step 2: Initialize x_{ij}.

Step 2.1 Select a random number k from set π, $\pi = \{1, 2, ..., mn\}$.

Step 2.2 Calculate the corresponding row and column numbers i and j by: $i = \lfloor (k-1)/n + 1 \rfloor$ and $j = (k-1) \bmod n + 1$;

Step 2.3 Assign the available amount to x_{ij}, i.e., $x_{ij} = \min\{b_i, c_j\}$.

Step 2.4 Update b_i and c_j: $b_i = b_i - x_{ij}$; $c_j = c_j - x_{ij}$; and delete k from π.

Step 2.5 Repeat Step 2.1 to 2.4 until there is no more element in π.

It is noted that the chromosome generated from the above steps is feasible, but may not be optimal. The chromosome population size is *psize* (a predetermined value), so the above procedure will repeat by *psize* times to generate *psize* chromosomes.

5.3.2 Evaluation function

In genetic algorithms, the chromosomes in the population and the offspring must be evaluated by some measures of fitness. In the PCB component grouping problem, the objective function in LP3-3 is used to measure the fitness. So, the fitness value for the kth chromosome is calculated as:

$$eval(X_k) = \max\left\{ s_i + \sum_{j=1}^{n} t_{ij} x_{i_k j_k} \quad | \, i = 1, 2, ..., m \right\} \tag{5.1}$$

A selection procedure is required to choose some chromosomes to perform the crossover operation. The selection procedure is listed as follows:

Step 1: Generate a random number r, $0 \le r \le 1$.

Step 2: If $q_k \le r \le q_{k+1}$, then chromosome X_{k+1} is selected to perform the crossover operation.

Here, the cumulative probability q_k for chromosome X_k is calculated from the formulation $q_k = \sum_{j=1}^{k} p_j$, where p_j is defined as the selection probability. For the kth chromosome, the selection probability $p_k = \dfrac{\sum_{l=1}^{psize} eval(X_l) - eval(X_k)}{\sum_{l=1}^{psize} eval(X_l) * (psize - 1)}$. It is noted that the better the fitness value, the more frequently it will be selected to perform the crossover operation.

5.3.3 Genetic operators

Similar to the general genetic algorithms, two genetic operations, crossover and mutation, are used in the algorithm as two search mechanisms. The number of chromosomes selected for performing crossover and mutation operations depends on the crossover rate and the mutation rate. If *cross* and *mut* denote the number of pairs of chromosomes to perform crossover, the number of chromosomes to perform mutation, respectively, then,

$$cross = \begin{cases} crossno/2 & \text{if } crossno \text{ is even} \\ (crossno-1)/2 & \text{otherwise} \end{cases},$$

where, *crossno* = ⌈crossover rate × *psize*⌉, and *mut* = round (mutation rate × *psize*).

5.3.3.1 The mechanism of crossover operation

Step 1: Follow the selection procedure at section 5.3.2 (Step 1 to Step 2), a pair of chromosomes, $\boldsymbol{X}_1 = (x_{i1j1})$ and $\boldsymbol{X}_2 = (x_{i2j2})$, are selected from the population for performing the crossover operation.

Step 2: Create two $m \times n$ temporary matrices, $\boldsymbol{D} = (d_{ij})$ and $\boldsymbol{R} = (r_{ij})$, where

$$d_{ij} = \lfloor (x_{i1j1} + x_{i2j2})/2 \rfloor \text{ and } r_{ij} = (x_{i1j1} + x_{i2j2}) \bmod 2.$$

Step 3: Divide matrix $\boldsymbol{R}$ into two matrices $\boldsymbol{R}_1 = (r_{i1j1})$ and $\boldsymbol{R}_2 = (r_{i2j2})$ under the constraints: $\sum_{i=1}^{m} r_{i_1 j_1} = \sum_{i=1}^{m} r_{i_2 j_2}$, $j = 1, 2, \ldots, n$ and $\boldsymbol{R} = \boldsymbol{R}_1 + \boldsymbol{R}_2$.

Step 4: Generate two offspring:

$$\boldsymbol{X}_1' = \boldsymbol{D} + \boldsymbol{R}_1 \text{ and } \boldsymbol{X}_2' = \boldsymbol{D} + \boldsymbol{R}_2$$

5.3.3.2 The mechanism of mutation operation

Step 1: Select a chromosome from the population randomly.

Step 2: Arbitrarily extract a sub-matrix, $\mathbf{W} = (w_{ij})$, from the selected chromosome, relocate w_{ij} in this sub-matrix. The reallocation procedure is the same as that in the initialization steps, i.e., Step

1 and Step 2 in section 5.3.1, but m now is the number of rows, and n is the number of columns in **W**, and $c_j = \sum_{i=1}^{m} w_{ij}$ (j = 1, 2, ..., n), respectively.

Step 3: An offspring is created by replacing the reallocated sub-matrix into the selected chromosome.

5.3.4 The algorithm

The procedure of the genetic algorithm for the PCB component grouping problem is as follows:

Step 1: Input the population size, *psize* and the number of iterations, *itno*.

Step 2: Generate *psize* initial chromosomes by the initialization procedure in Section 5.3.1.

Step 3: Calculate fitness value $eval(X_k)$ by using Eq. (5.1) for all chromosomes in the population.

Step 4: Use the selection procedure in Section 5.3.2 to choose *cross* pairs of chromosomes to perform the crossover operation in Section 5.3.3.1.

Step 5: Choose *mut* chromosomes randomly to perform the mutation operation in Section 5.3.3.2.

Step 6: Compare all offspring (generated from the crossover and the mutation operations) with the chromosomes in the population by the fitness values obtained from Eq. (5.1). Replace a

chromosome in the population by the offspring which has a better fitness value.

Step 7: Determine the best chromosome, that is, min{$eval(X_k)$, k = 1, 2, ..., $psize$} for each generation. Repeat Step 4 to Step 7, until $itno$ iterations are performed.

5.4 A Numerical Example

Example 2 in Chapter 3, the component grouping problem in the second placement station for the bottom side PCB, is selected to illustrate the above algorithm. The min-max formulation is as follows:

$$min\left\{\max\begin{pmatrix}110+3x_{11}+7x_{12}+7x_{13}+5x_{14}+90000x_{15}+90000x_{16}+90000x_{17};\\147+7x_{21}+12x_{22}+15x_{23}+16x_{24}+15x_{25}+15x_{26}+21x_{27};\\147+23x_{31}+38x_{32}+35x_{33}+35x_{34}+27x_{35}+33x_{36}+43x_{37};\end{pmatrix}\right\}$$

Subjec to:

$$x_{11}+x_{21}+x_{31}=324$$
$$x_{12}+x_{22}+x_{32}=37$$
$$x_{13}+x_{23}+x_{33}=12$$
$$x_{14}+x_{24}+x_{34}=5$$
$$x_{15}+x_{25}+x_{35}=7$$
$$x_{16}+x_{26}+x_{36}=5$$
$$x_{17}+x_{27}+x_{37}=4$$
$$x_{ij} \geq 0 \text{ and integer; } i = 1, 2, 3; j = 1, 2, ..., 7.$$

The operation parameters are set as, population size $psize$ = 25, crossover rate = 0.4 and mutation rate = 0.3, so $cross$ = 5, mut = 8. Only the first generation (iteration) will be shown in detail to demonstrate how the algorithm works in the next section.

5.4.1 The first iteration

The procedure of the genetic algorithm for determining the cycle time for the second station in example 2 is shown as follows:

Step 1: Input the population size, and the number of iteration. In this case, *psize* = 25 and *itno* = 1000.

Step 2: Generate 25 initial chromosomes. A chromosome is generated as follows (see Section 5.3.1):

Firstly, randomly assign an integer number to b_i, say b_i = {120, 58, 216}, such that $\sum_{i=1}^{3} b_i = \sum_{j=1}^{7} c_j = 394$. In this case, b_i represents the total number of components for machine i to assemble. For example, there are totally 120 components assigned to machine 1.

Then, select a random number k from set π, $\pi = \{1, 2, \ldots, 21\}$. If $k = 4$, $i = \lfloor (4 - 1)/7 + 1 \rfloor = 1$ and $j = (4 - 1) \bmod 7 + 1 = 4$, $x_{14} = \min\{b_1, c_4\} = \min\{120, 5\} = 5$. Now, update b_1, c_4 and π, $b_1 = 120 - 5 = 115$, $c_4 = 5 - 5 = 0$ and delete 4 from set π. Repeat the procedure until there is no more elements in π.

Repeat the above procedure, and a chromosome can be obtained as follows:

$$\mathbf{X}_1 = \begin{bmatrix} 115 & 0 & 0 & 5 & 0 & 0 & 0 \\ 41 & 0 & 12 & 0 & 0 & 5 & 0 \\ 168 & 37 & 0 & 0 & 7 & 0 & 4 \end{bmatrix}$$

Similarly, other 24 chromosomes can be generated.

Step 3: Calculate the fitness value by Eq. (5.1). Here, the fitness value for $\mathbf{X}_1$ is:

$$eval(X_1) = \max\begin{pmatrix} 370+110 \\ 542+147 \\ 5631+147 \end{pmatrix} = 5778$$

The fitness values for these 25 chromosomes are listed as follows:

$Eval(X_1)$=5778	$eval(X_6)$=2056	$eval(X_{11})$=66520	$Eval(X_{16})$=9637	$Eval(X_{21})$=7568
$Eval(X_2)$=9015	$eval(X_7)$=4360	$eval(X_{12})$=7568	$Eval(X_{17})$=1906	$Eval(X_{22})$=9639
$Eval(X_3)$=45752	$eval(X_8)$=8620	$eval(X_{13})$=5863	$Eval(X_{18})$=5778	$Eval(X_{23})$=6352
$Eval(X_4)$=5680	$eval(X_9)$=8562	$eval(X_{14})$=4879	$Eval(X_{19})$=8796	$Eval(X_{24})$=7652
$Eval(X_5)$=7854	$eval(X_{10})$=9632	$eval(X_{15})$=7986	$Eval(X_{20})$=5687	$Eval(X_{25})$=4356

Step 4: As the crossover rate = 0.4, 5 pairs of chromosomes (or 10 chromosomes) will be selected. To select the chromosomes for performing the crossover operation, the following data are needed (see Section 5.3.2.):

a) The selection probability p_k for chromosome $\mathbf{X}_k$ are:

p_1 = 0.040767	p_6 = 0.041346	p_{11} = 0.031305	p_{16} = 0.040166	p_{21} = 0.040488
p_2 = 0.040262	p_7 = 0.040988	p_{12} = 0.040488	p_{17} = 0.04137	p_{22} = 0.040166
p_3 = 0.03454	p_8 = 0.040324	p_{13} = 0.040753	p_{18} = 0.040767	p_{23} = 0.040677
p_4 = 0.040782	p_9 = 0.040333	p_{14} = 0.040907	p_{19} = 0.040297	p_{24} = 0.040475
p_5 = 0.040443	p_{10} = 0.040166	p_{15} = 0.040423	p_{20} = 0.040781	p_{25} = 0.040988

b) The cumulative probability q_k for chromosome $\mathbf{X}_k$ are:

q_1 = 0.040767	q_6 = 0.238141	q_{11} = 0.431257	q_{16} = 0.633993	q_{21} = 0.837694
q_2 = 0.081029	q_7 = 0.279128	q_{12} = 0.471744	q_{17} = 0.675363	q_{22} = 0.87786
q_3 = 0.115569	q_8 = 0.319452	q_{13} = 0.512498	q_{18} = 0.716129	q_{23} = 0.918537
q_4 = 0.156351	q_9 = 0.359785	q_{14} = 0.553404	q_{19} = 0.756426	q_{24} = 0.959012
q_5 = 0.196794	q_{10} = 0.399951	q_{15} = 0.593827	q_{20} = 0.797207	q_{25} = 1.00000

Now, in order to select 10 chromosomes, a random number, r, is generated 10 times. If two of the random numbers are 0.020384, and 0.657324,

for example, then $\mathbf{X}_1$ and $\mathbf{X}_{17}$ will be selected as two of the parents to perform the crossover operation (see Section 5.3.3.1):

$$\mathbf{X}_1 = \begin{bmatrix} 115 & 0 & 0 & 5 & 0 & 0 & 0 \\ 41 & 0 & 12 & 0 & 0 & 5 & 0 \\ 168 & 37 & 0 & 0 & 7 & 0 & 4 \end{bmatrix} \text{ and } \mathbf{X}_{17} = \begin{bmatrix} 251 & 37 & 0 & 5 & 0 & 0 & 0 \\ 4 & 0 & 12 & 0 & 7 & 5 & 0 \\ 69 & 0 & 0 & 0 & 0 & 0 & 4 \end{bmatrix}$$

Create two temporary matrices, $\mathbf{D} = (d_{ij})$ and $\mathbf{R} = (r_{ij})$

$$\mathbf{D} = \begin{bmatrix} 183 & 18 & 0 & 5 & 0 & 0 & 0 \\ 22 & 0 & 12 & 0 & 3 & 5 & 0 \\ 118 & 18 & 0 & 0 & 3 & 0 & 4 \end{bmatrix} \text{ and } \mathbf{R} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Then divide $\mathbf{R}$ into $\mathbf{R}_1$ and $\mathbf{R}_2$ as follows:

$$\mathbf{R}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \text{ and } \mathbf{R}_2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$\mathbf{R}_1$ and $\mathbf{R}_2$ must have the same numbers of 1's on each column.

Finally, two offspring $\mathbf{X}_1$' and $\mathbf{X}_{17}$' are

$$\mathbf{X}_1' = \begin{bmatrix} 183 & 18 & 0 & 5 & 0 & 0 & 0 \\ 23 & 0 & 12 & 0 & 3 & 5 & 0 \\ 118 & 19 & 0 & 0 & 4 & 0 & 4 \end{bmatrix} \text{ and } \mathbf{X}_{17}' = \begin{bmatrix} 183 & 19 & 0 & 5 & 0 & 0 & 0 \\ 22 & 0 & 12 & 0 & 4 & 5 & 0 \\ 119 & 18 & 0 & 0 & 3 & 0 & 4 \end{bmatrix}$$

Step 5: The mutation rate is 0.3, so 8 chromosomes will perform the mutation operation. Randomly select 8 chromosomes from the population. Assume $\mathbf{X}_1$ is one of the chromosomes selected as a parent for mutation (see Section 5.3.3.2):

$$\mathbf{X}_1 = \begin{bmatrix} 115 & 0 & 0 & 5 & 0 & 0 & 0 \\ 41 & 0 & 12 & 0 & 0 & 5 & 0 \\ 168 & 37 & 0 & 0 & 7 & 0 & 4 \end{bmatrix}$$

Selected two rows {1, 3} and three columns {1, 4, 7} randomly. The corresponding sub-matrix, $\mathbf{W} = (w_{ij})$, and the reallocated sub-matrix are:

Corresponding sub-matrix Reallocated sub-matrix

$$\begin{bmatrix} 115 & 5 & 0 \\ 168 & 0 & 4 \end{bmatrix}, \qquad \begin{bmatrix} 162 & 5 & 0 \\ 121 & 0 & 4 \end{bmatrix}$$

Replace the reallocated sub-matrix into $\mathbf{X}_1$, the offspring after mutation is:

$$\mathbf{X}_1'' = \begin{bmatrix} 162 & 0 & 0 & 5 & 0 & 0 & 0 \\ 41 & 0 & 12 & 0 & 0 & 5 & 0 \\ 121 & 37 & 0 & 0 & 7 & 0 & 4 \end{bmatrix}$$

Step 6: Calculate the fitness values for all offspring, and compare them with the population. In this case, the fitness values are: $\mathbf{X}_1'$=3863, $\mathbf{X}_{17}'$=3821 and $\mathbf{X}_1''$ = 4697. So, $\mathbf{X}_1'$, $\mathbf{X}_{17}'$ and $\mathbf{X}_1''$ will replace $\mathbf{X}_{11}$, $\mathbf{X}_3$, and $\mathbf{X}_{22}$ because they are the worst in the population.

Step 7: The best chromosome for this generation is $\mathbf{X}_{17}$, as it has the smallest fitness value. Repeat Step 4 to Step 6, until 1000 iterations (generations) are performed.

5.4.2 Result analysis

Fig. 5.4 shows the cycle time determined at each iteration. After 1000 iterations, the best solution is shown in Table 5.1 with the cycle time of 979, i.e., 97.9 seconds. It can be observed that the solution improves sharply in the first 100 iterations, and when it gets closer to the optimum, the improvement rate decreases quickly. Furthermore, the genetic algorithm can not guarantee that the optimal solution can be obtained and the best solution determined from the 1000 iterations (i.e., solution shown in Table 5.1) is still not optimal. Compared with

the optimal cycle time determined in Chapter 3, that is, 97.1 seconds, the GA solution has a less than 1 % error. Finally, if a longer cycle time is accepted in a real situation, the number of iterations can be greatly reduced. For example, to obtain a cycle time of 100 seconds, only 100 iterations are enough.

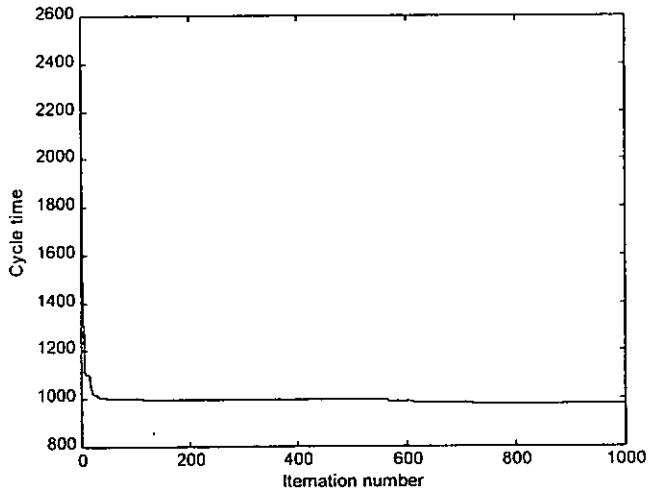


Figure 5.4: The best cycle time determined at each generation

x_{11}=269	x_{12}=0	x_{13}=6	x_{14}=4	x_{15}=0	x_{16}=0	x_{17}=0
x_{21}=46	x_{22}=37	x_{23}=3	x_{24}=0	x_{25}=0	x_{26}=0	x_{27}=1
x_{31}=9	x_{32}=0	x_{33}=3	x_{34}=1	x_{35}=7	x_{36}=5	x_{37}=3
				Cycle time:	97.9	

Table 5.1: Solution of the best chromosome

5.5 The Genetic Algorithm vs. the Linear Programming Solution

In this section, the comparison between the two algorithms (GA and LP) will be made such that a better algorithm can be recommended to the production planners to cope with the component grouping problem.

The decimal optimal solution for the above example obtained by the LP algorithm stated in Chapter 4 is 96.8 seconds. After rounding off the decimal

values to integers, the cycle time becomes 97.3 seconds. Table 5.2 and Table 5.3 show the optimal decimal solution and its rounding off solution. It is noted that the solution determined by the linear programming with rounding off (LPRO) seems to be better than that obtained from the GA. However, if the computing time is not a constraint in the GA and thus the number of iteration is unlimited, it is possible that the solution obtained from the GA is better. Since the solution from LPRO is unique whereas in the GA the final solution depends on the number of iteration, it is meaningless to make the comments based on the "quality" of the solution only. Moreover, the methodologies inside these two algorithms are different. A comparison can be made only based on the same methodology, either mathematical or heuristic approach. For example, the comparison between the simplex method and the linear programming algorithm proposed on Chapter 4 on the component grouping problem can be carried out because both of them are mathematical approaches and will produce the same solution. Similarly, the GA can be compared with other heuristic techniques, like ANN. Such a comparison can be made based on the accuracy of the solutions with the same computing time.

Besides the "quality" of the solution, the LPRO algorithm has the following advantages over the GA approach:

1. Smaller error

The LPRO algorithm is modified from the simplex algorithm, thus the solution is obtained by searching vertexes one by one along edges until an optimal vertex is reached. On the other hand, the GA approach searches the solution by making some adjustments randomly for the next iteration. This

means the solution may fall far away from the boundary and also far away from the optimal solution. Therefore, if the computing times for the both methods are the same, the solution from the LPRO algorithm solution will have a smaller error than the one from the GA.

2. More stable solution

The LPRO algorithm can only produce a unique solution for a particular problem. However, each time the GA may result in different solution, depending the GA parameters, such as, the number of iterations, the population of the chromosomes, the crossover rate and the mutation rate.

3. Easier control and implementation

The iteration in the LPRO algorithm will stop automatically once it reaches the optimal decimal solution. For the GA, a decision regarding the iteration number has to be made by the user before the algorithm starts since it is one of the input parameters, and it is difficult to make. Moreover, the parameters of the population size, the crossover and mutation rates also have to be decided at the beginning.

x_{11}=277.75	x_{12}=0	x_{13}=0	x_{14}=5	x_{15}=0	x_{16}=0	x_{17}=0
x_{21}=46.25	x_{22}=37	x_{23}=3.56	x_{24}=0	x_{25}=0	x_{26}=0	x_{27}=0
x_{31}=9	x_{32}=0	x_{33}=8.44	x_{34}=0	x_{35}=7	x_{36}=5	x_{37}=4
				Cycle time:	96.8	

Table 5.2: Optimal decimal solution

x_{11}=278	x_{12}=0	x_{13}=0	x_{14}=5	x_{15}=0	x_{16}=0	x_{17}=0
x_{21}=46	x_{22}=37	x_{23}=4	x_{24}=0	x_{25}=0	x_{26}=0	x_{27}=0
x_{31}=0	x_{32}=0	x_{33}=8	x_{34}=0	x_{35}=7	x_{36}=5	x_{37}=4
				Cycle time:	97.3	

Table 5.3: Rounding off solution

5.6 Conclusion

A genetic algorithm was designed to cope with the component assignment problem in this chapter. The following observations are made after the algorithm was implemented:

1. GA performs very well at the beginning. However, when it converges to the optimal solution, its improvement rate decreases sharply.
2. GA can not guarantee to obtain the optimal solution. The greater the iteration number, the better the solution. A decision on how many iterations should be performed is difficult to make.
3. Comparing GA with the solution from the linear programming with rounding off, the latter is more preferable.

The last chapter will have an overall conclusion and points out some possible further work in future.

Chapter Six

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

The component grouping problem in PCB assembly was studied in this project. The findings in this project can be summarized as follows:

(1) To optimize the production rate in PCB assembly, many problems should be considered. For a large-scale PCB assembly company, the main objective is to reduce the processing time per board (that is, the cycle time) and the cycle time depends on how the components are grouped and assigned to machines.

(2) Different mathematical models require different algorithms and thus have different computing efficiency. The component grouping problem for single-sided PCB assembly is formulated as various types of mathematical models, including

i) min-max type binary integer programming,

ii) binary integer linear programming,

iii) min-max type integer programming,

iv) integer linear programming.

In general, the problem size for binary integer programming is very large and not applicable in the real situation. Whereas, the computing efficiency for an integer linear programming is better than the min-max type one, due to the irregular behavior of the nonlinear functions in min-max type programming.

(3) Two objective functions, minimizing cycle time and balance delay, were considered for measuring a line performance. Since the model of minimizing balance delay requires more computing time and it may induce an inappropriate grouping, minimizing cycle time is more suitable for the component grouping problem.

(4) A procedure is presented to decompose the double-sided PCB assembly model into two sub-models, so that a shorter computing time can be performed. Besides, the structures of these two sub-models are the same to those defined in the single-side PCB assembly, and so another advantage is that an algorithm developed later is also applicable to the double-sided PCB assembly.

(5) Mathematical models for the component grouping problem are compared with some special problems. However, the effective algorithms for those special problems are not applicable to the component grouping problem.

(6) From the idea of the revised simplex method, an effective algorithm is developed to determine the optimal solution by releasing the integer constraint in the component grouping problem. This algorithm reduces the number of iterations required and also the unnecessary computing induced by the simplex method. Although the solution obtained is not integer, after rounding off the decimal values to integers, a near optimal solution, which is acceptable in practice, can be determined.

(7) A genetic algorithm procedure is designed to seek the optimal solution for the component grouping problem, but the solution from the GA technique is

not good as the one obtained by the linear programming approach with rounding off. In some case, the GA solution may be acceptable.

6.2 Future Work

The future work related to this project is suggested as follows:

(1) Developing a production planning system for PCB assembly.

A production planning system should be developed to integrate the component grouping problem with the problem of assigning PCB to assembly lines, so that a board can be assembled in an assembly line with the minimum cost and time.

(2) Implementing the system in a company.

The ultimate purpose of research is to apply the research results to industry. The component grouping problem is from a PCB assembly company, so it is reasonable to return the research solution to a PCB assembly company.

(3) Re-grouping some single components.

Sometimes, the optimal solution or the rounding off solution from the linear programming approach is not so reasonable in the real situation. For example, if one or few components are grouped to a machine and if it is not practical to set up a machine for processing only one component, this component should be assigned to another machine to form a bigger group with the same type of components. So, a system should be developed to tune up some groupings if necessary.

(4) Improving the quality of the linear programming solution.

If an upper bound can be determined for the component grouping problem, a better integer solution may be obtained by searching the integers between the linear programming solution and the upper bound.

(5) Enriching the genetic algorithm's performance.

In this project, some basic techniques in the genetic algorithms were implemented. It may be worth to add other genetic operations, like scaling, to improve the GA's performance or design other mechanisms to enrich crossover and mutation operations. Of course, other heuristics, like simulated annealing, may also provide a better solution.

References:

[Aar97] Aarts, E. and Lenstra, K.(editors): *Local Search in Combinatorial Optimization.* A Wiley-Interscience Publication, 1997.

[Ami94] Amini, M. M.: Vectorization of an auction algorithm for linear cost assigment problem. *Computers and Industrial Engineering,* v26 n1, 1994, 141-149.

[Amm97] Ammons, J. C., Carlyle, M., Cranmer, L., Depuy, G., Ellis, K., Mcginnis, L.F., Tovey, C. A. and Xu, H.: Component allocation to balance workload in printed circuit card assembly systems. *IIE Transactions*, v29, 1997, 265-275.

[Ash98] Ashayeri, J. and Van Dorp, A. M.: An allocation problem in printed circuit board assembly. *Proceedings of Eighth International FAIM Conference*, Portland, Oregon. 1998, 299-309.

[Ask94] Askin, R. G., Dror, M. and Vakharia, A. J.: Printed circuit board family grouping and component allocation for a multimachine, open-shop assembly cell. *Naval Research Logistic Quarterly*, v41, 1994, 587-608.

[Bal64] Balas, E. and Ivanescu, P. L.: On the generalized transportation problem. *Management Science*, v11 n1, 1964, 188-202.

[Bal66] Balas, E.: The dual method for the generalized transportation problem. *Management Science*, v12 n7, 1966, 555-568.

[Bal75a] Balachandran, V. and Thompson, G. L.: An operator theory of parametric programming for the generalized transportation problem: I. Basic theory. *Naval Research Logistic Quarterly*, v22, 1975, 79-100.

[Bal75b] Balachandran, V. and Thompson, G. L.: An operator theory of parametric programming for the generalized transportation problem - III - Weight operators. *Naval Research Logistic Quarterly*, v22, 1975, 317-339.

[Bal75c] Balachandran, V. and Thompson, G. L.: An operator theory of parametric programming for the generalized transportation problem - IV - Global operators. *Naval Research Logistic Quarterly*, v22, 1975, 101-125.

[Bal75d] Balachandran, V. and Thompson, G. L.: An operator theory of parametric programming for the generalized transportation problem: II-Rim, cost and bound operators. *Naval Research Logistic Quarterly*, v22, 1975, 297-315.

[Bal88] Ball, M. O. and Magazine, M. J.: Sequencing of insertions in printed circuit board assembly. *Operations Research*, v36 n2, March 1988, 192-201.

[Bar89] Bard, J. F.: Assembling line balancing with parallel workstations and dead time. *International Journal Production Research*, v27 n6, 1989, 1005-1018.

[Bar94] Bard, J. F., Clayton, R. W. and Feo, T. A.: Machine setup and component placement in printed circuit board assembly. *The International Journal of Flexible Manufacturing System*, v6, 1994, 5-31.

[Bay86a] Baybars, I.: A survey of exact algorithms for the simple assembly line balancing problem. *Management Science*, v32 n8, August 1986, 909-932.

[Bay86b] Baybars, I.: An efficient heuristic method for the simple assembly line balancing problem. *International Journal of Production Research*, v24 n1, 1986, 149-166.

[Bea88] Beale, E. M. L.: *Introduction to Optimization*, A Wiley-Interscience Publication. Lynne Mackley (editor). 1988.

[Ben90] Ben-Arieh, D., and Dror, M.: Part assignment to electronic insertion machines: two machine case. *International Journal of Production Research*, v28 n7, 1990, 1317-1327.

[Ber81] Bertsekas, D. P.: A new algorithm for the assignment problem. *Mathematical Programming*, v21, 1981, 152-171.

[Ber88] Bertsekas, D. P.: The auction algorithm: a distributed relaxation method for the assignment problem. *Annals of Operations Research,* v14, 1988, 105-123.

[Ber89] Bertsekas, D. P. and Castanon, D. A.: The auction algorithm for the transportation problem. *Annals of Operations Research,* v20, 1989, 67-96.

[Ber90] Bertsekas, D. P.: The auction algorithm for assignment and other network flow problems: A tutorial. *Interfaces,* v20 n4, 1990, 133-149.

[Ber91] Bertsekas, D. P. and Castanon, D. A.: Practical aspects and experiences Parallel synchronous and asynchronous implementations of the auction algorithm. *Parallel Computing*, v17, 1991, 707-732.

[Ber92a] Bertsekas, D. P. and Castanon, D. A.: A forward/reverse auction algorithm for asymmetric assignment problems. *Computational Optimization and Applications,* v1, 1992, 277-297.

[Ber92b] Bertsekas, D. P.: Auction algorithms for network flow problems: A tutorial introduction. *Computational Optimization and Applications*, v1, 1992, 7-66.

[Ber94] Bertsekas, D. P. and Castanon, D. A.: A generic auction algorithm for the minimum cost network flow problem. *Computational Optimization and Applications*, v2, 1993, 229-260.

[Buc92] Buckles, B. P. and Petry, F. E. (editors): *Genetic Algorithms.* IEEE Computer Society Press, 1992.

[Car89] Carmon, T. F., Maimon, O. Z. and Dar-El E. M.: Group set-up for printed circuit board assembly. *International Journal of Production Research,* v27 n10, 1989, 1795-1810.

[Cer85] Cerny, V.: Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, v45, 1985, 41-51.

[Cha89] Chan, D., and Mercier, D.: IC chip insertion: an application of the travelling salesman problem. *International Journal of Production Research*, v27 n10, 1989, 1837-1841.

[Cha95] Chambers, L. (editor): *Practical handbook of genetic algorithms, vII New Frontiers.* CRC press, 1995.

[Cun86] Cunningham, P. and Browne, J.: A LISP-based heuristic scheduler for automatic insertion in electronics assembly. *International Journal of Production Research,* v24 n6, 1986, 1395-1408.

[Dan51] Dantzig, G. B.: Maximization of a linear function variables subject to linear inequalities. *Activity Analysis of Production and Allocation.* edited by Koopmans, T. C., John Wiley, 1951.

[Dav90] Davis, T. and Selep, E.: Group technology for high mix printed circuit assembly. *IEEE/CHMT '90 symposium,* Washington, DC, USA, 1990, 264-269.

[Dav91] Davis, L.: *Handbook of genetic algorithm.* Van Nostrand Reinhold, 1991.

[Der85] Derigs, U.: The shortest augmenting path method for solving assignment problems - motivation and computational experience. *Annals of Operations Research,* v4, 1985/6, 57-102.

[Des95] Dessouky, M. M., Adiga, S., and Park, K.: Design and scheduling of flexible assembly lines for printed circuit boards. *International Journal of Production Research*, v33 n3, 1995, 757-775.

[Eis64] Eisemann, K.: The generalized stepping stone method for the machine loading model. *Management Science*, v11 n1, 1964, 154-176.

[Fel93] Feldmann, K., Franke, J., and Rothhaupt, A.: Automated generating and simulation of insertion sequencer. *IEEE/CHMT International Electronic Manufacturing Technology Symposium*, Senta Clara, CA, USA, 1993, 206-210.

[Fel94a] Feldmann, K., Franke, J., and Rothhaupt, A.: Optimization and Simulation of the printed circuit assembly. *IEEE Transactions On Components, Packaging, And Manufacturing Technology*, Part A, v17 n2, June 1994, 277-281.

[Fel94b] Feldmann, K. and Rothhaupt, A.: An intelligent optimization tool for the electronics production. *SMI Proceedings of The Technical Program, Surface Mount International*, San Jose, California, 1994, 704-710.

[Fel94c] Feldmann, K. and Rothhaupt, A.: Intelligent planning tool to support the electronic production. *Production Engineering*, vII/1, 1994, 169-172.

[Fer56] Ferguson, A. R. and Dantzig, G. B.: The allocation of aircraft to routes - an example of linear programming under uncertain demand. *Management Science*, v3, 1956, 45-73.

[Fis85] Fisher, M. L.: Applications oriented guide to lagrangian relaxation. *Interface*, v15 n2, 1985, 10-21.

[Gar70] Garfinkel, R. S.: An improved algorithm for the bottleneck assignment problem. *Operations Research*, v19, 1970, 1747-1751.

[Gar72] Garfinkel, R.: *Integer Programming*. New York : Wiley, 1972.

[Gar76] Garfinkel, R. S. and Rao, M.: Bottleneck linear programming. *Mathematical Programming*, v11, 1976, 291-298.

[Gar96] Garetti, M., Pozzetti, A. and Tavecchio, R.: Porduction scheduling in SMT electronic boards assembly. *Production Planning and Control*, v7 n2, 1996, 197-204.

[Gen97] Gen, M. and Cheng, R.: *Genetic Algorithms and Engineering Design*. John Wiley & Sons, 1997.

[Gil76] Gillett, B. E.: *Introduction to Operations Research: A Computer-Oriented Algorithmic Approach.* McGraw-Hill, 1976.

[Glo86] Glover, F. and McMillan, W. R.: The general employee scheduling problem: An integration of management science and artificial intelligence. *Computers and Operations Research*, v13 n5, 1986, 563-593.

[Glo89] Glover, F., and Laguna, M.: *Target analysis to improve a tabu search method of machine scheduling.* Technical Report, Advanced Knowledge Research Group, US West Advanced Technologies, Boulder, Co. September 1989.

[Glo90] Glover, F.,: Tabu search - Part II. *Operations Research Society of America*, v2 n1, 1990.

[Gol86] Golden, B., and Skiscim, C.: Using simulated-annealing to solve routing and location problems. *Naval Research Logistics Quarterly*, v33, 1986, 261-279.

[Gom58] Gomory, R. E.: *Outline of an algorithm for integer solutions to linear programs.* 1958.

[Gro59] Gross, O.: *The bottleneck assignment problem.* The Rand Corporation, 1959.

[Gün96] Günther, H. O., Gronalt, M. and Piller, F.: Component kitting in semi-automated printed circuit board assembly. *International Journal of Production Economics*, v43, 1996, 213-226.

[Hac89] Hackman, S. T., Magazine. M. J. and Wee, T. S.: Fast, effective algorithms for simple assembly line balancing problems. *Operations Research*, v37 n6, Nov 1989, 916-924.

[Had67] Hadley, G.: *Linear Programming.* Addison-Wesley, 1967.

[Hal93] Hallefjord, Å., Jörnsten, K. O. and Värbrand, P.: Solving large scale generalized assignment problems - an aggregation/disaggregation approach. *European Journal of Operational Research*, v64, 1993, 103-114.

[Hel61] Helgeson, W. B. and Birnie, D. P.: Assembly line balancing using the ranked positional weight technique. *The Journal of Industrial Engineering*, v12, 1961, 394-398.

[Hel63] Held, M., Karp R. M. and Shareisan, R.: Assembly line balancing-dynamic programming with precedence constraints. *Operations Research*, v11, 1963, 442-459.

[Hen86] Henig, M.: Extensions of the dynamic programming method in the deterministic and stochastic assembly line balancing problems. *Computers and Operations Research*, v13, 1986, 443-449.

[Hua93] Huang, Y. W. and Srihari, K.: A solution methodology for the multiple batch surface mount PCB placement sequence problem. *Advances in Electronic Packaging*, ASME, USA, EEP v4 n1, 1993, 373-379.

[Hun80] Hung, M. S. and Rom, W. O.: Solving the assignment problem by relaxation. *Operations Research*, v28, 1980, 969-982.

[Ign65] Ignall, E. and Schrange, L.: Application of the branch and bound technique to some flow-shop scheduling problems. *Operation Research*, v13, 1965, 400-412.

[Jai96] Jain, S. and Gea, H. C.: PCB layout design using a genetic algorithm. *Journal of Electronic Packaging*, v118 n1, March 1996, 11-15.

[Jip94] Ji, P., Wong, Y. S., Loh, H. T. and Lee, L. C.: SMT production scheduling: a generalized transportation approach. *International Journal of Production Research*, v32 n10, 1994, 2323-2333.

[Jiz91] Ji, Z., Leu, M. C. and Wong, H.: Application of linear assignment model for planning of robotic printed circuit board assembly. *ASME Manufacturing Processes and Materials Challenges in Microelectronic Packaging*, ADM-v131/EEP-v1, 1991, 35-41.

[Jiz93] Ji, Z., Leu, M. C. and Wong, H.: Development and implementation of linear assignment algorithm for assembly of PCB components. *Advances in Electronic Packaging, American Society of Mechanical Engineers*, ASME, USA, EEP v4 n1, 1993, 365-371.

[Joh89] Johnson, D, Argon, C., McGeoch, L. and Schevon, C.: Optimization by simulated-annealing: An experiental evaluation-part I, graph partitioning. *Operations Research*, v37 n6, 1989, 865-892.

[Kap94] Kapov, J. S.: Extensions of a tabu search adaptation to the quadratic assignment problem, *Computers and Operations Research*, v21 n8, 1994, 855-865.

[Kar84] Karmarkar, N.: A new polynomial-time algorithm for linear programming. *Combinatorica 4*, v4, 1984, 373-395.

[Kar87] Karwan, M. H. and Ram, B.: A lagrangean dual-based solution method for a special linear programming problem. *Computers and Operations Research*, v14 n1, 1987, 67-73.

[Ken91] Kennington, J. L. and Wang, Z.: An empirical analysis of the dense assignment problem: Sequential and parallel implementations. *ORSA Journal on Computing*, v3 n4, 1991, 299-306.

[Ken94] Kennington, J. L. and Mohammadi, F.: The singly constrained assignment problem: A Lagrangian relaxation heuristic algorithm. *Computational Optimization and Applications*, v3 n1, 1994, 7-26.

[Kha79] Khachiyan, L. G.: A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, v20, 1979, 191-194.

[Kim95] Kim, H. and Park, S.: Strong cutting plane algorithm for the robotic assembly line balancing problem. *International Journal of Production Research*, v33 n8, 1995, 2311-2323.

[Kun91] Kung, H. K., and Changchit, C.: A just-in-time simulation model of a PCB assembly line. *Computers and Industrial Engineering*, v20 n1, 1991, 17-26.

[Lφk96] Lφkketangen, A. and Woodruff, D. L.: Progressive hedging and tabu search applied to minxed integer (0, 1) multistage stochastic programming. *Journal of Heuristics*, v2, 1996, 111-128.

[Lan60] Land, A. H. and Doig, A. G.: An automatic method of solving discrete programming problems. *Econometrica*, v28 n3, July 1960, 497-520.

[Lap96] Laporte, G., Potvin, J. Y. and Quilleret, F.: A tabu search heuristic using genetic diversification for the clustered traveling salesman problem. *Journal of Heuristics*, v2, 1996, 187-200.

[Leu93] Leu, M. C., Wong, H. and Ji, Z.: Planning of component placement/insertion sequence and feeder setup in PCB assembly using genetic algorithm. *Journal of Electronic Packaging*, v115, Dec. 1993, 424-432.

[Lim97] Lim, J. M.: A genetic algorithm for a single hoist scheduling in the printed-circuit board electroplating line. *Computers and Industrial Engineering*, v33 n3-4, 1997, 789-792.

[Lin92] Lin, Y., Austin, L. M. and Burns, J. R.: An intelligent algorithm for mixed-integer programming models. *Computers and Operations Research*, v19 n6, 1992, 461-468.

[Lou64] Lourie, J. R.: Topology and computation of the generalized transportation problem. *Management Science*, v11 n1, 1964, 177-187.

[Luz93] Luzzatto, D. and Perona, M.: Cell formation in PCB assembly based on production quantitive data. *European Journal of Operational Research*, v69, 1993, 312-329.

[Mai91] Maimon, O. Z and Shtub A.: Grouping method for printed circuit board assembly. *International Journal of Production Research*, v29 n 7, 1991, 1379-1390.

[Mai93] Maimon, O. Z., Dar-El , E. M., and Maimon, O. Z.: Set-up schemes for printed circuit boards assembly. *European Journal of Operational Research*, v70, 1993, 177-190.

[Mar81] Martello, S. and Toth, P.: An algorithm for the generalized assignment problem. *Operational Research '81*, J. P. Brans (editor), 1981.

[Mar95] Martello, S. and Toth, P.: The bottleneck generalized assignment problem. *European Journal of Operational Research*, v83, 1995, 621-638.

[Maz88] Mazzola, J. B. and Neebe, A. W.: Bottleneck generalized assignment problems. *Engineering Costs and Production Economics*, v14, 1988, 61-65.

[Maz93] Mazzola, J. B. and Neebe, A. W.: An algorithm for the bottleneck generalized assignment problem. *Computers and Operations Research*, v20 n4, 1993, 355-362.

[McC89] McCormick, S. T., Pinedo, M. L., Shenker, S. and Wolf, B.: Sequencing in an assembly line with blocking to minimize cycle time. *Operations Research*, v37 n6, 1989, 925-935.

[Mcc92] McClain, J. O., Thomas, L. J. and Mazzola, J. B.: *Operations Management Production of Goods and Services*, Prentice Hall, 1992.

[Met89] Mettala, E. G., and Egbelu, P. J.: Alternative approach to sequencing robot moves for PCB assembly. *International Journal of Computer Integrated Manufacturing*, v2 n5, 1989, 243-256.

[Mit96] Mitchell, M. : *An introduction to genetic algorithms.* The MIT press, 1996.

[Muk92] Mukai, S.: PCB continuous line system proceeds from manufacture to inspection. *Journal of electronic engineering*, v29 n305, 1992, 34-39.

[Mur83] Murty, K. G.: *Linear Programming.* John Wiley & Sons, 1983.

[Mur95a] Murty, K. G.: *Operations Research.* Prentice Hall, 1995.

[Mur95b] Murray, A. T. and Church, R. L.: Heuristic solution approaches to the operational forest planning problem, *OR Spektrum*, v17, 1995, 193-203.

[Mur96] Murray, A. T. and Church, R. L.: Applying simulated annealing to location-planning models. *Journal of Heuristics*, v2, 1996, 31-53.

[Nas96] Nash, S. G. and Sofer, A.: *Linear and Nonlinear Programming.* The McGraw-Hill Companies, 1996.

[Naz95] Nazario, D. and Ramirez-Beltran: Integer programming to minimize labour costs. *Journal of Operational Research Society*, v46 n2, 1995. 139-146.

[Nel95] Nelson, K. M. and Wille, L. T.: Comparative study of heuristics for optimal printed circuit board assembly. *Southcon Conference Record*, 1995, 322-327.

[Nem88] Nemhauser, G. L. and Wolsey, L. A.: *Integer and Combinatorial Optimization.* A Wiley-Interscience Publication, John Wiley & Sons, 1988.

[Orl92] Orlin, J. B. and Ahuja, R. K.: New scaling algorithms for the assignment and minimum mean cycle problems. *Mathematical Programming*, v54, 1992, 41-56.

[Pap81] Papadimitriou, C. H.: On the complexity of integer programming. *Journal of the Association for Computing Machinery*, v28 n4, 1981, 765-768.

[Pet96] Peters, B. A. and Subramanian, G. S.: Analysis of partial setup strategies for solving the operational planning problem in parallel machine electronic assembly systems. *International Journal of Production Research,* v34 n4, 1996, 999-1021.

[Ran85] Randhawa, S. U., Mcdowell E. D., and Faruqui, S. D.: An integer programming application to solve sequencer mix problems in printed circuit board production. *International Journal of Production Research*, v23 n3, 1985, 543-552.

[Ree93] Reeves, C. R.: *Advanced Topics in Computer Science: Modern Heuristic Techniques for Combinatorial Problems.* Blackwell Scientific Publication, 1993.

[Ros75] Ross, G. T. and Soland, R. M.: A branch and bound algorithm for the generalized assignment problem. *Mathematical Programming*, v8, 1975, 92-103.

[Sad93] Sadiq, M. Landers T. L. and Don Taylor G.: A heuristic algorithm for minimizing total production time for a sequence of jobs on a surface mount placement machine. *International Journal of Production Research*, v31 n6, 1993, 1327-1341.

[San95] Santos, D., Kane, J., Caballero, F., and Nagarajan, K.: On the selection of a printed circuit board assembly line system. *Computers and Industrial Engineering*, v29 n1-4, 1995, 591-595.

[Sau94] Saunders, R.: A cell for assembly. *Manufacturing Engineer*, v73 n5, 1994, 234-237.

[Sch86] Schrijver, A.: *Theory of Linear and Integer Programming.* John Wiley & Sons, 1986.

[Sch94] Schwartz, B. L.: A computational analysis of the auction algorithm, *European Journal of Operational Research*, v74, 1994, 161-169.

[Sch96] Scholl, A. and Voβ, S.: Simple assembly line balancing – heuristic approach. *Journal of Heuristics*, v2, 1996, 217-244.

[Sht92] Shtub, A. and Maimon O. Z.: Role of similarity measures in PCB grouping procedures. *International Journal of Production Researches*, v30 n5, 1992, 973-983.

[Soh96] Sohn, J. and Park, S.: Efficient operation of a surface mounting machine with a multihead turret. *International Journal of Production Research*, v34 n4, 1996, 1131-1143.

[Sou95] Souza, R. D. and Lijun, W.: Intelligent optimization of component onsertion in multi-head concurrent operation PCBA machines. *Journal of Intelligent Manufacturing*, v6, 1995, 235-243.

[Tah97] Taha, H. A.: *Operations Research: An Introduction.* Prentice-Hall, sixth edition, 1997.

[Tho86] Thompson, G. L. and Sethi, A. P.: Solution of constrained generalized transportation problems using the pivot and probe algorithm. *Computers and Operations Research*, v13 n1, 1986, 1-9.

[Wat95] Watkins, R. E., and Cochran, J. K.: A line balancing heuristic case study for existing automated surface mount assembly line setups. *Computers and Industrial Engineering*, v29 n1-4, 1995, 681-685.

[Wee81] Wee, T. S. and Magazine, M. J.: An efficient branch and bound algorithm for assembly line balancing – Part 1: minimize the number of work stations. *Working Paper n150*, University of Waterloo, Waterloo, ONT, 1981.

[Wei91] Wein, J. M. and Zenios, S. A.: On the massively parallel solution of the assignment problem. *Journal of Parallel and Distributed Computing*, v13, 1991, 228-236.

[Whi61] White, W. W.: Comments on a paper by Bowman. *Operations Research*, v9, 1961, 274-276.

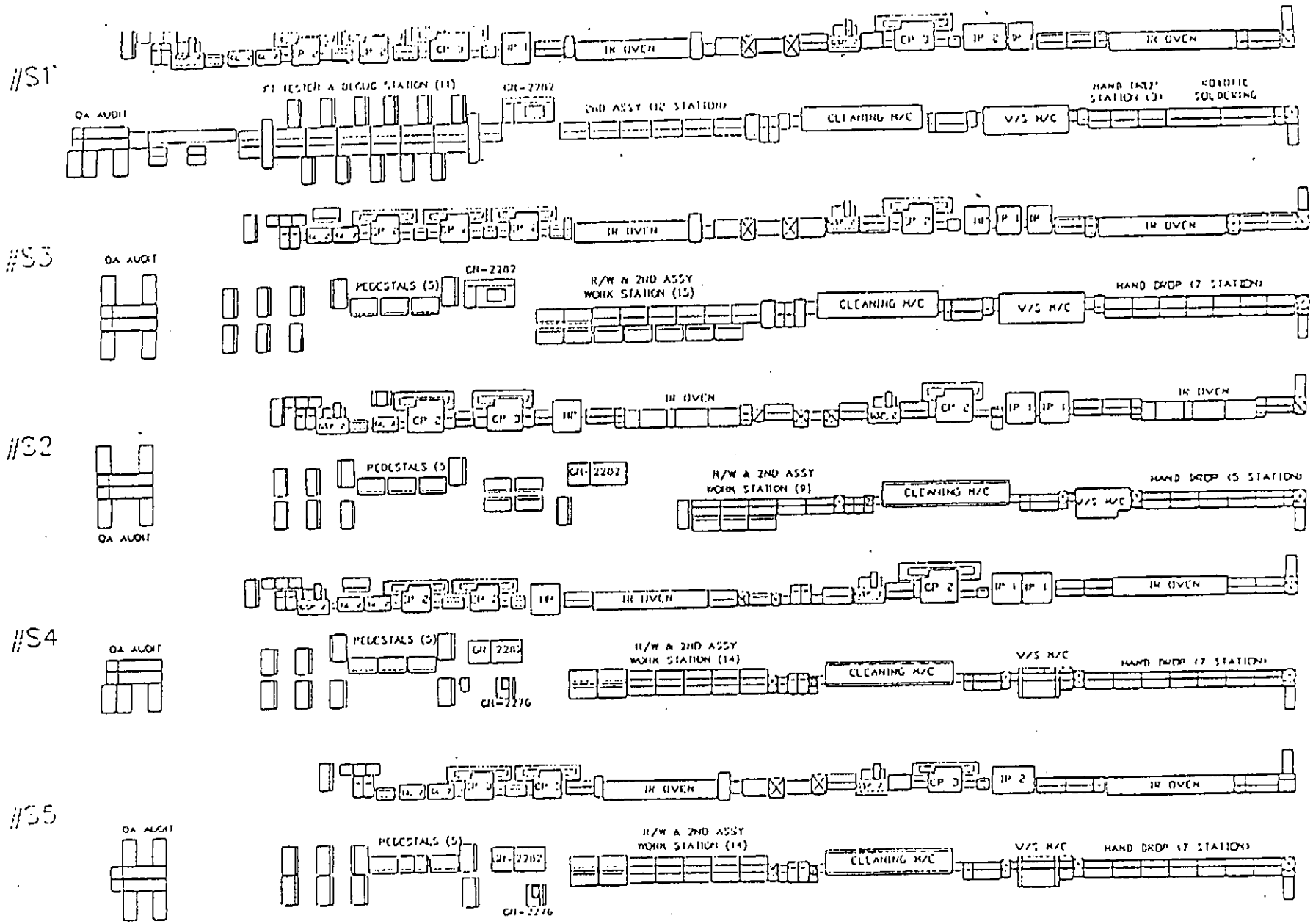
[Wil87] Wilhelm, M., and Ward, T.: Solving the quadratic assignment problems by simulated-annealing. *IIE Transactions*, v19 n1, March 1987, 107-119.

[Yug93] Yu, G. and Kouvelis, P.: Complexity results for a class of min-max problems with robust optimization applications. *Complexity in Numerical Optimization*, Panos M. Paradalos, Editor, 1993, 501-511.

[Zak95] Zaki, H. A.: A comparison of two algorithms for the assignment problem. *Computational Optimization and Applications*, v41, 1995, 23-45.

Appendix I

Line Layout in a PCB Assembly Company



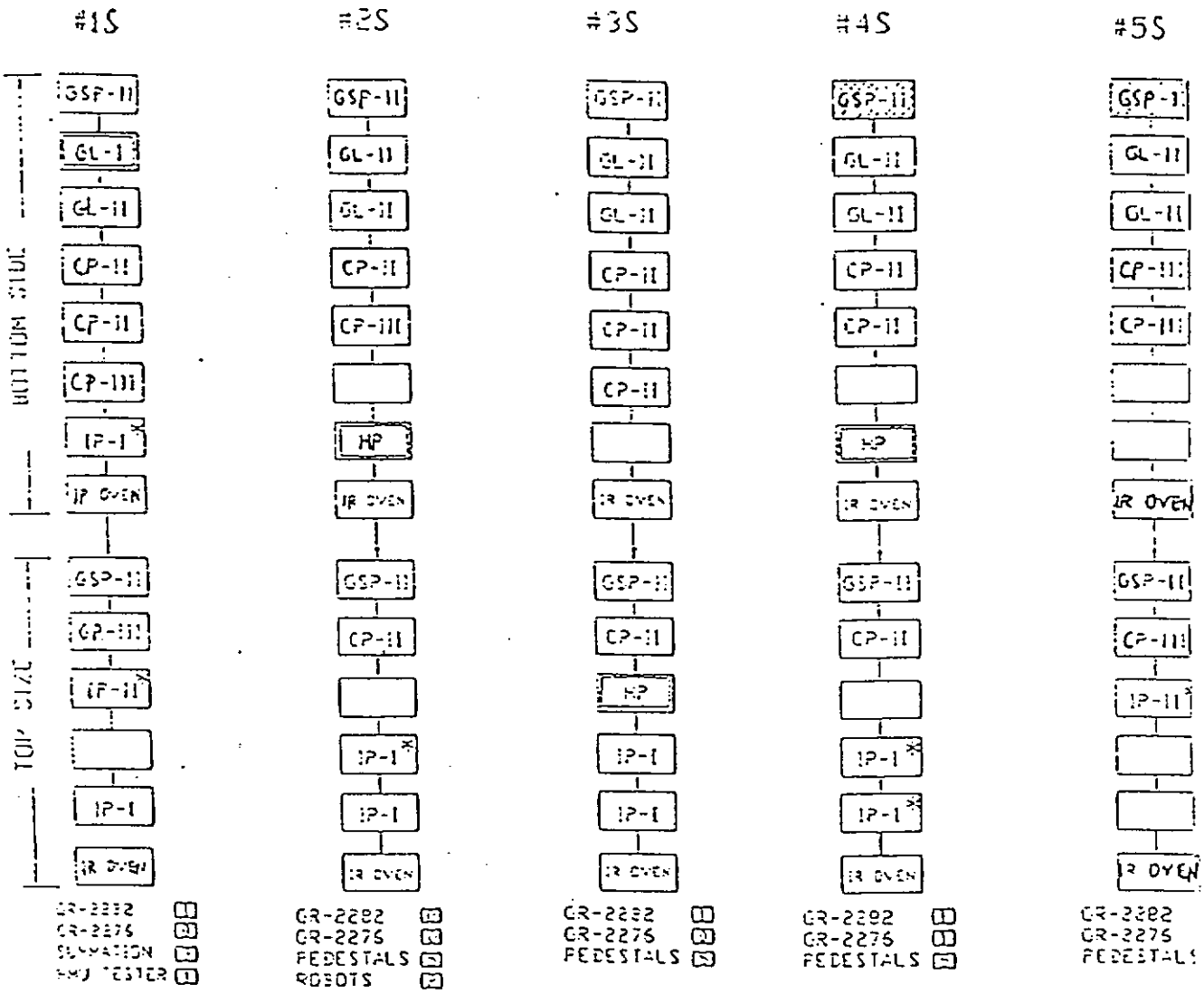
ATTACHMENT I: Plant Layout

Appendix II

Line Configurations

MANUFACTURING ENGINEERING XXX Company

SMT LINES CONFIGURATION



LEGEND :-

UNOCCUPIED MACHINE SPACE

AVAILABLE BY SEP

DIRECT DRIVE

REVISION : 03
FILE : E:\ONG\SI
DATE : . .

Appendix III

Component Placement Times

MACHINE BASIC TIME

A) CP II & CP III		TIME
1) Chip Components (Resistor, Capacitor, Diode, LED, Transistor)	:	0.3 sec/comp.
2) SOIC (8~28 PINS)	:	0.7 sec/comp.
3) Tantalum	:	0.7 sec/comp.
* Fudicial mark, load & unlod PCB board	:	11.0 sec/panel
B) IP-I		
1) Chip Components (Resistor, Capacitor, Diode, LED, Transistor)	:	1.5 sec/comp.
2) SOIC (8~28 PINS)	:	2.5 sec/comp.
3) SOJ IC (24 ~ 28 PINS)	:	3.5 sec/comp.
4) PLCC (20~84 PINS)	:	3.5 sec/comp.
5) PLCC SOCKET	:	3.5 sec/comp.
6) QFP IC (48~160 PINS)	:	3.5 sec/comp.
7) SM CONNECTOR (32~44 x 12~24 size)	:	3.5 sec/comp.
* Fudicial mark, load & unlod PCB board	:	14.67 sec/panel
C) IP-II		
1) Chip Components (Resistor, Capacitor, Diode, LED, Transistor)	:	0.7 sec/comp.
2) SOIC (8~28 PINS)	:	1.2 sec/comp.
3) SOJ IC (24 ~ 28 PINS)	:	1.7 sec/comp.
4) PLCC (20~84 PINS)	:	1.7 sec/comp.
5) PLCC SOCKET	:	1.7 sec/comp.
6) QFP IC (48~160 PINS)	:	1.7 sec/comp.
7) SM CONNECTOR (32~44 x 12~24 size)	:	1.7 sec/comp.
* Fudicial mark, load & unlod PCB board	:	14.67 sec/panel
D) HP		
1) Chip Components (Resistor, Capacitor, Diode, LED, Transistor)	:	2.3 sec/comp.
2) SOIC (8~28 PINS)	:	3.8 sec/comp.
3) SOJ IC (24 ~ 28 PINS)	:	3.8 sec/comp.
4) PLCC (20~84 PINS)	:	3.8 sec/comp.
5) PLCC SOCKET	:	3.8 sec/comp.
6) QFP IC (48~144 PINS)	:	3.8 sec/comp.
* Fudicial mark, load & unlod PCB board	:	14.67 sec/panel

Appendix IV

Source code for the Linear Programming Algorithm

```
#include <iostream.h>  //file at c:\bc5\bin\nothing1
#define MAX 30
struct Entercel{       //define the basic cell
          int machine;
  int compon;};
int getno(const char* infor)
 {
          int no;
  cout << "Enter the number of " << infor << " :";
  cin>>no;
  return no;
 }

 void mcsetup(double Setup[MAX], int M)
 {
                    cout<<"\n"<<"\n";
                    for (int i=1; i<=M; i++)
                    { cout << "Enter the set up time for machine M"<<i<<" in second:  ";
          cin >> Setup[i];
       }
    cout <<"\n"<<"\n";

 }
void comptype(double Quantity[MAX], int C)
{
            for (int i=1; i<=C; i++)
          {cout << "Enter the quantity for component type C" <<i<<": ";
    cin >> Quantity[i];
    }

  }

void plactime(double Time[MAX][MAX], int C, int M)
{
          cout << "\n\nPlease enter the component placement time. \n";
  cout << "Tij refers to component placement time for component j at machine i.\n";
    for (int j=1; j<=C; j++)
  { for (int i=1; i<=M; i++)
    {cout << "T["<<i<<"]["<<j<<"]:";
     cin >> Time[i][j];}
     cout << endl; }
}

void inform(double U[MAX],double V[MAX], double Bmatrix[MAX][MAX], Entercel
T[MAX], double A[MAX], double Quantity[MAX], double Setup[MAX],int C, int M)
{
                    for (int i=1; i<=M; i++)       // set initial solution for ui
    {if (i==1)
          U[i]=1.0;
     else
          U[i]=0.0;}
     for (int i=1; i<=C; i++)    //set initial solution for vj
          V[i]=0.0;
    for (int i=1; i<=M+C; i++)  // set initial B matrix
    {for (int j=1; j<=M+C; j++)
     {    if (i==1 && j<=M)
          Bmatrix[i][j]=1.0;
      else if (i==j && i!=1)
```

```
        Bmatrix[i][j]=-1.0;
      else
         Bmatrix[i][j]=0.0;}}


    for (int i=1; i<=M+C;i++) //set initial cell - Tau
    {if (i==1)
         {T[i].machine=M+1;
         T[i].compon=C+1;}
    else if (i>1 && i<=M)
         {T[i].machine=i;
         T[i].compon=0;}
    else
         {T[i].machine=0;
         T[i].compon=i-M;}}


    for (int i=1; i<=M+C; i++) //set A matrix
     {if (i<=M)
         A[i]=Setup[i]*(-1.0);
     else
          A[i]=Quantity[i-M]*(-1.0);}

}



void AB(double X[MAX], double A[MAX], double Bmatrix[MAX][MAX], int C, int M)
{
         for (int j=1; j<=M+C; j++)     //let X = A*B
                   {        X[j]=0.0;
                            for (int i=1; i<=M+C; i++)
                   {X[j]=X[j]+A[i]*Bmatrix[i][j];}}
}



  void prif(double U[MAX],double V[MAX], double Bmatrix[MAX][MAX], Entercel T[MAX],
double A[MAX],double X[MAX],int C, int M)
{

                   cout.precision(5);
   double obj1, obj2,obj3;
         obj1=0.0;
         obj2=0.0;
   for (int i=1;i<=M;i++)
          { obj1=obj1+U[i]*A[i];}
   for (int i=1;i<=C;i++)
   {obj2=obj2+V[i]*A[M+i];}

   obj3=obj1+obj2;
   cout<<endl<<"The objective is: "<<obj3;



    cout<<endl;
                   for (int i=1; i<=M; i++)        // print initial solution for ui
    {  cout.width(7);
```

[missing page]

Page 131

```
                    for (int i=1; i<=M; i++)
     {    p[i]=Bmatrix[i][k];}
    for (int j=1; j<=C; j++)
     {    q[j]=Bmatrix[j+M][k];}
    }
void Finobasic(double W[MAX][MAX], Entercel T[MAX], double Time[MAX][MAX], double
U[MAX], double V[MAX], double p[MAX], double q[MAX], int C, int M)          //compare the
cell with tau, find the non basic cell.
{
    int Ind[MAX][MAX];
    for (int i=1; i<=M; i++)
    {for (int j=1; j<=C;j++)
         Ind[i][j]=0;}    //index =0 means non-basic cell, =1 means basic cell.

                    for (int i=1; i<=M*C; i++)
    {for (int j=1; j<=M+C; j++)
    {for (int a=1; a<=M+C; a++)
         {if (i==T[a].machine && j==T[a].compon)
          Ind[i][j]=1;}}}

    cout<<endl<<"pi is: ";
    for (int i=1;i<=M;i++)
    {cout<<p[i]<<"  ";}


    cout<<endl<<"qi is: ";
    for (int i=1;i<=C;i++)
    {cout<<q[i]<<"  ";}

    double Ya[MAX][MAX];        //wij=10000 if Ya <=0 or it is basic.
    for (int i=1; i<=M; i++)
    {for (int j=1; j<=C; j++)
         if (Ind[i][j]==0)
                    {Ya[i][j]=-1*Time[i][j]*p[i]+q[j];
                    if (Ya[i][j]<0)
                    {W[i][j]=-Time[i][j]*U[i]/Ya[i][j]+V[j]/Ya[i][j];}
         else
                    W[i][j]=10000;}
     else
        W[i][j]=10000; }

 cout<< endl;
 for (int i=1;i<=M;i++)  //print wij
 {for (int j=1; j<=C; j++)
  if (W[i][j]!=10000)
   cout<<"W"<<i<<j<<" is:  "<< W[i][j]<<endl;}
         }


void Findx(int *a, int *b, double W[MAX][MAX],int C, int M)
{
    double temp=W[1][1];
                    *a=1;
    *b=1;
    for (int i=1; i<=M; i++)
    {for (int j=1; j<=C; j++)
         if (temp>W[i][j])
                    {W[*a][*b]=temp;
```

```
        temp=W[i][j];
          *a=i;
        *b=j;}}

}


 void Update(double Bnew[MAX][MAX],double Bmatrix[MAX][MAX], double
Time[MAX][MAX], double W[MAX][MAX], Entercel T[MAX], double p[MAX], double
q[MAX], double U[MAX], double V[MAX], int x, int y, int k, int C, int M)
 {
         double Bindex;
  Bindex=-Time[x][y]*p[x]+q[y];
  for (int i=1; i<=M+C; i++)
        {Bnew[i][k]= Bmatrix[i][k]/Bindex;}
  for (int j=1; j<=M+C; j++)
  {if (j!=k)                  //Find Bnew matrix
   {for (int i=1; i<=M+C;i++)
         {Bnew[i][j]=(Bmatrix[i][j]+(Time[x][y]*Bmatrix[x][j]-
Bmatrix[M+y][j])*Bnew[i][k]);}}}
  double temp;
  for (int i=1; i<=M+C;i++)
  {for (int j=1; j<=M+C;j++)
  {temp=Bmatrix[i][j];
  Bmatrix[i][j]=Bnew[i][j];
  Bnew[i][j]=temp;}}




  for (int i=1;i<=M;i++)          //find new Ui
   {U[i]=U[i]-W[x][y]*p[i];}

  for (int i=1;i<=C;i++)          //find new Vj
  { V[i]=V[i]-W[x][y]*q[i];}

  //find new tau
   T[k].machine=x;
   T[k].compon=y;

  }




        double Quantity[MAX], Setup[MAX], U[MAX],V[MAX],A[MAX],X[MAX];
  double p[MAX],q[MAX];
  double Time[MAX][MAX],Bmatrix[MAX][MAX],Bnew[MAX][MAX];
  double W[MAX][MAX];
  Entercel T[MAX];


void main()
```

```
{
  int C,M,k,x,y;
  int noiter=0;
  M = getno("machine"); //get machine number
  C = getno("component type"); //get component number
  mcsetup(Setup, M);
        comptype(Quantity,C);
  plactime(Time, C, M);
  inform(U,V,Bmatrix,T,A,Quantity,Setup,C,M);
  AB(X, A,Bmatrix,C,M);
  k=Selectk(X, C, M);
  prif(U,V,Bmatrix,T,A,X,C,M);
  for (;;)
  {
    char Ya;
                cout<<endl;
                cout<<"This is not optimal solution, do you want to cont? ";
    cin>>Ya;
    if (Ya=='n'||Ya=='N')
    {break;}
    else


          {     if (X[k]<=0)
                {cout << endl<<"Optimal solution find,the number of iteration required is :
"<<noiter<<endl;
       break;}
  else
  {
  noiter=noiter+1;
  SelPQ(p, q, Bmatrix,C,  M, k);
  Finobasic(W, T, Time, U, V, p, q, C, M);
  Findx(&x,&y,W, C,M);

  cout<<endl<<endl<<"x is "<<x<<endl;
   cout<<endl<<endl<<"y is "<<y<<endl;
   cout<<"wxy is "<<W[x][y]<<endl;
   Update(Bnew,Bmatrix, Time, W, T,p,q,U,V,x,y,k,C,M);

   AB(X, A,Bmatrix,C,M);
  k=Selectk(X, C, M);



  prif(U,V,Bmatrix,T,A,X,C,M);}

  } }
   }
```

Appendix V

Source Code for the Genetic Algorithm

```
%%%%% GET DATA: GA2.m %%%%%%%
disp('Please select the entering method:');
disp('    1. From screen input');
disp('    2. From data file');

pe=input('Enter 1 or 2 ===> ');
while (pe~=1&pe~=2)==1
 pe=input('Wrong Choice! Re-enter 1 or 2: ');
  if pe==1 | pe==2
    break;
  end
end

if pe==1
          gatdat;
  else
       [filename,path] = uigetfile('*.mat', 'Get File');
       eval(['load ' [path filename]]);
end
disp('Time is :')
disp(T)
disp('Quantity is :')
disp(Q)
disp('Setup time is:')
disp(Setup)


%%%% variables of GA algorithm %%%%%%%%%
disp('_____');
disp('');
disp('Genetic Algorithm:');
GA=input('Enter the required Population: ');
itno=input('Enter the no. of iteration required: ');


%%%%%%%%%        initialization      %%%%%%%%%%%%%%%
disp('')
disp('')

%%%
x=input('Would you like to enter a initial solution, Y/N? ','s');
f=1;
if x=='Y' | x=='y'
 X=input('Enter the deciamal desired solution ');
 [G1,G2,G3]=inytr(X);
 disp('');

 for i=1:3
        data=['G' int2str(i)];
        fitness(i)=max(sum((eval(data))'.*T')+Setup);

 end
 f=4;
end


%%%%%%%%%%%%%        CREAT RANDOM POPULATION
  Comp=sum(Q);
```

```
  for x=f:GA
          V=reassig(Comp,Q,M,C);
          eval(['G' int2str(x) '=V']);
          fitness(x)=max(sum(V'.*T')+Setup);

  end
          p=(sum(fitness)-fitness)/((GA-1)*sum(fitness));
          q=cumsum(p);


%%%%%%%%%%%%      ALGORITHM FOR CROSSOVER
for ite=1:itno
b=ceil(GA*0.5);              %% find the no. of chromosomes for crossover:0.5
  if rem(b,2)~=0
          b=b-1;
  end
Rno=rand([1,b]);          %%  select chromosomes for crossover
for i=1:b
   n(i)=min(find(Rno(i)<q));
end
%%%%%%%%%%  TEST FOR CROSSOVER
for x=1:+2:b
    if all(all(eval(['G' int2str(n(x))])==eval(['G' int2str(n(x+1))])))==0 & n(x)~=n(x+1)  % check
Ga~= Gb and a~=b
          [Da,Db]=crosover(eval(['G' int2str(n(x))]),eval(['G' int2str(n(x+1))]),C);          %call
sub-program%
          eval(['G' int2str(GA+x) '=Da']);
          eval(['G' int2str(GA+x+1) '=Db']);


    elseif all(all(eval(['G' int2str(n(x))])==eval(['G' int2str(n(x+1))])))==1 & n(x)~=n(x+1)
%check Ga=Gb and a~=b
              V=reassig(sum(Q),Q,M,C);        %call sub-program%
              eval(['G' int2str(n(x)) '=V']);
          fitness(n(x))=max(sum(V'.*T')+Setup);
              eval(['G' int2str(GA+x) '=[]']);
              eval(['G' int2str(GA+x+1) '=[]']);


    else
            eval(['G' int2str(GA+x) '=[]']);
            eval(['G' int2str(GA+x+1) '=[]']);

  end
end
%%%%%%%%%%%%      TEST FOR MUTATION
Muno=ceil(rand(1,round(GA*0.3))*GA);  %%%mutation rate:0.3
a=size(Muno,2);              % a is the no. of mutation


for i=1:a
          s=eval(['G' int2str(Muno(i))]);
          Mut=mutat(s);

    if all(all(Mut==s))==0
      eval(['G' int2str(GA+b+i) '=Mut']);
     else
      eval(['G' int2str(GA+b+i) '=[]']);
```

```
    end
end
%%%%%%%%%%%      FIND FITNESS
for i=1:a+b
    V=['G' int2str(GA+i)];
    if eval(V)==[]
          fitness(GA+i)=max(max(T))*Comp;
    else
      fitness(GA+i)=max(sum((eval(V))'.*T')+Setup);
    end
end
%%%%%%%%%%%       Select the best poopulation


[Y,index]=sort(fitness);
pp=find(index(1:GA)>GA);
qq=find(index(GA+1:GA+b+a)<=GA);

re=size(pp,2);

if re~=0
for i=1:re
 if pp(i)==1
          eval(['G' int2str(index(qq(i)+GA)) '=G' int2str(index(pp(i)))]);
 else
    if all(all(eval(['G' int2str(index(pp(i)))])==eval(['G' int2str(index(pp(i)-1))])))==0
          eval(['G' int2str(index(qq(i)+GA)) '=G' int2str(index(pp(i)))]);
    else
          if re~=1 | i~=re
        for j=re:-1:i+1
              qq(j)=qq(j-1);
        end
          end
    end
 end
end
end

%%%%%%%5   RENEW FITNESS AND P, Q
for i=1:GA
  data=['G' int2str(i)];
  fitness(i)=max(sum((eval(data))'.*T')+Setup);
end


bestfit(ite)=min(fitness);
fit=fitness(1:GA);
p=(sum(fit)-fit)/((GA-1)*sum(fit));
q=cumsum(p);


end
x=1:1:ite;
plot(x,bestfit(x))
title('Performance of GA at cycle time determination')
xlabel('Iteration number')
ylabel('Cycle time')
```

[blank page]

```
%%%%%%%    read machine and component no.    Gatdat.m%%%%%%%%%%%%%%

M=input('Enter the number of machines  : ');
C=input('Enter the number of components : ');
%%%%%%%  read time for component number  %%%%%%%%%%%%%%
a='Please enter the quantities for C';
for j=1:C
        J=num2str(j);
        p=[a,J,'  :'];
        Q(j)=input(p);
        end

%%%%%%%  read set up time for each machine  %%%%%%%%%%%%%5
a='Please enter the set up time for M';
for j=1:M
        J=num2str(j);
        p=[a,J,'  :'];
        Setup(j)=input(p);
        end

%%%%%%%%  read placement time for machine i to component j
a='Enter the time for T';
for i=1:M
 for j=1:C
        I=num2str(i);
        J=num2str(j);
        p=[a,I,J,'-----> '];
        T(i,j)=input(p);
 end
end

fil=input('Do you want to save your data, y/n?  ', 's');
if fil=='Y' | fil=='y'
   [filename, path] = uiputfile('*.mat', 'Put File');
   eval(['save ' [path filename]]);
end
```

```
%% Crosover.m%%%
%A  : one of the chromosome
%B  : one of the chromosome, where A!=B
%Ca : no. of column
%----------------------------------------%
%   D,R,j,I,no,Ra, Rb,Da, Db

function [Da,Db]=crosover(A,B,Ca)


D=fix((A+B)/2);
R=rem(A+B,2);
[a,b]=size(A);
Ra=zeros(a,b);
Rb=zeros(a,b);

if any(any®)==1
  for j=1:Ca
    I=find(R(:,j));
     if I~=[]
       no=ceil(rand(1,size(I,1)/2)*size(I,1));
           Ra(I(no),j)=R(I(no),j);
           R(I(no),j)=zeros(size(no,2),1);
         end
  end
 Rb=R;
 Da=D+Ra;
 Db=D+Rb;

else
  Da=D;
  Db=[];
end



%%%% output Da, Db
```

```
%%%Mutat.m%%%
function E=mutat(A)

[M,C]=size(A);
Rono=0;
Colno=0;
while Rono==0 & Colno==0
 Rono=ceil(rand(1)*M);
 Colno=ceil(rand(1)*C);
 if Rono>=1 & Colno>=1
    break;
 end
end

 StRono=ceil(rand(1)*(M-Rono+1))
 StCono=ceil(rand(1)*(C-Colno+1))
 Asub=A(StRono:StRono+Rono-1,StCono:StCono+Colno-1);
 [a,b]=size(Asub);
 su=sum(sum(Asub));
 if Rono==1
    c=Asub;
   else
    c=sum(Asub);
 end
 Asub=reassig(su,c,a,b);

 A(StRono:StRono+Rono-1,StCono:StCono+Colno-1)=Asub;
 E=A;
```

```
%%%Inytr.m%%%
function [G1,G2,G3]=inytr(X)
[m,n]=size(X);
Y=fix(X);
Z=X-Y;
Q=sum(Z);
[U,I]=sort(Z);
G1=Y;
for j=1:n           % assign to higher decimal number
 for i=1:round(Q(j))
        G1(I(m-i+1,j),j)=G1(I(m-i+1,j),j)+1;
 end
end


G2=Y;
for j=1:n           % assign to the lower decimal number
  a=1;
 for i=1:round(Q(j))
    while U(a,j)<=0
         a=a+1;
        end

        G2(I(a+i-1,j),j)=G2(I(a+i-1,j),j)+1;
 end
end

G3=Y;
for j=1:n           %random assign
 for i=1:round(Q(j))

        k=ceil(rand(1)*m);
        G3(k,j)=G3(k,j)+1;
 end
end
```