William Ho
Ping Ji

Springer Series in Advanced Manufacturing

Optimal Production Planning for PCB Assembly

Springer

Springer Series in Advanced Manufacturing

Other titles in this series

Assembly Line Design
B. Rekiek and A. Delchambre

Advances in Design
H.A. ElMaraghy and W.H. ElMaraghy (Eds.)

Effective Resource Management in Manufacturing Systems: Optimization Algorithms in Production Planning
M. Caramia and P. Dell'Olmo

Condition Monitoring and Control for Intelligent Manufacturing
L. Wang and R.X. Gao (Eds.)

William Ho and Ping Ji

Optimal Production Planning for PCB Assembly

With 41 Figures

Springer

William Ho, Ph.D.
Operations and Information Management Group
Aston Business School
Aston University
Aston Triangle
Birmingham, B4 7ET
UK

Ping Ji, Ph.D.
Department of Industrial and Systems Engineering
The Hong Kong Polytechnic University
Hung Hom
Kowloon
Hong Kong

Series Editor:
Professor D. T. Pham
Intelligent Systems Laboratory
WDA Centre of Enterprise in Manufacturing Engineering
University of Wales Cardiff
PO Box 688
Newport Road
Cardiff, CF2 3ET
UK

British Library Cataloguing in Publication Data
Ho, William
Optimal production planning for PCB assembly. - (Springer series in advanced manufacturing)
1.Printed circuits industry - Production control
2.Production planning - Mathematical models 3.Printed circuits - Design and construction
I.Title II.Ji, Ping
621.3'81531'0685
ISBN-13: 9781846284991
ISBN-10: 1846284996

Library of Congress Control Number: 2006933375

Springer Series in Advanced Manufacturing ISSN 1860-5168
ISBN-10: 1-84628-499-6 e-ISBN 1-84628-500-3 Printed on acid-free paper
ISBN-13: 978-1-84628-499-1

9 8 7 6 5 4 3 2 1

Springer Science+Business Media
springer.com

Preface

This book is written mainly for process planners in the electronics industry and for those who are concerned with printed circuit board (PCB) assembly line efficiency. We develop mathematical modeling techniques and heuristic solution approaches to optimize some critical PCB assembly problems arising in the industry. Line assignment, component allocation, component sequencing, and feeder arrangement problems are optimized so that line efficiency can be improved.

In addition, this book is written for undergraduate and postgraduate students in Operations Research and those who are interested in mathematical modeling techniques. We formulate the above mentioned PCB assembly problems as various types of mathematical models, including nonlinear and linear types. We also show the transformation of the minimax type formulation into a minimization one as well as the conversion of the nonlinear type formulation into a linear one. We believe this is the first book to cover the application of mathematical modeling techniques to PCB assembly problems extensively and successfully.

It is indisputable that PCBs play a vital role in our daily life. A wide variety of our "necessities", including personal computers, mobile phones and so on, use PCBs. Due to the fact that the applicability of PCBs nowadays is increasing, one of the crucial ways to increase a PCB company's competitiveness is to minimize production time so that products can be introduced to the market sooner. To attain this goal, the component placement process must be optimized due to the fact that it is generally the bottleneck of a PCB assembly line and dominates the line cycle time. However, there is hardly a book covering this issue or studying PCB assembly problems. This is our primary motivation for writing this book.

Component sequencing and feeder arrangement problems are two interrelated issues for optimizing the component placement process. Therefore, in this book, efforts are mainly made to study and solve these two problems in an appropriate manner. According to the literature published in the past two decades, a prevalent approach adopted by many researchers was to deal with the problems separately for the sequential pick-and-place machine and the concurrent chip shooter machine, which are the two placement machines to be studied in this book. This is definitely not the best approach because machines performance depends heavily on the position of the next component to be placed (*i.e.*, the component sequencing

problem) and which feeder stores the next component to be picked up (*i.e.*, the feeder arrangement problem). Therefore, the problems should be integrated rather than separated.

There are four objectives in this book. To tackle the real problem, the first objective is to integrate the component sequencing and the feeder arrangement problems for both the pick-and-place machine and the chip shooter machine. This is a sophisticated approach but coincides with the actual situation. The second objective is to optimize the component placement process using mathematical modeling techniques. Mathematical models are constructed for the integrated problems for both types of placement machines. Generally, it takes long computational time to find a global optimal solution of a complex optimization problem. So, developing an efficient and effective heuristic solution approach for integrated problems is our third objective. Finally, the fourth objective is to develop a prototype of the PCB assembly planning system. The system comprises the line assignment problem, the component allocation problem, and the integrated problems for both types of placement machines. It provides a user with an interactive interface to PCB assembly planning. With the user friendly system, process planners can get the solutions to the above planning problems easily and quickly.

All the materials in this book come from our research project. Certainly, financial support from The Hong Kong Polytechnic University and The Research Grants Council of Hong Kong made the project possible. Therefore, we express our sincere gratitude to these two organizations.

Furthermore, Dr. William Ho (one of the authors of this book) is genuinely thankful to Dr. Ping Ji (another author) for his persistent guidance and assistance, Madam Chun Fong Yeung for her continuous encouragement, and Miss Pui Ki Wong for her love and concern. I acknowledge all parties once again from the heart including those mentioned above together with those who helped me indeed but missed thanking before.

Contents

1

Introduction

1.1 PCB Assembly Process

In today's digital age, electronic products such as personal computers, mobile phones, and audio-video equipment are ubiquitous. A common point of these products is the application of the printed circuit board (PCB). Actually, a PCB consists of a pattern of electrical traces etched from copper that is laminated on an insulated base, which is typically rigid fiberglass. The PCB serves as the interconnection device with electrical currents traveling on the board and the different discrete electronic components that are essential to the functioning of an electronic product. Components from a few hundred to some thousands can be assembled on a single PCB.

The process of assembling electronic components on a PCB is called PCB assembly. It can be classified into two categories: plated-through-hole (PTH) technology and surface mount technology (SMT). For products where overall board size is not a major concern, the PTH technology is applied. The components are inserted into the holes drilled through the PCB. Then, the connections are soldered on the underside of the PCB between the component lead and the PCB pad. However, requirements from consumers, such as smaller product size and greater function and reliability, have forced the SMT to replace the PTH technology. The configuration and the size of surface mount components have permitted mounting a large number of components on a single PCB.

The PCB assembly process in the SMT environment consists of five operations. First of all, solder paste is applied where the components will be placed. Typically, it is applied by screen printing. Then, it is followed by the placement operation. A high-speed placement machine is used to mount small components such as chip resistors on the PCB first. A flexible placement machine is then used to mount large components such as integrated circuits (ICs) on the PCB. After all the components have been assembled, the PCB is inspected for missing components. Subsequently, the PCB is conveyed through an oven, which makes the solder paste reflow and form the solder joints. Finally, the PCB must be cleaned to remove the contaminants exposed during fabrication and assembly.

The PCB assembly line described above can be referred to as a single-sided assembly, and it is designed for PCBs that need mounting on one side. On the other hand, the assembly line must be redesigned if the PCBs need double-sided mounting. In this situation, more additional placement machines are required to mount the components on the other side of the PCB.

1.2 Assembly Equipment

There are mainly two types of placement machines in the SMT environment. Each type of machine possesses its own peculiarities as well as the operation. The first type of machine is called the sequential pick-and-place (PAP) machine. In this type of machine, components of the same type are stored in a single stationary feeder, whereas the PCB is secured on a fixed working table. During the placement operation, the assembly head travels to pick up one component at a time from a feeder, and then places it on the stationary board. The PAP machine can achieve high accuracy. Moreover, it is suitable for operating with large components such as ICs. The Fuji XP-241E machine belongs to this category.

The second type of machine is the concurrent chip shooter (CS) machine. It possesses an X-Y table carrying a PCB, a feeder carrier with several feeders holding components, and a rotary turret with multiple assembly heads to pick up and place components. Each assembly head has several nozzles of different sizes. A large nozzle is used to pick up and place large components. The major advantage of the CS machine is its high speed because the pickup and placement operations are performed concurrently. However, it is only preferable for operating with small components such as chip resistors. Because the placement of smaller components is given priority, this type of machine is arranged before the PAP machine in the assembly line. The Fuji CP-732E machine belongs to this category.

Although both the PAP machine and the CS machine are SMT placement machines, the configurations as well as the characteristics of both machines are totally different. Table 1.1 summarizes the differences between these two types of placement machines.

Table 1.1. Differences between the PAP and CS machines

	The PAP machine	**The CS machine**
Feeders	Stationary	Movable
PCB	Stationary	Movable
Number of assembly heads	Single	Multiple
Speed	Moderate	High
Assembly operation	Sequential	Concurrent

1.3 PCB Assembly Problems

PCB assembly planning consists of seven decision problems, which can be classified into two closely related issues: setup management and process optimization (Ellis *et al.*, 2001). The meanings of the decision problems are discussed below.

A. Setup management

- Line assignment: assigning board types to assembly lines;
- Machine grouping: grouping placement machines;
- PCB grouping: grouping PCBs into families;
- PCB sequencing: sequencing the production of PCBs.

B. Process optimization

- Component allocation: allocating component types to placement machines;
- Component sequencing: determining the sequence of component placements;
- Feeder arrangement: assigning component types to feeders at each machine.

Among the five assembly operations, component placement is generally the most time-consuming (De Souza and Wu, 1995; Ong and Khoo, 1999; Ong and Tan, 2002). In addition, it is frequently a bottleneck in an assembly line (Dikos *et al.*, 1997; Csaszar *et al.*, 2000; Ong and Tan, 2002) and determines the line cycle time (Wilhelm and Tarmy, 2003). Due to the large production volumes, a minor reduction in cycle time will save significant production time. For example, to produce 50,000 boards, a reduction of six seconds in cycle time will save 5,000 minutes, that is, more than 80 working hours. Therefore, to increase the efficiency or minimize the cycle time of the line, the component placement process must be optimized. Optimization of the component placement process includes two interrelated problems: the component sequencing problem and the feeder arrangement problem. So, the focus of this book is mainly confined to these problems.

Generally, the component sequencing problem is akin to the traveling salesman problem (TSP). This is the problem of finding the optimal placement order to visit a set of components and return home with a minimum assembly distance or assembly time for the PAP and the CS machines. On the other hand, the feeder arrangement problem is very like the quadratic assignment problem (QAP). This is the problem of determining the optimal arrangement to assign a set of component types to feeders with a minimum assembly distance or assembly time for both types of machines.

For the PTH technology, the sequence of the insertion operation in the auto-insertion machine can be simply formulated as the TSP (Chan and Mercier, 1989), and it is not necessary to consider the feeder arrangement problem. However, in the SMT environment, the efficiency of the component placement process is also dependent on a feeder to hold which types of components besides the pick and placement sequence. If the arrangement of components to feeders is not done carefully, even if the pick and placement sequencing is optimally solved, it can

cause significant deterioration in machines performance (Altinkemer *et al.*, 2000). So, certainly, the component sequencing problem as well as the feeder arrangement problem should be solved simultaneously. And, this is what this book is going to do.

1.4 Scope of This Book

In this book, four special tasks associated with the optimization of the component placement process are performed. First of all, the component sequencing and the feeder arrangement problems should be integrated rather than separated as many researchers did. The integrated problem is extremely intricate because it is somewhat similar to the combination of the TSP and the QAP; both of them belong to NP-hard problems (Burkard *et al.*, 1991; Freisleben and Merz, 1996a,b).

The second task is to build a mathematical model for the integrated problem. It aims at obtaining the optimal sequence of component placements and the optimal assignment of component types to feeders simultaneously for each type of machine. It was revealed in Table 1.1 that the PAP machine is completely different from the CS machine, especially in assembly operation. So, it is certain that the mathematical model formulated for one machine type is not appropriate to another. Besides, the feasibility of transforming mathematical models for both types of machines into simpler form is also studied.

A PCB has hundreds of components to be placed. The computational time of the mathematical models for integrated problems is tremendous. So, the third task of this book is to develop a heuristic method to solve the integrated problem efficiently and effectively to get a reasonably good solution in a reasonable time. In particular, the genetic algorithm approach is studied due to its applicability as well as flexibility.

The fourth task is to develop a prototype of the PCB assembly planning system. The system comprises three levels in which the problems are closely related. After different board types have been assigned to multiple assembly lines, that is, the line assignment problem (Level 1), the components or the component types of the board are allocated to multiple placement machines in a particular line, that is, the component allocation problem (Level 2). At the last stage, Level 3, the component sequencing problem and the feeder arrangement problem in each of the placement machines are determined. The integration of these problems is regarded as a PCB assembly planning system.

The structure of the book is organized as follows. In Chapter 2, a detailed description of the optimization techniques used for solving PCB assembly problems is presented. The techniques include mathematical programming, commonly used exact algorithms, metaheuristics, and commercial packages.

In Chapter 3, the component sequencing and the feeder arrangement problems for the PAP machine are studied. First of all, a comprehensive literature review is presented to show how the previous researchers tackled the problems during the past two decades. Several mathematical models, including the individual and the integrated, are then formulated for the problems after the operation of the machine is investigated thoroughly. In addition, two different approaches are compared with

respect to the capability of obtaining a global optimal solution. They are the iterative approach (*i.e.*, solving the individual mathematical models sequentially) and the integrated approach (*i.e.*, optimizing the integrated mathematical model). The verification of the models is carried out by commercial packages, including BARON and CPLEX. The complexity of the models is also compared to find the best formulation that requires the least computational time to reach the global optimum. Besides, a genetic algorithm hybridized with several improved heuristics is developed to solve the problems simultaneously for the PAP machine. The performance of the algorithm is compared with the approaches proposed by other researchers and also compared with the optimal solution obtained by solving the integrated mathematical models.

In Chapter 4, the content is similar to that in Chapter 3 except that the focus is confined to the CS machine. Mathematical models are constructed for the component sequencing and the feeder arrangement problems in advance. After that, the genetic algorithm used to optimize the PAP machine performance is modified so that it is desirable for the CS machine.

In Chapter 5, the line assignment and the component allocation problems are studied. To optimize each of the problems, mathematical modeling and genetic algorithms are applied. The procedure of the algorithms is described step by step with the aid of numerical examples. Furthermore, the performance of the algorithms is compared with the optimal solutions generated by solving the mathematical models.

In Chapter 6, a prototype of the PCB assembly planning system (PCBAPS) is developed. A guide to using the PCBAPS and graphical user interfaces of the system are presented.

In summary, the objectives of this book are

- to integrate the component sequencing and the feeder arrangement problems for both the PAP and the CS machines,
- to apply mathematical programming to the optimization of the component placement process,
- to develop a modern heuristic approach to the integrated problems, and
- to develop a PCB assembly planning system.

2

Optimization Techniques

2.1 Introduction

There are mainly two approaches to optimizing PCB assembly problems. The first one is to formulate the problems as mathematical models and then solve them to optimality using exact algorithms or commercial packages. The second one is simply to generate good solutions of the problems using metaheuristics. Although the latter approach can generate solutions efficiently, no one knows how good the solutions are unless the optimal solution is known in advance.

This chapter is organized in the following way: Section 2.2 presents several types of mathematical programming in Operations Research, which can be applied to PCB assembly problems. Section 2.3 and Section 2.4 survey commonly used exact algorithms and metaheuristics, respectively. Section 2.5 describes the commercial packages including those adopted in this book and some other prevalently used ones. Finally, some remarks concerning this chapter are summarized in Section 2.6.

2.2 Mathematical Programming

Many researchers used mathematical programming models in dealing with the PCB assembly problems. A mathematical programming model is a mathematical representation of the actual situation that may be used to make better decisions or simply to understand the actual situation better (Winston and Venkataramanan, 2003). The common feature which mathematical programming models have is that they all involve optimization (Williams, 1999). In PCB assembly problems, optimization includes the minimization of something (*e.g.*, setup time, placement time, and so on) or the maximization of something (*e.g.*, throughput, workload balance, and so on), under certain constraints (*e.g.*, machine capacity, available production time, and so on).

In the following subsections, attention is confined to linear programming models, integer linear programming models, and nonlinear programming models.

They are presented and studied because the component sequencing and the feeder arrangement problems for the sequential pick-and-place machine (in Chapter 3) and the concurrent chip shooter machine (in Chapter 4), and the line assignment and the component allocation problems (in Chapter 5) can be formulated with these types of models.

2.2.1 Linear Programming

A model is defined as linear program (LP) when the objective function and the constraints involve linear expressions and the decision variables are continuous. Comparatively, LP models are given so much attention in comparison with nonlinear programming models because they are much easier to solve. The transportation model, first described by Hitchcock in 1941, is a special class of LP (Williams, 1999). Suppose that a number of suppliers (i = 1, 2, …, m) provides a commodity to a number of customers (j = 1, 2, …, n). The transportation problem is how to meet each customer's requirement, d_j, while not exceeding the capacity of any supplier, s_i, at minimum cost, c_{ij}. By introducing variables x_{ij} to represent the quantity of the commodity sent from supplier i to customer j, the transportation model can be written as (Winston and Venkataramanan, 2003)

$$\text{Minimize } z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.1}$$

subject to

$$\sum_{j=1}^{n} x_{ij} \le s_i \qquad i = 1, 2, \ldots, m \tag{2.2}$$

$$\sum_{i=1}^{m} x_{ij} \ge d_j \qquad j = 1, 2, \ldots, n \tag{2.3}$$

$$\text{All } x_{ij} \ge 0. \tag{M2-1}$$

The objective function (2.1) is to minimize the total transportation cost. Constraint set (2.2) is known as a supply or availability constraint, whereas constraint set (2.3) is known as a demand or requirement constraint. M2-1 is referred to as the transportation model. If the total supply equals total demand, then the problem is said to be a balanced transportation problem. In this case, constraint sets (2.2), and (2.3) are treated as both "=" instead of "≤" and "≥", respectively.

2.2.2 Integer Linear Programming

Integer linear programming or integer programming (IP) is widely adopted as a method of modeling because some variables are not continuous but integers in many cases in real life. Actually, IP is a subset of LP, with an additional constraint that some or all decision variables are restricted to integral values depending on the

type of IP. Generally, there are three types of IP. First, IP is called pure integer linear programming if all variables must be integers. Second, IP is called mixed integer linear programming if only some of the variables must be integers. Third, IP is called binary integer linear programming if all the variables must be either 0 or 1 (Winston and Venkataramanan, 2003).

IP has important practical applications. However, it was pointed out that computational experience with IP has been less than satisfactory (Taha, 2003).

The traveling salesman problem (TSP) is one of the most widely studied IP problems. The TSP can be easily stated as follows. A salesman wants to visit n distinct cities and then return home. He wants to determine the sequence of the travel so that the overall travel distance is minimized while visiting each city not more than once. Although the TSP is conceptually simple, it is difficult to obtain an optimal solution. In an n-city situation, any permutation of n cities yields a possible solution. As a consequence, $n!$ possible tours must be evaluated in the search space. By introducing variables x_{ij} to represent the tour of the salesman travels from city i to city j, one of the common IP formulations for the TSP can be written as (Winston and Venkataramanan, 2003)

$$\text{Minimize } z = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} c_{ij} x_{ij} \tag{2.4}$$

subject to

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, 2, \ldots, n; i \neq j \tag{2.5}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, 2, \ldots, n; i \neq j \tag{2.6}$$

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad i, j = 2, 3, \ldots, n; i \neq j \tag{2.7}$$

All $x_{ij} = 0$ or 1. All $u_i \geq 0$ and is a set of integers. (M2-2)

The distance between city i and city j is denoted as c_{ij}. The objective function (2.4) is simply to minimize the total distance traveled in a tour. Constraint set (2.5) ensures that the salesman arrives once at each city. Constraint set (2.6) ensures that the salesman leaves each city once. Constraint set (2.7) is to avoid the presence of a subtour.

The TSP formulated for the component sequencing problem is known as the Euclidean TSP, in which the distance matrix c is expected to be symmetrical, that is, $c_{ij} = c_{ji}$ for all i, j, and to satisfy the triangle inequality, that is, $c_{ik} \leq c_{ij} + c_{jk}$ for all distinct i, j, k.

2.2.3 Nonlinear Programming

In the previous subsections, LP as well as IP has been studied. For an LP, the objective is to minimize or maximize a linear function subject to linear constraints. Although LP problems are very common and cover a wide range of problems, the objective function may not be a linear function, or some of the constraints may not be linear in a real-life situation. Such an optimization problem is called a nonlinear programming (NLP) problem.

The quadratic assignment problem (QAP) is a generalization of the linear assignment problem. The major difference between them is that the objective function of the QAP is in a nonlinear expression. Therefore, it is comparatively difficult to solve. The QAP can be described as follows. Consider a set of facilities (i, k = 1, 2, …, n) placed uniquely in a set of locations (j, l = 1, 2, …, n). The workflow intensity between each pair of facilities is a_{ik} while the distance between each pair of locations is b_{jl}. Also, a fixed cost c_{ij} associated with the placement of facility i in location j is specified. The formulation of the QAP can be written as (Burkard *et al.*, 1991; Williams, 1999)

$$\text{Minimize } z = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{\substack{k=1\\k\neq i}}^{n}\sum_{\substack{l=1\\l\neq j}}^{n} a_{ik}b_{jl}x_{ij}x_{kl} + \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{2.8}$$

subject to

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, 2, \ldots, n \tag{2.9}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, 2, \ldots, n \tag{2.10}$$

$$\text{All } x_{ij} = 0 \text{ or } 1. \tag{M2-3}$$

The decision variables x_{ij} represent the placement of facility i in location j. Often, the objective function (2.8) is to assign facilities to locations so that the travel distance of material flow is minimized, while assuming that the cost of assigning a facility does not depend upon the location, that is, $c_{ij} = 0$. Constraint set (2.9) ensures that each location must be occupied by only one facility. Constraint set (2.10) ensures that each facility must be assigned only to one location.

2.3 Exact Algorithms

A set of fixed computational rules for solving a particular class of problems or models is known as an algorithm. It applies the rules repetitively to the problem or the model, each iteration moves the solution closer to the optimum. In Operations Research, there does not exist an algorithm that solves all types of mathematical models. For example, the simplex algorithm is the general method for solving LP

models, whereas the branch-and-bound algorithm is the general technique for solving IP models.

2.3.1 Algorithms for Linear Programming

Finding an optimal solution to an LP model can be regarded as assigning values to the decision variables so that the specified objective is achieved and the constraints are not violated. In the following, two commonly used algorithms for solving the LP models are discussed: the simplex algorithm and the interior point algorithm.

2.3.1.1 The Simplex Algorithm
The simplex algorithm has proved highly efficient in practice and therefore was widely adopted in commercial optimization packages for solving any LP model (Jensen and Bard, 2003). Its development was based on the graphical method that the optimal solution is always associated with a corner point of the solution space. The idea of the simplex algorithm is to move the solution to a new corner that has the potential to improve the value of the objective function in each iteration. The process terminates when the optimal solution is found (Taha, 2003).

2.3.1.2 The Interior Point Algorithm
The simplex algorithm searches for the optimal solution along the corner points of the solution space, whereas the interior point algorithm looks for the optimum through the interior of the feasible region (Jensen and Bard, 2003). The interior point algorithm has theoretical importance that it provides a bound on the computational effort required to solve a problem that is a polynomial function of its size. But, there is no polynomial bound available in the simplex algorithm (Carter and Price, 2001; Jensen and Bard, 2003).

2.3.2 Algorithms for Integer Linear Programming

Unlike LP with the simplex algorithm, a good IP algorithm for a very wide class of IP problems has not been developed (Williams, 1999). Different algorithms are good with different types of problem. Generally, IP algorithms are based on exploiting the tremendous computational success of LP. Thus, before applying an IP algorithm, the integer restriction on the problem should be relaxed first to form an LP model. Starting from the continuous optimum point obtained from the LP model, integer constraints are incorporated repeatedly to modify the LP solution space in a manner that will eventually render the optimum extreme point satisfying the integer requirements.

2.3.2.1 The Branch-and-Bound Algorithm
In practice, the branch-and-bound (B&B) algorithm is widely used for solving IP models, especially mixed integer linear programming (MIP) models (Williams, 1999). The idea of the B&B algorithm is to perform the enumeration efficiently so that not all combinations of decision variables must be examined. Sometimes, the terms “implicit enumeration”, “tree search”, and “strategic partitioning” are used depending on the implementation of the algorithm (Jensen and Bard, 2003).

The B&B algorithm starts with solving an IP model as an LP model by relaxing the integrality conditions. In case the resultant LP solution or the continuous optimum is an integer, this solution will also be the integer optimum. Otherwise, the B&B algorithm sets up lower and upper bounds for the optimal solution. The branching strategy repetitively decreases the upper bound and increases the lower bound. The process terminates, provided that the processing list is empty (Castillo *et al.*, 2002).

2.3.2.2 The Cutting Plane Algorithm

As with the B&B algorithm for solving IP models, the cutting plane algorithm relaxes the integrality requirements of the IP models and solves the resultant LP. But rather than repetitively imposing restrictions on the fractional variables, as is done in the B&B algorithm, extra constraints (*i.e.*, cutting planes) are systematically added to the model, and the model is then resolved. The new solution to the further constrained model may or may not be an integer. By continuing the process until an integer solution is found or the model is shown to be infeasible, the IP model can be solved (Williams, 1999; Jensen and Bard, 2003).

2.3.3 Algorithms for Nonlinear Programming

The quadratic assignment problem (QAP), as shown in Section 2.2.3, belongs to the NLP model because there is a nonlinear expression in the objective function. In addition, the QAP is a binary NLP model as the decision variable is either 0 or 1. But, if an NLP model consists of both integer and continuous variables, it is regarded as mixed integer nonlinear programming model (MINLP). In this book, the integrated problem for the concurrent chip shooter machine will be formulated as this type of model. Therefore, the algorithms for solving the MINLP models are discussed.

Actually, the MINLP problems are the most difficult optimization problems of all. They combine all the difficulties of both the MIP as well as the NLP. Also, they do not have the properties of the MIP or the NLP. For example, a local minimum is equivalent to the global minimum for convex NLP problems. But this result does not hold for MINLP problems. Therefore, MINLP problems belong to the class of NP-complete problems (Kallrath, 1999).

There are two categories of MINLP problems: convex and nonconvex. In the following, the generalized benders decomposition, an algorithm for solving convex MINLP problems, and the branch-and-reduce algorithm, an algorithm for optimizing nonconvex MINLP problems, are described in brief.

2.3.3.1 The Generalized Benders Algorithm

In the generalized benders decomposition, two sequences of updated upper and lower bounds are generated. The upper bounds correspond to solving subproblems in continuous variables by fixing the integer variables, while the lower bounds are based on duality theory. The algorithm terminates if the lower and the upper bounds equal or cross each other (Floudas, 2000).

2.3.3.2 The Branch-and-Reduce Algorithm

The branch-and-reduce algorithm is the extended version of the B&B algorithm for optimizing nonconvex MINLP problems in which valid convex underestimating NLPs can be constructed for the nonconvex relaxation. Due to the fact that nonconvex NLPs must be underestimated at each node, convergence can be achieved only if the continuous variables are branched. A number of tests are suggested to speed up the reduction of the solution space, including the optimality-based range reduction tests and the feasibility-based range reduction tests (Floudas, 2000; Tawarmalani and Sahinidis, 2002).

2.4 Metaheuristics

For many exact algorithms, the computational effort required is an exponential function of the problem size. In a sense, therefore, it may be necessary to abandon the search for the optimal solution using the exact algorithms and simply seek a good solution in a reasonable computational time using heuristics. In Operations Research, the term "heuristic" refers to the methods for the problem under study, based on rule of thumb, common sense, or adaptations of exact methods for simpler models. They are used to find reasonable solutions when the problems are complex and difficult to solve. In optimization, a heuristic method refers to a practical and quick method based on strategies that are likely to (but not guaranteed to) lead to a solution that is approximately optimal or near-optimal (Murty, 1995).

Heuristics can be classified as either constructive (greedy) heuristics or as local search heuristics (Walser, 1999). First, greedy heuristics, such as the nearest neighbor heuristic, simply list all the feasible solutions of the problem under study, evaluate their objective functions, and pick the best as the output of the model. This approach of complete enumeration is likely to be grossly inefficient especially when the number of possible solutions to the problem is vast. So, greedy heuristics are not desirable for solving combinatorial optimization problems, and conversely, local search heuristics are more suitable.

Second, local search heuristics, such as the 2-opt local search heuristic, are based on the concept of exploring the vicinity of the current solution. Neighboring solutions are generated by a move-generation mechanism. If the generated neighbor has a better objective value, it becomes a new current solution, or otherwise the current solution is retained. The process is iterated until there is no possibility of improvement in the neighboring solution. The method then terminates at a point called local optimum, which may be far from any global optimum, as shown in Figure 2.1. This is one of the disadvantages of simple local search methods.

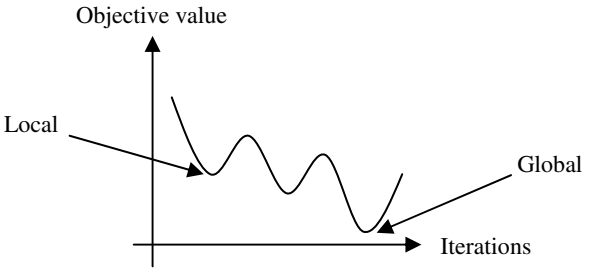

Figure 2.1. Global and local optimum

To avoid getting trapped at a local optimum, a number of conceptual metalevel strategies have been developed for local search heuristics. These strategies are referred to as metaheuristics (Osman and Kelly, 1996). A metaheuristic is an iterative generation process that guides a subordinate or simple heuristic by combining intelligence, biological evolution, neural systems, and statistical mechanics for exploring and exploiting the search spaces using learning strategies to structure information to find near-optimal solutions efficiently. The families of metaheuristics include genetic algorithms, the greedy random adaptive search procedure, problem-space search, neural networks, simulated annealing, tabu search, threshold algorithms, and their hybrids.

In the following subsections, three metaheuristics will be described briefly. These three approaches are very general and applicable to a wide range of problems, while yielding reasonable performance in terms of speed and good performance in terms of the quality of the solutions generated. In addition, the Committee on the Next Decade of Operations Research has singled out these approaches as "extremely promising" for the future treatment of practical applications (Glover *et al.*, 1993). They are simulated annealing (SA), tabu search (TS), and genetic algorithms (GAs).

2.4.1 Simulated Annealing

Simulated annealing (SA), introduced by Kirkpatrick, Gelatt, and Vecchi in 1983, is a technique combining the concepts of statistical mechanics. SA is based on an analogy between the annealing process and the technique of solving the combinatorial optimization problem. SA starts with an initial solution and repeatedly generates a neighbor solution. A neighbor solution is always accepted if there is an improvement in the objective value. However, if it is worse, the solution may be accepted and this acceptance will depend on the control parameter (temperature). To apply SA to a practical problem, there are several choices to be made. The choices can be divided into two main categories: problem-specific and generic. Table 2.1 illustrates the several choices (Johnson *et al.*, 1989; Rayward *et al.*, 1993; Osman and Kelly, 1996).

Table 2.1. Choices to be made in implementing simulated annealing

Problem-specific:	- What is the solution representation?
	- What is the objective function?
	- What are the neighborhood generation mechanisms?
	- How do we determine an initial solution?
Generic:	- How do we determine an initial temperature?
	- What is the temperature update rule?
	- How many iterations must be performed at each temperature?
	- What is the stopping criterion?

2.4.2 Tabu Search

Tabu search (TS) was developed by Glover and Hansen in 1986 for solving combinatorial optimization problems. TS, like SA, is based on local search heuristics with local-optima avoidance, but in a deterministic way which tries to model human memory processes. In other words, TS is an iterative metaheuristic search procedure combining the concepts of artificial intelligence.

TS starts with an initial current solution, which can be generated randomly. Then, the method generates a list of all neighborhood solutions, which is known as the candidate list, from the current solution. Next, all solutions in the candidate list are evaluated, and the best solution from the candidate list will be selected. Sometimes, the best solution may not be selected if the solution is in the tabu memory lists. The lists can be divided into two parts, which are recency (short-term) memory, and frequency (long-term) memory. Both memories are responsible for recording the history of the search, and especially the moves, called attributes, have participated in generating past solutions. The mechanism attempts to avoid the cycling behavior of the method. If the selection is forbidden (tabu), the method proceeds to select the second best solution in the candidate list as the new current solution. On the other hand, if the current solution is better than the specified aspiration level or best fitness value found so far, the solution's tabu status is overridden and the solution is still admissible as the next current solution. The current best solution is updated if necessary, and then a new list of candidate solutions is generated around the new current solution. The procedure continues until the stopping criteria are satisfied (Glover *et al.*, 1993; Glover and Laguna, 1997).

2.4.3 Genetic Algorithms

Genetic algorithms (GAs) were developed by Holland in the 1960s. Only recently, their potential for solving combinatorial optimization problems has been explored. Similar to SA and TS, GAs can avoid getting trapped in a local optimum by the aid of one of the genetic operators called mutation. Actually, the basic idea of GAs is to maintain a population of candidate solutions that evolves under selective

pressure. Hence, they can be viewed as a class of local search based on a solution-generation mechanism operating on attributes of a set of solutions rather than attributes of a single solution by the move-generation mechanism of the local search methods, like SA and TS (Osman and Kelly, 1996). As GA is selected as a heuristic method to solve the problems in this book, it will be described more thoroughly in the following chapters.

In recent years, many researchers discovered that a simple GA was not desirable for solving combinatorial optimization problems with a large problem size. Therefore, they incorporated local search heuristics into the GA, which is called the genetic local search (GLS), for solving the TSP (Freisleben and Merz, 1996a,b) and the QAP (Huntley and Brown, 1996; Ahuja *et al.*, 2000). Experimental results showed that the GLS could solve the TSP and the QAP effectively. For instance, it was found that the GLS obtained the optimal solution for a TSP with 1,400 cities after 200 iterations (Freisleben and Merz, 1996b).

Commonly used local search heuristics can be classified into three main categories: 2-opt, 3-opt, and Lin-Kernighan (LK). For the 2-opt local search heuristic, a neighboring solution is obtained from the current solution by deleting two edges, reversing and reconnecting the two resultant paths in a different way to form a new tour. For the 3-opt local search heuristic, three edges are deleted instead of two. The resultant paths are combined in the best possible way. 3-opt is much more effective than 2-opt, though the size of the neighborhood is larger, and hence more time-consuming to search. To improve 3-opt further, Lin and Kernighan developed a sophisticated edge exchange procedure where the number of edges to be exchanged is variable (Reinelt, 1994).

2.5 Commercial Packages

It is worth writing very sophisticated and efficient computer programs for algorithms when they are used frequently for solving many different models. Such programs, usually consisting of a number of algorithms collected together, are called a “package” of computer routines. Many such package programs are available commercially for solving mathematical programming models. When a mathematical programming model is built, it is usually worth using an existing package to solve it rather than getting diverted onto the task of programming the computer to solve the model oneself (Williams, 1999).

In this book, integrated problems for the sequential pick-and-place machine and the concurrent chip shooter machine are formulated as integer nonlinear programming models presented in Chapter 3 and Chapter 4, respectively. Then, each of the models is equivalently converted into an integer linear programming model. To verify the models, two commercial packages are used. First, BARON is adopted to solve integer nonlinear programming models, whereas CPLEX is used for optimizing integer linear programming models. In the following, the characteristics and the working principles of both commercial packages are described briefly. In addition, some existing commercial packages not used in this book are discussed.

2.5.1 BARON

BARON is a computational system for solving nonconvex optimization models to global optimality. Purely continuous NLP models, purely integer NLP models, and MINLP models can be solved with the software. This is the reason why it is adopted to solve the integer nonlinear programming models presented in Chapter 3 and Chapter 4. BARON combines constraint propagation, interval analysis, and duality for efficient range reduction with rigorous relaxation constructed by enlarging the feasible region and/or underestimating the objective function.

2.5.2 CPLEX

CPLEX is used as an integer linear programming solver in this book because it is powerful in solving LP and MIP problems. For problems with integer variables, CPLEX uses a branch-and-bound search with modern algorithmic features, such as cuts and heuristics, to solve a series of LP subproblems. Because a single MIP generates many LP subproblems, even a small MIP can be very computationally intensive and requires significant amounts of physical memory.

2.5.3 Others

Nowadays, there are numerous commercial packages available for tackling different types of mathematical programming problems. For instance, apart from BARON, DICOPT is a framework for solving MINLP models using standard MIP and NLP solvers to solve MIP and NLP subproblems generated by the algorithm. SBB is another MINLP solver. It is based on a combination of the standard B&B algorithm and some of the standard NLP solvers for subproblems.

On the other hand, except for CPLEX, LINDO is also widely used as an MIP solver. The base version includes primal and dual simplex solvers. For models with integer restrictions, LINDO includes an exceptional integer solver with default settings selected to work well on broad classes of integer models. OSL includes a set of stand-alone solvers for the MIP problem. A branch-and-bound technique is used for MIP, whereas the simplex algorithm is used to solve LP subproblems.

2.6 Summary

Various optimization techniques appropriate for the PCB assembly problems have been discussed in this chapter. Some remarks concerning these techniques are summarized as follows:

1. PCB assembly problems can be formulated as different types of mathematical programming models, including linear programming, integer linear programming, and nonlinear programming models.
2. An algorithm does not exist that solves all types of mathematical models. For instance, the simplex algorithm is the general method for solving linear programming models but not integer linear programming models.

3. Simulated annealing, tabu search, and the genetic algorithms (GAs) are commonly used metaheuristics. Note that each metaheuristic possesses its own characteristics and there is no special one that is acknowledged as the best.
4. GAs will be adopted to solve integrated problems for both types of placement machines, the line assignment problem, and the component allocation problem. The reason is that GAs have been applied successfully in a wide variety of optimization problems such as the TSP, the QAP, and the minimum spanning tree problem (Gen and Cheng, 1997). In addition, the merits of GAs, including simplicity, ease of operation, and flexibility, are the encouraging factors for applying it.
5. BARON and CPLEX are commercial packages used to solve integer nonlinear programming models and integer linear programming models to be formulated in this book, respectively.

In the next chapter, the component sequencing problem and the feeder arrangement problem are studied for the sequential pick-and-place machine. Each of the problems is formulated as an individual mathematical model first. Because of their inseparable relationship, one cannot be solved unless the solution of the other one is obtained beforehand. Therefore, two mathematical models in nonlinear form are constructed for the integrated problem. The nonlinear programming models are also converted equivalently into two linear programming models. These models are compared in terms of computing complexity as well as the computational time taken for obtaining the global optimal solution. To achieve this goal, BARON and CPLEX are used. Besides applying mathematical modeling, a genetic algorithm incorporating several improved heuristics is developed.

3

The Sequential Pick-and-Place (PAP) Machine

3.1 Introduction

Mathematical modeling is a powerful tool in today's life. Without applying it, the optimal solution of a particular problem cannot be obtained. Although heuristic methods and simulation are alternative tools for solving a problem, no one can assure that the solution generated using these tools is optimal or even knows how good the solution is before the optimal solution has been found. In this chapter, mathematical modeling is applied to optimize the performance of the sequential pick-and-place (PAP) machine so that the highest throughput can be achieved.

The placement head, which is the only movable mechanism in this type of machine, has to move to a feeder, pick up a component from the feeder, move to the desired location on the PCB, and place the component there to assemble a component. The distance traveled by the placement head or the placement time is dependent on the position of the next component to be placed (*i.e.*, the component sequencing problem) together with which feeder stores the next component to be picked up (*i.e.*, the feeder arrangement problem). So, to optimize the performance of the PAP machine, both problems should be considered and solved simultaneously.

This chapter is organized as follows. First of all, Section 3.2 presents a comprehensive review of how previous researchers attempted to solve the component sequencing and the feeder arrangement problems for the PAP machine. Section 3.3 describes the operating sequence of the PAP machine. Section 3.4 summarizes all notation adopted in the mathematical models presented in this chapter. Section 3.5 presents both individual and integrated models for the problems and examines whether the iterative approach (*i.e.*, sequentially solving individual models) can yield the global optimal solution of the integrated approach. In addition, all integrated models are compared in terms of computing complexity as well as computational time spent to reach the global optimum. Section 3.6 develops a genetic algorithm (GA) incorporating several improved heuristics to solve the integrated problem for the PAP machine. Performance of the algorithm will be studied and compared with that of other researchers. Finally, some remarks are summarized in Section 3.7.

3.2 Literature Review

As mentioned earlier, the PAP machine performance is dependent on both the component sequencing and the feeder arrangement problems. So, certainly, both problems should be taken into consideration simultaneously. Nevertheless, many researchers studied the individual problems and/or solved the problems separately.

3.2.1 The Component Sequencing Problem

Ball and Magazine (1988) were the first researchers to study the component sequencing problem for the sequential PAP machine. The assumption was that the feeder arrangement was given. The problem was modeled as the rural postman problem. A heuristic approach was then used to solve the problem, which assured that the solution was optimal if the movement of the assembly head was rectilinear.

3.2.2 The Integrated Problem

Ji *et al.* (1992) studied the component sequencing and the feeder arrangement problems for the PAP machine. In their approach, the authors separated the pickup operation from the placement operation. They first modeled the component sequencing problem as a linear assignment problem with an assumption that the feeder arrangement was provided. Then, they formulated the feeder arrangement problem as a linear assignment problem again. They adopted the existing linear programming algorithms and the heuristic methods to solve the component sequencing model and the feeder arrangement model, respectively.

Foulds and Hamacher (1993) determined the sequence of component placements and the assignment of component types to feeders for the PAP machine to minimize the total cost of placement head travel in assembling all components on a board. The feeder arrangement problem, which was formulated as a number of one-facility location models, was solved first. Then, the component sequencing problem, which was formulated as the TSP, was solved. Because it was proved that both the individual problems are NP-hard, the authors developed a heuristic method to tackle the problems separately.

Leu *et al.* (1993) studied the component sequencing and the feeder arrangement problems simultaneously for three types of PCB assembly machines, including the insertion machine, the PAP machine, and the CS machine. The authors presented a GA to solve the problems for the three types of machines. The genetic operations adopted in the GA included the crossover, the mutation, the inversion, and the rotation.

Francis *et al.* (1994) studied the component sequencing and the feeder arrangement problems for the PAP machine. The component sequencing problem was formulated as the TSP with a special structure to minimize the total assembly time. A heuristic method called the “clock sequence” was developed to solve the problem. Once the sequence of component placements was obtained, the feeder arrangement problem was also known with reference to the placement sequence.

Kumar and Li (1995) studied the component sequencing and the feeder arrangement problems for the PAP machine. The authors formulated the problems

as a quadratic programming model. However, it was unable to find the first component and therefore could not calculate the distance between the starting point and the first component. Besides, the authors did not solve it because they found that the model was computationally intractable. Therefore, they solved the problems separately. First, the component sequencing problem was referred to as the TSP. The nearest neighbor, the nearest insertion, the furthest insertion, and random generation were used to generate an initial placement sequence. Then, the 2-opt, the 3-opt, and the or-opt heuristics were used to improve the placement sequence. Second, the feeder arrangement problem was referred to as a minimum weight matching problem. They used commercial software to obtain an optimal solution for the problem.

Broad *et al.* (1996) agreed that the component sequencing and the feeder arrangement problems for the PAP machine should be simultaneously solved as they are interrelated. The authors therefore formulated the integrated problem as an integer linear programming model. However, the solution generated might be infeasible because the subtour elimination constraint was not included in the model. Besides, the sequence of placement head movements was still unknown after a solution had been generated. This could be solved provided that the starting point and the finishing point were given.

Egbelu *et al.* (1996) studied the component sequencing and the feeder arrangement problems for the PAP machine to reduce assembly cycle time. Four different robotic assembly cell designs were investigated, and several heuristic procedures were presented to obtain solutions of the designs or the models. In the heuristic procedures, the feeder arrangement problem modeled as the QAP was solved first using the cutting plane algorithm and the exchange algorithm. Then, the component sequencing problem was formulated as the TSP. The furthest insertion algorithm and the 3-opt algorithm were adopted to solve the problem.

Magyar *et al.* (1999) developed several local search heuristics to solve the component sequencing and the feeder arrangement problems separately for the PAP machine. The authors first determined the feeder arrangement and then the sequence of component placements to maximize the throughput of the machine.

Ong and Khoo (1999) developed a GA to determine the sequence of component placements and the arrangement of component types to feeders simultaneously for the PAP machine. The objective of the approach was to minimize the total travel distance of the placement head. The genetic operations adopted in the GA were the crossover, the mutation, and the inversion. In their approach, components of the same type could be stored in more than one feeder.

Deo *et al.* (2002) also studied the component sequencing and the feeder arrangement problems for the PAP machine. Although an integer linear programming model was formulated for the integrated problem, the model had two drawbacks. The distance between the starting point and the first component was not included, and also the solution generated might be infeasible due to the occurrence of a subtour. Instead of verifying the model, the authors applied a GA to solve the problems simultaneously. The genetic operations adopted in the GA were the crossover and the mutation.

3.3 Operating Sequence

The sequential PAP machine can achieve high accuracy and is suitable for operating with large components such as ICs. The Fuji XP-241E machine belongs to the class of PAP machines. In this type of placement machine, an image camera is installed on the placement head. The head can therefore move directly from the pickup points (*i.e.*, the stationary feeders) to the placement points (*i.e.*, the position of components on the PCB) without a stop for part image acquisition.

The operating sequence of the PAP machine is described as follows. At the beginning, the placement head starts from its original location or starting point, moves to a feeder that carries components, and picks up a component from the feeder. Then, it moves to the desired placement location on the stationary board, and places it there. After that, the head moves back to the previous feeder, if the next component is the same type as the previous one, or moves to another feeder, if it is different from the previous one, to pick up the next component and repeats the operating procedure. After completing all component placements on a board, the head returns to its original location, and waits for the next board to be assembled, as shown in Figure 3.1 for 10 components.

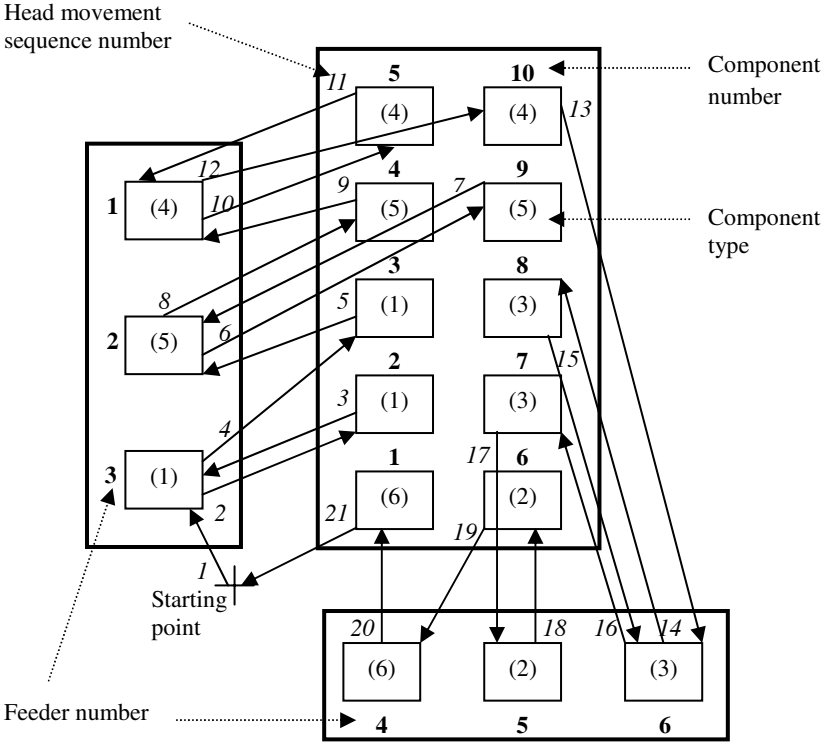

Figure 3.1. The assembly sequence of the placement head

Consider a board with 10 components of six types that requires assembly using the PAP machine, as illustrated in Figure 3.1. The number inside the bracket represents the component type. For example, component 1 or c_1 is of type 6. Furthermore, each of the component types is assigned to a feeder. For instance, component type 4 is stored in feeder 1 or f_1. If the sequence of placements starts with component 2, then components 3, 9, 4, 5, 10, 8, 7, 6, and finally component 1, then the entire assembly sequence of the placement head will be starting point $\rightarrow f_3 \rightarrow c_2 \rightarrow f_3 \rightarrow c_3 \rightarrow f_2 \rightarrow c_9 \rightarrow f_2 \rightarrow c_4 \rightarrow f_1 \rightarrow c_5 \rightarrow f_1 \rightarrow c_{10} \rightarrow f_6 \rightarrow c_8 \rightarrow f_6 \rightarrow c_7 \rightarrow f_5 \rightarrow c_6 \rightarrow f_4 \rightarrow c_1 \rightarrow$ starting point.

3.4 Notation

Although the operating sequence of the PAP machine is easy to describe verbally, it is hard and complex to find the shortest distance traveled by the placement head for assembling all components on the PCB. The reason is that the distance is dependent on the position of the next component to be placed, that is, the component sequencing problem together with which feeder stores the next component to be picked up, that is, the feeder arrangement problem. So, the problems of the component sequencing as well as the feeder arrangement should be studied and solved simultaneously to optimize the performance of the PAP machine. To achieve this goal, mathematical modeling must be applied. Because there are many variables adopted in the mathematical models presented in the following section, the interpretation of the notation is summarized here.

Consider a PCB to be assembled by a PAP machine. The PCB has n components with μ different types. Each of the component types must be stored in a feeder. But a feeder can store only a unique type of component. Because a component type must be assigned to a feeder, μ feeders are needed to store μ types of components. The objective of the integrated problem for the PAP machine is to minimize the total travel distance of the placement head, which includes the distance from the starting point to a feeder at the beginning (*i.e.*, d_{0l}), the distances from a feeder to a component's position on the PCB (*i.e.*, d_{lj}), the distances from a component's position to a feeder (*i.e.*, d_{il}), and the distance from the last component's position to the starting point (*i.e.*, d_{i0}). Note that the starting point can be referred to as component 0 (*i.e.*, $i, j = 0$). The notation used in both individual and integrated models has been summarized in Table 3.1.

Table 3.1. Notation

Indexes:

i, j: components ($i, j = 0, 1, \ldots, n$).

t: component types ($t = 1, 2, \ldots, \mu$).

l: feeders ($l = 1, 2, \ldots, \mu$).

p: placement order or placement position ($p = 1, 2, \ldots, n$).

Distances:

d_{0l}: distance traveled from starting point to feeder l.

d_{lj}: distance traveled from feeder l to the position of component j on the PCB.

d_{il}: distance traveled from the position of component i to feeder l.

d_{i0}: distance traveled from the position of component i to starting point.

Subtour elimination constraint:

u_i: placement order of component i.

Decision variables:

x_{ij} = 1 if component i is placed immediately prior to component j; 0 otherwise.

x_{ip} = 1 if component i is placed in the pth position; 0 otherwise.

$y_{t_j l}$ = 1 if component j with component type t is stored in feeder l; 0 otherwise.

3.5 Mathematical Models

In the following subsections, individual component sequencing and feeder arrangement problems are constructed in advance. It is then followed by the formulation of the integrated mathematical models. To determine the best way to optimize the performance of the PAP machine, the iterative and the integrated approaches are compared. Furthermore, a computational analysis of all integrated models, including the nonlinear and linear types, is carried out.

3.5.1 A Component Sequencing Model

Suppose that the assignment of component types to feeders (*i.e.*, the feeder arrangement problem) is solved beforehand. Then, the component sequencing model can be formulated to find the minimal distance traveled by the placement head for assembling all components on the PCB. To achieve this goal, a decision variable is defined as

$$x_{ij} = \begin{cases} 1 & \text{if component } i \text{ is placed immediately prior to component } j, \\ 0 & \text{otherwise} \end{cases}$$

Actually, the component sequencing problem is somewhat similar to the traveling salesman problem (TSP) except for the objective function. For the TSP, the objective is simply to minimize $\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} c_{ij}x_{ij}$, where c_{ij} is the distance between cities i and j. For the PAP machine, the objective is not to minimize the distance between components i and j because the placement head is unable to place the next component on the PCB immediately without picking up a component from a feeder first. Therefore, the objective for the PAP machine should be to minimize the summation of different distances, including

- The distance between the position of component i on the PCB and feeder l (if $i = 0$, it is the distance between the starting point at the beginning and feeder l);
- The distance between feeder l and the position of the next component j;
- The distance between the position of the last component i and the starting point at the end.

For example, if the sequence of component placements starts with component 2 and finishes with component 1, as shown in Figure 3.1, then both decision variables x_{02} and x_{10} are equal to 1. As mentioned before, it is assumed that the feeder arrangement problem is solved beforehand. If the type of component 2 is stored in feeder 3, then the placement head travels from the starting point to feeder 3 initially to pick up a component, and then moves from feeder 3 to the position of component 2 to place the component. So, the distances for assembling component 2 include the distance from the starting point to feeder 3 (*i.e.*, $d_{il} = d_{03}$) and the distance from feeder 3 to the position of component 2 (*i.e.*, $d_{lj} = d_{32}$). The idea of calculating the distances for assembling the remaining $(n - 1)$ components is the same. Besides, the distance for the placement head to return from the position of the last component to the starting point should be included (*i.e.*, $d_{i0} = d_{10}$). The mathematical model for the component sequencing problem can be formulated as

$$\text{Minimize } z = \sum_{i=0}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n}\sum_{l=1}^{\mu}\left(d_{il}+d_{lj}\right)x_{ij} + \sum_{i=1}^{n} d_{i0}\,x_{i0} \tag{3.1}$$

subject to

$$\sum_{i=0}^{n} x_{ij} = 1 \qquad \text{for } j = 0, 1, \ldots, n;\ i \neq j \tag{3.2}$$

$$\sum_{j=0}^{n} x_{ij} = 1 \qquad \text{for } i = 0, 1, \ldots, n;\ i \neq j \tag{3.3}$$

$$u_i - u_j + nx_{ij} \leq n-1 \qquad \text{for } i, j = 1, 2, \ldots, n;\ i \neq j \tag{3.4}$$

All $x_{ij} = 0$ or 1. All $u_i \geq 0$ and is a set of integers. (M3-1)

In M3-1, the objective function (3.1) is to minimize the total travel distance of the placement head. If the moving speed of the placement head is incorporated, then the objective can be to minimize the total placement time for assembling all components on the PCB. Constraint set (3.2) ensures that exactly one component must be placed immediately before component *j*. Constraint set (3.3) ensures that exactly one component must be placed immediately after component *i*. Although the solution drawn satisfies both constraint sets (3.2) and (3.3), it may still be infeasible due to the occurrence of subtours. Therefore, constraint set (3.4) is added to eliminate subtours. Because the starting point must be visited first, it is redundant to include *i* and/or *j* = 0 in constraint set (3.4). This is very similar to the classic TSP except that the placement head has to pick up a component from a feeder before placing the component to its position.

According to Section 3.2, it was noticed that researchers formulated the component sequencing problem for the PAP machine as the TSP. The major advantage is that the computational effort is less because the mathematical model contains only one set of decision variables (*i.e.*, x_{ij}). However, an assumption must be made for this approach. The assumption is that the feeder arrangement is predetermined. Because both problems are interrelated and dependent, they should be solved simultaneously rather than separately.

3.5.2 A Feeder Arrangement Model

If the component placement sequence is known, that is, x_{ij} in M3-1 is known, we need to arrange a component type to a feeder, and this is the second problem to be studied, called the feeder arrangement problem. It is to assign the component types to feeders so that the total distance traveled by the placement head is minimized. To achieve this goal, a decision variable is defined as

$$y_{t_j l} = \begin{cases} 1 & \text{if component type } t \text{ of component } j \text{ is stored in feeder } l, \\ 0 & \text{otherwise} \end{cases}$$

As explained earlier, the number of component types is equivalent to that of feeders. Therefore, the mathematical model for the feeder arrangement problem is somewhat similar to the quadratic assignment problem (QAP) except the objective function. Similar to that in the component sequencing model, the objective is to minimize the total distance traveled by the placement head. Using the same example in Figure 3.1, component 2 is of type 1 (*i.e.*, $t_j = t_2 = 1$), whereas component type 1 is stored in feeder 3. So, the decision variable y_{13} is 1. Besides, it is assumed that the sequence of component placements (*i.e.*, x_{ij}) is predetermined. Suppose that both decision variables x_{02} and x_{10} are equal to 1, which means that component 2 and component 1 are placed first and last, respectively. In this situation, the placement head starts traveling from the starting point to feeder 3 to pick up a component of type 1 and then travels from feeder 3 to the position of component 2 to place the component. Therefore, the distances traveled are the summation of d_{03} and d_{32}. Because the placement head must return to its starting

point after assembling all components on the PCB, the distance d_{10} must be taken into consideration. The feeder arrangement model can be formulated as

$$\text{Minimize } z = \sum_{i=0}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{t=1}^{\mu} \sum_{l=1}^{\mu} \left(d_{il} + d_{lj}\right) y_{t_j l} + d_{i0} \tag{3.5}$$

subject to

$$\sum_{t=1}^{\mu} y_{tl} = 1 \qquad \text{for } l = 1, 2, \ldots, \mu \tag{3.6}$$

$$\sum_{l=1}^{\mu} y_{tl} = 1 \qquad \text{for } t = 1, 2, \ldots, \mu \tag{3.7}$$

All $y_{tl} = 0$ or 1. (M3-2)

In M3-2, the objective function (3.5) is to calculate the total distance traveled by the placement head for assembling all components. Constraint set (3.6) ensures that exactly one component type is stored in one feeder. Constraint set (3.7) ensures that exactly one feeder holds one component type.

3.5.3 Integrated Mathematical Models

Before solving M3-1, it is essential to obtain the solution of the feeder arrangement problem (*i.e.*, M3-2) first. On the other hand, M3-2 cannot be solved until the solution of the component sequencing problem (*i.e.*, M3-1) is known. There is no doubt that the component sequencing problem and the feeder arrangement problem are interrelated. Moreover, the objective function in M3-1 and M3-2 is to minimize the total distance traveled by the placement head. Note that the amount of distance traveled is dependent on the position of the next component to be placed together with which feeder stores the next component to be picked up. Therefore, to obtain the optimal solution, a model which integrates both problems should be built. Here, two integer nonlinear programming models are constructed for the integrated problem first. Each of them is then converted into an integer linear programming model.

A pure integer nonlinear programming model for the integrated problem can be formulated as

$$\text{Minimize } z = \sum_{i=0}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{t=1}^{\mu} \sum_{l=1}^{\mu} \left(d_{il} + d_{lj}\right) x_{ij}\ y_{t_j l} + \sum_{i=1}^{n} d_{i0}\ x_{i0} \tag{3.8}$$

subject to

$$\sum_{i=0}^{n} x_{ij} = 1 \qquad \text{for } j = 0, 1, \ldots, n;\ i \neq j \tag{3.9}$$

$$\sum_{j=0}^{n} x_{ij} = 1 \qquad \text{for } i = 0, 1, \ldots, n;\ i \neq j \tag{3.10}$$

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad \text{for } i, j = 1, 2, \ldots, n;\ i \neq j \tag{3.11}$$

$$\sum_{t=1}^{\mu} y_{tl} = 1 \qquad \text{for } l = 1, 2, \ldots, \mu \tag{3.12}$$

$$\sum_{l=1}^{\mu} y_{tl} = 1 \qquad \text{for } t = 1, 2, \ldots, \mu \tag{3.13}$$

All x_{ij} and $y_{tl} = 0$ or 1. All $u_i \geq 0$ and is a set of integers. (M3-3)

Because there is a nonlinear term $x_{ij}\, y_{t,l}$ in the objective function and the model contains both binary variables (*i.e.*, x_{ij} and $y_{tl} = 0$ or 1) as well as integer variables (*i.e.*, $u_i = 1, 2, \ldots, n$), M3-3 can be regarded as a pure integer nonlinear programming model. The objective function (3.8) calculates the total travel distance of the placement head, whereas the interpretation of constraint sets (3.9) to (3.13) was mentioned in M3-1 and M3-2.

Because the decision variable x_{ij} is used, the solutions generated may form subtours, if the constraint set (3.11) in M3-3 is not incorporated. For example, the decision variables for the 10-component problem are $x_{02} = x_{23} = x_{39} = x_{94} = x_{45} = x_{50} = x_{10,8} = x_{87} = x_{76} = x_{61} = x_{1,10} = 1$. In this case, two subtours are formed:

1st subtour: starting point $\rightarrow c_2 \rightarrow c_3 \rightarrow c_9 \rightarrow c_4 \rightarrow c_5 \rightarrow$ starting point;

2nd subtour: $c_{10} \rightarrow c_8 \rightarrow c_7 \rightarrow c_6 \rightarrow c_1 \rightarrow c_{10}$.

Note in the first subtour that the placement head moves back to the starting point after placing component 5 or c_5. Moreover, there is no indication which component is placed next after assembling all components in the first subtour. So, the above solution is unacceptable, and the constraint set (3.11) must be included.

Although the constraint set (3.11) can guarantee that the solution generated is feasible, it increases the complexity of the model as there are $n(n-1)$ constraints in this subtour elimination constraint. To reduce the burden of the model, it is essential to find a way to replace the bulky constraint. Here, M3-3 is remodeled or the constraint set (3.11) in M3-3 is discarded using another decision variable x_{ip} instead of x_{ij}. The interpretation of x_{ip} is that

$$x_{ip} = \begin{cases} 1 & \text{if component } i \text{ is placed in the } p\text{th position,} \\ 0 & \text{otherwise} \end{cases}$$

The idea is to assign n components to n positions, which means that there are totally n^2 decision variables in which only n variables are 1 and all others are 0. Because each component must be placed in exactly one position, no subtour will appear in this situation. Referring to Figure 3.1, x_{21} (*i.e.*, component 2 in the first position) and x_{32} (*i.e.*, component 3 in the second position) are both 1 because component 2 and component 3 are placed first and second, respectively.

A binary integer nonlinear programming model can be formulated as

$$\text{Minimize } z = \sum_{j=1}^{n}\sum_{t=1}^{\mu}\sum_{l=1}^{\mu}\left(d_{0l} + d_{lj}\right)x_{j1}\ y_{t_j l}$$

$$+ \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n}\sum_{p=1}^{n-1}\sum_{t=1}^{\mu}\sum_{l=1}^{\mu}\left(d_{il} + d_{lj}\right)x_{ip}\ x_{j,p+1}\ y_{t_j l} + \sum_{i=1}^{n} d_{i0}\ x_{in} \tag{3.14}$$

subject to

$$\sum_{i=1}^{n} x_{ip} = 1 \qquad \text{for } p = 1, 2, \ldots, n \tag{3.15}$$

$$\sum_{p=1}^{n} x_{ip} = 1 \qquad \text{for } i = 1, 2, \ldots, n \tag{3.16}$$

$$\sum_{t=1}^{\mu} y_{tl} = 1 \qquad \text{for } l = 1, 2, \ldots, \mu \tag{3.17}$$

$$\sum_{l=1}^{\mu} y_{tl} = 1 \qquad \text{for } t = 1, 2, \ldots, \mu \tag{3.18}$$

All x_{ip} and $y_{tl} = 0$ or 1. (M3-4)

Because there are nonlinear terms $x_{j1}\ y_{t_j l}$ and $x_{ip}\ x_{j,p+1}\ y_{t_j l}$ in the objective function and the model contains only binary variables (*i.e.*, x_{ip} and $y_{tl} = 0$ or 1), M3-4 can be regarded as a binary integer nonlinear programming model. The objective function (3.14) calculates the total distance traveled by the placement head. Constraint set (3.15) ensures that exactly one component is placed in one position. Constraint set (3.16) ensures that one position has exactly one component placed. Constraint set (3.17) ensures that exactly one component type is stored in one feeder. Constraint set (3.18) ensures that exactly one feeder holds one component type.

Because M3-3 and M3-4 contain only nonlinear functions in the form of products of binary variables, they can be reformulated as linear programming models by implementing the following steps:

- Introducing a binary variable w to replace the product term xy;
- Using the extra constraints: $w \le x$, $w \le y$, and $w \ge x + y - 1$ to reflect the logical condition: $w = 1$ if and only if $x = 1$ and $y = 1$.

For M3-3, the nonlinear term in the objective function can be rewritten as a linear one by introducing an extra binary variable w_{ijl} as well as three extra constraint sets. The interpretation of the decision variable w_{ijl} is

$$w_{ijl} = \begin{cases} 1 & \text{if component } j \text{ is placed just after component } i \text{ and} \\ & \text{the type of component } j \text{ is stored in feeder } l, \\ 0 & \text{otherwise} \end{cases}$$

M3-3 can be converted into a linear programming model as

$$\text{Minimize } z = \sum_{i=0}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{l=1}^{\mu} \left(d_{il} + d_{lj}\right) w_{ijl} + \sum_{i=1}^{n} d_{i0}\, x_{i0} \tag{3.19}$$

subject to

$$\sum_{i=0}^{n} x_{ij} = 1 \qquad \text{for } j = 0, 1, \ldots, n;\ i \neq j \tag{3.20}$$

$$\sum_{j=0}^{n} x_{ij} = 1 \qquad \text{for } i = 0, 1, \ldots, n;\ i \neq j \tag{3.21}$$

$$u_i - u_j + n x_{ij} \le n - 1 \qquad \text{for } i, j = 1, 2, \ldots, n;\ i \neq j \tag{3.22}$$

$$\sum_{t=1}^{\mu} y_{tl} = 1 \qquad \text{for } l = 1, 2, \ldots, \mu \tag{3.23}$$

$$\sum_{l=1}^{\mu} y_{tl} = 1 \qquad \text{for } t = 1, 2, \ldots, \mu \tag{3.24}$$

$$w_{ijl} \le x_{ij} \qquad \text{for } i = 0, 1, \ldots, n; \quad \text{for } j = 1, 2, \ldots, n; i \neq j; \quad \text{for } l = 1, 2, \ldots, \mu \tag{3.25}$$

$$w_{ijl} \le y_{t_j l} \qquad \text{for } i = 0, 1, \ldots, n; \quad \text{for } j = 1, 2, \ldots, n; i \neq j; \quad \text{for } l, t = 1, 2, \ldots, \mu \tag{3.26}$$

$$w_{ijl} \ge x_{ij} + y_{t_j l} - 1 \qquad \text{for } i = 0, 1, \ldots, n; \quad \text{for } j = 1, 2, \ldots, n; i \neq j; \quad \text{for } l, t = 1, 2, \ldots, \mu \tag{3.27}$$

All x_{ij}, y_{tl}, and $w_{ijl} = 0$ or 1. All $u_i \ge 0$ and is a set of integers. (M3-5)

Because all the polynomial expressions are converted into linear expressions in M3-5, it can be regarded as a pure integer linear programming model. In M3-5, objective function (3.19) and constraint sets (3.25) to (3.27) are the linear expression of the objective function (3.8) of M3-3. The interpretation of the constraint sets (3.20) to (3.24) in M3-5 is the same as that of the constraint sets (3.9) to (3.13) in M3-3.

Similarly, M3-4 can be reformulated to a linear programming model. In the objective function (3.14) of M3-4, the first nonlinear term (*i.e.*, $x_{j1}\ y_{t_j l}$) is in the form of products of two binary variables. Therefore, it can be rewritten as a linear term by introducing an extra binary variable w_{j1l} as well as three extra constraint sets. The interpretation of the decision variable w_{j1l} is

$$w_{j1l} = \begin{cases} 1 & \text{if component } j \text{ is placed first and} \\ & \text{the type of component } j \text{ is stored in feeder } l, \\ 0 & \text{otherwise} \end{cases}$$

However, in the objective function (3.14) of M3-4, the second nonlinear term (*i.e.*, $x_{ip}\ x_{j,p+1}\ y_{t_j l}$) is in the form of products of three binary variables. So, the steps for converting it into linear type need to be modified in this case. The major difference is that four instead of three extra constraints are introduced. Similarly, a decision variable $w_{ij(p+1)l}$ is introduced. The decision variable $w_{ij(p+1)l}$ is defined as

$$w_{ij(p+1)l} = \begin{cases} 1 & \text{if component } j \text{ is placed just after component } i \text{ and the type of} \\ & \text{component } j \text{ placed in the } (p+1)\text{th position is stored in feeder } l, \\ 0 & \text{otherwise} \end{cases}$$

M3-4 can be converted into a linear programming model as

$$\text{Minimize } z = \sum_{j=1}^{n}\sum_{t=1}^{\mu}\sum_{l=1}^{\mu}\left(d_{0l} + d_{lj}\right) w_{j1l}$$

$$+ \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n}\sum_{p=1}^{n-1}\sum_{t=1}^{\mu}\sum_{l=1}^{\mu}\left(d_{il} + d_{lj}\right) w_{ij(p+1)l} + \sum_{i=1}^{n} d_{i0}\ x_{in} \tag{3.28}$$

subject to

$$\sum_{i=1}^{n} x_{ip} = 1 \qquad \text{for } p = 1, 2, \ldots, n \tag{3.29}$$

$$\sum_{p=1}^{n} x_{ip} = 1 \qquad \text{for } i = 1, 2, \ldots, n \tag{3.30}$$

$$\sum_{t=1}^{\mu} y_{tl} = 1 \qquad \text{for } l = 1, 2, \ldots, \mu \tag{3.31}$$

$$\sum_{l=1}^{\mu} y_{tl} = 1 \qquad \text{for } t = 1, 2, \ldots, \mu \tag{3.32}$$

$$w_{j1l} \leq x_{j1} \qquad \text{for } j = 1, 2, \ldots, n; \text{ for } l = 1, 2, \ldots, \mu \tag{3.33}$$

$$w_{j1l} \leq y_{t_j l} \qquad \text{for } j = 1, 2, \ldots, n; \text{ for } l, t = 1, 2, \ldots, \mu \tag{3.34}$$

$$w_{j1l} \geq x_{j1} + y_{t_j l} - 1 \qquad \text{for } j = 1, 2, \ldots, n; \text{ for } l, t = 1, 2, \ldots, \mu \tag{3.35}$$

$$w_{ij(p+1)l} \leq x_{ip} \qquad \text{for } i, j = 1, 2, \ldots, n;\ i \neq j; \text{ for } p = 1, 2, \ldots, n-1; \text{ for } l = 1, 2, \ldots, \mu \tag{3.36}$$

$$w_{ij(p+1)l} \leq x_{j,p+1} \qquad \text{for } i, j = 1, 2, \ldots, n;\ i \neq j; \text{ for } p = 1, 2, \ldots, n-1; \text{ for } l = 1, 2, \ldots, \mu \tag{3.37}$$

$$w_{ij(p+1)l} \leq y_{t_j l} \qquad \text{for } i, j = 1, 2, \ldots, n;\ i \neq j; \text{ for } p = 1, 2, \ldots, n-1; \text{ for } l, t = 1, 2, \ldots, \mu \tag{3.38}$$

$$w_{ij(p+1)l} \geq x_{ip} + x_{j,p+1} + y_{t_j l} - 2 \qquad \text{for } i, j = 1, 2, \ldots, n;\ i \neq j;\ \text{for } p = 1, 2, \ldots, n-1;\ \text{for } l, t = 1, 2, \ldots, \mu \qquad (3.39)$$

All x_{ip}, y_{tl}, w_{j1l}, and $w_{ij(p+1)l} = 0$ or 1. (M3-6)

Because all the polynomial expressions are converted into linear ones in M3-6, it can be regarded as a binary integer linear programming model. In M3-6, constraint sets (3.33) to (3.35) and the first term in the objective function (3.28) are the linear expression of the first term in the objective function (3.14) of M3-4. Furthermore, constraint sets (3.36) to (3.39) and the second term in the objective function (3.28) are the linear expression of the second term in the objective function (3.14) of M3-4. The interpretation of the constraint sets (3.29) to (3.32) in M3-6 is the same as that of the constraint sets (3.15) to (3.18) in M3-4.

3.5.4 Iterative Approach vs. Integrated Approach

According to the literature discussed in Section 3.2, the most prevalent approach is to solve the component sequencing and the feeder arrangement problems for the PAP machine individually rather than simultaneously. For instance, the component sequencing problem is tackled in advance, followed by the feeder arrangement problem with respect to the placement sequence. One of the motivating factors for adopting this iterative approach is its simplicity as only one single problem instead of two is focused on at a time. Nevertheless, the question is, Can this approach generate the optimal solution of the original problem? To answer this critical question, an investigation on the effectiveness of the iterative approach is carried out. The results of this comparison, definitely, provide valid evidence to answer the above query, and most importantly, direct us to an appropriate way (*i.e.*, integrated or iterative?) of optimizing the machine performance.

An iterative approach, as illustrated in Figure 3.2, is going to be studied in which the component sequencing model (*i.e.*, M3-1) is solved beforehand while assuming that the feeder arrangement, generated randomly, is predetermined. After that, the feeder arrangement model (*i.e.*, M3-2) is solved based on the optimal sequence of component placements to find the minimum travel distance for assembling the four components on the PCB. The data of the four-component problem are listed in Table 3.2; the coordinates of the starting point are (0, 0). If there is an improvement in the solution's quality, the above procedure will be repeated. The final best solution obtained by the approach is compared with the global optimal solutions generated by solving any one of the integrated models (*i.e.*, M3-3 to M3-6).

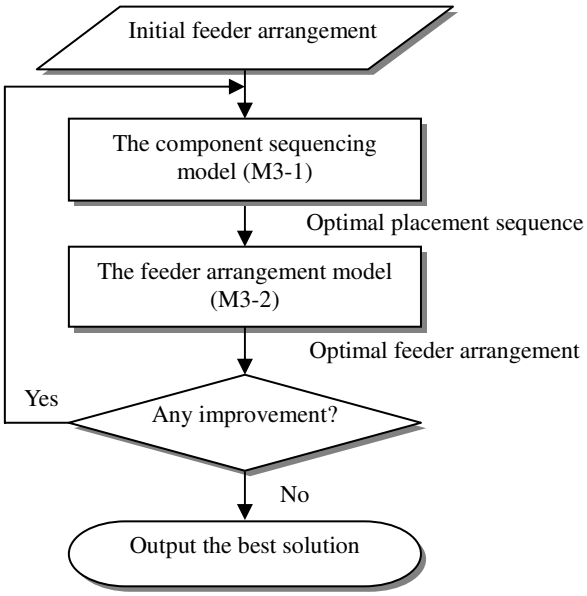

Figure 3.2. An iterative approach

Table 3.2. Data of the four-component problem

Components	**Types**	**Coordinates (mm)**		**Feeders**	**Coordinates (mm)**	
(***i***)	(***t*** $_i$)	***x***	***y***	(***l***)	***x***	***y***
0	N/A	0	0	1	10	30
1	4	30	40	2	10	20
2	3	30	60	3	20	10
3	1	50	20	4	30	10
4	2	50	40			

Suppose that the assignment of component types to feeders is generated arbitrarily so that component types 1, 3, 2, and 4 are stored in feeders 1, 2, 3, and 4, respectively (*i.e.*, $y_{11} = y_{32} = y_{23} = y_{44} = 1$). In this case, the procedure is repeated two times more (*i.e.*, totally three iterations). The solutions together with the objective value in each of the three iterations are summarized in Table 3.3.

Table 3.3. Best solutions obtained in each iteration of the iterative approach

	Model	Solutions	Objective value
1st iteration:	M3-1	$x_{10} = x_{31} = x_{42} = x_{23} = x_{04} = 1$	333.88 mm
	M3-2	$y_{31} = y_{12} = y_{23} = y_{44} = 1$	329.16 mm
2nd iteration:	M3-1	$x_{20} = x_{31} = x_{12} = x_{43} = x_{04} = 1$	327.37 mm
	M3-2	$y_{31} = y_{22} = y_{43} = y_{14} = 1$	314.24 mm
3rd iteration:	M3-1	$x_{30} = x_{01} = x_{12} = x_{43} = x_{24} = 1$	314.11 mm
	M3-2	$y_{31} = y_{22} = y_{43} = y_{14} = 1$	314.11 mm

The objective value remains the same as 314.11 mm after the third iteration, and the procedure is therefore terminated. Although the individual models are solved sequentially several times, this iterative approach cannot generate the global optimal solution of the integrated problem. The optimal assembly sequence of the placement head is as follows: starting point $\rightarrow f_3 \rightarrow c_3 \rightarrow f_4 \rightarrow c_4 \rightarrow f_2 \rightarrow c_2 \rightarrow f_1 \rightarrow c_1 \rightarrow$ starting point, whereas the total distance traveled by the placement head is 310.26 mm. Note that all four integrated models (*i.e.*, M3-3 to M3-6) generate the same solution of the four-component problem. Therefore, it is proved that the iterative approach widely adopted by many researchers cannot guarantee that the solution is globally optimal.

Besides, note that the initial feeder arrangement plays a vital role in the approach. If the initial feeder arrangement is not done carefully, even if the component sequencing problem is solved to optimality, it could result in poor performance. According to the above observations, it is concluded that the component sequencing and the feeder arrangement problems must be integrated to optimize the PAP machine performance.

3.5.5 Computational Analysis

The solutions of the four integrated models (*i.e.*, M3-3 to M3-6) are exactly the same due to their equivalent physical meanings. This gives rise to a question: Which model has the best performance? In general, a linear programming model is better than a nonlinear programming model, as we discussed in Chapter 2. Here in either linear programming or nonlinear programming, we have formulated two models for each of them. Therefore, to find the best among the four models, we can first compare the models in each class (*i.e.*, M3-3 vs. M3-4, and M3-5 vs. M3-6) in terms of computing complexity. Then, the models with less complexity in each class are compared with respect to computational time (*e.g.*, M3-4 vs. M3-5). The model regarded as the best requires less time for computation.

3.5.5.1 Computing Complexity

To examine the complexity of the models, it is essential to find the numbers of variables and constraints in each of the models. In M3-3, it can be seen that the model is very sophisticated. The objective function is nonlinear, and also its

enumeration is huge. M3-3 has $(n^2 + n + \mu^2)$ binary variables, n integer variables, and $(n^2 + n + 2\mu + 2)$ constraints. In the objective function (3.8), the terms are $n\mu + n\mu(n-1) + n$ or $n^2\mu + n$. After adopting another decision variable (*i.e.*, x_{ip}) in M3-4, the bulky subtour elimination constraint disappears. Therefore, the complexity of M3-4 is lower than that of M3-3 because both numbers of variables and constraints are reduced significantly. M3-4 has only $(n^2 + \mu^2)$ binary variables and $(2n + 2\mu)$ constraints.

For M3-5, although the model becomes linear, the numbers of variables and constraints are much greater than those in M3-3. For the number of variables, $n^2\mu$ of w_{ijl} are introduced in M3-5 besides $(n^2 + n + \mu^2)$ binary variables and n integer variables. For the number of constraints, besides $(n^2 + n + 2\mu + 2)$ constraints, M3-5 has $3n^2\mu$ constraints more for constraint sets (3.25) to (3.27). Because there are two nonlinear terms in the objective function (3.14) in M3-4, two additional decision variables and seven extra constraints are necessary to convert it into the equivalent linear programming model, M3-6. As a result, M3-6 becomes enormous and very complex. Both numbers of variables and constraints are even much greater than those in M3-5. For the number of binary variables, $n\mu$ of w_{j1l} and $n(n-1)^2\mu$ of $w_{ij(p+1)l}$ are introduced in M3-6 besides $(n^2 + \mu^2)$ binary variables. For the number of constraints, M3-6 has $3n\mu + 4n(n-1)^2\mu$ constraints more in which there are $3n\mu$ constraints for constraint sets (3.33) to (3.35) and $4n(n-1)^2\mu$ constraints for constraint sets (3.36) to (3.39), besides $(2n + 2\mu)$ constraints. The numbers of variables and constraints in the nonlinear programming models (*i.e.*, M3-3 and M3-4) and the linear programming models (*i.e.*, M3-5 and M3-6) are listed in Table 3.4 and Table 3.5, respectively.

Table 3.4. Numbers of variables and constraints in M3-3 and M3-4

	M3-3	**M3-4**
No. of variables	$n^2 + 2n + \mu^2$	$n^2 + \mu^2$
No. of constraints	$n^2 + n + 2\mu + 2$	$2n + 2\mu$

Table 3.5. Numbers of variables and constraints in M3-5 and M3-6

	M3-5	**M3-6**
No. of variables	$(n^2 + 2n + \mu^2) + n^2\mu$	$(n^2 + \mu^2) + n\mu + n(n-1)^2\mu$
No. of constraints	$(n^2 + n + 2\mu + 2) + 3n^2\mu$	$(2n + 2\mu) + 3n\mu + 4n(n-1)^2\mu$

For a realistically sized problem of 100 components and 10 component types, M3-3 has 10,300 variables and 10,122 constraints. Comparatively, M3-4 is a better nonlinear programming formulation because it consists of 10,100 variables together with 220 constraints. For the linear programming formulation, M3-5 is much more desirable than M3-6. M3-5 has 110,300 variables and 310,122 constraints. However, both numbers of variables and constraints in M3-6 are much

greater. It has 9,812,100 variables as well as 39,207,220 constraints! So, the model may not be solved to optimality in a reasonable time.

3.5.5.2 Computational Time

Because M3-4 and M3-5 are better nonlinear and linear formulations in terms of complexity, respectively, these two models are solved to global optimality using BARON and CPLEX. By these two commercial packages, the models are tested by several small examples, and both have the same solutions of the same examples. According to Table 3.6, it is found that the pure integer linear programming model (*i.e.*, M3-5) is more desirable than the binary integer nonlinear programming model (*i.e.*, M3-4) in terms of the amount of computational time spent. For instance, it only requires 10½ hours by CPLEX to solve M3-5 to optimality with eight components and eight types. But it takes more than 15 days by BARON to solve M3-4 to optimality with the same problem size. Therefore, M3-5 is the best mathematical model for the integrated problem for the PAP machine.

Although both models can obtain the global optimum, both of them are not efficient approaches because the computational time grows exponentially with problem size, as seen in Table 3.6.

Table 3.6. Computational time spent in solving M3-4 and M3-5

Numbers of components and types	Optimal solution CPU time (hh:mm:ss) by BARON for M3-4	Optimal solution CPU time (hh:mm:ss) by CPLEX for M3-5
4×4	0.31 second	0.16 second
5×5	00:00:04	00:00:01
6×6	00:01:52	00:00:41
7×7	01:21:39	00:02:26
8×8	379:02:50	10:30:51

3.6 Genetic Algorithms

Various types of mathematical models have been formulated for the integrated problem. These include the pure integer nonlinear programming model, the binary integer nonlinear programming model, the pure integer linear programming model, and the binary integer linear programming model. Because both the integer nonlinear and the integer linear programming models are among the class of theoretically difficult problems (NP-complete) (Kallrath, 1999), the integrated problem is therefore NP-complete. Solving it can be a challenging task because it requires an extremely long time to find the global optimum, as shown in Section 3.5.5.2. To solve the models efficiently, it is necessary to develop a heuristic method.

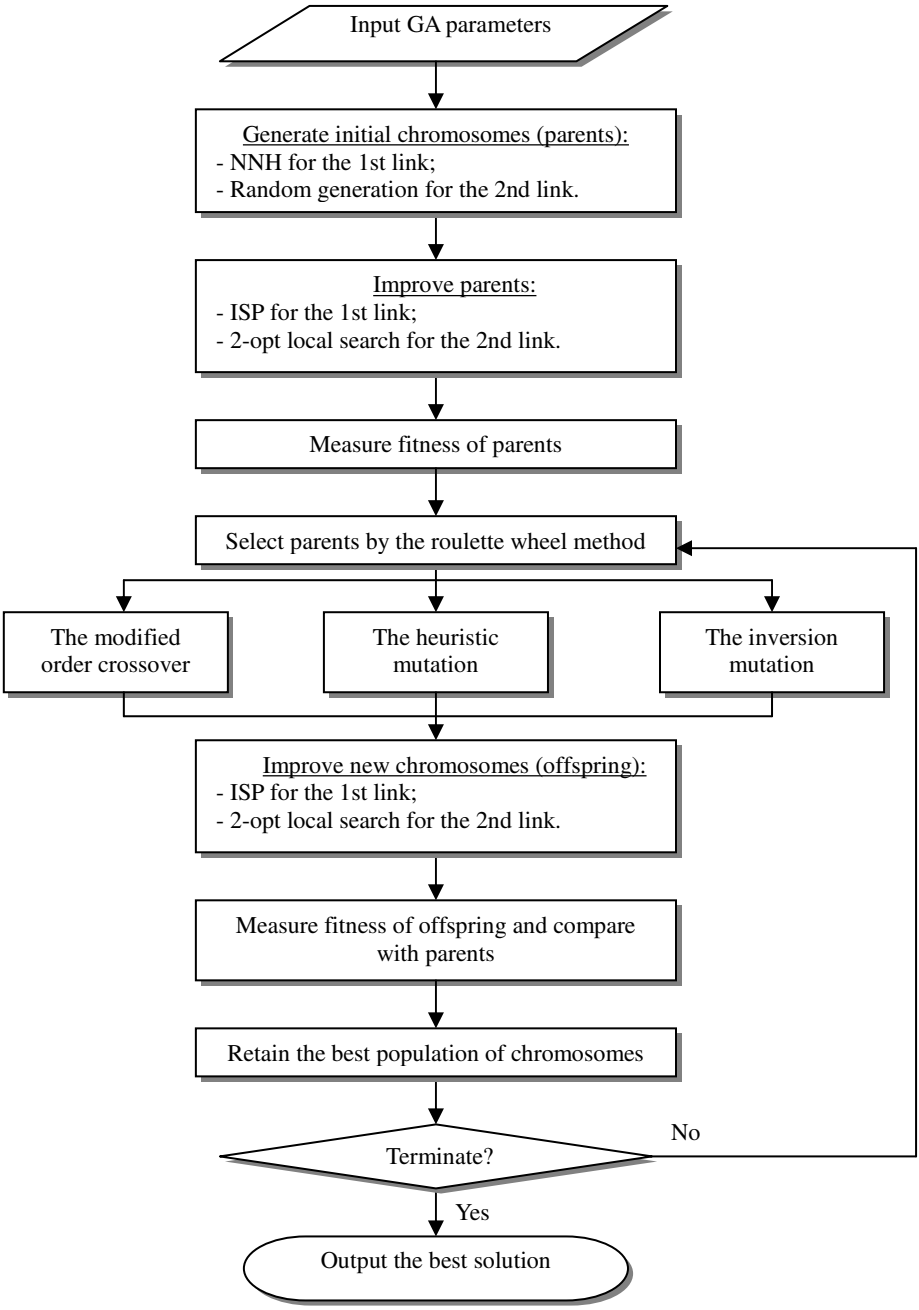

Figure 3.3. The flowchart of the HGA

Moreover, a particular mathematical model is limited to finding the solution for a single type of placement machine. The integrated models for the PAP machine are inappropriate for the CS machine. To eliminate this limitation, it is desirable to develop a flexible method, which is capable of solving the integrated problem for different types of placement machines.

To fulfil the above criteria, a GA is developed. Ideas of the general GAs are described thoroughly in Goldberg (1989). One of the advantages of GAs is their great level of flexibility (Davis, 1991; Mitchell, 1996). They can be hybridized with other heuristics to improve the solution further. Because several improved heuristics are hybridized with the GA, the algorithm developed is called the hybrid GA (HGA).

The idea of the HGA, as shown in Figure 3.3, for the integrated problem is described as follows. After the parameters such as the iteration number, the population size, the crossover rate, and the mutation rate, have been set, the HGA generates the initial chromosomes (*i.e.*, solutions) for the integrated problem. Each chromosome contains two links. The first link representing the sequence of component placements is generated using the nearest neighbor heuristic (NNH). The second link indicating the feeder arrangement is generated randomly. After that, a new improved heuristic, called the iterated swap procedure (ISP), is performed on the first link, and the 2-opt local search heuristic is applied to the second link. Each chromosome is then measured by an evaluation function. The roulette wheel selection operation is adopted to select some chromosomes for the genetic operations, including the modified order crossover, the heuristic mutation, and the inversion mutation. After an offspring is produced, the first link is improved by the ISP, and the second link is improved by the 2-opt local search heuristic. The fitness of the offspring will be measured, and it may become a member of the population if it possesses relatively good quality. These steps form an iteration, and then the roulette wheel selection is performed again to start the next iteration. The HGA will not stop unless the predetermined number of iterations is conducted. The detailed algorithm is discussed in the following subsections.

The procedure of the HGA for the integrated problem is listed as follows:

Step 1: Set the GA parameters, including the population size (*psize*), the number of iterations (*itno*), the crossover rate (*cr*), and the mutation rate (*mr*).

Step 2: Generate *psize* initial chromosomes with two-link encoding discussed in Section 3.6.1. For each chromosome, the first link is generated by the nearest neighbor heuristic (NNH) as shown in Section 3.6.2.1, and the second link is generated randomly.

Step 3: The 2-opt local search heuristic presented in Section 3.6.2.2 is performed on the second link for each initial chromosome.

Step 4: The iterated swap procedure (ISP) addressed in Section 3.6.2.3 is performed on the first link for each initial chromosome.

Step 5: Evaluate the fitness value $eval(X_h)$ for all chromosomes in the population, as illustrated in Section 3.6.3 for the PAP machine.

Step 6: Follow the selection procedure in Section 3.6.4 to select chromosomes to perform the modified order crossover operation in Section 3.6.5.1.

Step 7: Follow the selection procedure to select chromosomes to perform the heuristic mutation operation in Section 3.6.5.2.
Step 8: Follow the selection procedure to select chromosomes to perform the inversion mutation operation in Section 3.6.5.3.
Step 9: The 2-opt local search heuristic is performed on the second link for each offspring generated in Steps 6, 7, and 8.
Step 10: The ISP is performed on the first link for each offspring generated in Steps 6, 7, and 8.
Step 11: Compare all offspring with the chromosomes in the population by the fitness values $eval(X_h)$. Retain the best *psize* chromosomes in the population.
Step 12: Determine the best chromosome at each iteration. Repeat Step 6 to Step 12 until *itno* iterations are performed.

3.6.1 Encoding

The first decision needs to be made when implementing GAs is which representation of the chromosomes (*i.e.*, encoding) is designed. Generally, the binary representation is adopted in which the building blocks or the genes are 0 or 1. On the other hand, the matrix representation can be adopted if the solutions of the problems are in table form such as the transportation model (*i.e.*, M2-1). In this book, the path representation is selected to encode the solutions of the integrated problem for both types of placement machines.

For the component sequencing problem, the idea of the path representation is that the components are listed in the order in which they are placed. Consider a PCB with 10 components of six types. If the sequence of placements starts with component 1, then components 2, 3, 4, 5, 10, 9, 8, 7, and finally component 6, its sequence can be represented as (1 2 3 4 5 10 9 8 7 6). For the feeder arrangement problem, the idea of the path representation is that the component types are listed in the location in which they are assigned. If the feeder arrangement is represented by (6 5 1 3 4 2), it means that the component type 6 is stored in feeder 1, the component type 5 is stored in feeder 2, and so on.

Because the component sequencing and the feeder arrangement problems are considered simultaneously, a chromosome should include both path representations. A chromosome with a two-link representation is illustrated in Figure 3.4, in which Link 1, or the first link, represents the sequence of component placements, whereas Link 2, or the second link, represents the assignment of component types to feeders.

Assembly sequence	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Component number	1	2	3	4	5	10	9	8	7	6

Link 1

Component type	6	5	1	3	4	2
Feeder	**1**	**2**	**3**	**4**	**5**	**6**

Link 2

Figure 3.4. The two-link representation for a chromosome

3.6.2 Improved Heuristics

Solving the component sequencing and the feeder arrangement problems concurrently is extremely sophisticated as the integrated problem is akin to the combination of the TSP and the QAP. In addition, some researchers discovered that a simple GA was not desirable for solving combinatorial optimization problems with a large problem size (Freisleben and Merz, 1996a,b). So, it is necessary to develop an improved heuristic. In the HGA, three types of heuristics are adopted to improve the solution, including the nearest neighbor heuristic (NNH), the 2-opt local search heuristic, and the iterated swap procedure (ISP).

3.6.2.1 Nearest Neighbor Heuristic

The NNH is used to generate an initial solution only for the first link, which is the sequence of component placements. The principle of the NNH is to start with the first component randomly, then to select the next component as close as possible to the previous one from those unselected components to form the placement sequence until all components are selected.

3.6.2.2 2-Opt Local Search Heuristic

Compared with the total number of components on a PCB, the number of component types is much fewer. Therefore, it is desirable to perform the 2-opt local search heuristic only for the second link, that is, the feeder arrangement. The principle of this heuristic is very straightforward. For one parent, all possible two swaps are examined to generate offspring and the best offspring will replace the parent if it has higher quality. The process will not stop until there is no further improvement in the quality of the solution.

3.6.2.3 Iterated Swap Procedure

The computational effort will be high if the 2-opt local search is performed for the first link, which is the sequence of component placements, because the number of components is quite large, normally several hundreds. As a consequence, a “fast” improved heuristic is developed, which is called the iterated swap procedure (ISP). The ISP, as illustrated in Figure 3.5, is performed for the first link of each initial solution generated by the NNH as well as each offspring generated by the three genetic operators. The procedure of the ISP is as follows:

Step 1: Select two genes randomly from the first link of a parent.
Step 2: Exchange the positions of the two genes to form an offspring.
Step 3: Swap the neighbors of the two genes to form four more offspring.
Step 4: Evaluate all offspring and find the best one.
Step 5: If the best offspring is better than the parent, replace the parent with the best offspring and go back to Step 1; otherwise, stop.

			Select two genes randomly							
Parent:	1	2	3	4	5	6	7	8	9	10
Offspring 1:	1	2	**8**	4	5	6	7	**3**	9	10
Offspring 2:	1	**8**	**2**	4	5	6	7	3	9	10
Offspring 3:	1	2	**4**	**8**	5	6	7	3	9	10
Offspring 4:	1	2	8	4	5	6	**3**	**7**	9	10
Offspring 5:	1	2	8	4	5	6	7	**9**	**3**	10

Figure 3.5. The iterated swap procedure

3.6.3 Evaluation

The objective function of an optimization problem can be used for evaluation. Its goal is to find the objective value or fitness value of a solution of the problem. Without evaluation, it is impossible to compare and then find the best solution. Therefore, evaluation is often regarded as the most important element of GAs. The fitness function or evaluation for the PAP machine is described in the following.

For the PAP machine, the fitness function used is the total travel distance of the placement head. This function calculates the distance from the starting point to a feeder at the beginning, the distance from a feeder to a component's position, the distance from a component's position to a feeder, and the distance from the last component's position to the starting point at the end. Let $eval(X_h)$ be the fitness function for chromosome X_h in the integrated problem, and let $D(a, b)$ be the distance from point a to point b; then,

$$eval(X_h) = D[o, f(1)] + \sum_{i=1}^{n} D[f(i), c_i] + \sum_{i=1}^{n-1} D[c_i, f(i+1)] + D(c_n, o)$$

where

n is the number of components,

$f(i)$ is the feeder location for the ith component,

c_i is the location in the board for the ith component, and

o is the starting point.

Here, $f(i)$ represents the feeder location for the ith component. For example, component 3 is stored in feeder 1, so $f(3) = 1$, that is, f_1.

3.6.4 Selection

The roulette wheel selection operation (Goldberg, 1989) is adopted to choose some chromosomes to undergo genetic operations. The approach is based on an

observation that a roulette wheel has a section allocated for each chromosome in the population, and the size of each section is proportional to the chromosome's fitness. The fitter the chromosome, the higher the probability of being selected. It is true that the roulette wheel selection mechanism chooses chromosomes probabilistically, instead of deterministically. For example, although one chromosome has the highest fitness, there is no guarantee it will be selected. The only certain thing is that, on average, a chromosome will be chosen with the probability proportional to its fitness. Suppose that the population size is *psize*; then the selection procedure is as follows:

Step 1: Calculate the total fitness of the population:

$$F = \sum_{h=1}^{psize} eval(X_h)$$

Step 2: Calculate the selection probability p_h for each chromosome X_h:

$$p_h = \frac{F - eval(X_h)}{F \times (psize - 1)}, \qquad h = 1, 2, ..., psize$$

Step 3: Calculate the cumulative probability q_h for each chromosome X_h:

$$q_h = \sum_{j=1}^{h} p_j, \qquad h = 1, 2, \ldots, psize$$

Step 4: Generate a random number r in the range (0, 1].

Step 5: If $q_{h-1} < r \le q_h$, then chromosome X_h is selected.

3.6.5 Genetic Operations

The genetic search progress is obtained by two essential genetic operations: exploitation (or intensification) and exploration (or diversification). Generally, the crossover operator exploits a better solution, whereas the mutation operator explores a wider search space. The genetic operators used in the algorithm for integrated problems are one crossover and two mutations, which are called the heuristic mutation and the inversion mutation, respectively. Two links in a chromosome are required to perform these genetic operations. The number of chromosomes selected to perform the crossover and the mutation operations depends on the crossover rate and the mutation rate, respectively, which are predetermined by the GA user. Let *crossno* and *mut* denote the numbers of chromosomes selected to undergo the crossover and the mutation, respectively; then $crossno = \text{round}(cr \times psize)$ and $mut = \text{round}(mr \times psize)$, where cr is the crossover rate, and mr is the mutation rate. Because a pair of chromosomes is required to undergo the crossover operation, the number of pairs of chromosomes, denoted as *cross*, is an integer, so

$$cross = \begin{cases} \dfrac{crossno}{2} & \text{if } crossno \text{ is even} \\ \dfrac{crossno - 1}{2} & \text{otherwise} \end{cases}$$

3.6.5.1 The Modified Order Crossover

As shown in Figures 3.6 and 3.7, the crossover operator adopted in the HGA is a modified version of the classical order crossover operator, and two offspring will be generated each time. The procedure of the modified order crossover operation is as follows:

Step 1: Select a substring from the first parent randomly.

Step 2: Produce a protochild by copying the substring into the corresponding positions in the protochild.

Step 3: Find the gene right prior to the first gene of the substring from the second parent. If the gene is one of the genes in the substring, go to Step 4. Otherwise, place it in front of the substring in the protochild.

Step 4: Find the gene right behind the last gene of the substring from the second parent. If the gene is one of the genes in the substring, go to Step 5. Otherwise, place it just after the substring in the protochild.

Step 5: Delete those genes that are already in the protochild from the second parent. The resulting genes, that is, the genes not yet in the protochild, form a sequence.

Step 6: Place the genes into the unfilled positions of the protochild from left to right according to the resulting sequence of genes in Step 5 to produce an offspring, as illustrated in Figure 3.6.

Step 7: Repeat Step 1 to Step 6 to produce the second offspring by exchanging the two parents, as shown in Figure 3.7.

				Selected substring						
Parent 1:	1	2	3	4	5	6	7	8	9	10
Parent 2:	6	8	1	9	10	4	5	2	7	3
Offspring 1:	8	1	10	**4**	**5**	**6**	**7**	3	9	2

Figure 3.6. The modified order crossover operator (Offspring 1)

				Selected substring						
Parent 2:	6	8	1	9	10	4	5	2	7	3
Parent 1:	1	2	3	4	5	6	7	8	9	10
Offspring 2:	1	2	8	**9**	**10**	**4**	**5**	6	3	7

Figure 3.7. The modified order crossover operator (Offspring 2)

3.6.5.2 The Heuristic Mutation

A heuristic mutation (Gen and Cheng, 1997) is designed with the neighborhood technique to produce a better offspring. A set of chromosomes transformed from a parent by exchanging some genes is regarded as the neighborhood. Only the best one in the neighborhood is used as the offspring produced by the mutation. However, the purpose of the mutation operation is to promote diversity of the

population. Therefore, it is necessary to change the original heuristic mutation for the integrated problem. The modification, as illustrated in Figure 3.8, is that all neighbors generated are used as offspring. The procedure of the heuristic mutation operation is as follows:

Step 1: Pick up three genes in a parent at random.

Step 2: Generate neighbors for all possible permutations of the selected genes, and all neighbors generated are regarded as offspring.

			Select three genes at random							
Parent:	1	2	3	4	5	6	7	8	9	10
Offspring 1:	1	2	**3**	4	5	**8**	7	**6**	9	10
Offspring 2:	1	2	**6**	4	5	**3**	7	**8**	9	10
Offspring 3:	1	2	**6**	4	5	**8**	7	**3**	9	10
Offspring 4:	1	2	**8**	4	5	**3**	7	**6**	9	10
Offspring 5:	1	2	**8**	4	5	**6**	7	**3**	9	10

Figure 3.8. The heuristic mutation operator

3.6.5.3 The Inversion Mutation

The inversion operator, shown in Figure 3.9, selects a substring from a parent and flips it to form an offspring. However, the inversion operator operates with only one chromosome, so it is similar to the heuristic mutation and thus lacks the interchange of characteristics between chromosomes. So, the inversion operator is a mutation operation, which is used to increase the diversity of the population rather than to enhance the quality of the population.

				Selected substring						
Parent:	1	2	3	4	5	6	7	8	9	10
Offspring:	1	2	3	**7**	**6**	**5**	**4**	8	9	10

Figure 3.9. The inversion mutation operator

3.6.6 Performance Analysis

The performance of the HGA is evaluated using the PCB example in Leu *et al.* (1993). The example has 200 components with 10 different component types. The parameters of the HGA for the integrated problem for the PAP machine are preset as *psize* = 25, *itno* = 3000, *cr* = 0.4, and *mr* = 0.2. Therefore, five pairs of chromosomes are selected to perform the modified order crossover operation, whereas five chromosomes perform the heuristic mutation and the inversion operation. The total number of offspring produced per iteration will be 40, 10 from

the modified order crossover operation, 25 from the heuristic mutation operation, and 5 from the inversion mutation operation.

3.6.6.1 Comparison to Other Approaches

The performance of the HGA is shown in Figure 3.10, whereas the comparison between the results obtained from the HGA and those obtained from other researchers (Leu *et al.*, 1993; Ong and Khoo, 1999) is shown in Table 3.7. It is found that the performance of the HGA is superior. First, the best chromosome (6,275.4 cm) in the initial population obtained by the HGA is better than those of the simple GAs (both larger than 6,900 cm). Second, the HGA can obtain a better solution with a smaller population size, only 25, whereas the other two used 100. Third, the HGA can obtain a better solution with a smaller population size and also with fewer iterations, 3,000 vs. 6,150. Finally and the most important, the HGA obtained a better solution than any previous methods, 5,660.5 cm vs. 5,673.7 cm or 6,129 cm.

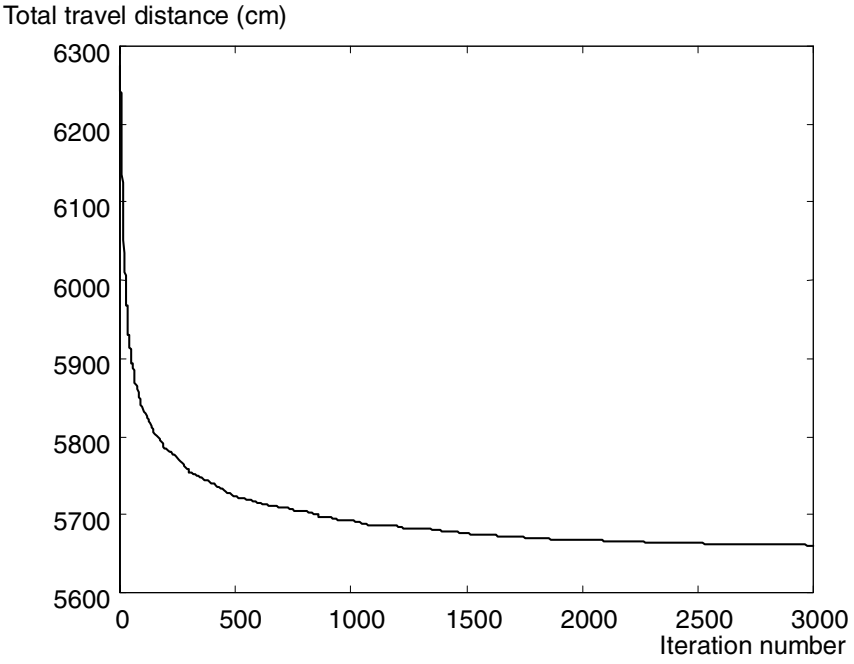

Figure 3.10. The minimum travel distance at each iteration

Table 3.7. A comparison of the experimental results for the PAP machine

	Leu *et al.* (1993)	**Ong and Khoo (1999)**	**HGA**
Best one in the initial solution (cm)	About 6,950	6,952.2	6,275.4
Population size	100	100	25
Iteration number	6,150	6,150	3,000
Final best solution (cm)	About 6,129	5,673.7	5,660.5

3.6.6.2 Effect of Population Size

Population size is the number of chromosomes in the population. A chromosome represents a point in the search space. Therefore, a larger population size means more search points. The more the search points, the higher the chance of finding the optimal solution. However, a larger population size needs longer computational time and more computer storage space. To identify the effect of population size, the HGA program was run with the above 200-component problem using three different population sizes: 5, 25, and 50. The effect of population size on the 200-component problem is shown in Figure 3.11.

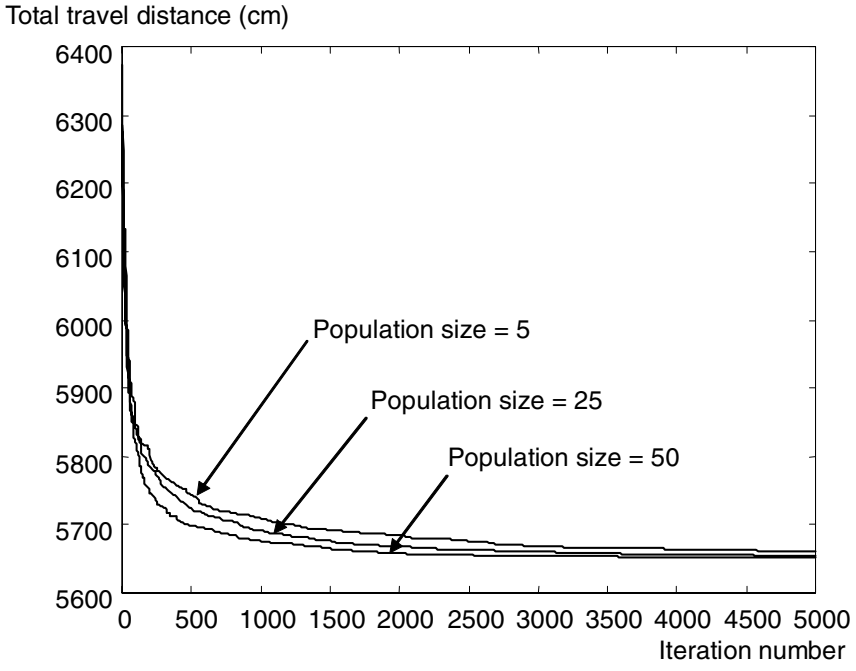

Figure 3.11. The effect of population size for the PAP machine

It can be seen from Figure 3.11 that the curve representing the population size of 50 is lower than those representing the population sizes of 5 and 25. This phenomenon shows that the HGA with a large population size can obtain a better final solution. This may be due to the fact that more offspring are produced at each iteration. The effect of different population sizes on the 200-component problem is summarized in Table 3.8.

Table 3.8. A comparison of different population sizes for the PAP machine

Population size	5	25	50
Best one in the initial population (cm)	6,374	6,275	6,285
Final best solution (cm)	5,661	5,655	5,651
Iteration number	4,835	4,923	4,984

3.6.6.3 Comparison to Optimal Solution

In Section 3.5.5.2, several sizes of problems are solved to optimality. In this section, the performance of the HGA is examined and compared with the optimal solution. It is found that the HGA achieves the optimal solution for all problems. As shown in Table 3.9, the longest computational time spent is only nine seconds for the eight-component problem. It shows that the HGA takes much less time. However, there is no guarantee that the solution generated is globally optimal because the HGA is a heuristic method.

Table 3.9. Comparisons of the HGA with BARON and CPLEX for the PAP machine

Numbers of components and types	**Optimal solution CPU time (hh:mm:ss) by BARON for M3-4**	**Optimal solution CPU time (hh:mm:ss) by CPLEX for M3-5**	**Best solution CPU time (hh:mm:ss) by HGA**
4×4	0.31 second	0.16 second	0.15 second
5×5	00:00:04	00:00:01	0.90 second
6×6	00:01:52	00:00:41	00:00:04
7×7	01:21:39	00:02:26	00:00:06
8×8	379:02:50	10:30:51	00:00:09

3.6.6.4 Integrated Problem with Feeder Duplication

In the integrated problem, the number of feeders supplied is exactly the same as the number of component types required. But, in real-life situations, the number of feeders available may be greater, so components of the same type can be stored in more than one feeder. In this case, the feeder arrangement problem no longer belongs to the category of a 1-to-1 assignment problem. Also, besides the component sequencing and the feeder arrangement problems, it is necessary to determine the feeder from which a component should be retrieved because more

than one feeder hold the same type of component, that is, the component retrieval problem.

When a component type is assigned to two feeders, a retrieval plan must be set up to determine the feeder from which a component should be retrieved. If a component is stored in two feeders, it should be retrieved from a feeder so that the distances traveled by the placement head are minimized. For example, the component type of component j or c_j is stored in two feeders, feeder r or f_r and feeder s or f_s, whereas component i or c_i is placed just prior to c_j. It is necessary to calculate the amount of distance, D, traveled by the placement head, and then select a feeder for component retrieval that incurs a shorter travel distance as follows:

- If $D(c_i, f_r) + D(f_r, c_j) < D(c_i, f_s) + D(f_s, c_j)$, then retrieve from f_r.
- If $D(c_i, f_r) + D(f_r, c_j) > D(c_i, f_s) + D(f_s, c_j)$, then retrieve from f_s.
- In case $D(c_i, f_r) + D(f_r, c_j) = D(c_i, f_s) + D(f_s, c_j)$, select a feeder for retrieval arbitrarily.

The performance of the HGA for solving the component sequencing, the feeder arrangement, and the component retrieval problems simultaneously is evaluated by use of the 200-component problem (Leu *et al.*, 1993) again in which three surplus feeders are available. In this case, three types of components can be assigned to two feeders. In the 200-component problem, component types 2, 6, and 8 are the most frequently used. These three types of components can therefore be stored in two feeders.

The performance of the HGA for the three problems is shown in Figure 3.12, and its result is listed in Table 3.10. It is found that the solution is even better if the number of feeders available is more than that of component types required. The best solution generated by the HGA after 1,000 iterations is just 4,855.5 cm for the case with three surplus feeders. Because the three most frequently used component types are assigned to more than one feeder, they can be retrieved from a closer feeder. So, the total distance traveled by the placement head can be reduced.

Table 3.10. A comparison of HGAs for the PAP machine

	HGA	**HGA**
Number of surplus feeders	N/A	3
Population size	25	25
Iteration number	3,000	1,000
Final best solution (cm)	5,660.5	4,855.5

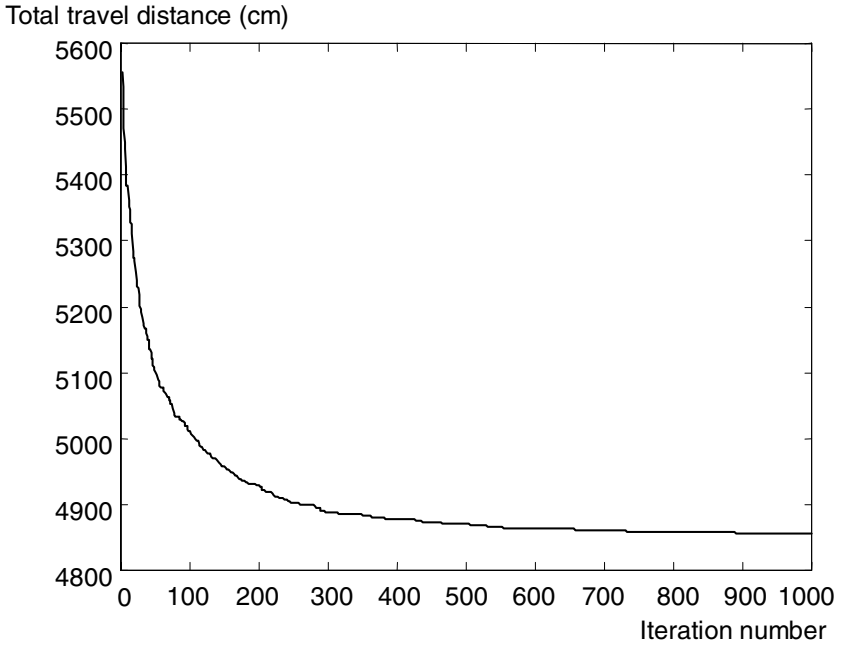

Figure 3.12. The performance of the HGA for the integrated problem with feeder duplication for the PAP machine

3.7 Summary

In this chapter, the focus is confined to the optimization of the PAP machine performance. First, various types of mathematical models have been formulated to find the global optimal solution. Second, a HGA has been developed to obtain the near-optimal solution. Some remarks are summarized in the following.

1. The component sequencing and the feeder arrangement problems are interrelated and inseparable. One cannot be solved unless the solution of the other one is obtained beforehand.
2. The iterative approach that sequentially solves the individual component sequencing and feeder arrangement models, adopted by many researchers, is not the best way to optimize the PAP machine performance. The approach cannot guarantee that the solution is globally optimal.
3. Due to their close relationship, two integer nonlinear programming models were formulated to solve the integrated problem for the PAP machine. It is feasible to convert the integer nonlinear programming models into integer linear programming models equivalently.
4. All four mathematical models were verified using commercial packages, and all of them generated the same optimal solutions of the same problems.

5. Different types of models took different times for computation. In terms of the amount of computational time spent, the linear type model is more desirable.
6. Although the optimal solution can be found using commercial packages, it was proved that the computational time grows exponentially with problem size.
7. After the genetic algorithm was combined with several improved heuristics, the algorithm gave a better solution using fewer number of iterations compared with other approaches.
8. The algorithm with a larger population size can obtain a better final solution, but, at the same time, it requires more computational time.
9. Although the algorithm cannot guarantee obtaining the optimal solution, it was proved that the HGA can reach the global optimum of several problems with small sizes quickly.
10. The solution was even better when component types could be stored in more than one feeder.

Chapter 4 will study the component sequencing problem and the feeder arrangement problem for another type of placement machine called the concurrent chip shooter (CS) machine. Similar to that in Chapter 3, mathematical models are constructed for the problems first. The computing complexity and the computational time of the integrated models are studied, and then are followed by the HGA for the integrated problem.

4

The Concurrent Chip Shooter (CS) Machine

4.1 Introduction

There are normally different sizes of components on a PCB. The sequential PAP machine, studied in the former chapter, assembles large components such as ICs, whereas the concurrent chip shooter (CS) machine, the focus in this chapter, picks up and places small components including chip resistors on the PCB. To optimize the component placement process, the performance of both types of placement machines should therefore be considered. In this chapter, mathematical modeling is adopted to optimize the CS machine performance so that the highest productivity can be achieved. In the CS machine, the assembly time is dependent on three movable mechanisms: the movement of the X-Y table or the PCB (*i.e.*, the component sequencing problem), the movement of the feeder carrier (*i.e.*, the feeder arrangement problem), and the movement of the turret. Naturally, it is more difficult than that of the PAP machine. Besides, the component sequencing problem and the feeder arrangement problem should definitely be studied and solved simultaneously for the machine. Furthermore, the hybrid genetic algorithm (HGA), as presented in Chapter 3, is modified to deal with the integrated problem for the CS machine.

The organization of this chapter is as follows: Section 4.2 presents a comprehensive review of the way previous researchers tackled the component sequencing and the feeder arrangement problems for the CS machine. Section 4.3 describes the operating sequence of the CS machine in detail with the aid of an example. Section 4.4 summarizes the interpretation of all notation used in the mathematical models. Section 4.5 presents both individual and integrated models for the problems and examines whether the iterative approach (*i.e.*, sequentially solving individual models) can yield the global optimal solution of the integrated approach. In addition, all integrated models are compared in terms of computing complexity as well as computational time spent to reach the global optimum. Section 4.6 shows the modification of the HGA developed in the previous chapter for solving the integrated problem for the CS machine. Performance of the algorithm will be studied and compared with that of other researchers. Finally, some remarks are listed in Section 4.7.

4.2 Literature Review

The CS machine is another type of SMT placement machine to be studied in this book. Unlike the configuration of the PAP machine, the CS machine consists of three movable mechanisms: a feeder carrier holding components, a rotary turret with multiple assembly heads, and an X-Y table carrying a PCB. During assembly, the three mechanisms move at the same time. So, certainly, the assembly time of the machine is dependent on these three mechanisms. Nevertheless, many researchers studied the machine performance separately. Generally, they formulated the movement of the X-Y table (*i.e.*, the component sequencing problem) as the TSP, and the movement of the feeder carrier (*i.e.*, the feeder arrangement problem) as the QAP. Then, the problems were solved individually.

4.2.1 The Component Sequencing Problem

De Souza and Wu (1994) studied the component sequencing problem only for the CS machine. A knowledge-based component placement system (CPS), incorporated with TSP algorithms, was developed to solve the problem. The objective was to minimize the total travel distance of the X-Y table. Furthermore, De Souza and Wu (1995) pointed out that the CPS is more practical and effective compared with the machine proprietary algorithm.

Moyer and Gupta (1997) agreed that the component sequencing problem for the CS machine could be formulated as the TSP provided that the locations of the components on the board were assigned prior to the determination of the placement sequence. The board sequencing heuristic (BSH) was developed to solve the problem. The idea of the BSH was to rearrange the placement order by swapping a pair of components in the current tour to obtain a better solution.

4.2.2 The Feeder Arrangement Problem

Moyer and Gupta (1996a) studied the feeder arrangement problem only for the CS machine based on the assumption that the sequence of component placements was predetermined. The problem was formulated as the QAP. Two heuristic methods were proposed to solve the problem.

Dikos *et al.* (1997) also formulated the feeder arrangement problem for the CS machine as the QAP, and made an assumption that an optimal component placement sequence was first specified. The authors employed GAs to find a near-optimal solution for the problem.

Klomp *et al.* (2000) treated the problem of determining an optimal feeder arrangement for a line of CS machines as finding the shortest Hamiltonian path. An insertion heuristic and a local search heuristic were employed to solve the problem.

4.2.3 The Integrated Problem

Bard *et al.* (1994) used an iterative two-step heuristic approach to determine the component placement sequence, the feeder arrangement, and the retrieval plan for the CS machine. Initially, a placement sequence was generated with the weighted

nearest neighbor heuristic, whereas the remaining two problems were then formulated as an integer quadratic programming model and solved with a Lagrangian relaxation scheme. In the final step, the previous feeder arrangement was used to update the placement sequence, and the entire process was repeated.

Moyer and Gupta (1996b) developed a heuristic approach to solve the component sequencing and the feeder arrangement problems separately for the CS machine. In the algorithm, the nearest neighbor heuristic was applied to generate an initial placement sequence first. The pairwise exchange method was then applied to improve the initial solution. Following that, the feeder arrangement was generated randomly. Similarly, the pairwise exchange method was applied to improve the initial feeder arrangement.

Sohn and Park (1996) studied the component sequencing and the feeder arrangement problems for the CS machine. They pointed out that it was difficult to solve the problems concurrently. Therefore, they focused on the machine with only one assembly head instead of multiple heads, and formulated the integrated problem as a mixed integer nonlinear programming model. For the machine with one head, the component sequencing problem was modeled as the TSP, whereas the feeder arrangement problem was formulated as the QAP. Then, a heuristic approach, similar to that developed by Leipälä and Nevalainen (1989), was applied to solve the problems separately.

Yeo *et al.* (1996) developed a rule-based frame system to generate the feeder arrangement first and then the component placement sequence for the CS machine. The approach was based on the one-pitch incremental feeder heuristic and the nearest neighbor heuristic.

Crama *et al.* (1997) proposed a solution procedure to tackle the component sequencing, the feeder arrangement, and the component retrieval problems for the CS machine. The authors stated that the individual problems are already very hard in terms of computational complexity, so they solved the problems individually and heuristically. The feeder arrangement problem was heuristically solved first, and then the remaining two problems were solved using constructive heuristics and local search methods.

Ellis *et al.* (2001) developed a heuristic approach to determine the component placement sequence and the feeder arrangement for the CS machine. The nearest neighbor heuristic was used to generate the initial placement sequence first, and then the QAP greedy heuristic was used to generate the initial feeder arrangement. The 2-opt local search heuristic was adopted to improve both types of initial solutions.

Ong and Tan (2002) developed a GA incorporated with different types of crossover and mutation operations to determine the sequence of component placements on a PCB and the arrangement of component types to feeders simultaneously for the CS machine. The objective of the approach was to minimize the total assembly time.

Wilhelm and Tarmy (2003) also applied a set of heuristics to tackle the integrated problem for the CS machine. Each component type was assigned to a feeder first. Then, the sequence of component placements was determined by solving an asymmetrical TSP.

4.3 Operating Sequence

The Fuji CP-732E machine belongs to the class of CS machines. It is a concurrent type because the feeder carrier, the X-Y table, and the turret move simultaneously during assembly. The turret is equipped with multiple windmill-style nozzle holders, called assembly heads, each of which can be fitted with up to six nozzles. During the pickup operation, the machines can index to the appropriate nozzle size for each component to achieve high-accuracy placement. The major advantage of the CS machine is its high speed; it can achieve a maximum placing speed of 0.068 second per shot.

As illustrated from the top to the bottom in Figure 4.1, a CS machine has a movable feeder carrier holding components, a rotary turret with multiple assembly heads (usually 10 or 12), and a movable X-Y table carrying a PCB. Each assembly head has several (normally five) nozzles of different sizes. A large nozzle is used to pick up and place large components.

The operating sequence of the CS machine is described as follows. As the first board of a batch enters the machine, the first nozzle of the turret picks up a component from a feeder. Then the turret indexes one step and the next nozzle picks up the second component. After that, the turret indexes again to pick up the next component, and so on. At the same moment, the PCB is moved to the placement location waiting for the first component to be placed on the board. When the sixth component is being picked up, if the turret has 10 heads, the first component is being placed on the board at the same time. These operations continue so that the turret indexes one step, the feeder carrier moves to the location containing the next pickup component, and the X-Y table moves to the next placement location. In the assembly of the last five components, there is no need to pick up components for the board being assembled. However, the nozzles of the turret can pick up the first five components for the next board to be assembled, if necessary. For the first few components assembled in a batch of PCBs, there are only pickup movements and no placement movement. For the last few components of the same batch, there are only placement movements and no pickup movement. Therefore, if the quantity of PCB in a batch is very large, these boundary effects can be neglected (Leu *et al.*, 1993).

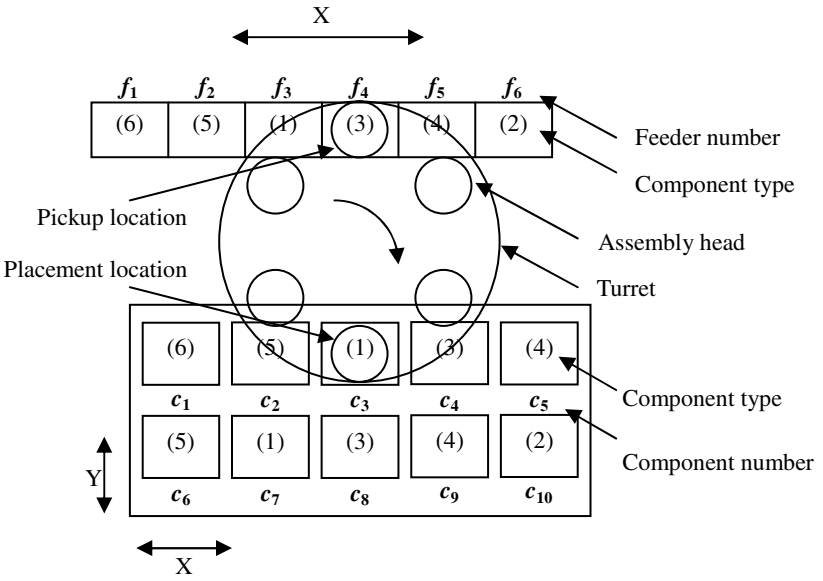

Figure 4.1. The schematic diagram of the CS machine

The operation of the CS machine is more sophisticated than that of the PAP machine, so a detailed description of its operation is provided in the following with the aid of an example. Consider a board with 10 components of six types that requires assembly using the CS machine, as illustrated in Figure 4.1. The number inside the bracket represents the component type. For instance, component 1 or c_1 is of type 6. Furthermore, each of the component types is assigned to a feeder. For instance, component type 6 is stored in feeder 1 or f_1. If the sequence of placements starts with component 1, then components 2, 3, 4, 5, 10, 9, 8, 7, and finally component 6, then the entire assembly sequence of the CS machine is as follows:

1. f_1 is moved to the pickup location (PUL).
2. The first nozzle picks up a component of type 6, as shown in Figure 4.2.

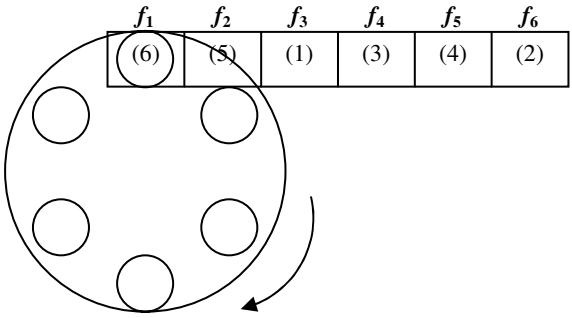

Figure 4.2. The schematic diagram of assembly sequence number 2

3. The turret rotates one step.
4. f_2 is moved to the PUL.
5. The second nozzle picks up a component of type 5, as shown in Figure 4.3.

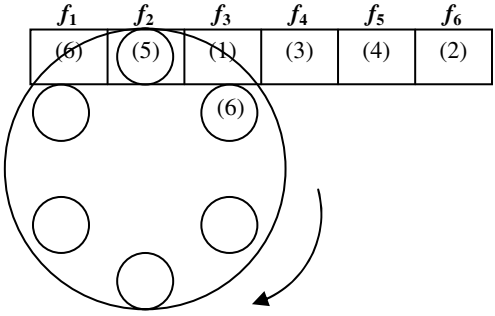

Figure 4.3. The schematic diagram of assembly sequence number 5

6. The turret rotates one step.
7. f_3 is moved to the PUL.
8. The third nozzle picks up a component of type 1, as shown in Figure 4.4.

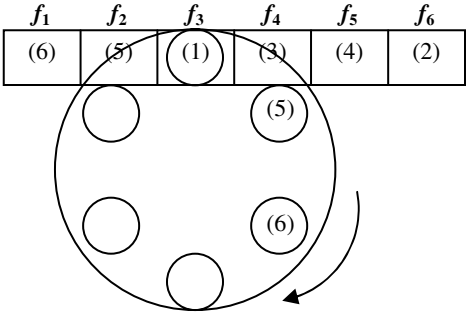

Figure 4.4. The schematic diagram of assembly sequence number 8

9. The turret rotates one step.
10. f_4 is moved to the PUL.
11. c_1 is moved to the placement location (PL).
12. When the fourth nozzle is picking up a component of type 3, c_1 is being placed, as shown in Figure 4.5.

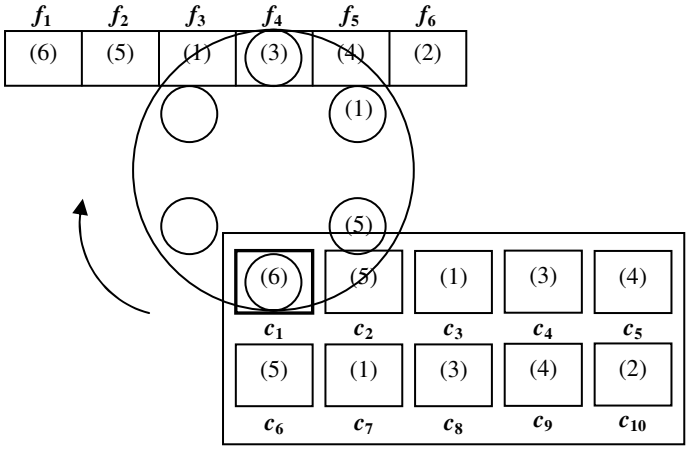

Figure 4.5. The schematic diagram of assembly sequence number 12

13. The turret rotates one step.
14. f_5 is moved to the PUL.
15. c_2 is moved to the PL.
16. When the fifth nozzle is picking up a component of type 4, c_2 is being placed, as shown in Figure 4.6.

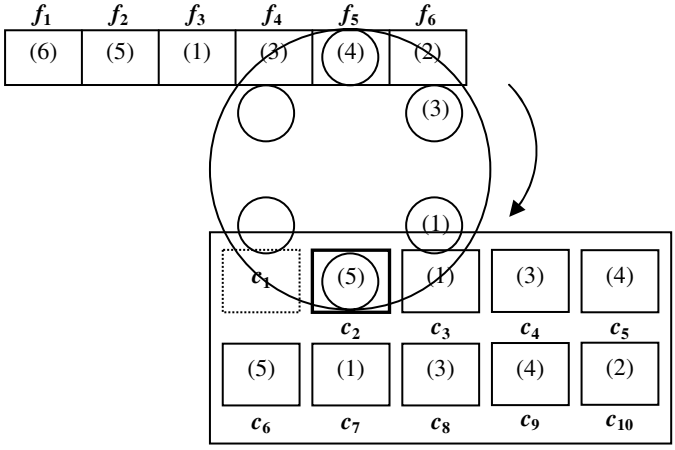

Figure 4.6. The schematic diagram of assembly sequence number 16

17. The turret rotates one step.
18. f_6 is moved to the PUL.
19. c_3 is moved to the PL.
20. When the sixth nozzle is picking up a component of type 2, c_3 is being placed, as shown in Figure 4.7.

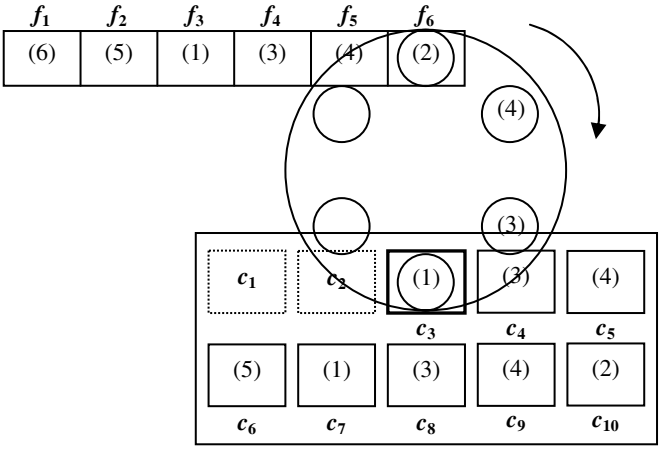

Figure 4.7. The schematic diagram of assembly sequence number 20

21. The turret rotates one step.
22. f_5 is moved to the PUL.
23. c_4 is moved to the PL.
24. When the seventh nozzle is picking up a component of type 4, c_4 is being placed, as shown in Figure 4.8.

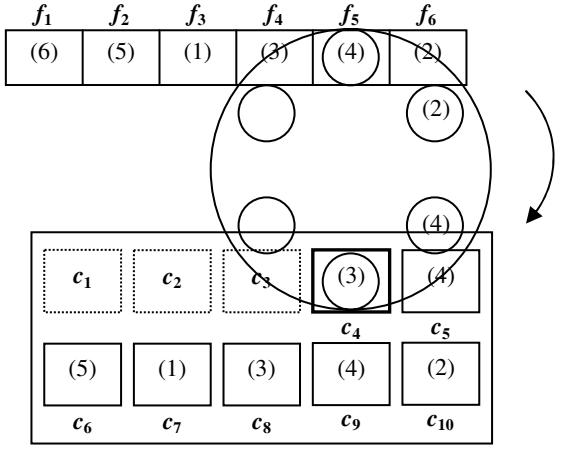

Figure 4.8. The schematic diagram of assembly sequence number 24

25. The turret rotates one step.
26. f_4 is moved to the PUL.
27. c_5 is moved to the PL.
28. When the eighth nozzle is picking up a component of type 3, c_5 is being placed, as shown in Figure 4.9.

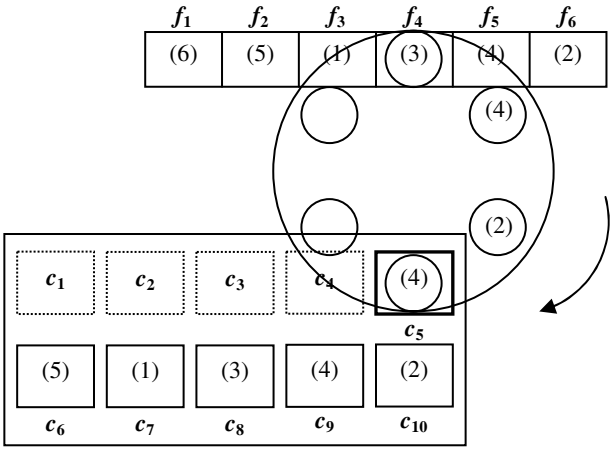

Figure 4.9. The schematic diagram of assembly sequence number 28

29. The turret rotates one step.
30. f_3 is moved to the PUL.
31. c_{10} is moved to the PL.
32. When the ninth nozzle is picking up a component of type 1, c_{10} is being placed, as shown in Figure 4.10.

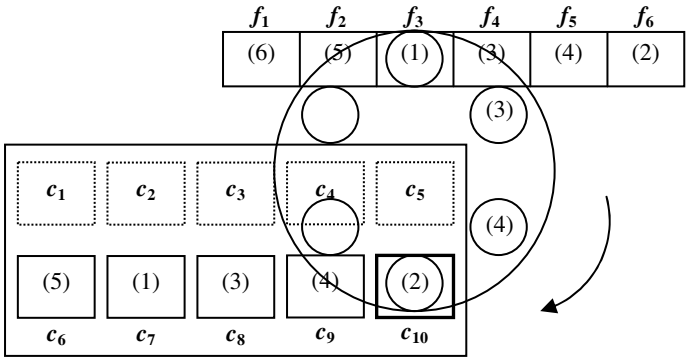

Figure 4.10. The schematic diagram of assembly sequence number 32

33. The turret rotates one step.
34. f_2 is moved to the PUL.
35. c_9 is moved to the PL.
36. When the tenth nozzle is picking up a component of type 5, c_9 is being placed, as shown in Figure 4.11.

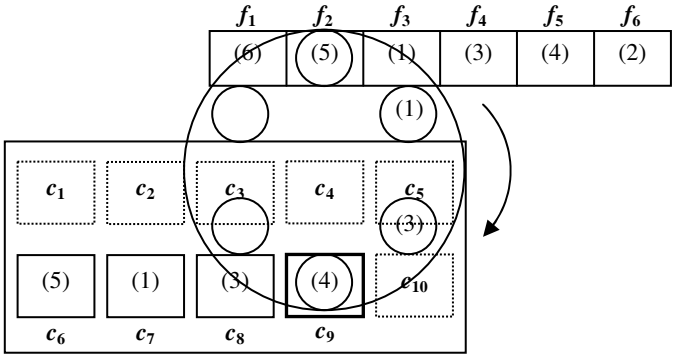

Figure 4.11. The schematic diagram of assembly sequence number 36

37. The turret rotates one step.
38. c_8 is moved to the PL.
39. The eighth nozzle places c_8, as shown in Figure 4.12.

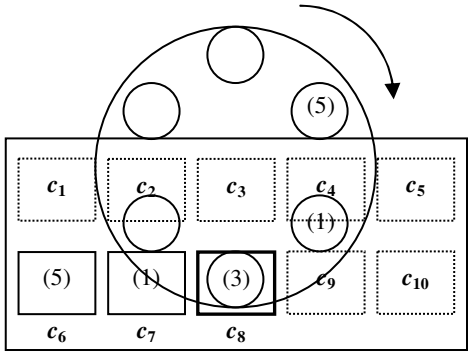

Figure 4.12. The schematic diagram of assembly sequence number 39

40. The turret rotates one step.
41. c_7 is moved to the PL.
42. The ninth nozzle places c_7, as shown in Figure 4.13.

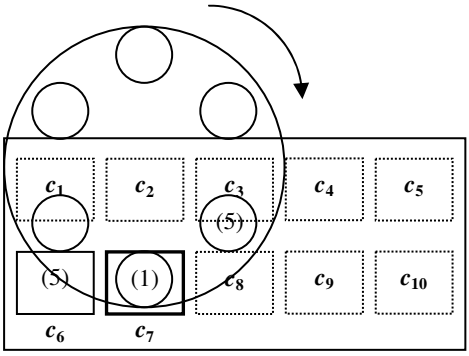

Figure 4.13. The schematic diagram of assembly sequence number 42

43. The turret rotates one step.
44. c_6 is moved to the PL.
45. The tenth nozzle places c_6, as shown in Figure 4.14.

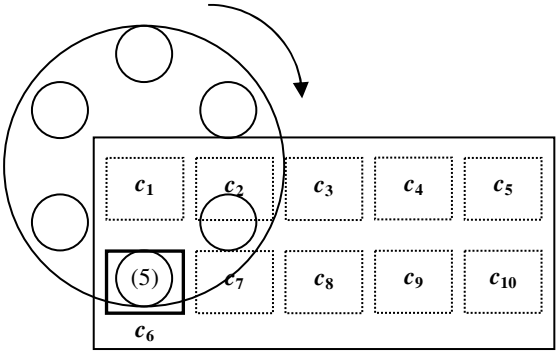

Figure 4.14. The schematic diagram of assembly sequence number 45

Note that all mechanisms, including the feeder carrier, the turret, and the X-Y table, move concurrently during the assembly of each component, except the first few and the last few components. For the first few components assembled on the PCB, there are only pickup movements and no placement movement. For the last few components of the same board, there are only placement movements and no pickup movement. However, these boundary effects can be neglected provided that the number of the components to be placed is very large or the batch size is very large (Leu *et al.*, 1993).

4.4 Notation

Before formulating the individual and the integrated mathematical models, the meaning of the notation is explained first in this section. If there are g components between the pickup location and the placement location, then there are $(2g + 2)$ assembly heads in the rotary turret. Each of the PCBs in the batch has n components with μ different types. Besides, each of the component types must be stored in a feeder, but a feeder can only hold a unique type of component, so μ feeders are needed to hold μ types of components.

Three mechanisms of the CS machine move at different speeds, so the travel time of the PCB or the X-Y table (*i.e.*, $C1_{ij}$), the travel time of the feeder carrier (*i.e.*, $C2_{rs}$), and the indexing time of the turret (*i.e.*, $C3$) are different from each other. These three times can be calculated as follows:

$C1_{ij}$ = time used by the X-Y table for traveling from component i (location) to component j (location).

$$= \max\left(\frac{|X_j - X_i|}{V_x}, \frac{|Y_j - Y_i|}{V_y}\right).$$

$C2_{rs}$ = time used by the feeder carrier for traveling from feeder r to feeder s.

$$= \frac{|X_s - X_r|}{V_f}.$$

$C3$ = time used by the turret for rotating one step,

where

X_i and X_j are the x-coordinates of components i and j, respectively,
Y_i and Y_j are the y-coordinates of components i and j, respectively,
V_x and V_y are speeds of the X-Y table in the x and y directions, respectively,
X_r and X_s are the x-coordinates of the feeders r and s, respectively, and
V_f is the speed of the feeder carrier.

The longest one among the three times in one step is called the dominating time. So, for the CS machine, the objective of the integrated problem is obviously to minimize the total assembly time, which is the summation of all dominating times of components so that the highest productivity of the machine can be achieved. The notation used in the models is summarized in Table 4.1.

Table 4.1. Notation

Indexes:

i, j: components ($i, j = 1, 2, \ldots, n$).

t: component types ($t = 1, 2, \ldots, \mu$).

r, s: feeders ($r, s = 1, 2, \ldots, \mu$).

p: placement order or placement position ($p = 1, 2, \ldots, n$).

Travel times:

$C1_{ij}$: travel time of the X-Y table.

$C2_{rs}$: travel time of the feeder carrier.

$C3$: indexing time of the turret.

T_p: the longest travel time among three for assembling the pth component.

Feeder constraint:

g: number of components between the pickup and the placement locations.

Decision variables:

$x_{ip} = 1$ if component i is placed in the pth position; 0 otherwise.

$y_{t_i r} = 1$ if component i with component type t is stored in feeder r; 0 otherwise.

4.5 Mathematical Models

Similar to that for the PAP machine, mathematical modeling is applied to optimize the performance of the CS machine. In the following subsections, individual component sequencing and feeder arrangement problems are constructed in advance. After that, several integrated models, including nonlinear and linear types, are constructed. To determine the best way to optimize machine performance, the iterative approach is compared with the integrated one with respect to the capability of obtaining a global optimal solution. In addition, the computational analysis of all integrated models is carried out.

4.5.1 A Component Sequencing Model

Assuming that the feeder arrangement problem is solved beforehand, the component sequencing problem is frequently formulated as the TSP for finding the placement order of components on a PCB so that the total travel distance or time of the X-Y table is minimized. In the TSP model, a decision variable, x_{ij}, is normally used to indicate that component i is placed immediately before component j if $x_{ij} = 1$. However, subtours may be formed, and thus the bulky subtour elimination constraints are essential in the model, as discussed in Chapter 3. Here, x_{ip} instead of x_{ij} is used as the decision variable. The interpretation of x_{ip} is that

$$x_{ip} = \begin{cases} 1 & \text{if component } i \text{ is placed in the } p\text{th position,} \\ 0 & \text{otherwise} \end{cases}$$

The idea is to assign n components to n positions, which means that there are totally n^2 decision variables in which only n variables are 1 and all others are 0. Each component must be placed in exactly one position, so no subtour will appear in this situation. If component 1 and component 2 are placed first and second, respectively, then x_{11} (*i.e.*, component 1 in the first position) and x_{22} (*i.e.*, component 2 in the second position) are both 1.

For the component sequencing model, only x_{ip} is incorporated, whereas the assignment of component types to feeders (*i.e.*, $y_{t_i r}$) is predetermined. After the feeder arrangement has been generated, the objective function of the model can then be constructed as

$$\text{Minimize } z = \max \left(\sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} C1_{ij}\ x_{i,p-1}\ x_{j,p}, \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} C2_{rs}\ x_{i,p+g}\ x_{j,p+g+1}, C3 \right)$$

$$\text{for } p = 1, 2, \ldots, n$$

The above objective function is to minimize the summation of all dominating times among $C1_{ij}$, $C2_{rs}$, and $C3$ of the components. First of all, $C1_{ij}$ calculates the time used by the X-Y table for traveling from the position of component i on the PCB to the position of component j, which is placed in the pth position. As described earlier, one nozzle is placing a component on the PCB while another nozzle is picking up another component from a feeder at the same time. Here, when the pth component is being placed, the type of another component to be placed in the $(p + g + 1)$th position is being picked up from a feeder. It is assumed that the types of the $(p + g)$th and the $(p + g + 1)$th components are stored in feeders r and s, respectively. So, $C2_{rs}$ calculates the time used by the feeder carrier for traveling from feeder r to feeder s. Third, $C3$ is the indexing time of the turret.

After introducing T_p, which is the dominating time for assembling the pth component, and incorporating the constraint sets for determining the sequence of component placements, the mathematical model for the component sequencing problem can be represented as

$$\text{Minimize } z = \sum_{p=1}^{n} T_p \tag{4.1}$$

subject to

$$T_p - \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} C1_{ij}\ x_{i,p-1}\ x_{j,p} \geq 0 \qquad \text{for } p = 1, 2, \ldots, n \tag{4.2}$$

$$T_p - \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} C2_{rs}\, x_{i,p+g}\, x_{j,p+g+1} \geq 0 \qquad \text{for } p = 1, 2, \ldots, n \qquad (4.3)$$

$$T_p - C3 \geq 0 \qquad \text{for } p = 1, 2, \ldots, n \qquad (4.4)$$

$$\sum_{i=1}^{n} x_{ip} = 1 \qquad \text{for } p = 1, 2, \ldots, n \qquad (4.5)$$

$$\sum_{p=1}^{n} x_{ip} = 1 \qquad \text{for } i = 1, 2, \ldots, n \qquad (4.6)$$

All $x_{ip} = 0$ or 1; $T_p \geq 0$ and is a set of integers. (M4-1)

Constraint set (4.2) is to calculate the travel time of the X-Y table for assembling the pth component. Constraint set (4.3) is to calculate the travel time of the feeder carrier for assembling the pth component. Constraint set (4.4) is the indexing time of the turret. For example, when $p = 1$, if $T_1 \geq 5$, $T_1 \geq 4$, and $T_1 \geq 3$ in constraint sets (4.2), (4.3), and (4.4), respectively, then T_1 will become 5 to satisfy all constraints. This is the idea of obtaining the value of T_p. Besides, constraint set (4.5) is to guarantee that exactly one component is placed in one position, and constraint set (4.6) is to guarantee that one position has exactly one component placed.

The assembly time of the CS machine is dependent on all the $C1_{ij}$, $C2_{rs}$, and $C3$. These three travel times must be incorporated together in the objective function of the component sequencing model. Focusing only on the travel time of the X-Y table cannot represent the actual situation, and therefore the TSP may not be desirable for the component sequencing problem.

4.5.2 A Feeder Arrangement Model

Assuming that the sequence of component placements is predetermined, some researchers formulated the feeder arrangement problem as the QAP for assigning the component types to feeders so that the number of feeder carrier's movements or the total travel time of the feeder carrier is minimized. As in the component sequencing model, the travel times of the X-Y table and the feeder carrier and the indexing time of the turret should be considered simultaneously in the feeder arrangement model. So, the QAP may not be desirable in this case.

In the model, only $y_{t_i r}$ is incorporated, whereas the sequence of component placements (*i.e.*, x_{ip}) is known in advance. It is to determine which component type is stored in which feeder. It is assumed that the number of feeders available is equivalent to that of component types required, so a single component type can only be assigned to a feeder. The interpretation of $y_{t_i r}$ is that

$$y_{t_i r} = \begin{cases} 1 & \text{if component type } t \text{ of component } i \text{ is stored in feeder } r, \\ 0 & \text{otherwise} \end{cases}$$

After x_{ip} has been known, the objective function of the feeder arrangement model is constructed as

$$\text{Minimize } z = \max \left(C1_{ij}, \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{t=1}^{\mu} \sum_{r=1}^{\mu} \sum_{s=1}^{\mu} C2_{rs}\ y_{t_i r}\ y_{t_j s}, C3 \right)$$

for $p = 1, 2, \ldots, n$

The above objective function calculates the total assembly time for assembling all components on a PCB. It is the summation of all dominating times among $C1_{ij}$, $C2_{rs}$, and $C3$ of the components. The placement order of each component is known, so the times used by the X-Y table for traveling from the position of component i to that of component j (*i.e.*, $C1_{ij}$) can be obtained directly. Note that the index j in $C1_{ij}$ refers to component j to be placed in the pth position. When the position of the pth component is being moved to the placement location, the feeder holding the type of component to be placed in the $(p + g + 1)$th position is being moved to the pickup location. Here, the indexes i and j in $C2_{rs}$ refer to components i and j, respectively. Also, component j is placed in the $(p + g + 1)$th position, and component i is placed immediately prior to component j. If the type of component i (*i.e.*, t_i) is stored in feeder r and the type of component j (*i.e.*, t_j) is assigned to feeder s, $C2_{rs}$ is equivalent to the time used by the feeder carrier to travel from feeder r to feeder s.

By introducing T_p and incorporating the constraint sets for determining the assignment of component types to feeders, the mathematical model for the feeder arrangement problem can be formulated as

$$\text{Minimize } z = \sum_{p=1}^{n} T_p \tag{4.7}$$

subject to

$$T_p - C1_{ij} \geq 0 \qquad \text{for } p = 1, 2, \ldots, n \tag{4.8}$$

$$T_p - \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{t=1}^{\mu} \sum_{r=1}^{\mu} \sum_{s=1}^{\mu} C2_{rs}\ y_{t_i r}\ y_{t_j s} \geq 0 \qquad \text{for } p = 1, 2, \ldots, n \tag{4.9}$$

$$T_p - C3 \geq 0 \qquad \text{for } p = 1, 2, \ldots, n \tag{4.10}$$

$$\sum_{t=1}^{\mu} y_{tr} = 1 \qquad \text{for } r = 1, 2, \ldots, \mu \qquad (4.11)$$

$$\sum_{r=1}^{\mu} y_{tr} = 1 \qquad \text{for } t = 1, 2, \ldots, \mu \qquad (4.12)$$

All $y_{tr} = 0$ or 1; $T_p \geq 0$ and is a set of integers. (M4-2)

Similar to M4-1, the minimax type objective function is transformed into a simple objective function (4.7) while subject to three constraint sets (4.8), (4.9), and (4.10) in M4-2. Constraint set (4.11) ensures that exactly one component type is stored in one feeder. Constraint set (4.12) ensures that exactly one feeder holds one component type.

4.5.3 Integrated Mathematical Models

Before solving the component sequencing model (*i.e.*, M4-1), it is essential to obtain the solution of the feeder arrangement problem (*i.e.*, M4-2) first. On the other hand, M4-2 cannot be solved until the solution of M4-1 is known. Therefore, there is no doubt that the component sequencing and the feeder arrangement problems are interrelated, and it is more suitable to consider the two problems simultaneously rather than separately, as many researchers did.

Consider a PCB to be assembled by the CS machine. The problem here is to determine the placement order of components (*i.e.*, x_{ip}) and to assign component types to feeders (*i.e.*, $y_{t_i r}$) at the same time so that the total assembly time (*i.e.*, z) is minimized. The X-Y table, the feeder carrier, and the turret move simultaneously, so the three travel times of these movable mechanisms need to be taken into consideration. The longest travel time among the three, that is, the dominating time, is the assembly time of the component. The summation of all dominating times of the components on the PCB is the total assembly time. The movements of the two faster mechanisms need to wait until the slowest movement has been completed. For example, if the travel time of the feeder carrier is the longest for assembling a particular component, then obviously it becomes the assembly time of that component. The turret and the X-Y table will delay for a while. Also, there is no pickup and placement operation until the next feeder which holds the next required component type has been moved to the pickup location. This leads to the time lag between each component's pickup and placement.

First, to obtain the value of $C1_{ij}$ (*i.e.*, the travel time of the X-Y table), the placement order of components must be known beforehand. For example, $C1_{ij}$ is the time used by the X-Y table to travel from the position of component i on the PCB to the position of component j if component i is placed in the $(p - 1)$th position followed by component j. If component 1 is placed first (*i.e.*, x_{11}) followed by component 2 (*i.e.*, x_{22}), then the travel time is $C1_{12}$. Therefore, the travel time of the X-Y table for assembling the pth component is

$$\sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} C1_{ij}\ x_{i,p-1}\ x_{j,p}$$

Comparatively, it is difficult to obtain the value of $C2_{rs}$ (*i.e.*, the travel time of the feeder carrier). The assignment of component types to feeders and also the placement order of components must be known in advance to calculate $C2_{rs}$. When the pth component is being moved to the placement location, the feeder holding the type of $(p + g + 1)$th component is being moved to the pickup location. The pickup and placement operations take place when the movements have been finished. For example, $C2_{rs}$ is the time used by the feeder carrier to travel from feeder r to feeder s if the type of component i placed in the $(p + g)$th position is stored in feeder r and the type of component j placed in the $(p + g + 1)$th position is stored in feeder s. Consider the 10-component problem in Figure 4.6, in which g equals to 2. While the X-Y table is moving from the position of c_1 to the position of c_2, if $p = 2$, the feeder carrier is moving from f_4 [because it holds the type of component 4 to be placed in the $(p + g)$th position or the fourth position] to f_5 [because it holds the type of component 5 to be placed in the $(p + g + 1)$th position or the fifth position]. Therefore, the travel time of the feeder carrier for assembling the pth component is

$$\sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{t=1}^{\mu} \sum_{r=1}^{\mu} \sum_{s=1}^{\mu} C2_{rs}\ x_{i,p+g}\ x_{j,p+g+1}\ y_{t_i r}\ y_{t_j s}$$

The indexing time of the turret is a constant, so the value of $C3$ remains the same for the assembly of every component on the PCB. For the integrated model, the objective is to minimize the summation of all dominating times among $C1_{ij}$, $C2_{rs}$, and $C3$ of all components on the PCB. So, the integrated model can be formulated as

Minimize $z =$

$$\max\left(\sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} C1_{ij}\ x_{i,p-1}\ x_{j,p}, \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{t=1}^{\mu} \sum_{r=1}^{\mu} \sum_{s=1}^{\mu} C2_{rs}\ x_{i,p+g}\ x_{j,p+g+1}\ y_{t_i r}\ y_{t_j s}, C3 \right)$$

for $p = 1, 2, \ldots, n$ (4.13)

subject to

$$\sum_{i=1}^{n} x_{ip} = 1 \qquad \text{for } p = 1, 2, \ldots, n \qquad (4.14)$$

$$\sum_{p=1}^{n} x_{ip} = 1 \qquad \text{for } i = 1, 2, \ldots, n \qquad (4.15)$$

$$\sum_{t=1}^{\mu} y_{tr} = 1 \qquad \text{for } r = 1, 2, \ldots, \mu \qquad (4.16)$$

$$\sum_{r=1}^{\mu} y_{tr} = 1 \qquad \text{for } t = 1, 2, \ldots, \mu \qquad (4.17)$$

All x_{ip} and $y_{tr} = 0$ or 1. (M4-3)

The objective function (4.13) is a minimax type formulation. After incorporating T_p and adding the constraints for determining the placement order of components and the assignment of component types to feeders, the mathematical model for the integrated problem in the form of minimization is formulated as

$$\text{Minimize } z = \sum_{p=1}^{n} T_p \qquad (4.18)$$

subject to

$$T_p - \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} C1_{ij}\, x_{i,p-1}\, x_{j,p} \geq 0 \qquad \text{for } p = 1, 2, \ldots, n \qquad (4.19)$$

$$T_p - \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{t=1}^{\mu} \sum_{r=1}^{\mu} \sum_{s=1}^{\mu} C2_{rs}\, x_{i,p+g}\, x_{j,p+g+1}\, y_{t_i r}\, y_{t_j s} \geq 0$$

$$\text{for } p = 1, 2, \ldots, n \qquad (4.20)$$

$$T_p - C3 \geq 0 \qquad \text{for } p = 1, 2, \ldots, n \qquad (4.21)$$

$$\sum_{i=1}^{n} x_{ip} = 1 \qquad \text{for } p = 1, 2, \ldots, n \qquad (4.22)$$

$$\sum_{p=1}^{n} x_{ip} = 1 \qquad \text{for } i = 1, 2, \ldots, n \qquad (4.23)$$

$$\sum_{t=1}^{\mu} y_{tr} = 1 \qquad \text{for } r = 1, 2, \ldots, \mu \qquad (4.24)$$

$$\sum_{r=1}^{\mu} y_{tr} = 1 \qquad \text{for } t = 1, 2, \ldots, \mu \qquad (4.25)$$

All x_{ip} and $y_{tr} = 0$ or 1; $T_p \geq 0$ and is a set of integers. (M4-4)

There is a nonlinear expression in each of the constraint sets (4.19) and (4.20), and the model contains both binary variables (*i.e.*, x_{ip} and y_{tr} = 0 or 1) and integer variables (*i.e.*, T_p), so the above mathematical programming formulation can be regarded as a pure integer nonlinear programming model. The objective function (4.18) calculates the total assembly time. Constraint set (4.19) is to calculate the travel time of the X-Y table. Constraint set (4.20) is to calculate the travel time of the feeder carrier. Constraint set (4.21) is the turret indexing time. The interpretation of constraint sets (4.22) to (4.25) was mentioned in M4-1 and M4-2.

Note that M4-4 is very complex and very challenging because we have to consider both the component sequencing and the feeder arrangement problems simultaneously. Nevertheless, the complexity of the model is reduced provided that some assumptions are made. First, if the travel speed of the X-Y table is assumed to be much slower than the other two, including the feeder carrier's travel speed and the turret indexing speed, then $C1_{ij}$ is always the dominating time, and $C2_{rs}$ together with $C3$ can be neglected. Therefore, we can simply solve the model as the TSP for finding the minimum $C1_{ij}$. Second, the model's complexity can also be reduced if the travel speed of the feeder carrier is supposed to be much slower than the other two. In this case, we can simply treat the problem as finding the minimum $C2_{rs}$. However, in general, the travel speeds of both the X-Y table and the feeder carrier are different, and no one of them is much slower than the other one; therefore, the component sequencing problem and the feeder arrangement problem should be considered at the same time.

The nonlinear terms in the constraint sets (4.19) and (4.20) are in the form of products of binary variables, so M4-4 can be reformulated as a linear one by implementing the following steps:

- Replace the product term xy by a binary variable w;
- Impose the logical condition: $w = 1$ if and only if $x = 1$ and $y = 1$ by means of the extra constraints: $w \le x$, $w \le y$, and $w \ge x + y - 1$.

In the constraint set (4.19) of M4-4, the nonlinear term is in the form of products of two binary variables. Therefore, it can be rewritten as a linear constraint by introducing an extra binary variable w_{ijp} as well as three extra constraint sets. The interpretation of the decision variable w_{ijp} is

$$w_{ijp} = \begin{cases} 1 & \text{if component } j \text{ is placed just after component } i \text{ and} \\ & \text{component } j \text{ is placed in the } p\text{th position,} \\ 0 & \text{otherwise} \end{cases}$$

However, in the constraint set (4.20) of M4-4, the nonlinear term is in the form of products of four binary variables. So, the steps for converting it into a linear type need to be modified in this case. The major difference is that five instead of three extra constraints are introduced. Similar to that for the constraint set (4.19), a decision variable $w_{ij(p+g+1)rs}$ is introduced:

$$w_{ij(p+g+1)rs} = \begin{cases} 1 & \text{if the type of component } i \text{ is stored in feeder } r \text{ and the type of} \\ & \text{component } j \text{ placed in the } (p+g+1)\text{th position is stored in feeder } s, \\ 0 & \text{otherwise} \end{cases}$$

The mathematical model for the integrated problem in the form of a linear function can be formulated as

$$\text{Minimize } z = \sum_{p=1}^{n} T_p \tag{4.26}$$

subject to

$$T_p - \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} C1_{ij}\ w_{ijp} \geq 0 \qquad \text{for } p = 1, 2, \ldots, n \tag{4.27}$$

$$T_p - \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \sum_{r=1}^{\mu} \sum_{s=1}^{\mu} C2_{rs}\ w_{ij(p+g+1)rs} \geq 0 \qquad \text{for } p = 1, 2, \ldots, n \tag{4.28}$$

$$T_p - C3 \geq 0 \qquad \text{for } p = 1, 2, \ldots, n \tag{4.29}$$

$$\sum_{i=1}^{n} x_{ip} = 1 \qquad \text{for } p = 1, 2, \ldots, n \tag{4.30}$$

$$\sum_{p=1}^{n} x_{ip} = 1 \qquad \text{for } i = 1, 2, \ldots, n \tag{4.31}$$

$$\sum_{t=1}^{\mu} y_{tr} = 1 \qquad \text{for } r = 1, 2, \ldots, \mu \tag{4.32}$$

$$\sum_{r=1}^{\mu} y_{tr} = 1 \qquad \text{for } t = 1, 2, \ldots, \mu \tag{4.33}$$

$$w_{ijp} \leq x_{i,p-1} \qquad \text{for } i, j, p = 1, 2, \ldots, n;\ i \neq j \tag{4.34}$$

$$w_{ijp} \leq x_{jp} \qquad \text{for } i, j, p = 1, 2, \ldots, n;\ i \neq j \tag{4.35}$$

$$w_{ijp} \geq x_{i,p-1} + x_{jp} - 1 \qquad \text{for } i, j, p = 1, 2, \ldots, n;\ i \neq j \tag{4.36}$$

$$w_{ij(p+g+1)rs} \le x_{i,p+g} \quad \text{for } i, j, p = 1, 2, \ldots, n;\ i \neq j;\ \text{for } r, s = 1, 2, \ldots, \mu \tag{4.37}$$

$$w_{ij(p+g+1)rs} \le x_{j,p+g+1} \quad \text{for } i, j, p = 1, 2, \ldots, n;\ i \neq j;\ \text{for } r, s = 1, 2, \ldots, \mu \tag{4.38}$$

$$w_{ij(p+g+1)rs} \le y_{t_i r} \quad \text{for } i, j, p = 1, 2, \ldots, n;\ i \neq j;\ \text{for } r, s = 1, 2, \ldots, \mu \tag{4.39}$$

$$w_{ij(p+g+1)rs} \le y_{t_j s} \quad \text{for } i, j, p = 1, 2, \ldots, n;\ i \neq j;\ \text{for } r, s = 1, 2, \ldots, \mu \tag{4.40}$$

$$w_{ij(p+g+1)rs} \ge x_{i,p+g} + x_{j,p+g+1} + y_{t_i r} + y_{t_j s} - 3 \quad \text{for } i, j, p = 1, 2, \ldots, n;\ i \neq j;\ \text{for } r, s = 1, 2, \ldots, \mu \tag{4.41}$$

All x_{ip}, y_{tr}, w_{ijp}, and $w_{ij(p+g+1)rs} = 0$ or 1; $T_p \ge 0$ and is a set of integers. (M4-5)

In M4-5, constraint sets (4.27) and (4.34) to (4.36) are the linear expression of the constraint set (4.19). They calculate the time taken by the X-Y table to assemble the *p*th component. Besides, constraint sets (4.28) and (4.37) to (4.41) are the linear expression of the constraint set (4.20). They calculate the time spent by the feeder carrier to assemble the *p*th component. The interpretation of the objective function and the remaining constraint sets can be found in M4-4.

4.5.4 Iterative Approach vs. Integrated Approach

There is no doubt that the component sequencing and the feeder arrangement problems are closely related. Nevertheless, according to the literature discussed in Section 4.2, solving the problems for the CS machine individually rather than simultaneously is still prevalent. For example, the feeder arrangement problem is tackled in advance, followed by the component sequencing problem with respect to the optimal assignment of component types to feeders. If there is improvement in the solution's quality, the individual problems or mathematical models are solved repeatedly.

The iterative approach is comparatively easy to solve as only one single problem instead of two is focused on at a time. However, Altinkemer *et al.* (2000) pointed out that both problems should be considered and solved simultaneously, otherwise poor performance can result. To examine this and prove that the iterative approach may not generate a global optimal solution, an investigation of the effectiveness of the iterative approach was carried out.

In the following, an iterative approach, as illustrated in Figure 4.15, is applied to optimize the CS machine performance. The approach is similar to that proposed

in Section 3.5.4 (refer to Chapter 3), except that the feeder arrangement model (*i.e.*, M4-2) is solved first here while assuming that the placement sequence, generated randomly, is obtained beforehand. After that, the component sequencing model (*i.e.*, M4-1) is solved based on the optimal assignment of component types to feeders to find the minimum assembly time for assembling the four components on the PCB. This procedure forms one iteration and will continue unless there is no further improvement in the quality of the solution. The effectiveness of this iterative approach is then examined by comparing its final best solution with the global optimal solution generated by solving any one of the integrated models (*i.e.*, M4-4 and M4-5). The comparison directs us to the best way (*i.e.*, integrated or iterative?) for optimizing the CS machine performance.

The data of the four-component problem are listed in Table 4.2. The indexing time of the turret (*i.e.*, $C3$) is 0.25 second per step. The number of components between the pickup and the placement locations (*i.e.*, g) is 2. The X-Y table moves simultaneously and independently in the x and y directions, so the Chebyshev metric is adopted. Herein, the table's speed in both x and y directions (*i.e.*, V_x and V_y) is 60 mm/s. The linear speed of the feeder carrier (*i.e.*, V_f) is also 60 mm/s.

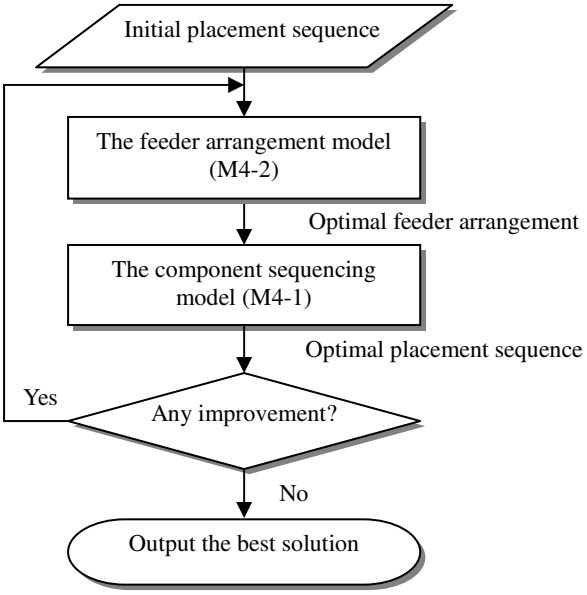

Figure 4.15. An iterative approach

Table 4.2. Data of the four-component problem

Components	Types	Coordinates (mm)		Feeders	Coordinates (mm)	
(i)	(t_i)	x	y	(r)	x	y
1	4	10	40	1	10	10
2	3	30	70	2	30	10
3	1	50	50	3	50	10
4	2	70	60	4	70	10

Suppose that an initial placement sequence is generated arbitrarily that starts with component 1, then components 4, 2, and finally component 3 (*i.e.*, $x_{11} = x_{42} = x_{23} = x_{34} = 1$). Based on this initial placement sequence, the optimal feeder arrangement can be obtained by solving M4-2. In this case, the optimal assignment of component types to feeders is that component types 4, 3, 2, and 1 are stored in feeders 1, 2, 3, and 4, and the total placement time is 2.67 seconds. Using the optimal feeder arrangement, the optimal placement sequence can then be found by solving M4-1. Surprisingly, note that the optimal placement sequence starts with component 1, components 4, 2, and finally component 3, which is exactly the same as the initial one. Because there is no further improvement in the objective value, the above procedure is terminated. Although the individual models are solved sequentially, the approach cannot generate the global optimal solution of the integrated models (*i.e.*, M4-4 and M4-5). The optimal sequence of component placements is $x_{21} = x_{12} = x_{43} = x_{34} = 1$, whereas the optimal feeder arrangement is $y_{41} = y_{22} = y_{13} = y_{34} = 1$. The minimum placement time is 2.16 seconds. Table 4.3 illustrates the calculation of the placement times for the global optimal solution.

Table 4.3. Calculation of the three travel times of the global optimal solution

	$C1_{ij}$	$C2_{rs}$	$C3$	T_p
$p = 1$	$C1_{32} = 0.33$	$C2_{23} = 0.33$	0.25	0.33
$p = 2$	$C1_{21} = 0.50$	$C2_{34} = 0.33$	0.25	0.50
$p = 3$	$C1_{14} = 1.00$	$C2_{41} = 1.00$	0.25	1.00
$p = 4$	$C1_{43} = 0.33$	$C2_{12} = 0.33$	0.25	0.33

As a consequence, it is proved that the iterative approach widely adopted by many researchers is not the best way because it cannot guarantee that the solution is globally optimal. Besides, note from the above experiment that the initial placement sequence plays a vital role in the approach. If the initial placement sequence is not done carefully, even if the feeder arrangement problem is solved to optimality, it could result in poor performance. According to the above observations, it is concluded that the component sequencing and the feeder arrangement problems must be integrated rather than separated to optimize the CS machine performance or find the shortest placement time.

4.5.5 Computational Analysis

Although the formulations of M4-4 and M4-5 are different, the objective values of both models are the same due to their equivalent physical meanings. In this section, the computing complexity of the models is studied first. The numbers of variables and constraints of the models are listed. After that, the computational time spent in solving the models with different problem sizes is presented.

4.5.5.1 Computing Complexity

The complexity of a model increases when the numbers of variables and constraints increase. Generally, a complex model is more difficult to solve and takes more time for computation. Therefore, the amount of computational time spent is directly proportional to the complexity of a model. The numbers of variables and constraints in each of M4-4 and M4-5 are listed in Table 4.4.

M4-4 has $(n^2 + \mu^2)$ binary variables, n integer variables, and $(5n + 2\mu)$ constraints. In each of the constraint sets (4.19) and (4.20), the terms are $n(n - 1)$ and $n\mu^2(n - 1)$, respectively. For a realistically sized problem of 100 components and 10 component types, M4-4 has 10,100 binary variables, 100 integer variables, and 520 constraints. In addition, M4-4 has 9,900 and 990,000 terms in each of the constraint sets (4.19) and (4.20), respectively!

Although M4-5 becomes linear, both numbers of variables and constraints increase greatly. For the number of binary variables, $n^2(n - 1)$ of w_{ijp} and $n^2\mu^2(n - 1)$ of $w_{ij(p+g+1)rs}$ are introduced in M4-5 besides $(n^2 + \mu^2)$ binary variables and n integer variables. For the number of constraints, besides $(5n + 2\mu)$ constraints, M4-5 has $[3n^2(n - 1)] + [5n^2\mu^2(n - 1)]$ constraints more in which there are $[3n^2(n - 1)]$ constraints for constraint sets (4.34) to (4.36) and there are $[5n^2\mu^2(n - 1)]$ constraints for constraint sets (4.37) to (4.41). For a realistically sized problem of 100 components and 10 component types, M4-5 has 100,000,200 variables, and 497,970,520 constraints.

Table 4.4. Numbers of variables and constraints in M4-4 and M4-5

	M4-4	**M4-5**
No. of variables	$n^2 + \mu^2 + n$	$(n^2 + \mu^2 + n) + [n^2(n - 1)] + [n^2\mu^2(n - 1)]$
No. of constraints	$5n + 2\mu$	$(5n + 2\mu) + [3n^2(n - 1)] + [5n^2\mu^2(n - 1)]$

4.5.5.2 Computational Time

BARON and CPLEX are adopted to verify the integer nonlinear programming model (*i.e.*, M4-4) and the integer linear programming model (*i.e.*, M4-5), respectively. By these two commercial packages, both M4-4 and M4-5 are tested by several small examples, and both are correct. According to Table 4.5, it is found that M4-4 is better than M4-5 in terms of the amount of computational time spent to obtain the global optimum. For example, it only takes 14½ hours by BARON to solve M4-4 with the example of eight components and eight types to optimality. However, it takes more than 13 days by CPLEX to solve M4-5 for the same problem to optimality.

Although both models can obtain the global optimum, they are not efficient approaches because the computational time grows exponentially with the problem size, as seen in Table 4.5.

Table 4.5. Computational time spent in solving M4-4 and M4-5

Numbers of components and types	Optimal solution CPU time (hh:mm:ss) by BARON for M4-4	Optimal solution CPU time (hh:mm:ss) by CPLEX for M4-5
4×4	00:00:01	00:00:01
5×5	00:00:14	00:00:20
6×6	00:02:27	00:07:19
7×7	00:47:20	04:37:39
8×8	14:30:25	328:07:51

4.6 Genetic Algorithms

Two types of mathematical models have been formulated for the integrated problem, including the pure integer nonlinear programming and the pure integer linear programming models. Because both types of models are among the class of theoretically difficult problems (NP-complete) (Kallrath, 1999), the integrated problem is therefore NP-complete. Solving it can be a challenging task because it requires an extremely long time to find the global optimum, as shown in Section 4.5.5.2. To solve the models efficiently, it is necessary to apply a heuristic method.

The HGA presented in the previous chapter is adopted here to solve the integrated problem for the CS machine. Actually, the algorithm is so flexible that it can be modified to optimize different types of placement machines. The only modification is the evaluation function. So, the evaluation function for the CS machine is discussed only in this section, whereas for the remaining elements of the HGA, refer to Chapter 3.

4.6.1 Evaluation

The objective of the integrated problem for the CS machine is to minimize the total placement time for assembling all electronic components on a PCB. So certainly, the fitness function for the CS machine should be the total placement time, which is the summation of all dominating times of components because the three mechanisms of the machine move at different speeds: the travel time of the PCB or the X-Y table, the travel time of the feeder carrier, and the indexing time of the turret. The longest among the three is the dominating time needed in the assembly of the component. Let T_i be the time needed for the placement of component i and $eval(X_h)$ be the total placement time or the fitness function for chromosome X_h in the integrated problem. Then,

$$T_i = \max\{t_1[c(i-1), c(i)],\ t_2[f(i+g), f(i+g+1)],\ t_3\}$$

$$eval(X_h) = \sum_{i=1}^{n} T_i$$

where

$t_1(a, b)$ is the travel time of the X-Y table from component a to component b

$$t_1(a, b) = \max\left(\frac{|x_b - x_a|}{V_x}, \frac{|y_b - y_a|}{V_y}\right) \text{ for the Chebyshev metric}$$

V_x and V_y are the speeds of the X-Y table in the x and y directions, respectively

$t_2(u, v)$ is the travel time of the feeder carrier from feeder u to feeder v

$$t_2(u, v) = \frac{|x_v - x_u|}{V_f}$$

V_f is the speed of the feeder carrier

t_3 is the indexing time of the turret

n is the number of components

$c(i)$ is the location in the board for the ith component

$f(i)$ is the feeder location for the ith component type

g is the number of components in the gap between the pickup component and the placement component in the turret, and normally, $g = 5$ or 6

In the above expression, $c(i - 1) = c(n)$ when $i = 1$. When $f(l)$ has $l > n$, where $l = i + g$ or $i + g + 1$, $f(l)$ is replaced by $f(l - n)$, which represents the initial g components of the next board in the batch.

4.6.2 Performance Analysis

The performance of the HGA for the CS machine is evaluated using the PCB example in Leu *et al.* (1993). The example has 50 components with 10 different types. The parameters of the HGA for the integrated problem for the CS machine are preset as *psize* = 25, *itno* = 1000, *cr* = 0.4, and *mr* = 0.2. Therefore, five pairs of chromosome are selected to perform the modified order crossover operation, whereas five chromosomes perform the heuristic mutation and the inversion operation. The total number of offspring produced per iteration will be 40, 10 from the modified order crossover operation, 25 from the heuristic mutation operation, and 5 from the inversion mutation operation.

4.6.2.1 Comparison to Other Approaches

The performance of the HGA is shown in Figure 4.16, whereas the comparison between the results obtained from the HGA and those obtained from other researchers (Leu *et al.*, 1993; Ong and Tan 2002) is shown in Table 4.6. According to Table 4.6, the performance of the HGA is superior to that of the simple GAs used in Leu *et al.* (1993) and Ong and Tan (2002) in three aspects. First, the best chromosome (30 seconds) in the initial population obtained by the HGA is better than that of the simple GAs (both more than 60 seconds). Second, the HGA can yield a better solution with fewer iterations, 323 vs. 1,750 or 5,000. Finally and the

most important, the HGA gave a better solution than the previous methods, 26 seconds vs. 51.5 seconds or 26.9 seconds.

Table 4.6. A comparison of the experimental results for the CS machine

	Leu *et al.* (1993)	**Ong and Tan (2002)**	**HGA**
Best one in the initial solution (s)	70	About 60	30
Population size	100	10	25
Iteration number	About 1,750	5,000	323
Final best solution (s)	About 51.5	26.9	26

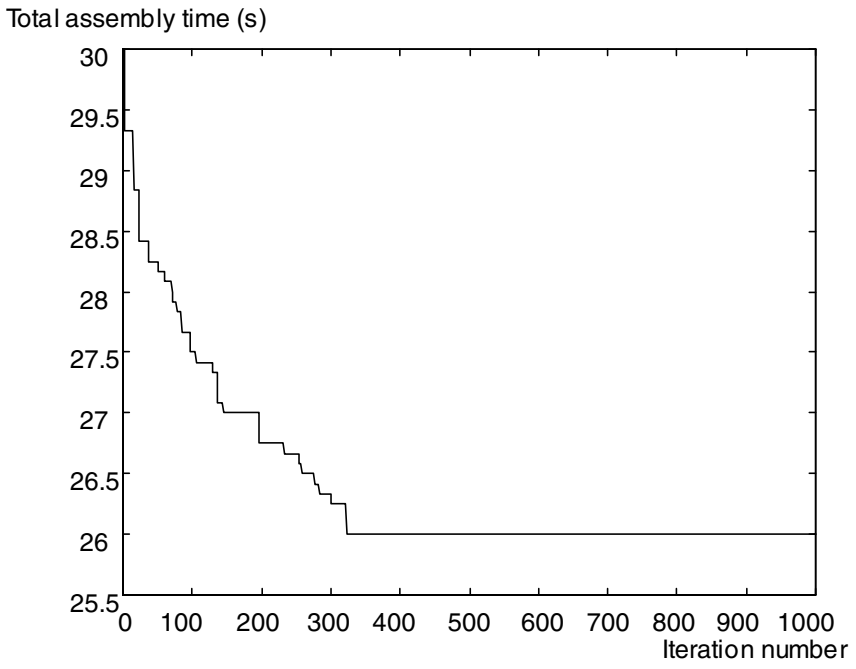

Figure 4.16. The minimum assembly time at each iteration

4.6.2.2 Effect of Population Size

To identify the effect of population size, the HGA program was run with the above 50-component problem using three different population sizes: 5, 25, and 50. It can be seen from Figure 4.17 that the HGA with population size of 50 obtains a better chromosome in the initial population, and also it requires a fewer number of iterations to obtain a better final solution. This may be due to the fact that more offspring are produced at each iteration. There are totally 80 offspring (20 from the modified order crossover operation, 50 from the heuristic mutation operation, and 10 from the inversion mutation operation) produced at each iteration. However,

there are only eight offspring (two from the modified order crossover operation, five from the heuristic mutation operation, and one from the inversion mutation operation) produced at each iteration if the population size is five. The effect of different population sizes on the 50-component problem is summarized in Table 4.7.

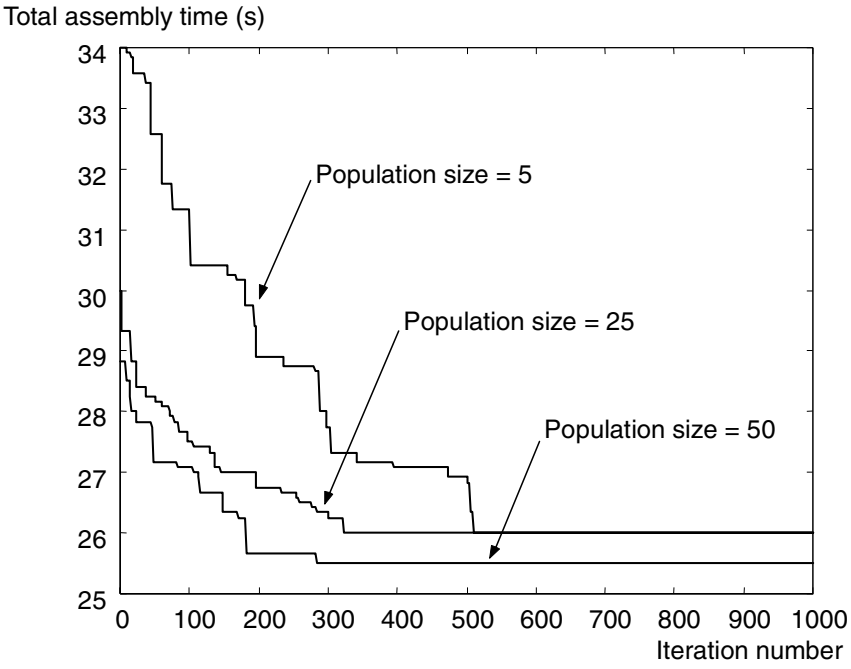

Figure 4.17. The effect of population size for the CS machine

Table 4.7. A comparison of different population sizes for the CS machine

Population size	5	25	50
Best one in the initial population (s)	34	30	28.83
Final Best Solution (s)	26	26	25.5
Iteration number	510	323	283

4.6.2.3 Comparison to Optimal Solution

In Section 4.5.5.2, several sizes of problems are solved to optimality. In this section, the performance of the HGA is examined and compared with the optimal solution. It is found that the HGA achieves the optimal solution for all problems. As shown in Table 4.8, the longest computational time spent is only five seconds for the eight-component problem. It shows that the HGA takes much less time, but, there is no guarantee that the optimal solution can be generated because the HGA is simply a heuristic method.

Table 4.8. Comparisons of the HGA with BARON and CPLEX for the CS machine

Numbers of components and types	Optimal solution CPU time (hh:mm:ss) by BARON for M4-4	Optimal solution CPU time (hh:mm:ss) by CPLEX for M4-5	Best solution CPU time (hh:mm:ss) by HGA
4×4	00:00:01	00:00:01	0.15 second
5×5	00:00:14	00:00:20	0.40 second
6×6	00:02:27	00:07:19	00:00:01
7×7	00:47:20	04:37:39	00:00:03
8×8	14:30:25	328:07:51	00:00:05

4.6.2.4 Integrated Problem with Feeder Duplication

In this section, besides the component sequencing and the feeder arrangement problems, an additional optimization problem is also considered for the CS machine. It is due to the fact that some components of same types may be in hundreds or even more. The movement of the feeder carrier can be reduced if two feeders are arranged for holding such frequently used component types. As a consequence, assembly time can be minimized. If some component types are assigned to two feeders, it is then necessary to determine from which feeder a component should be retrieved, that is, the component retrieval problem. So, the problem we face in this section is to solve the component sequencing, the feeder arrangement, and the component retrieval problems simultaneously for the CS machine.

When a component type is assigned to two feeders, a retrieval plan must be set up to determine from which feeder a component should be retrieved. The retrieval plan is similar to the NNH. For the first component, if its type is stored in two feeders, then select a feeder randomly. But if the types of the remaining components are stored in two feeders, then select the feeders as close as possible to the previous ones so that the movements of the feeder carrier are minimized. For example, if a component type is stored in two feeders, say $v1$ and $v2$, and the previous feeder to be visited is u, then select $v1$ if $|x_{v1} - x_u| < |x_{v2} - x_u|$, or select $v2$ provided that $|x_{v1} - x_u| > |x_{v2} - x_u|$. When $|x_{v1} - x_u| = |x_{v2} - x_u|$, then select a feeder randomly.

The performance of the HGA for solving the component sequencing, the feeder arrangement, and the component retrieval problems simultaneously is evaluated by use of the 50-component problem (Leu *et al.*, 1993) again in which three surplus feeders are available. In this case, three types of components can be assigned to two feeders. According to the 50-component problem, component types 4, 9, and 10 are the most frequently used; these three types of components can therefore be stored in two feeders.

The performance of the HGA for the three problems is shown in Figure 4.18, and its result is listed in Table 4.9. It is found that the solution is even better if the number of feeders available is more than that of component types required, 25

seconds vs. 26 seconds. Because the three most frequently used component types are assigned to more than one feeder, they can be retrieved from a closer feeder according to the retrieval plan. So, the travel time of the feeder carrier can be shortened. As a result, the total assembly time can be minimized, too.

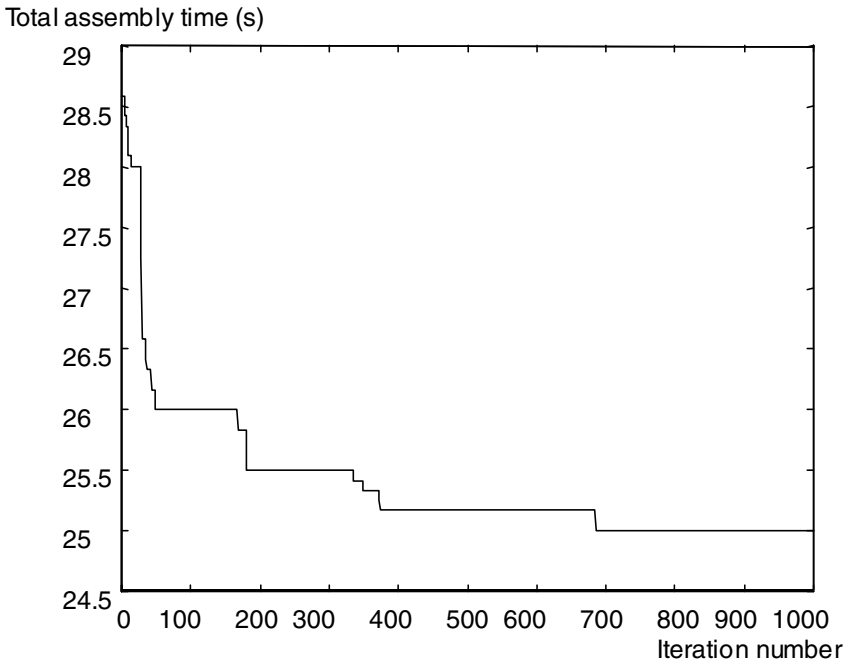

Figure 4.18. The performance of the HGA for the integrated problem with feeder duplication for the CS machine

Table 4.9. A comparison of the HGAs for the CS machine

	HGA	**HGA**
Number of surplus feeders	N/A	3
Population size	25	25
Iteration number	323	685
Final best solution (s)	26	25

4.7 Summary

In this chapter, the optimization of the CS machine performance is studied thoroughly and successfully. Mathematical modeling has been applied, and also a HGA has been used to solve the component sequencing and the feeder arrangement problems. Some remarks are summarized in the following.

1. Due to the CS machine's configuration, the travel time of the X-Y table, the travel time of the feeder carrier, and the indexing time of the turret must be considered simultaneously. The TSP and the QAP are therefore not suitable for the individual component sequencing and feeder arrangement models, respectively.
2. The component sequencing and the feeder arrangement problems are interrelated and inseparable. Each of the individual component sequencing and feeder arrangement models cannot be solved unless the solution of the other one is predetermined in advance.
3. The iterative approach, that is, sequentially solving the individual component sequencing and feeder arrangement models, adopted by many researchers, is not the best way to optimize the CS machine performance. The approach cannot guarantee that the solution is globally optimal.
4. Two mathematical models were formulated for the integrated problem, which considers simultaneously both the component sequencing problem and the feeder arrangement problem.
5. For integrated models, although the integer nonlinear programming model can be converted into the integer linear programming model equivalently, both numbers of variables and constraints increase significantly.
6. The nonlinear and the linear programming models were verified using commercial packages, and both of them generated the same optimal solutions of the same problems.
7. Different types of models took different times for computation. In terms of the amount of computational time spent in solving the model to global optimality, the integer nonlinear type is more desirable.
8. Although the optimal solution can be found using commercial packages, it was proved that the computational time grows exponentially with problem size.
9. The HGA developed for solving the integrated problem for the PAP machine can be modified easily to solve that for the CS machine. The only modification that needs to be made is the evaluation function.
10. The HGA proved superior to the simple GAs proposed by other researchers in terms of effectiveness as well as efficiency.
11. The algorithm with larger population size can yield a better final solution, but, at the same time, it requires more computational time.
12. Although the algorithm cannot guarantee obtaining the optimal solution, it was proved that the HGA can reach the global optimum of several problems with small sizes quickly.
13. The solution was even better when component types could be stored in more than one feeder.

In the next chapter, the focus is confined to another two PCB assembly problems called the line assignment problem and the component allocation problem. Similar to that in Chapter 3 and Chapter 4, mathematical modeling and the heuristic method are adopted to optimize the problems.

5

The Line Assignment and the Component Allocation Problems

5.1 Introduction

One of the objectives in this book is to develop a prototype of the PCB assembly planning system to be discussed in Chapter 6. The system, as shown in Figure 5.1, comprises three levels in which the problems are closely related. After different board types have been assigned to multiple assembly lines, that is, the line assignment problem (Level 1), the components or the component types of the

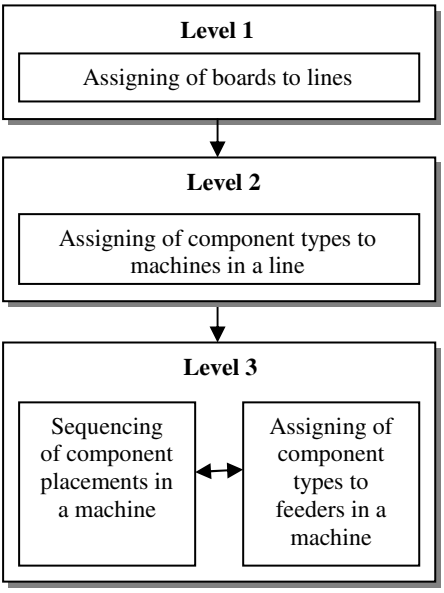

Figure 5.1. Overall structure of a PCB assembly planning system

board are allocated to multiple placement machines in a particular line, that is, the component allocation problem (Level 2). At the last stage, Level 3, the component sequencing problem and the feeder arrangement problem in each of the placement machines are determined. The integration of these problems is regarded as a PCB assembly planning system.

The integrated problems for both types of machines (*i.e.*, Level 3) have been studied thoroughly in the previous chapters (refer to Chapter 3 and Chapter 4), so only the remaining two problems (*i.e.*, Levels 1 and 2) are covered in this chapter. In Section 5.2 and Section 5.3, attention will be confined to several areas for the line assignment problem as well as the component allocation problem, respectively. The mathematical models are formulated for the problems first, followed by the solution approaches for solving them with numerical examples. Last, some remarks concerning this chapter are summarized in Section 5.4.

5.2 The Line Assignment Problem

Comparatively, the number of research projects on the line assignment problem is few. Rajkumar and Narendran (1997) presented a similarity-based heuristic to assign a given set of PCBs to any assembly machine among an available set of identical assembly machines, with the twin objectives of minimizing the makespan and balancing the load.

Hillier and Brandeau (1998) formulated an integer linear programming model for assigning the boards and components to the machines and manual process so as to minimize the total cost for PCB and component setup. An optimal solution technique was developed for the single-machine case and for the multimachine case where boards were not allowed to be set up on more than one process. Also, a heuristic approach was developed to obtain a near-optimal solution.

Balakrishnan and Vanderbeck (1999) developed an integer linear programming model that minimized the setup costs of the placement machines while ensuring that the total processing workload on each line did not exceed a predetermined limit. The model was to assign product families to parallel surface mount assembly lines. An optimization-based method incorporated with the initial product assignment heuristic, the column generation, and the lower bound procedures, was adopted to obtain a near-optimal solution.

Hillier and Brandeau (2001) formulated an integer linear programming model for assigning boards and components to the machines and manual process to minimize the production cost while at the same time balancing machine workloads. The machine capacity was also taken into consideration. A heuristic called the cost minimizing workload balancing was developed to generate the upper bounds. The branch-and-bound method was used to find the optimal solutions of small and medium-sized problems.

Ellis and Bhoja (2002) formulated the line assignment problem as a mixed integer linear programming model to minimize the total assembly time, including the setup time and the processing time for each of the board types. The problem was then solved using problem decomposition along with the branch-and-bound algorithm.

According to the above literature, all of them focused on the problem with a small production volume and a high variety of board types. In such a situation, boards of same type can only be assigned to a single assembly line so that the setup time is minimized. However, up to now, no researcher has studied the problem in a high-volume environment. So, it is believed that we are the first to investigate this.

A PCB manufacturing company may have several SMT production lines. It receives production orders for many distinct products every month. The production volume for each type of product is high. The scheduler has to determine which product to produce on which line and also the quantity of the product to be produced on the line so that the production cost is minimized. A product may be produced on one line only, or more than one lines, depending on the product order and the availability of the production lines. The line configurations are different from each other. Figure 5.2 shows *n* board types to be assigned to three assembly lines.

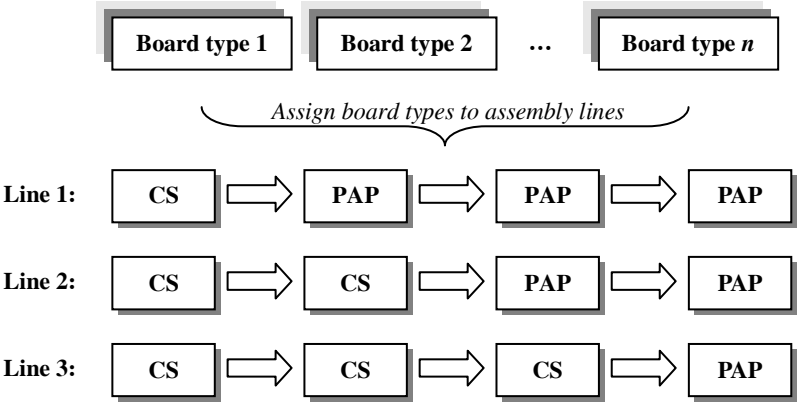

Figure 5.2. An example of the line assignment problem

If these three lines are assigned to produce a product, the times required by these three lines to produce one unit of the product are not identical, neither are the efficiency nor the cost.

The assignment of board types to multiple SMT production lines is addressed as a line assignment problem (*i.e.*, Level 1). Here, the problem is formulated as the generalized transportation problem (GTP). Actually, the GTP is an extension of the linear transportation problem, which is one of the famous linear programming models, as discussed in Section 2.2.1. The GTP (Balas and Ivanescu, 1964; Lourie, 1964) or the machine loading problem (Eisemann, 1964), proposed by Ferguson and Dantzig (1956), has been studied for a long time because of its wide applicability (Eisemann, 1964; Ji *et al.*, 1994).

5.2.1 A Mathematical Model

The line assignment problem is formulated as the GTP. The mathematical model of the GTP assumes that *m* SMT production lines are available for the production of *n*

types of products. Here, production means that each product is processed using a single line instead of using a specified sequence of lines. Furthermore, a product can be produced using any line. When line i is assigned to produce product j, it requires a_{ij} (> 0) hours and costs c_{ij} (> 0) dollars for one unit j. Besides, line i has a maximum of t_i hours available for production, and product j has a volume requirement of s_j. The problem here is to determine the quantity x_{ij} of product j to be produced on line i to minimize the total production cost. A pure integer linear programming model can be formulated as

$$\text{Minimize } z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{5.1}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_{ij} \le t_i \qquad \text{for } i = 1, 2, \ldots, m \tag{5.2}$$

$$\sum_{i=1}^{m} x_{ij} = s_j \qquad \text{for } j = 1, 2, \ldots, n \tag{5.3}$$

$$x_{ij} \ge 0 \text{ and is a set of integers.} \tag{M5-1}$$

The objective function (5.1) is to minimize the total production cost. Constraint set (5.2) is due to the limited available time, which means that each line must be operated within the fixed time period. Constraint set (5.3) is from the product volume requirement. M5-1 is referred to as the GTP.

Table 5.1 below shows a GTP tableau, which has four SMT production lines and five types of products. The northwest corner in cell (i, j) indicates the unit time a_{ij}, and the northeast corner represents the unit cost c_{ij}. For example, in cell (1, 1), a_{11} is 5 and c_{11} is 3.

Table 5.1. A generalized transportation problem

	Product j										
Line i	1		2		3		4		5		t_i
1	5	3	2	6	2	6	5	7	6	4	
	x_{11}		x_{12}		x_{13}		x_{14}		x_{15}		48000
2	2	6	7	5	8	6	7	15	6	8	
	x_{21}		x_{22}		x_{23}		x_{24}		x_{25}		48000
3	4	5	2	3	2	10	1	6	6	7	
	x_{31}		x_{32}		x_{33}		x_{34}		x_{35}		48000
4	3	4	2	4	4	7	5	8	2	9	
	x_{41}		x_{42}		x_{43}		x_{44}		x_{45}		48000
s_j	12000		8000		10000		8000		6000		

Many researchers presented a number of methods to solve the GTP. Lourie (1964) used the stepping stone algorithm associated with altering topology to solve the GTP. Eisemann (1964) proposed a generalized stepping stone algorithm, which is the extension of the loop technique of the stepping stone method for the Hitchcock transportation problem. Balas (1966) adopted the ideas underlying the usual stepping stone algorithm to specialize the dual method and the poly-ω technique for the GTP. Balachandran and Thompson (1975a–d) studied an operator theory of parametric programming for the problem. Besides, Thompson and Sethi (1986) developed a pivot and probe algorithm (PAPA) to solve an uncapacitated GTP with some side constraints. Ji *et al.* (1994) developed an algorithm for the dual form of the GTP from the idea of the revised simplex method. It was found that the performance of the algorithm is much better. However, all of these algorithms are based on the linear programming model of the GTP and the integer solution requirement is relaxed. Therefore, a heuristic approach is developed to solve M5-1 efficiently.

5.2.2 A Genetic Algorithm

Similar to that for the integrated problems for both types of placement machines in Level 3, a GA is adopted to deal with the line assignment problem in Level 1 or M5-1. However, because the type of encoding for the problem is different from that for the integrated problems, a tailor-made GA is developed and explained in the following.

The general structure of a GA for the line assignment problem is illustrated in Figure 5.3. The GA starts with an initial population in which the chromosomes are generated randomly. The fitness of chromosomes is then measured using the objective function of the model. Roulette wheel selection operation is performed to select some chromosomes for the next procedure, called genetic operations. These operations consist of crossover and mutation. However, the techniques applied are not the common type. The genetic operations, including those used in Chapters 3 and 4 (*i.e.*, the modified order crossover, the heuristic mutation, and the inversion mutation), are not suitable because the representation of the chromosomes in the line assignment problem is in the form of a matrix instead of a path representation. After the offspring is produced, their fitness will be measured and may become a member of population if it possesses a relatively good "quality". A new roulette wheel is then performed. These procedures form a cycle and the cycle will not be terminated until the predetermined number of iterations is conducted.

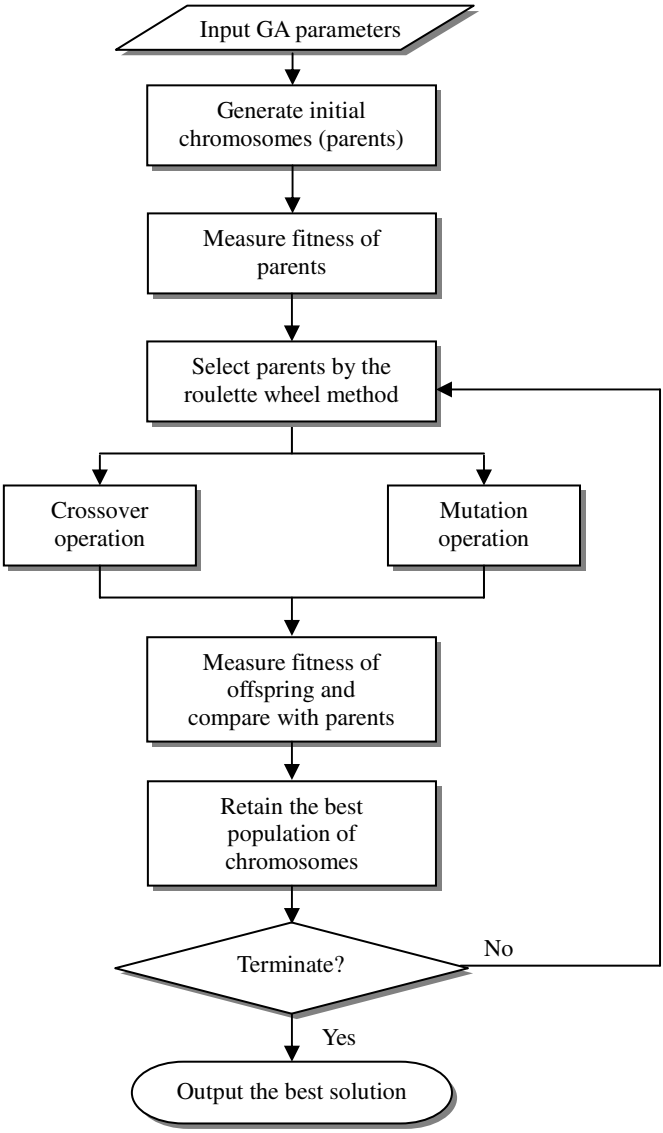

Figure 5.3. The general structure of the genetic algorithm

The procedure of the GA for the line assignment problem is as follows:

Step 1: Set the GA parameters, including the population size (*psize*), the number of iterations (*itno*), the crossover rate (*cr*), and the mutation rate (*mr*).

Step 2: Generate *psize* initial chromosomes using the initialization procedure to be discussed in Section 5.2.2.1.

Step 3: Evaluate the fitness value $eval(X_h)$ for all chromosomes in the population to be addressed in Section 5.2.2.2.

Step 4: Follow the selection procedure in Section 5.2.2.3 to select chromosomes to perform the crossover operation in Section 5.2.2.4.

Step 5: Follow the selection procedure to select chromosomes to perform the mutation operation in Section 5.2.2.5.

Step 6: Compare all offspring, including the chromosomes generated from both the crossover and the mutation operations, with the chromosomes in the population by the fitness values $eval(X_h)$. Retain the best *psize* chromosomes in the population.

Step 7: Determine the best chromosome at each iteration. Repeat Step 4 to Step 7 until *itno* iterations are performed.

The GA for the line assignment problem is proposed in the following. Here, a matrix is used to represent a chromosome in the GA:

$$X_h = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}$$

5.2.2.1 Initialization

The following initialization procedure is used to generate a feasible chromosome for the line assignment problem represented by model M5-1.

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

Step 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 3: Assign the available amount of units to x_{ij}:

$$x_{ij} = \min\left(\frac{t_i}{a_{ij}}, s_j\right).$$

Step 4: Update t_i and s_j:

$t_i = t_i - x_{ij} \times a_{ij}$; $s_j = s_j - x_{ij}$; and delete k from π.

Step 5: Repeat Step 1 to Step 4 until π becomes empty.

The above initialization procedure should be repeated *psize* times to generate *psize* chromosomes for the problem. All chromosomes generated from the above steps are in the form of a matrix. Because the chromosomes satisfy constraint sets (5.2) and (5.3), they are feasible but may not be optimal to the line assignment problem.

5.2.2.2 Evaluation

In a GA, both parent and offspring chromosomes must be evaluated by some measures of fitness. In the line assignment problem, the objective function (5.1) is used to measure the fitness. Let $eval(X_h)$ be the fitness function for chromosome X_h (h = 1, 2, …, *psize*) in the problem; then the fitness function for the problem is

$$eval(X_h) = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}$$

5.2.2.3 Selection

The roulette wheel approach is applied to choose some chromosomes probabilistically instead of deterministically for performing the genetic operations. The selection procedure has been discussed in Section 3.6.4 (in Chapter 3).

5.2.2.4 Crossover Operator

Step 1: Implement the selection procedure twice to choose a pair of chromosomes, $X_1 = (x_{ij}^1)$ and $X_2 = (x_{ij}^2)$, from the population to perform the crossover operation.

Step 2: Create two temporary matrices, $D = (d_{ij})$ and $R = (r_{ij})$, as follows:
$d_{ij} = \lfloor (x_{ij}^1 + x_{ij}^2)/2 \rfloor$ and $r_{ij} = (x_{ij}^1 + x_{ij}^2) \bmod 2$.

Step 3: Divide matrix R into two matrices $R_1 = (r_{ij}^1)$ and $R_2 = (r_{ij}^2)$ so that $R = R_1 + R_2$ and

$$\sum_{j=1}^{n} r_{ij}^1 = \sum_{j=1}^{n} r_{ij}^2 = \frac{1}{2}\sum_{j=1}^{n} r_{ij} \quad \text{for } i = 1, 2, ..., m$$

$$\sum_{i=1}^{m} r_{ij}^1 = \sum_{i=1}^{m} r_{ij}^2 = \frac{1}{2}\sum_{i=1}^{m} r_{ij} \quad \text{for } j = 1, 2, ..., n$$

Step 4: Produce two offspring, X_1' and X_2', as follows:
$X_1' = D + R_1$ and $X_2' = D + R_2$

The offspring generated from the crossover operation are still feasible for M5-1. Note that $x_{ij}^1 + x_{ij}^2 = 2d_{ij} + r_{ij}$, where $r_{ij} = 0$ if $(x_{ij}^1 + x_{ij}^2)$ is even, and $r_{ij} = 1$ if $(x_{ij}^1 + x_{ij}^2)$ is odd. On the other hand, for any column j, $\sum_{i=1}^{m} x_{ij}^1 = \sum_{i=1}^{m} x_{ij}^2$, therefore $2\sum_{i=1}^{m} x_{ij}^1 = 2\sum_{i=1}^{m} d_{ij} + \sum_{i=1}^{m} r_{ij}$, that is, $\sum_{i=1}^{m} x_{ij}^1 = \sum_{i=1}^{m} d_{ij} + \frac{1}{2}\sum_{i=1}^{m} r_{ij} = \sum_{i=1}^{m} X_1$. Therefore, it can be concluded that the offspring (X_1' or X_2') is feasible for M5-1. Besides, a $\sum_{i=1}^{m} r_{ij}$ (j = 1, 2, …, n) should be an even number because both $\sum_{i=1}^{m} x_{ij}^1$ and $\sum_{i=1}^{m} d_{ij}$ are integers; then $\frac{1}{2}\sum_{i=1}^{m} r_{ij}$ has to be an integer.

5.2.2.5 Mutation Operator

Step 1: Implement the selection procedure to select a chromosome.

Step 2: Extract a submatrix Y from the parent matrix by randomly selecting m rows and n columns.

Step 3: Reallocate the submatrix Y. Use the initialization procedure in Section 5.2.2.1 to assign new values to the submatrix so that all constraints are satisfied.

Step 4: Create an offspring by replacing the appropriate elements of the parent matrix with the new elements from the reallocated submatrix Y.

5.2.3 A Numerical Example

The GTP example in Table 5.1 is used to illustrate how the GA works. A pure integer linear programming model for the problem in the form of M5-1 can be formulated as

$$\begin{aligned}\text{Minimize} \quad & 3x_{11} + 6x_{12} + 6x_{13} + 7x_{14} + 4x_{15} \\ & + 6x_{21} + 5x_{22} + 6x_{23} + 15x_{24} + 8x_{25} \\ & + 5x_{31} + 3x_{32} + 10x_{33} + 6x_{34} + 7x_{35} \\ & + 4x_{41} + 4x_{42} + 7x_{43} + 8x_{44} + 9x_{45}\end{aligned}$$

subject to

$$5x_{11} + 2x_{12} + 2x_{13} + 5x_{14} + 6x_{15} \le 48{,}000$$

$$2x_{21} + 7x_{22} + 8x_{23} + 7x_{24} + 6x_{25} \le 48{,}000$$

$$4x_{31} + 2x_{32} + 2x_{33} + x_{34} + 6x_{35} \le 48{,}000$$

$$3x_{41} + 2x_{42} + 4x_{43} + 5x_{44} + 2x_{45} \le 48{,}000$$

$$x_{11} + x_{21} + x_{31} + x_{41} = 12{,}000$$

$$x_{12} + x_{22} + x_{32} + x_{42} = 8{,}000$$

$$x_{13} + x_{23} + x_{33} + x_{43} = 10{,}000$$

$$x_{14} + x_{24} + x_{34} + x_{44} = 8{,}000$$

$$x_{15} + x_{25} + x_{35} + x_{45} = 6{,}000$$

$$x_{ij} \ge 0 \text{ and is a set of integers.} \qquad \text{(M5-2)}$$

As mentioned before, there are seven steps in the GA for solving the line assignment problem and finding the minimum production cost. The first iteration is described in detail to demonstrate how the GA works. The steps are as follows:

Step 1: In this case, *psize* = 25, *itno* = 1000, *cr* = 0.4, and *mr* = 0.3. Therefore, the number of pairs of chromosomes selected to undergo the crossover operation (*i.e.*, *cross*) = 5, and the number of chromosomes selected to undergo the mutation operation (*i.e.*, *mut*) = 8.

Step 2: Generate 25 initial chromosomes using the initialization procedure in Section 5.2.2.1. One initial chromosome is generated as follows:

Step 2.1: Select a random number k from set π, $\pi = \{1, 2, ..., 20\}$. For example, $k = 8$.

Step 2.2: Calculate the corresponding row i and column j by $i = \lfloor (8 - 1) / 5 + 1 \rfloor = 2$; $j = (8 - 1) \bmod 5 + 1 = 3$.

Step 2.3: Assign the available amount of units to x_{23}:

$$x_{23} = \min \left(\frac{t_2}{a_{23}}, s_3 \right) = \min (6000, 10000) = 6000.$$

Step 2.4: Now, update t_2 and s_3 and delete $k = 8$ from set π: $t_2 = 48{,}000 - 6{,}000 \times 8 = 0$; $s_3 = 10{,}000 - 6{,}000 = 4{,}000$.

Repeat the above procedure (Step 2.1 to Step 2.4) until π becomes empty, and the initial chromosome is obtained as

$$X_1 = \begin{bmatrix} 6400 & 8000 & 0 & 0 & 0 \\ 0 & 0 & 6000 & 0 & 0 \\ 0 & 0 & 0 & 8000 & 6000 \\ 5600 & 0 & 4000 & 0 & 0 \end{bmatrix}$$

Similarly, the remaining 24 chromosomes can be generated by following the above procedure.

Step 3: Evaluate the fitness value $eval(X_h)$ for all 25 chromosomes in the population. Here, the fitness value for X_1 is

$$eval(X_1) = 6{,}400 \times 3 + 8{,}000 \times 6 + 6{,}000 \times 6 + 8{,}000 \times 6 + 6{,}000 \times 7 + 5{,}600 \times 4 + 4{,}000 \times 7 = 243{,}600$$

Also the fitness values of all initial chromosomes are as follows:

$eval(X_1)$=243600	$eval(X_6)$=272500	$eval(X_{11})$=248500	$eval(X_{16})$=228800	$eval(X_{21})$=266000
$eval(X_2)$=271200	$eval(X_7)$=276000	$eval(X_{12})$=293200	$eval(X_{17})$=291600	$eval(X_{22})$=264400
$eval(X_3)$=253000	$eval(X_8)$=204000	$eval(X_{13})$=250000	$eval(X_{18})$=274000	$eval(X_{23})$=331600
$eval(X_4)$=235000	$eval(X_9)$=306000	$eval(X_{14})$=277000	$eval(X_{19})$=331600	$eval(X_{24})$=276000
$eval(X_5)$=276000	$eval(X_{10})$=312800	$eval(X_{15})$=266000	$eval(X_{20})$=244400	$eval(X_{25})$=295000

Step 4: Five (*cross* = 5) pairs of chromosomes (or say 10 chromosomes) are selected to perform the crossover operation. To select the chromosomes, the following procedure is implemented:

Step 4.1: Calculate the total fitness for the 25 chromosomes:

$$F = \sum_{h=1}^{25} eval(X_h) = 6{,}788{,}200$$

Step 4.2: Calculate the selection probability p_h for each chromosome:

$$p_h = \frac{F - eval(X_h)}{F \times (psize - 1)}, \qquad h = 1, 2, ..., 25$$

The selection probability p_h for chromosome X_h are

$p_1 = 0.0402$	$p_6 = 0.0400$	$p_{11} = 0.0401$	$p_{16} = 0.0403$	$p_{21} = 0.0400$
$p_2 = 0.0400$	$p_7 = 0.0400$	$p_{12} = 0.0399$	$p_{17} = 0.0399$	$p_{22} = 0.0400$
$p_3 = 0.0401$	$p_8 = 0.0404$	$p_{13} = 0.0401$	$p_{18} = 0.0400$	$p_{23} = 0.0396$
$p_4 = 0.0402$	$p_9 = 0.0398$	$p_{14} = 0.0400$	$p_{19} = 0.0396$	$p_{24} = 0.0400$
$p_5 = 0.0400$	$p_{10} = 0.0397$	$p_{15} = 0.0400$	$p_{20} = 0.0402$	$p_{25} = 0.0399$

Step 4.3: Calculate the cumulative probability q_h for each chromosome:

$$q_h = \sum_{j=1}^{h} p_j, \qquad h = 1, 2, ..., 25$$

The cumulative probability q_h for chromosome X_h are

$q_1 = 0.0402$	$q_6 = 0.2405$	$q_{11} = 0.4405$	$q_{16} = 0.6408$	$q_{21} = 0.8405$
$q_2 = 0.0802$	$q_7 = 0.2805$	$q_{12} = 0.4804$	$q_{17} = 0.6807$	$q_{22} = 0.8805$
$q_3 = 0.1203$	$q_8 = 0.3209$	$q_{13} = 0.5205$	$q_{18} = 0.7207$	$q_{23} = 0.9202$
$q_4 = 0.1605$	$q_9 = 0.3607$	$q_{14} = 0.5605$	$q_{19} = 0.7603$	$q_{24} = 0.9601$
$q_5 = 0.2005$	$q_{10} = 0.4004$	$q_{15} = 0.6005$	$q_{20} = 0.8005$	$q_{25} = 1.0000$

Step 4.4: If the first two random numbers are 0.0218 and 0.8782, then X_1 and X_{22} are selected as the first pair of chromosomes (totally five pairs) to perform the crossover operation (refer to Section 5.2.2.4). Here,

$$X_1 = \begin{bmatrix} 6400 & 8000 & 0 & 0 & 0 \\ 0 & 0 & 6000 & 0 & 0 \\ 0 & 0 & 0 & 8000 & 6000 \\ 5600 & 0 & 4000 & 0 & 0 \end{bmatrix}$$

$$X_{22} = \begin{bmatrix} 0 & 0 & 10000 & 5600 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 12000 & 0 & 0 & 0 & 0 \\ 0 & 8000 & 0 & 2400 & 6000 \end{bmatrix}$$

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

$$D = \begin{bmatrix} 3200 & 4000 & 5000 & 2800 & 0 \\ 0 & 0 & 3000 & 0 & 0 \\ 6000 & 0 & 0 & 4000 & 3000 \\ 2800 & 4000 & 2000 & 1200 & 3000 \end{bmatrix}$$

$$R = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

In this case, note that there is no "1" in matrix R. Therefore, two offspring X_1'and X_{22}' are the same:

$$X_1' \text{ and } X_{22}' = \begin{bmatrix} 3200 & 4000 & 5000 & 2800 & 0 \\ 0 & 0 & 3000 & 0 & 0 \\ 6000 & 0 & 0 & 4000 & 3000 \\ 2800 & 4000 & 2000 & 1200 & 3000 \end{bmatrix}$$

Step 5: Eight chromosomes are selected to perform the mutation operation. Assume X_1 is one of the chromosomes selected as a parent for mutation (refer to Section 5.2.2.5); then,

$$X_1 = \begin{bmatrix} 6400 & 8000 & 0 & 0 & 0 \\ 0 & 0 & 6000 & 0 & 0 \\ 0 & 0 & 0 & 8000 & 6000 \\ 5600 & 0 & 4000 & 0 & 0 \end{bmatrix}$$

Randomly select rows 1 and 4, columns 1 and 3. The corresponding submatrix Y and the reallocated submatrix are

Corresponding submatrix Y Reallocated submatrix

$$\begin{bmatrix} 6400 & 0 \\ 5600 & 4000 \end{bmatrix} \qquad \begin{bmatrix} 4800 & 4000 \\ 7200 & 0 \end{bmatrix}$$

Replacing the reallocated submatrix into X_1, the new offspring is

$$X_1'' = \begin{bmatrix} 4800 & 8000 & 4000 & 0 & 0 \\ 0 & 0 & 6000 & 0 & 0 \\ 0 & 0 & 0 & 8000 & 6000 \\ 7200 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Step 6: Calculate the fitness values for all offspring (totally 18), and compare them with the parents in the population. In this case, the fitness values of the three offspring are $X_1' = X_{22}' = 254{,}000$, and $X_1'' = 241{,}200$. So, X_1', X_{22}', and X_1'' will replace parents X_{19}, X_{23}, and X_{10} because these three parents are the worst in the population.

Step 7: The best chromosome for the first iteration is X_8 because it has the smallest fitness value. Repeat Step 4 to Step 7 until 1,000 iterations are performed.

After 1,000 iterations, the best solution is shown in Table 5.2 with the production cost of 203,200. Although the GA cannot guarantee that the optimal solution can be found, the best solution obtained by the GA for this specific problem is optimal. The optimal integer solution is generated from a commercial package, CPLEX, by solving the pure integer linear programming model, M5-2.

Table 5.2. The best chromosome after 1,000 iterations for M5-2

$x_{11} = 800$	$x_{12} = 0$	$x_{13} = 4{,}000$	$x_{14} = 0$	$x_{15} = 6{,}000$
$x_{21} = 0$	$x_{22} = 0$	$x_{23} = 6{,}000$	$x_{24} = 0$	$x_{25} = 0$
$x_{31} = 0$	$x_{32} = 8{,}000$	$x_{33} = 0$	$x_{34} = 8{,}000$	$x_{35} = 0$
$x_{41} = 11{,}200$	$x_{42} = 0$	$x_{43} = 0$	$x_{44} = 0$	$x_{45} = 0$
Total production cost = 203,200				

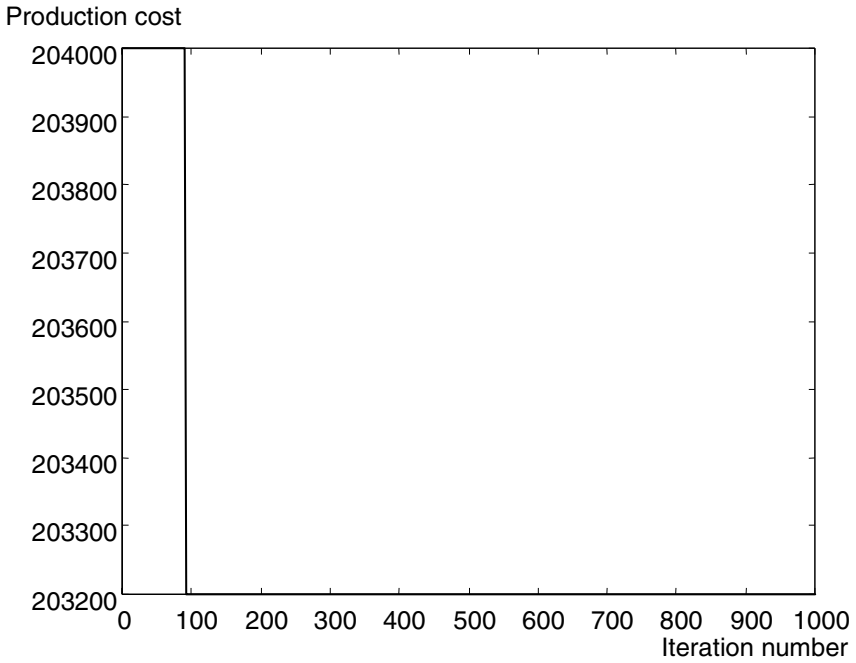

Figure 5.4. The minimum production cost at each iteration

The performance of the GA is shown in Figure 5.4. From the graph, it can be noticed that the objective value drops sharply at the first 90 iterations. Because the population size (*psize*) is small, only 25, the GA can only produce some not-so-good chromosomes at the beginning. Later, the GA generates good offspring quickly from those highly fit parents. This phenomenon is called rapid convergence. When the objective reaches the best solution (*i.e.*, 203,200), the curve levels off because the best solution is already the optimal solution.

5.3 The Component Allocation Problem

Here, the last element of the PCB assembly planning system is discussed. There were a number of researchers focusing on the component allocation problem. Ben-Arieh and Dror (1990) studied the problem of assigning components to insertion machines. An integer linear programming model was formulated for the problem to maximize the output. The problem was divided into two cases: the same component can be assigned to only a single machine, and the same component can be assigned to more than one machine. Two heuristic approaches were applied to solve the problem.

Crama *et al.* (1990) focused on the assembly of a single type of PCB in a line of PAP machines. A mixed integer linear programming model was formulated for the component allocation problem to minimize the workload of the bottleneck machine. A heuristic approach was developed to solve the problem.

Brandeau and Billington (1991) studied the component allocation problem for a set of capacitated insertion machines. The authors formulated the problem as a mixed integer linear programming model to minimize the total setup and processing costs for assembling all boards. It was assumed that the setup and the processing costs, by machine, were the same over all boards and components. Two and four different heuristic approaches were developed to solve the problem with single and multiple machines, respectively.

Klincewicz and Rajan (1994) formulated an integer linear programming model for the component allocation problem. The problem was to determine which components should be assigned to each robotic work cell to minimize the number of visits by circuit boards to work cells. Two heuristic approaches, based on the so-called greedy random adaptive search procedures (GRASP), were adopted to solve the problem. With GRASP, the local search heuristic was replicated many times with different starting points. The best result was then kept as the solution.

Günther *et al.* (1996) presented a mixed integer linear programming model for two PCB assembly problems. The main objective was to minimize the setup time. The problem first concerned the grouping of jobs for processing at the same assembly station. Following that, the problem was to allocate components among various identical assembly stations while taking production time and number of feeders available into consideration. A heuristic solution procedure was developed to solve the problem.

Ammons *et al.* (1997) considered the problem of assigning component types to multiple nonidentical assembly machines. A mixed integer linear programming model was formulated for the problem to balance the workload. Two alternative solution approaches were presented. First, a list-processing-based heuristic was used to solve the problem in the PTH line. Second, a linear-programming-based branch-and-bound heuristic was used to solve the problem in the SMT line.

Gronalt *et al.* (1997) formulated a mixed integer linear programming model for the component allocation problem to minimize total production time. A heuristic solution procedure was developed to solve the problem. First, the component setup was determined for a given sequence of board types to be processed on a single placement machine by applying a modification of the so-called “keep component

needed soonest" policy. Next, components were assigned to feeders of the placement machine.

Lapierre *et al.* (2000) studied the problem of allocating and arranging components on several PAP machines, while considering a different assembly time if components were located at different feeders. A mixed integer linear programming model was formulated for the problem to balance the workload among the placement machines. The Lagrangian relaxation algorithm was used to transfer the constraints into the objective function and to generate a lower bound for the optimal solution.

Ji *et al.* (2001) studied the problem of allocating components to a PCB assembly line, which had several nonidentical placement machines in series. The authors pointed out that the unit assembly times were the same for the same component with the same machine, but assembly times for different machines in the line were not the same. A minimax type integer linear programming model to minimize the cycle time of the assembly line was formulated. The model was proved to be NP-complete, so a GA was used to solve the problem.

Sze *et al.* (2001) formulated several integer linear programming models to obtain the best assignment of components to several nonidentical placement machines in a PCB assembly line. The placement times by different machines varied for the same type of components. The objective of the models was to minimize the cycle time of the assembly line. A numerical example was provided to illustrate the models and was solved by a commercial package.

Wan and Ji (2001) discussed the component allocation problem for multiple nonidentical assembly machines in an SMT line. An integer linear programming model was formulated. The objective was to minimize the cycle time of the assembly line. A tabu search heuristic was used to solve the problem.

Although there were a number of research projects on the component allocation problem (*i.e.*, Level 2), only a few of them focused on the problem for multiple machines. Moreover, these researchers assumed that the component processing times are identical even for different types of placement machines. But actually, different machines have different unit assembly times for the same kind of surface mount components. Therefore, the component allocation problem has not been studied thoroughly yet.

After board types have been assigned to production lines or the line assignment problem has been solved, components on the board should be grouped and allocated to appropriate machines to achieve better line performance in terms of cycle time. Due to the various configurations of machines, different machines have different unit assembly times for the same type of surface mount components. Actually, there is an occasion that a machine cannot handle a particular type of component. In this case, the unit assembly time should be assigned to be infinite (∞). Figure 5.5 shows n component types to be allocated to four placement machines in a line.

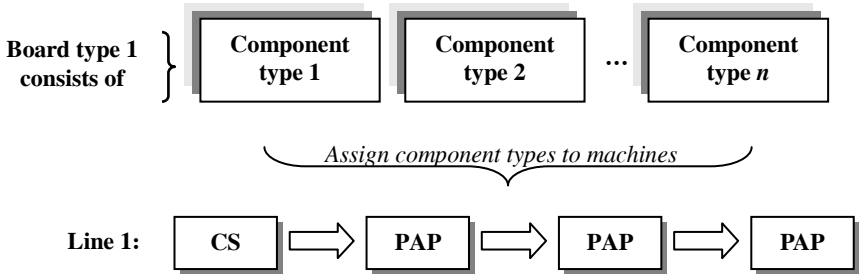

Figure 5.5. An example of the component allocation problem

5.3.1 A Mathematical Model

Suppose that m nonidentical placement machines are in an SMT production line and a board with n types of surface mount components is going to be assembled on that line. It requires t_{ij} time per unit to place if component type j is assigned to machine i. By introducing s_i to denote the setup time for machine i and c_j for the quantity of component type j, respectively, the above component allocation problem with the objective of minimizing the cycle time can be formulated as

$$\text{Minimize } z = \max\left(s_i + \sum_{j=1}^{n} t_{ij}x_{ij}\right) \qquad \text{for } i = 1, 2, \ldots, m \tag{5.4}$$

subject to

$$\sum_{i=1}^{m} x_{ij} = c_j \qquad \text{for } j = 1, 2, \ldots, n \tag{5.5}$$

$$x_{ij} \geq 0 \text{ and is a set of integers.} \tag{M5-3}$$

The decision variable x_{ij} is introduced to indicate the number of component type j to be assembled on machine i. The objective function (5.4) is to minimize the assembly time for the machine with the largest assembly time, including the machine setup time, that is, the cycle time. As usual, the cycle time is defined as the maximum assembly time among all the placement machines in a line. Constraint set (5.5) is to guarantee that all of the components will be assembled.

Consider seven types of components that must be allocated to three different placement machines, as shown in Table 5.3. The northeast corner in cell (i, j) indicates the unit assembly time t_{ij} (The time unit is 0.1 second in the table). For example, in cell (1, 1), t_{11} is 0.3 second. If a machine cannot handle a particular type of component, the unit assembly time is assigned to be infinite (∞).

Table 5.3. A component allocation problem

Machine i	Component Type j							Setup Time s_i
	1	2	3	4	5	6	7	
1	3 x_{11}	7 x_{12}	7 x_{13}	5 x_{14}	∞ x_{15}	∞ x_{16}	∞ x_{17}	110
2	7 x_{21}	12 x_{22}	15 x_{23}	16 x_{24}	15 x_{25}	15 x_{26}	21 x_{27}	147
3	23 x_{31}	38 x_{32}	35 x_{33}	35 x_{34}	27 x_{35}	33 x_{36}	43 x_{37}	147
No. of c_j	324	37	12	5	7	5	4	

Formulation M5-3 is a minimax type integer linear programming model. The complexity of a minimax type problem was discussed in Yu and Kouvelis (1993). Yu and Kouvelis proved that the minimax assignment problem is NP-hard. However, the data structure in M5-3 is similar to but not exactly the same as the minimax assignment problem. Actually, M5-3 is more difficult than the minimax assignment problem in two aspects. First, the decision variables x_{ij} in the minimax assignment problem must be either 0 or 1, and the number of jobs should be equal to the number of tasks (*i.e.*, $n = m$). However, the decision variables can be any nonnegative integer value rather than just 0 or 1, and $n \neq m$ in M5-3. Second, the objective of the minimax assignment problem is to minimize only the maximum x_{ij}, whereas the objective of M5-3 is to minimize the maximum summation of $t_{ij}x_{ij}$, let alone the component s_{ij}. Therefore, M5-3 is a typical general integer linear programming model, and it belongs to an NP-complete problem because Papadimitriou concluded that a general integer linear programming model is NP-complete (Papadimitriou, 1981). As a consequence, it is necessary to adopt a heuristic algorithm to solve the problem.

5.3.2 A Genetic Algorithm

As for the line assignment problem, a GA is applied to deal with the component allocation problem in Level 2 or M5-3, too. Actually, the general structure and the procedure of the GA for the component allocation problem are the same as those for the line assignment problem, as discussed in Section 5.2.2. The major differences are the way of initializing a feasible chromosome or initialization and the method of evaluating a chromosome or evaluation. Therefore, only the initialization and the evaluation are discussed in the following. For the remaining elements, such as the selection, the crossover operator, and the mutation operator, refer to Section 5.2.2.

5.3.2.1 Initialization

The procedure for generating a feasible initial chromosome for the component allocation problem represented by M5-3 is as follows:

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

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

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

Step 3: 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 4: Assign the available amount of units to x_{ij}: $x_{ij} = \min(b_i, c_j)$.

Step 5: Update b_i and c_j: $b_i = b_i - x_{ij}$; $c_j = c_j - x_{ij}$; and delete k from π.

Step 6: Repeat Step 2 to Step 5 until π becomes empty.

The above initialization procedure should be repeated *psize* times to generate *psize* chromosomes for the problem. All chromosomes generated from the above steps are in the form of a matrix. The chromosomes satisfy the constraint sets (5.5), so they are feasible but may not be optimal to the component allocation problem.

5.3.2.2 Evaluation

In the GA, the objective function of M5-3 is used for evaluation for the component allocation problem. Let $eval(X_h)$ be the fitness function for chromosome X_h in the problem; then the fitness function for the problem is

$$eval(X_h) = \max\left(s_i + \sum_{j=1}^{n} t_{ij} x_{ij} \right) \qquad \text{for } i = 1, 2, \ldots, m$$

5.3.3 A Numerical Example

The example in Table 5.3 is used to illustrate how the GA works. A minimax type integer linear programming model for the problem can be formulated as

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

subject to

$$x_{11} + x_{21} + x_{31} = 324$$

$$x_{12} + x_{22} + x_{32} = 37$$

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

$$x_{14} + x_{24} + x_{34} = 5$$

$$x_{15} + x_{25} + x_{35} = 7$$

$$x_{16} + x_{26} + x_{36} = 5$$

$$x_{17} + x_{27} + x_{37} = 4$$

$$x_{ij} \geq 0 \text{ and is a set of integers.} \qquad \text{(M5-4)}$$

There are seven steps in the GA for solving the component allocation problem and finding the minimum cycle time. The first iteration is described in the following to demonstrate how the GA works.

Step 1: In this case, *psize* = 25, *itno* = 1000, *cr* = 0.4, and *mr* = 0.3. Therefore, *cross* = 5, and *mut* = 8.

Step 2: Generate 25 initial chromosomes by following the initialization procedure in Section 5.3.2.1. For instance, one initial chromosome is generated as follows:

Step 2.1: Assign an integer number to b_i randomly, say

$b_i = \{374, 4, 16\}$, so 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 374 components allocated to machine 1, and so on.

Step 2.2: Select a random number k from set π, $\pi = \{1, 2, ..., 21\}$. For example, $k = 4$.

Step 2.3: Calculate the corresponding row i and column j by $i = \lfloor (4 - 1) / 7 + 1 \rfloor = 1$; $j = (4 - 1) \bmod 7 + 1 = 4$.

Step 2.4: Assign the available amount of units to x_{14}: $x_{14} = \min(b_1, c_4) = \min(374, 5) = 5$.

Step 2.5: Now, update b_1 and c_4 and delete $k = 4$ from set π: $b_1 = 374 - 5 = 369$; $c_4 = 5 - 5 = 0$.

Repeat the above procedure (Step 2.2 to Step 2.5) until π becomes empty, and the initial chromosome is obtained as

$$X_1 = \begin{bmatrix} 324 & 37 & 0 & 5 & 7 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 12 & 0 & 0 & 0 & 4 \end{bmatrix}$$

Similarly, the remaining 24 chromosomes can be generated by following the above procedure.

Step 3: Evaluate the fitness value $eval(X_h)$ for all 25 chromosomes in the population. Here, the fitness value for X_1 is

$$eval(X_1) = \max \begin{pmatrix} 721256 + 110 \\ 60 + 147 \\ 592 + 147 \end{pmatrix} = 721{,}366$$

Also the fitness values of all initial chromosomes are as follows:

$eval(X_1)$=721366	$eval(X_6)$=1287	$eval(X_{11})$=360596	$eval(X_{16})$=450164	$eval(X_{21})$=810525
$eval(X_2)$=270110	$eval(X_7)$=990654	$eval(X_{12})$=361128	$eval(X_{17})$=1081270	$eval(X_{22})$=631335
$eval(X_3)$=990972	$eval(X_8)$=9751	$eval(X_{13})$=4051	$eval(X_{18})$=810776	$eval(X_{23})$=450837
$eval(X_4)$=810713	$eval(X_9)$=5846	$eval(X_{14})$=450164	$eval(X_{19})$=1261002	$eval(X_{24})$=361341
$eval(X_5)$=811303	$eval(X_{10})$=450180	$eval(X_{15})$=1351351	$eval(X_{20})$=630801	$eval(X_{25})$=2432

Step 4: Five (*cross* = 5) pairs of chromosomes (or say 10 chromosomes) are selected to perform the crossover operation. The procedure of the roulette wheel approach for selection is as follows:

Step 4.1: Calculate the total fitness for the 25 chromosomes:

$$F = \sum_{h=1}^{25} eval(X_h) = 14{,}079{,}955$$

Step 4.2: Calculate the selection probability p_h for each chromosome:

$$p_h = \frac{F - eval(X_h)}{F \times (psize - 1)}, \qquad h = 1, 2, ..., 25$$

The selection probability p_h for chromosome X_h are

$p_1 = 0.0395$	$p_6 = 0.0417$	$p_{11} = 0.0406$	$p_{16} = 0.0403$	$p_{21} = 0.0393$
$p_2 = 0.0409$	$p_7 = 0.0387$	$p_{12} = 0.0406$	$p_{17} = 0.0385$	$p_{22} = 0.0398$
$p_3 = 0.0387$	$p_8 = 0.0416$	$p_{13} = 0.0417$	$p_{18} = 0.0393$	$p_{23} = 0.0403$
$p_4 = 0.0393$	$p_9 = 0.0416$	$p_{14} = 0.0403$	$p_{19} = 0.0379$	$p_{24} = 0.0406$
$p_5 = 0.0393$	$p_{10} = 0.0403$	$p_{15} = 0.0377$	$p_{20} = 0.0398$	$p_{25} = 0.0417$

Step 4.3: Calculate the cumulative probability q_h for each chromosome:

$$q_h = \sum_{j=1}^{h} p_j, \qquad h = 1, 2, ..., 25$$

The cumulative probability q_h for chromosome X_h are

$q_1 = 0.0395$	$q_6 = 0.2394$	$q_{11} = 0.4422$	$q_{16} = 0.6428$	$q_{21} = 0.8376$
$q_2 = 0.0804$	$q_7 = 0.2781$	$q_{12} = 0.4828$	$q_{17} = 0.6813$	$q_{22} = 0.8774$
$q_3 = 0.1191$	$q_8 = 0.3197$	$q_{13} = 0.5245$	$q_{18} = 0.7206$	$q_{23} = 0.9177$
$q_4 = 0.1584$	$q_9 = 0.3613$	$q_{14} = 0.5648$	$q_{19} = 0.7585$	$q_{24} = 0.9583$
$q_5 = 0.1977$	$q_{10} = 0.4016$	$q_{15} = 0.6025$	$q_{20} = 0.7983$	$q_{25} = 1.0000$

Step 4.4: Generate a number r, ranging from 0 to 1, 10 times randomly to select 10 chromosomes from the population. If the first two random numbers are 0.8912 and 0.4881, for example, then X_{23} and X_{13} are selected as the first pair of chromosomes to perform the crossover operation. Here,

$$X_{23} = \begin{bmatrix} 234 & 0 & 0 & 5 & 0 & 5 & 0 \\ 40 & 37 & 0 & 0 & 7 & 0 & 0 \\ 50 & 0 & 12 & 0 & 0 & 0 & 4 \end{bmatrix}$$

$$X_{13} = \begin{bmatrix} 146 & 0 & 0 & 0 & 0 & 0 & 0 \\ 77 & 0 & 12 & 0 & 7 & 5 & 4 \\ 101 & 37 & 0 & 5 & 0 & 0 & 0 \end{bmatrix}$$

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

$$D = \begin{bmatrix} 190 & 0 & 0 & 2 & 0 & 2 & 0 \\ 58 & 18 & 6 & 0 & 7 & 2 & 2 \\ 75 & 18 & 6 & 2 & 0 & 0 & 2 \end{bmatrix}$$

$$R = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Then divide R into R_1 and R_2 as follows:

$$R_1 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

R_1 and R_2 must have the same total number of 1's in each column. Therefore, two offspring X_{23}'and X_{13}' are

$$X_{23}' = \begin{bmatrix} 190 & 0 & 0 & 3 & 0 & 2 & 0 \\ 58 & 19 & 6 & 0 & 7 & 3 & 2 \\ 76 & 18 & 6 & 2 & 0 & 0 & 2 \end{bmatrix}$$

$$X_{13}' = \begin{bmatrix} 190 & 0 & 0 & 2 & 0 & 3 & 0 \\ 59 & 18 & 6 & 0 & 7 & 2 & 2 \\ 75 & 19 & 6 & 3 & 0 & 0 & 2 \end{bmatrix}$$

Step 5: Assume that X_{23} is one of the eight chromosomes selected as a parent to perform the mutation operation; then,

$$X_{23} = \begin{bmatrix} 234 & 0 & 0 & 5 & 0 & 5 & 0 \\ 40 & 37 & 0 & 0 & 7 & 0 & 0 \\ 50 & 0 & 12 & 0 & 0 & 0 & 4 \end{bmatrix}$$

Randomly select rows 1, 2, and 3, columns 2 and 3. The corresponding submatrix Y and the reallocated submatrix are

Corresponding submatrix Y Reallocated submatrix

$$\begin{bmatrix} 0 & 0 \\ 37 & 0 \\ 0 & 12 \end{bmatrix} \qquad \begin{bmatrix} 37 & 4 \\ 0 & 5 \\ 0 & 3 \end{bmatrix}$$

Replacing the reallocated submatrix into X_{23}, the new offspring is

$$X_{23}'' = \begin{bmatrix} 234 & 37 & 4 & 5 & 0 & 5 & 0 \\ 40 & 0 & 5 & 0 & 7 & 0 & 0 \\ 50 & 0 & 3 & 0 & 0 & 0 & 4 \end{bmatrix}$$

Step 6: Calculate the fitness values for all offspring (totally 18), and compare them with the parents in the population. In this case, the fitness values of the three offspring are X_{23}' = 180,695, X_{13}' = 270,690, and X_{23}'' = 451,124. So, X_{23}', X_{13}', and X_{23}'' will replace parents X_{15}, X_{19}, and X_{17} because these three parents are the worst in the population.

Step 7: The best chromosome for the first iteration is X_6 because it has the smallest fitness value. Repeat Step 4 to Step 7 until 1,000 iterations are performed.

After 1,000 iterations, the best solution is shown in Table 5.4 with the cycle time of 978 (*i.e.*, 97.8 seconds). The solution obtained for this specific problem is good because it has a less than 1% error. The optimal integer solution shown in Table 5.5 is generated from a commercial package, CPLEX, by solving the minimax type integer linear programming model, M5-4.

Table 5.4. The best chromosome after 1,000 iterations for M5-4

$x_{11} = 276$	$x_{12} = 0$	$x_{13} = 4$	$x_{14} = 2$	$x_{15} = 0$	$x_{16} = 0$	$x_{17} = 0$
$x_{21} = 45$	$x_{22} = 37$	$x_{23} = 1$	$x_{24} = 0$	$x_{25} = 0$	$x_{26} = 1$	$x_{27} = 2$
$x_{31} = 3$	$x_{32} = 0$	$x_{33} = 7$	$x_{34} = 3$	$x_{35} = 7$	$x_{36} = 4$	$x_{37} = 2$
Total cycle time = 97.8						

Table 5.5. The optimal solution

$x_{11} = 274$	$x_{12} = 0$	$x_{13} = 2$	$x_{14} = 5$	$x_{15} = 0$	$x_{16} = 0$	$x_{17} = 0$
$x_{21} = 50$	$x_{22} = 37$	$x_{23} = 1$	$x_{24} = 0$	$x_{25} = 1$	$x_{26} = 0$	$x_{27} = 0$
$x_{31} = 0$	$x_{32} = 0$	$x_{33} = 9$	$x_{34} = 0$	$x_{35} = 6$	$x_{36} = 5$	$x_{37} = 4$
Total cycle time = 97.1						

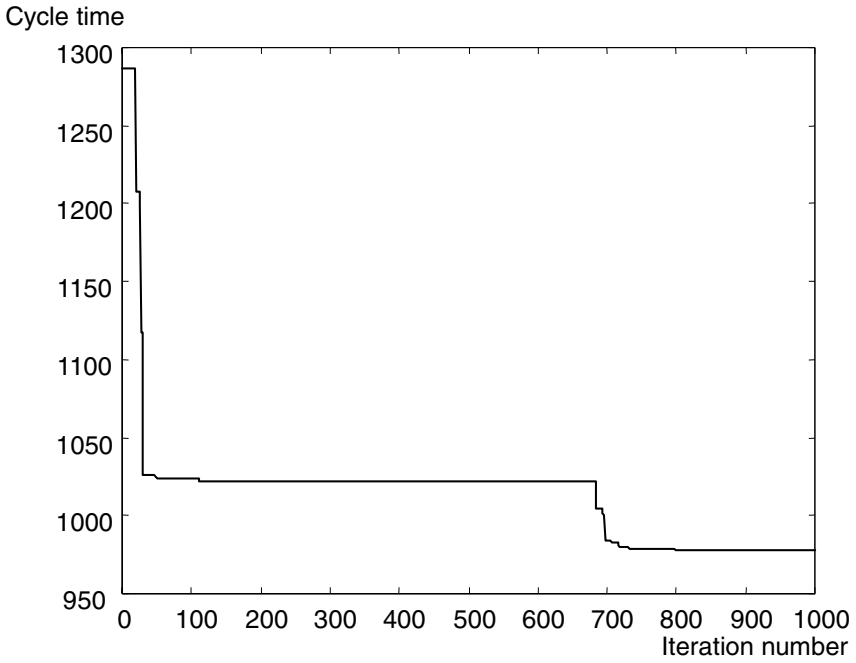

Figure 5.6. The minimum cycle time at each iteration

The performance of the GA is shown in Figure 5.6. From the graph, it can be noticed that the solution improves significantly at the first 30 iterations. After that, the curve becomes horizontal until around the 700th iteration. Finally, the curve levels off after the GA obtains the best solution of 978 (*i.e.*, 97.8 seconds) at about the 800th iteration.

5.4 Summary

The line assignment and the component allocation problems are tackled in this chapter. To optimize each of the PCB assembly problems, both mathematical modeling and heuristic methods have been applied. Some observations concerning these problems are made:

1. None of the researchers have focused on the line assignment problem arising in a high-volume production environment.
2. The generalized transportation problem was formulated as the line assignment problem to minimize the total production cost.
3. Although there were a number of research projects on the component allocation problem, only a few of them focused on the problem for multiple machines.

4. Besides, these researchers assumed that component processing times are identical even for different types of placement machines. This does not coincide with the real-life situation.
5. Genetic algorithms developed to solve the line assignment and the component allocation problems are different from those for the integrated problems because the matrix instead of the path representation is used.

The next chapter will develop a prototype of the "Printed Circuit Board Assembly Planning System" (PCBAPS).

6

A Prototype of the Printed Circuit Board Assembly Planning System (PCBAPS)

6.1 The PCBAPS Framework

A prototype of the "Printed Circuit Board Assembly Planning System" (PCBAPS) is developed in this chapter. As mentioned in Section 5.1 (in Chapter 5), the PCBAPS includes three levels of PCB assembly problems. Level 1 is the line assignment problem, and Level 2 is the component allocation problem. Level 3 consists of the integrated problems (*i.e.*, solving the component sequencing and the feeder arrangement problems simultaneously) for the PAP and the CS machines. The algorithms for solving the problems on all levels have been presented thoroughly in the previous chapters.

The PCBAPS is a program written by a computer language, Matlab. It is developed with some interfaces between a user and the system. The PCBAPS can be launched after typing the "PCBAPS" in the Matlab Command Window.

6.2 A Guide to Using the PCBAPS

The overall procedure for using the PCBAPS can be simply described as follows:

1. Select a specific problem to be solved.
2. Input the data for the problem to be solved.
3. Input the GA parameters.
4. Execute the programs.
5. The PCBAPS uses the GAs developed in Sections 5.2.2 and 5.3.2 to solve the line assignment problem and the component allocation problem, respectively. Besides, the system uses the HGAs in Sections 3.6 and 4.6 to solve the integrated problems for the PAP machine and the CS machine, respectively.
6. Obtain the results of the problems, including the solutions in Tables 5.2 and 5.4 and the performance of the system in Figures 5.4 and 5.6.

6.3 Graphical User Interfaces

After typing "PCBAPS" in the Matlab Command Window, a graphical user interface (GUI) figure will appear, as shown in Figure 6.1. It is for the users to select a specific problem to be solved. For example, click the "LAP" button if the users want to solve the line assignment problem. "CAP" refers to the component allocation problem, whereas "CSP and FAP" represent the component sequencing problem and the feeder arrangement problem (*i.e.*, the integrated problem).

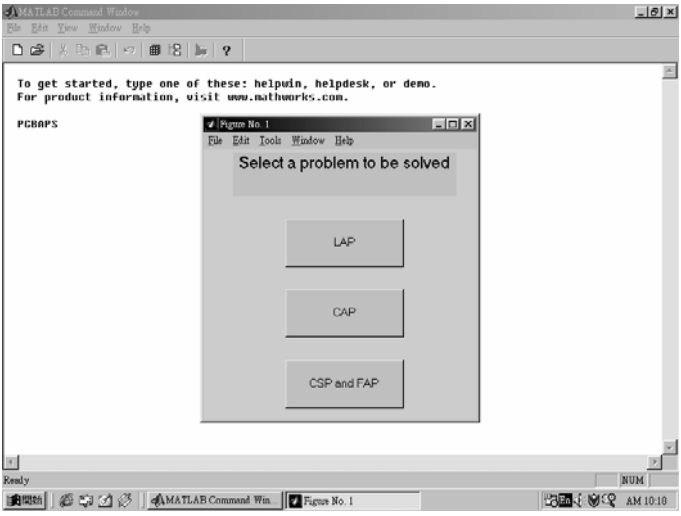

Figure 6.1. Problem input interface

As mentioned earlier, two types of placement machines are considered in the integrated problem. Therefore, another GUI figure, as shown in Figure 6.2, appears when the users click the "CSP and FAP" button. It is mainly for the users to select a specific type of placement machine to be considered, including the pick-and-place machine as well as the chip shooter machine.

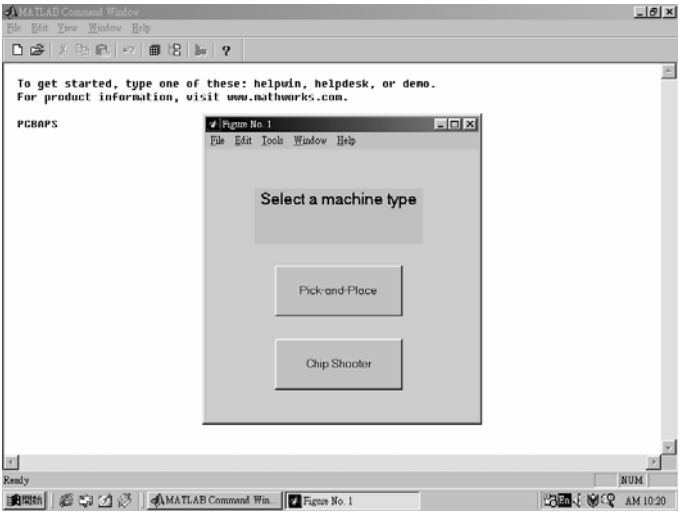

Figure 6.2. Machine input interface

After selecting a problem to be solved, the users have to prepare the data to input to the system. Actually, there are two ways to input data, as illustrated in Figure 6.3. One is to input the data by screen input. The second type is to retrieve the data from a data file. Generally, a problem with the same sets of data may need to be solved several times. So, this type of input method can save users a lot of time for inputting the same sets of data again and again.

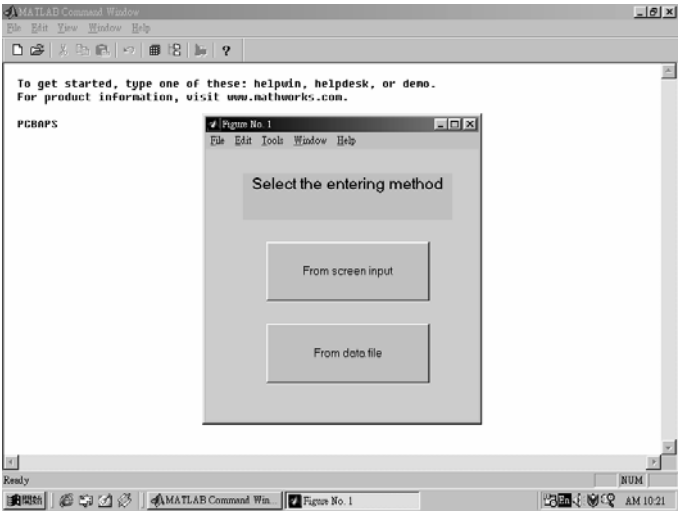

Figure 6.3. Data input interface

Then, it is necessary for users to input the GA parameters to the system, as shown in Figure 6.4. The parameters include the population size (*psize*), the number of iterations (*itno*), the crossover rate (*cr*), and the mutation rate (*mr*). In general, it is very difficult to find the best setting of the GA parameters. A good setting can be achieved using the trial-and-error method. If the setting of the GA parameters is confirmed, users can click the "Run" button. After pressing it, the system uses the GA or the HGA to solve the problem to be selected.

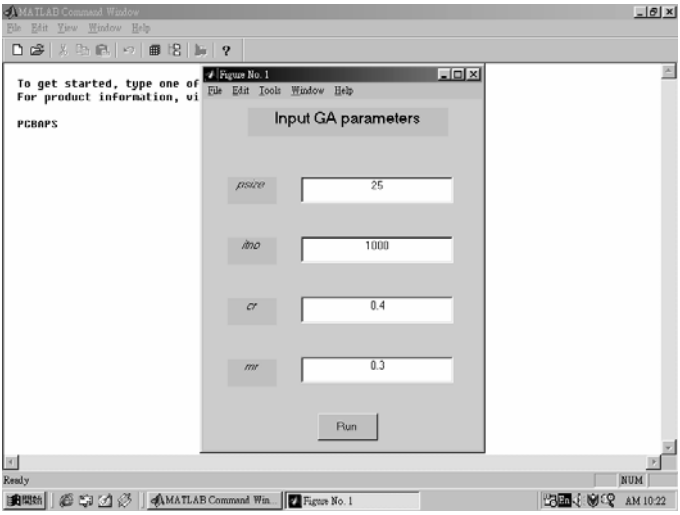

Figure 6.4. GA parameters input interface

The system terminates if the number of iterations is conducted. Two types of output will be generated. The first type is the best solution of the problem, like those in Tables 5.2 and 5.4. The second type is the performance of the system, like those in Figures 5.4 and 5.6.

6.4 Summary

A prototype of the "Printed Circuit Board Assembly Planning System" (PCBAPS) has been developed in this chapter. The PCBAPS uses genetic algorithms to solve the line assignment problem, the component allocation problem, and the integrated problems for both types of placement machines. Some observations are made after the system has been developed:

1. Genetic algorithms are so flexible that they can be applied to solve three different types of PCB assembly problems effectively.
2. The problems on three levels are simple to describe but are very complex to solve. However, with the input by users, the PCBAPS can perform planning tasks quickly.

3. The PCBAPS provides users friendly interfaces and aids the process planner in determining the assignment of product types to assembly lines, the allocation of component types to placement machines in an assembly line, and the sequence of component placements and the feeder arrangement.

References

Ahuja RK, Orlin JB, Tiwari A (2000) A greedy genetic algorithm for the quadratic assignment problem. Computers & Operations Research 27:917–934

Altinkemer K, Kazaz B, Köksalan M, Moskowitz H (2000) Optimization of printed circuit board manufacturing: integrated modeling and algorithms. European Journal of Operational Research 124:409–421

Ammons JC, Carlyle M, Cranmer L, DePuy G, Ellis K, McGinnis LF, Tovey CA, Xu H (1997) Component allocation to balance workload in printed circuit card assembly systems. IIE Transactions 29:265–275

Balachandran V, Thompson GL (1975a) An operator theory of parametric programming for the generalized transportation problem: I. basic theory. Naval Research Logistics 22:79–100

Balachandran V, Thompson GL (1975b) An operator theory of parametric programming for the generalized transportation problem: II. rim, cost and bound operators. Naval Research Logistics 22:101–126

Balachandran V, Thompson GL (1975c) An operator theory of parametric programming for the generalized transportation problem: III. weight operators. Naval Research Logistics 22:297–316

Balachandran V, Thompson GL (1975d) An operator theory of parametric programming for the generalized transportation problem: IV. global operators. Naval Research Logistics 22:317–340

Balakrishnan A, Vanderbeck F (1999) A tactical planning model for mixed-model electronics assembly operations. Operations Research 47:395–409

Balas E, Ivanescu PL (1964) On the generalized transportation problem. Management Science 11:188–202

Balas E (1966) The dual method for the generalized transportation problem. Management Science 12:555–568

Ball MO, Magazine MJ (1988) Sequencing of insertions in printed circuit board assembly. Operations Research 36:192–201

Bard JF, Clayton RW, Feo TA (1994) Machine setup and component placement in printed circuit board assembly. International Journal of Flexible Manufacturing Systems 6:5–31

Ben-Arieh D, Dror M (1990) Part assignment to electronic insertion machines: two machine case. International Journal of Production Research 28:1317–1327

Brandeau ML, Billington CA (1991) Design of manufacturing cells: operation assignment in printed circuit board manufacturing. Journal of Intelligent Manufacturing 2:95–106

Broad K, Mason A, Rönnqvist M, Frater M (1996) Optimal robotic component placement. Journal of the Operational Research Society 47:1343–1354

Burkard RE, Karisch SE, Rendl F (1991) QAPLIB – a quadratic assignment problem library. European Journal of Operational Research 55:115–119
Carter MW, Price CC (2001) Operations Research: A Practical Introduction. CRC Press, Boca Raton
Castillo E, Conejo AJ, Pedregal P, Garciá R, Alguacil N (2002) Building and Solving Mathematical Programming Models in Engineering and Science. Wiley, New York
Chan D, Mercier D (1989) IC insertion: an application of the travelling salesman problem. International Journal of Production Research 27:1837–1841
Crama Y, Kolen AWJ, Oerlemans AG, Spieksma FCR (1990) Throughput rate optimization in the automated assembly of printed circuit boards. Annals of Operations Research 26:455–480
Crama Y, Flippo O, Klundert JVD, Spieksma FCR (1997) The assembly of printed circuit boards: a case with multiple machines and multiple board types. European Journal of Operational Research 98:457–472
Csaszar P, Tirpak TM, Nelson PC (2000) Optimization of a high-speed placement machine using tabu search algorithms. Annals of Operations Research 96:125–147
Davis L (1991) Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York
Deo S, Javadpour R, Knapp GM (2002) Multiple setup PCB assembly planning using genetic algorithms. Computers & Industrial Engineering 42:1–16
De Souza R, Wu LJ (1994) CPS: a productivity tool for component placement in multi-head concurrent operation PCBA machines. Journal of Electronics Manufacturing 4:71–79
De Souza R, Wu LJ (1995) Intelligent optimization of component onsertion in multi-head concurrent operation PCBA machines. Journal of Intelligent Manufacturing 6:235–243
Dikos A, Nelson PC, Tirpak TM, Wang W (1997) Optimization of high-mix printed circuit card assembly using genetic algorithms. Annals of Operations Research 75:303–324
Egbelu PJ, Wu CT, Pilgaonkar R (1996) Robotic assembly of printed circuit boards with component feeder location consideration. Production Planning & Control 7:162–175
Eisemann K (1964) The generalized stepping stone method for the machine loading model. Management Science 11:154–176
Ellis KP, Vittes FJ, Kobza JE (2001) Optimizing the performance of a surface mount placement machine. IEEE Transactions on Electronics Packaging Manufacturing 24:160–170
Ellis KP, Bhoja S (2002) Optimization of the assignment of circuit cards to assembly lines in electronics assembly. International Journal of Production Research 40:2609–2631
Ferguson AR, Dantzig GB (1956) The allocation of aircraft to routes - an example of linear programming under uncertain demand. Management Science 3:45–73
Floudas CA (2000) Deterministic Global Optimization: Theory, Methods and Applications. Kluwer Academic, Dordrecht
Foulds LR, Hamacher HW (1993) Optimal bin location and sequencing in printed circuit board assembly. European Journal of Operational Research 66:279–290
Francis RL, Hamacher HW, Lee CY, Yeralan S (1994) Finding placement sequences and bin locations for cartesian robots. IIE Transactions 26:47–59
Freisleben B, Merz P (1996a) A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In: Proceedings of the 1996 IEEE International Conference on Evolutionary Computation. Nagoya, Japan, pp 616–621
Freisleben B, Merz P (1996b) New genetic local search operators for the traveling salesman problem. In: Proceedings of the 4th Conference on Parallel Problems Solving from Nature. Springer, pp 890–900
Gen M, Cheng R (1997) Genetic Algorithms and Engineering Design. Wiley, New York
Glover F, Taillard E, Werra DD (1993) A user's guide to tabu search. Annals of Operations Research 41:3–28
Glover F, Laguna M (1997) Tabu Search. Kluwer Academic, Boston

Goldberg DE (1989) Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, New York

Gronalt M, Grunow M, Günther HO, Zeller R (1997) A heuristic for component switching on SMT placement machines. International Journal of Production Economics 53:181–190

Günther HO, Gronalt M, Piller F (1996) Component kitting in semi-automated printed circuit board assembly. International Journal of Production Economics 43:213–226

Hillier MS, Brandeau ML (1998) Optimal component assignment and board grouping in printed circuit board manufacturing. Operations Research 46:675–689

Hillier MS, Brandeau ML (2001) Cost minimization and workload balancing in printed circuit board assembly. IIE Transactions 33:547–557

Huntley CL, Brown DE (1996) Parallel genetic algorithms with local search. Computers & Operations Research 23:559–571

Jensen PA, Bard JF (2003) Operations Research: Models and Methods. Wiley, New York

Ji P, Wong YS, Loh HT, Lee LC (1994) SMT production scheduling: a generalized transportation approach. International Journal of Production Research 32:2323–2333

Ji P, Sze MT, Lee WB (2001) A genetic algorithm of determining cycle time for printed circuit board assembly lines. European Journal of Operational Research 128:175–184

Ji Z, Leu MC, Wong H (1992) Application of linear assignment model for planning of robotic printed circuit board assembly. Journal of Electronic Packaging 114:455–460

Johnson DS, Aragon CR, McGeoch LA, Schevon C (1989) Optimization by simulated annealing: an experimental evaluation; Part 1, graph partitioning. Operations Research 37:865–892

Kallrath J (1999) Mixed-integer nonlinear programming applications. In: Ciriani TA, Gliozzi S, Johnson EL, Tadei R (eds.) Operational Research in Industry. Macmillan, Hampshire, pp 42–76

Klincewicz JG, Rajan A (1994) Using GRASP to solve the component grouping problem. Naval Research Logistics 41:893–912

Klomp C, Klundert JVD, Spieksma FCR, Voogt S (2000) The feeder rack assignment problem in PCB assembly: a case study. International Journal of Production Economics 64:399–407

Kumar R, Li H (1995) Integer programming approach to printed circuit board assembly time optimization. IEEE Transactions on Components, Packaging, and Manufacturing Technology – Part B 18:720–727

Lapierre SD, Debargis L, Soumis F (2000) Balancing printed circuit board assembly line systems. International Journal of Production Research 38:3899–3911

Leipälä T, Nevalainen O (1989) Optimization of the movements of a component placement machine. European Journal of Operational Research 38:167–177

Leu MC, Wong H, Ji Z (1993) Planning of component placement/insertion sequence and feeder setup in PCB assembly using genetic algorithm. Journal of Electronic Packaging 115:424–432

Lourie JR (1964) Topology and computation of the generalized transportation problem. Management Science 11:177–187

Magyar G, Johnsson M, Nevalainen O (1999) On solving single machine optimization problems in electronics assembly. Journal of Electronics Manufacturing 9:249–267

Mettala EG, Egbelu PJ (1989) Alternative approaches to sequencing robot moves for PCB assembly. International Journal of Computer Integrated Manufacturing 2:243–256

Mitchell M (1996) Introduction to Genetic Algorithms. MIT Press, London

Moyer LK, Gupta SM (1996a) SMT feeder slot assignment for predetermined component placement paths. Journal of Electronics Manufacturing 6:173–192

Moyer LK, Gupta SM (1996b) Simultaneous component sequencing and feeder assignment for high speed chip shooter machines. Journal of Electronics Manufacturing 6:271–305

Moyer LK, Gupta SM (1997) An efficient assembly sequencing heuristic for printed circuit board configurations. Journal of Electronics Manufacturing 7:143–160
Murty KG (1995) Operations Research: Deterministic Optimization Models. Prentice Hall, New Jersey
Ong NS, Khoo LP (1999) Genetic algorithm approach in PCB assembly. Integrated Manufacturing Systems 10:256–265
Ong NS, Tan WC (2002) Sequence placement planning for high-speed PCB assembly machine. Integrated Manufacturing Systems 13:35–46
Osman IH, Kelly JP (1996) Meta-Heuristics: Theory & Applications. Kluwer Academic, Boston
Papadimitriou CH (1981) On the complexity of integer programming. Journal of the Association for Computing Machinery 28:765–768
Rajkumar K, Narendran TT (1997) A bi-criteria model for loading on PCB assembly machines. Production Planning & Control 8:743–752
Rayward SVJ, Warren FB, Reeves CR (1993) Modern Heuristic Techniques for Combinatorial Problems. Oxford, New York
Reinelt G (1994) The Traveling Salesman: Computational Solutions for TSP Applications. Springer, New York
Sohn J, Park S (1996) Efficient operation of a surface mounting machine with a multihead turret. International Journal of Production Research 34:1131–1143
Sze MT, Ji P, Lee WB (2001) Modeling the component assignment problem in PCB assembly. Assembly Automation 21:55–60
Taha HA (2003) Operations Research: An Introduction. Prentice Hall, New Jersey
Tawarmalani M, Sahinidis NV (2002) Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications. Kluwer Academic, Dordrecht
Thompson GL, Sethi AP (1986) Solution of constrained generalized transportation problems using the pivot and probe algorithm. Computers & Operations Research 13:1–9
Walser JP (1999) Integer Optimization by Local Search: A Domain-Independent Approach. Springer, New York
Wan YF, Ji P (2001) A tabu search heuristic for the component assignment problem in PCB assembly. Assembly Automation 21:236–240
Wilhelm WE, Tarmy PK (2003) Circuit card assembly on tandem turret-type placement machines. IIE Transactions 35:627–645
Williams HP (1999) Model Building in Mathematical Programming. Wiley, New York
Winston WL, Venkataramanan M (2003) Introduction to Mathematical Programming: Operations Research. Brooks/Cole-Thomson Learning, California
Yeo SH, Low CW, Yong KH (1996) A rule-based frame system for concurrent assembly machines. International Journal of Advanced Manufacturing Technology 12:370–376
Yu G, Kouvelis P (1993) Complexity results for a class of min-max problems with robust optimization applications. In: Paradalos PM (ed.) Complexity in Numerical Optimization. World Scientific Publishing, Singapore, pp 501–511

Index