National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Canada

Minimizing Machine Set-up Time When Manufacturing Printed Circuit Boards

by

Michael Ellsworth Weedmark

A THESIS
submitted to the School of Graduate Studies and Research
in partial fulfillment of the requirement
for the degree of

Master
in
Computer Science

Ottawa-Carleton Institute for Computer Science
Department of Computer Science
Faculty of Science
University of Ottawa
OTTAWA, ONTARIO

ISBN 0-612-04890-X

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Sylvia Boyd. Since the undertaking of my thesis, Dr. Boyd has been a constant source of support, guidance, and advice for me. I am most appreciative of her patient assistance. I would also like to thank Peter Oulton, Mike Root, Dave Watson, and Tom Buskey at the Mitel Corporation for their time and help.

Lastly, I would like to thank my wife, Angèle, and my family for their constant moral support and encouragement. Had they not been there for me, this would have been a difficult undertaking.

Abstract

Technical advances in the past decade have enabled the development of very fast but expensive component placement machines for the production of Printed Circuit Boards (PCBs). However, when these fast machines have to assemble small volumes of many different types of circuit boards, the machine set-up time becomes much more important than the assembly rate of each board. In order to minimize this set up time, we must try to solve the set-up/sequencing decision problem according to either the MCS (Minimizing Component Switches) or MSI (Minimizing Switching Instants) performance criterion, or both.

We examine the set-up/sequencing decision problem for a single machine (work cell) which has high mix low volume production schedules. We compare and improve methods which attempt to solve this problem, and we develop our own heuristics for solving this problem when both performance criteria are of great importance. We use lower bounds to determine how good our results are, and we discuss how to adapt methods which we have looked at to different manufacturing environments.

Table of Contents

Chapter 1

Introduction

1.1 Introduction and motivation

In today's society, automation has become more of a necessity than a luxury. This is apparent in the world of electronics. Take for example the assembly of Printed Circuit Boards (PCBs), where a PCB is a laminated board with electronic components inserted or mounted onto it. In the 60's, PCBs and the components that had to be placed on them were large and thus could be assembled manually. However, with the development of surface mount technology in the early 70's, both the PCBs and their components were reduced substantially in size. Consequently, the density of components on each board increased dramatically. As a result, the need for automated assembly of PCBs became apparent.

This automation, though, could only be realized by highly sophisticated machines which are very expensive. With such large capital investments, the minimization of machine idle time becomes a very important issue. This *idle time*, or down time, results from either machine breakage or machine set-up. For the assembly of a particular sequence of PCBs, a machine set-up specifies which component types have to be added to and removed from the machine between the different PCB types in order to ensure that each consecutive board in the sequence can be assembled. The set-up time specifies how much time is required to perform

all the required switches of component types for a given sequence of PCBs. A more formal definition of these terms will be given in Chapter 2.

Minimization of set-up time, which is the topic of this thesis, is controllable because we make the two fundamental decisions which determines the set-up time required for a group of PCBs. If we make intelligent decisions, we can greatly reduce the amount of machine idle time due to machine set-up. This reduction in set-up time is very important because the set-up time required for a sequence of PCBs may be as large or larger than the actual time spent on assembling the sequence of PCBs. Note that an assembly machine can place as many as a few thousand components per hour.

If we assume there are many different types of PCBs (jobs) to assemble, the first decision is to determine the order in which to assemble these different types of PCBs. Once we have decided this ordering, the second decision is to determine which components will be accessible by the assembly machine while each PCB type is being assembled. This may necessitate the switching of component types to and from the assembly machine. This second decision would not be necessary if the machine could hold all the required components for every type of board, however, this is usually impossible because of the large number of available components. As a result, this decision must ensure that the machine has access to all the required components for each PCB as it is being assembled.

Together, the two decisions above are known as the *set-up/sequencing decision problem* for flexible manufacturing machines, and a solution to this problem determines the set-up time required for a list of PCBs. A solution to this problem is called *optimal* if set-up time is minimized by it. Today, these decisions

are becoming even more important than before because of the current trend towards high mixes of PCBs with low volumes to assemble. This high mix of PCBs with low volumes has come about because of the fierce competition in the electronics industry. This competition has forced the customization of products, and it is this customization that has resulted in high mixes and low volumes of PCBs. As a consequence, there will be more switching of component types to and from the assembly machine, and fewer boards assembled per PCB type. Hence, more time is spent on machine set-up, and less time on assembling.

One company that has to deal with the production of high mixes and low volumes of PCBs is the Mitel Corporation in Ottawa, Ontario. This company is a Canadian-based international telecommunication solutions supplier. They provide such products as telephones, semiconductors, and integrated circuits. As part of their operation, they assemble PCBs. A number of highly sophisticated machines are used to assemble these boards at Mitel. However, there is one production line in particular that is dedicated to assembling high mixes and low volumes of PCBs. The machine used in this production line is the Panasonic Mv-II (LL) model NM-2559B. In order to model the Mitel environment, we consider the set-up/sequencing decision problem for the single machine case, and we assume that each PCB is loaded only once onto the assembly machine in a particular sequence.

When solving the single machine set-up/sequencing decision problem, one of two possible performance criteria are used to represent the corresponding set-up time for a job sequence. One uses the number of component switches to represent set-up time, while the other uses the number of times the assembly machine must be shut down to perform component switches without worrying about how many switches take place at each shut down. The first criterion is used when set-up time

is directly proportional to the number of component switches, while the second is used whenever component switches can be accomplished in parallel. In either case, finding an optimal solution for the set-up/sequencing decision problem has been shown to be NP-hard in [TD88a] for the first criterion, and NP-hard in [TD88b] for the second criterion. If we consider the second criterion, the set-up/sequencing decision problem has been shown to be equivalent to the job grouping problem [TD88b].

Since the problem of finding an optimal solution for the set-up/sequencing decision problem (i.e. minimizing set-up time) is known to be NP-hard for either criterion, it is unlikely that an efficient method will be found to solve it. As a result, a heuristic approach is taken, i.e. an efficient method is sought which finds a "good" solution for the set-up/sequencing decision problem. This solution may not be optimal, but is hopefully close to optimal. Several such heuristics have already appeared in the literature. Some of these take linear programming based approaches ([Bar88]), while others take combinatorial optimization approaches ([TD88a], [TD88b], [CB86], [LM86b], [MS91], [BS93]). We have examined and compared some of these combinatorial optimization methods. Note that all of these methods are solving the problem for a single machine for an idealized model.

1.2 Contributions

In this thesis, our main contributions are as follows:

1) COMPARING METHODS. Surprisingly, there seems to be no performance comparison of any of the heuristics for the set-up/sequencing decision problem currently in the literature, and thus no way to judge which heuristics perform the

best. We choose and study four of these methods which we thought looked most promising. We implement and test these methods, and report on the results.

2) IMPROVING CURRENT METHODS. While studying the four methods from the literature, we discovered ways to improve the performance of some of the methods. We implement and test our improvements, and report on the results.

3) DEVELOPMENT OF A HEURISTIC METHOD WHICH PERFORMS WELL WITH RESPECT TO BOTH PERFORMANCE CRITERIA. All of the methods in the literature concentrate on one performance criterion or the other. As a result, they tended to perform very poorly for the other criterion. However, there are situations in which both criteria are important in accurately reflecting the set-up time for a PCB list. To the best of our knowledge, there are currently no methods for dealing with such situations. We develop two heuristic methods which take both criteria into consideration. We also implement and test both heuristics, and report on the results.

4) IMPROVING LOWER BOUNDS. One way to show a heuristic solution is "good" (i.e. close to optimal) is to provide a lower bound. A *lower bound* for a minimization problem is a number k such that the cost of any solution for the problem is greater than or equal to k. We discuss how to improve a simple method for obtaining a lower bound for one of the performance criteria. We implement and report on the results.

5) APPLYING METHODS TO MANUFACTURING ENVIRONMENTS. The methods in the literature are all described for an idealized model. Typically, when actually applying these methods in industry, the environment does not fit this model. For example, in the idealized model we assume that any component type can be switched with any other. However, in some manufacturing environments, this is not the case, which complicates matters considerably. As a result, we discuss how to adapt the methods so that they can actually be applied in such manufacturing environments. In particular, we discuss how to adapt these methods so that they can be applied to the manufacturing environment at Mitel.

1.3 Overview of the thesis

We begin in Chapter 2 by carefully defining the set-up/sequencing decision problem for each of the two performance criteria. In Chapter 3, we examine heuristics from the literature which attempt to find an optimal solution for this problem using one performance criterion or the other. In Chapter 4, we develop two methods which perform well with respect to both performance criteria. In Chapter 5, we develop and test an improved method for obtaining lower bounds for the set-up/sequencing decision problem. In Chapter 6, we describe how to adapt and change the heuristics so that they can be used in industry. Finally, in Chapter 7, we summarize our work, and propose some future research topics.

The remainder of this chapter is devoted to background information, such as definitions and notation which are used throughout the thesis.

1.4 Background information

In this section, we will describe some notation and definitions that we use throughout the thesis. In addition, we describe two problems that arise frequently as subproblems in the heuristics for the set-up/sequencing decision problem, namely the minimum weight Hamiltonian path problem and the travelling salesman problem.

1.4.1 General notations and definitions

Set : A set is a collection of distinguishable objects.

$|s|$: This represents the cardinality of the set s.

$A \setminus B$: This represents the set which is the difference between two sets A and B; i.e. $\{x : x \in A \text{ and } x \notin B\}$

$A \cap B$: This represents the intersection of two sets A and B; i.e. $\{x : x \in A \text{ and } x \in B\}$

$[x]^+$: This represents the maximum of x and zero.

$\lfloor i \rfloor$: This represents the largest integer that is less than or equal to the real number i .

$\lceil i \rceil$: This represents the smallest integer that is greater than or equal to the real number i.

1.4.2 Graph theoretical definitions [CLR90]

Graph

A graph is a pair (V,E), where V is the set of nodes and E is a set of unordered pairs of nodes representing edges.

Weighted graph

A weighted graph is a graph for which each edge has an associated weight which is a real number.

Path

A path in a graph is a finite non-null sequence of consecutive edges.

Cycle

A cycle in a graph is any path which starts and finishes at the same node.

Simple Path

A simple path in a graph is a path in which all nodes are distinct.

Hamiltonian Path

A Hamiltonian path in a graph is a simple path that visits every node exactly once.

Hamiltonian Cycle

A Hamiltonian cycle in a graph is a cycle which visits every node exactly once.

Complete Graph

A complete graph is a graph with an edge between each pair of nodes.

Incident

An edge $e = uv$ is said to be incident to nodes u and v.

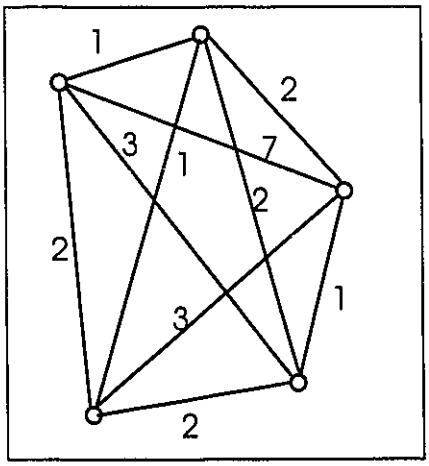
Degree of a node

The degree of a node in a graph is the number of edges incident to it.

1.4.3 Converting the minimum weight Hamiltonian path problem into a travelling salesman problem

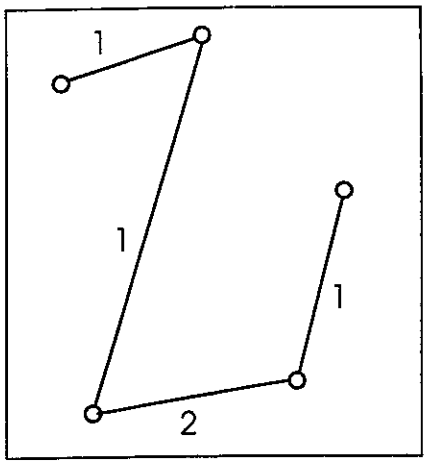
Given a weighted graph G, the weight of a path or cycle in G is the sum of the weights of the edges in that path or cycle. Given a complete weighted graph G, the *minimum weight Hamiltonian path problem* is the problem of finding a Hamiltonian path in G of minimum weight. The *Travelling Salesman Problem* (henceforth TSP) for G is the problem of finding a minimum weight Hamiltonian cycle in G. Both of these problems are known to be NP-hard [LLRS86].

Consider the graph G shown below:

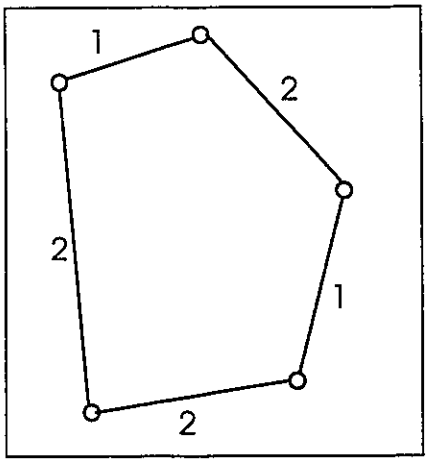


Graph G

For the graph G, a minimum weight Hamiltonian path and Hamiltonian cycle are shown below:

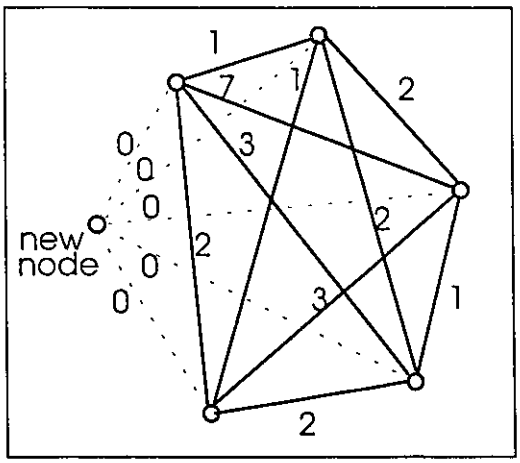


A minimum weight Hamiltonian path

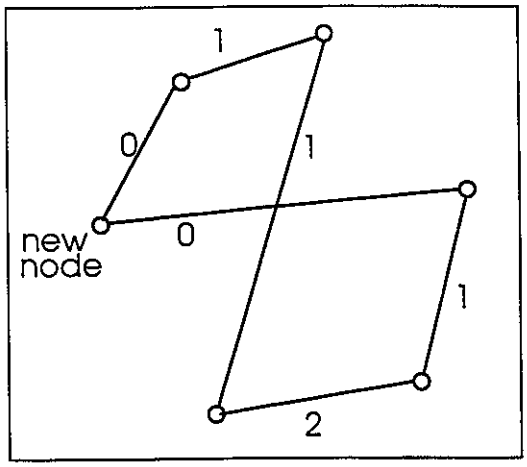


A minimum weight Hamiltonian cycle

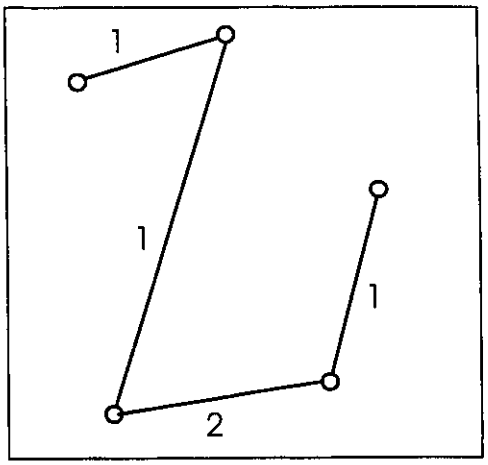
Many good heuristic methods are known for the TSP, such as the Lin-Kernighan method ([LLRS86]). These heuristics can also be used for the minimum weight Hamiltonian path problem, since it is possible to convert this problem into a TSP as follows. Generate a new graph G' by simply adding a new node to the complete graph G with an edge of weight zero between this new node and every other node. Let *W* be a minimum weight Hamiltonian cycle in G'. Then the path obtained by deleting the two edges in *W* incident with the new node is a minimum weight Hamiltonian path. For example, for the graph G illustrated previously, G', and the corresponding minimum weight Hamiltonian path are shown below.



G'



A minimum weight Hamiltonian cycle in G'



A minimum weight Hamiltonian path in G'

Chapter 2

Problem definition and complexity

2.1 Idealized model of the problem

In developing a solution strategy for a specific application in operations research, an idealized model of the problem is often considered in order to keep the complexities of the solution strategy to a minimum. We describe this model for the set-up/sequencing decision problem in the following sections.

2.1.1 Description of the equipment

We begin by describing a typical machine. Following this description, we specify the five fundamental operations performed by any of these machines.

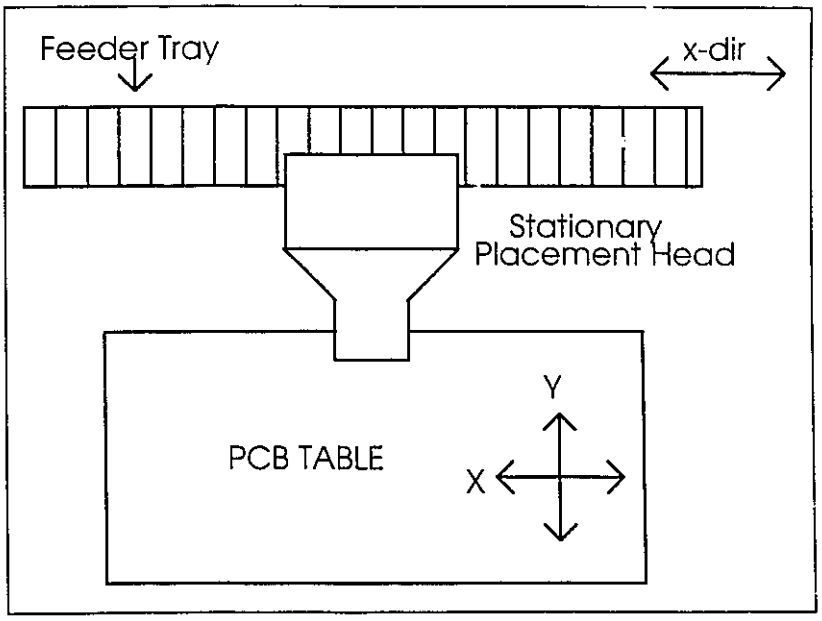


FIGURE 2.1

A typical high speed pick-and-place machine is shown in Figure 2.1. This machine has three main parts: the feeder tray, the placement head, and the PCB table. The *feeder tray* is a tray with a finite number of slots, where the number of slots is called the *feeder tray capacity.* Typically, there are between 50 and 150 slots in a feeder tray, and we will use the variable C to denote this capacity. These slots are used to hold *component reels*, where a reel contains many components of a single type. The function of this tray is to supply all the necessary components for a PCB to the placement head. This is accomplished by the movement of the tray back and fourth in the x-direction.

The *placement head* is a mechanism which places the components on the PCBs. It has three main functions. Its first function is to pick-up a component from the feeder tray. Secondly, it must process this component. *Component processing*

involves such things as verifying correctness, the rotation of a component, etc.. Lastly, it must place the component on the PCB.

The *PCB table* holds the PCB and allows the correct placement of components. This table moves in both the x and y directions so that the PCB can be positioned correctly under the placement head whenever a component is to be placed.

Now that we know the basic parts of a high speed pick-and-place machine, we can enumerate its five fundamental operations.

i) Positioning of the feeder tray for the retrieval of a component;

ii) Retrieval of a component;

iii) Component processing;

iv) Positioning of the PCB for component placement;

v) Placement of a component.

Because there are many different types of high speed pick-and-place machines, the description of each machine may vary a little from the one described above. For example, another common type of high speed pick-and-place machine has many placement heads located on a turret rather than having a single stationary head, as described above. Obviously, this changes the description of the machine, but the fundamental operations for both machines are still the same.

2.1.2 Description of the problem

We consider a *job* to be a set of PCBs all of one type. Given a list of jobs which must be processed, a *job sequence* is an ordering of these jobs. As each job has different component requirements, certain component reels will have to be added to the feeder tray in order to process the next job. This may necessitate the removal of other component reels. A *component switch* consists of a component reel being removed from the feeder tray and replaced by a different component reel, and a *component mix* specifies which component reels will be on the feeder tray before a job is to be processed. A *set-up* for a job sequence $(j_1,j_2,...,j_k)$ includes an initial component mix and a specification of component switches between jobs which will enable the processing of job j_{i+1} after job j_i, for i=1,2,...,k-1. These component switches determine the component mixes required to process each job in the production list. A *feeder tray assignment* specifies where each component reel in a component mix is to be placed on the feeder tray. Furthermore, *instant* n, n=1,...,k-1, specifies a moment in time just after processing the n^{th} job, but before any component switches occur for the $n+1^{st}$ job. A *switching instant* is an instant in which at least one component switch occurs between the processing of two jobs .

Set-up time is the time required to perform all the necessary component switches for a particular job sequence. Note that we do not include the time required to initially load the feeder tray into the set-up time, as it is constant for all our solutions (see Section 2.1.3). Generally, the set-up time is directly proportional to the number of switches, however, if switches can be performed in parallel, this is not the case. Finally, *makespan* is the total time required to process a list of jobs.

Now that some of the terminology that will be used throughout this thesis has been defined, we can discuss the problems that must be solved. Typically, a high speed pick-and-place machine that assembles PCBs will have a *production list* which specifies the types and numbers of PCBs that must be produced. This machine, however, requires machine instructions to operate. The generation of these instructions is a difficult task because they must minimize the makespan for a production list. To generate these instructions, the following problems must be solved:

i) The set-up/sequencing decision problem;

ii) The component insertion sequence/feeder tray assignment problem.

The *set-up/sequencing decision problem* is the following: Given a production list, find a job sequence and set-up. An *optimal* solution to this problem is one for which set-up time is minimized. In solving this problem, we will use two different performance criteria for set-up time. The first criterion called *Minimizing Component Switches* (henceforth MCS) considers minimizing the number of component switches, while the second called *Minimizing Switching Instants* (henceforth MSI) considers minimizing the number of switching instants, or the number of times at which the assembly machine must be stopped in order to perform component switches.

The MCS criterion is used whenever the set-up time is directly proportionate to the number of component switches. However, if all the component switches can be accomplished in parallel, we do not need to be concerned with how many component switches take place each time the machine is stopped for switching, in which case the MSI criterion is used. Interestingly, the

solution strategies for each performance criterion have led to completely different approaches, as will be seen in the following chapters. In any case, optimally solving the set-up/sequencing decision problem has been shown to be NP-hard in [TD88a] and [TD88b] for each of the MCS and MSI criteria, respectively. Hence, the need for heuristics becomes apparent.

By optimally solving the set-up/sequencing decision problem for a production list, we have minimized set-up time. Now, we must concentrate on minimizing the assembly time required for all the jobs in the production list. The *component insertion sequence/feeder tray assignment problem* is the following: Given a solution to the set-up/sequencing decision problem, find for each job in the job sequence an order for the placement of components on the PCB and an assignment of component reels to feeder slots. An *optimal* solution to this problem is one for which assembly time for each job is minimized. Optimally solving this problem is also NP-hard [BCF94]. Note that the solution to the feeder tray assignment problem could increase set-up time if between jobs component reels on the feeder tray are swapped to decrease assembly time.

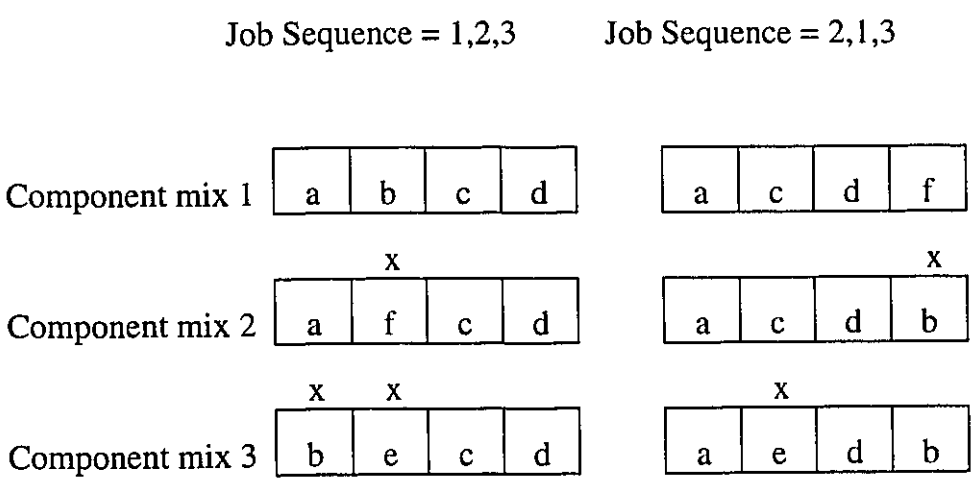
The solution to the above problems are important, but they become less or more important depending on the type of production list. If we have a production list where we assemble large numbers of only a few types of PCBs, we are more concerned with the time it takes to assemble each board (the component insertion sequence/feeder tray assignment problem). On the other hand, if we have to assemble small numbers of a wide variety of PCB types, we are more concerned with the amount of set-up time required (the set-up/sequencing decision problem).

Because of the fierce competition in the electronics industry, the customization of products has become a general trend. As a result, we find ourselves in the situation where we generally have production lists that have low volumes of a high mix of PCBs. In this case, the assembly time required for a sequence of jobs may be less than the set-up time required for that same job sequence. Consequently, we must try to reduce set-up time by solving the set-up/sequencing decision problem. Once this problem is solved, we have to generate a component insertion sequence and feeder tray assignment such that set-up time does not increase. One way to do this is to simply use the component mixes for each PCB generated by the solution to the set-up/sequencing decision problem as feeder tray assignments, and decide on a component placement order for each PCB based on these feeder tray assignments. Recall that assembly time is not a big concern when assembling small numbers of a wide variety of PCBs.

To demonstrate how different job sequences and component mixes can affect the number of component switches, we will use the following component/job matrix. This matrix specifies which component reels are required by each job. Let us assume that the feeder tray capacity (C) equals four, and that the feeder tray is initially empty.

Component Reels	Jobs 1	2	3
a	1	1	0
b	1	0	1
c	1	1	0
d	0	1	1
e	0	0	1
f	0	1	0

To illustrate the affect job sequences have on the number of component switches, consider the two job sequences 1,2,3 and 2,1,3. If we minimize the number of component switches required, we obtain the following component mixes for each of these job sequences. Note that the initial *C* component reels placed on the feeder tray are not considered to be component switches.

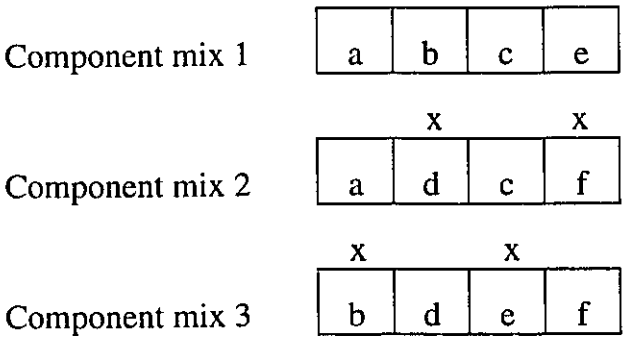


Every place there is an "x" above a component reel, there is a required component switch. As a result, job sequence 1,2,3 requires three component switches, whereas job sequence 2,1,3 requires only two. Consequently, the job sequence that should

be used is 2,1,3 if we consider only the two job sequences above. In fact, there are at most six job sequences for this example, and the minimum number of required component switches is two.

Obviously, the set-up for a job sequence can also affect the number of component switches. For example, as shown above, the job sequence 1,2,3 requires three component switches. On the other hand, if we look at the following example, job sequence 1,2,3 now requires four component switches with this new set-up.

Job Sequence = 1,2,3



Consequently, we must be very careful when determining which component reels will be added and removed from the feeder tray. Obviously, component reels that are on the feeder tray and are required furthest in the future should be removed, while component reels that are required by the next job and not on the feeder tray should be added.

2.1.3 Assumptions made for the idealized model

The following assumptions have been made so that we can minimize the complexity of the set-up/sequencing decision problem. Of course, some of these assumptions will need to be discarded once we adapt our heuristics to the Mitel environment in Chapter 6, but many of them are valid for any type of environment.

To begin, we assume that only one component reel per component type is allowed on the feeder tray at any one time, and that each component reel occupies a single slot. In reality, there are many cases where, due to component sizes, a component reel requires two or more slots in the feeder tray. However, if we do not make this assumption, component switching becomes much more complicated because we can not necessarily remove one component reel and replace it with another. Secondly, we assume that the time it takes to remove one component reel from the feeder tray and replace it with another is constant. As a result, it is not beneficial to leave any feeder slots empty. Thirdly, we assume that the initial feeder tray is empty, and thus the initial component mix, which is part of the set-up, requires C component reels to be placed onto the feeder tray. Because this time is constant for all solutions, it will not be included into the set-up time. Hence, the initial component mix is ignored when considering either the MCS or MSI performance criterion. Finally, we assume that the size of each job to be manufactured requires less than the feeder tray capacity (C) of the machine, and that *split boards,* which are PCBs that are loaded more than once onto the assembly machine, are not allowed. Not allowing split boards ensures the precise assembly and reduced handling of PCBs.

2.2 Non-linear integer programming formulations

One way to approach solving the set-up/sequencing decision problem which we do not examine in this thesis is to first formulate the problem as a Non-Linear Integer Program (NLIP), and then to try to use standard numerical methods and heuristics for such formulations. We describe two such formulations, one for each performance criterion, which are similar to the formulations given in [Tan86] and [Bar88]. Although we do not make use of these formulations, we include them here for completeness. Before describing the NLIPs for both the MCS and MSI criteria, the following notation is required.

Let:

J = number of jobs in the production list

M = number of required component types in the production list

C = feeder tray capacity

(i.e. number of component types that can be loaded simultaneously)

$$r_{ij} = \begin{cases} 1 \text{ if component type i is required by job j} \\ 0 \text{ otherwise} \end{cases}$$

$i = 1,...,M ; \; j = 1,...,J$

Decision Variables

$$x_{jn} = \begin{cases} 1 \text{ if job j is processed } n^{th} \text{ in the job sequence} \\ 0 \text{ otherwise} \end{cases}$$

$$y_{in} = \begin{cases} 1 \text{ if component type i is on the feeder tray at instant n} \\ 0 \text{ otherwise} \end{cases}$$

$i = 1,...,M \; ; \; j = 1,...,J \; ; \; n = 1,...,J$

2.2.1 The NLIP formulation using the MCS criterion

$$\min \sum_{n=2}^{J} \sum_{i=1}^{M} y_{in} \cdot (1 - y_{i,n-1}) \qquad (1)$$

s.t.

$$\sum_{n=1}^{J} x_{jn} = 1 \qquad j=1,...,J \qquad (2)$$

$$\sum_{j=1}^{J} x_{jn} = 1 \qquad n=1,...,J \qquad (3)$$

$$\sum_{i=1}^{M} y_{in} = C \qquad n=1,...,J \qquad (4)$$

$$r_{ij} \cdot x_{jn} \leq y_{in} \qquad \text{for all } i=1,...,M \text{ , } j=1,...,J, \; n=1,...,J \qquad (5)$$

$$x_{jn}, y_{in} \in \{0,1\} \qquad \text{for all } i=1,...,M \text{ , } j=1,...,J, \; n=1,...,J \qquad (6)$$

The non-linear integer objective function (1) corresponds to minimizing the number of component switches. As mentioned earlier, the time required to place the initial *C* component reels on the feeder tray is not included into the set-up time

(see Section 2.1.3). Constraint (2) ensures that each job j is assigned to only one position in the job sequence, while constraint (3) ensures that only one job is assigned to each position. Constraint (4) represents the capacity constraint due to the limited number of component reels that can be loaded simultaneously onto the machine. This constraint ensures that there are always C component reels on the feeder tray at any instant n because there is no benefit in leaving feeder slots empty (assuming that the time required for a component switch is constant). Constraint (5) ensures that the component types required for each PCB type are on the feeder tray when required, while constraint (6) ensures that the decision variables are either zero or one.

2.2.2 The NLIP formulation using the MSI criterion

In order to minimize the number of switching instants, we need to introduce a new 0-1 variable w_n into the formulation for the MCS criterion, where

$$w_n = \begin{cases} 1 \text{ if there is a switching instant between job n and job n+1} \\ 0 \text{ otherwise} \end{cases}$$

and w_n satisfies the following:

$$y_{in} - y_{i,n-1} \leq w_{n-1} \qquad n=2,...,J \; ; \; i=1,...,M \tag{7}$$

$$w_n \in \{0,1\} \quad n=1,...,J-1 \tag{8}$$

Constraint (7) ensures that w_{n-1} is equal to one if a component switch is required at instant n-1.

Therefore, the NLIP formulation using the MSI criterion is

$$\min \sum_{n=1}^{J-1} w_n \tag{9}$$

subject to constraints (2) - (8).

Chapter 3

Methods from the literature and proposed improvements

3.1 Introduction

Many different heuristic methods have been proposed in the literature for solving the set-up/sequencing decision problem. In this chapter, we describe four of these methods. The first two, called *greedy perturbation* [TD88a] and *labelling* [LM86a], deal with minimizing component switches (the MCS criterion), while the third and fourth, called *matrix diagonalization* [MS91] and *maximal intersection minimal union* [TD88b], deal with minimizing switching instants (the MSI criterion). In addition to describing these methods in detail, we will also discuss some possible improvements that we have developed, along with a summary of empirical results. These results will be obtained by applying these methods to our test set that consists of 10 groups of 15 different PCBs, for which the data was given to us by Mitel.

3.2 The Greedy Perturbation Method

The greedy perturbation method was proposed by Tang and Denardo in [TD88a]. In this paper, the authors do not deal with PCB assembly, but instead deal with machines that require tools rather than component reels to perform their

operations. Also, they talk about producing parts rather than assembling PCBs. However, if we consider a tool to be a component reel and a part to be a PCB, then the method presented in this paper can be used as a heuristic for the MCS criterion.

The greedy perturbation method consists of three major steps. The first step finds a good job sequence. This job sequence is then used in the second step to determine which component reels should be placed onto the feeder tray before each job is processed in order to minimize the total number of component switches required. The third step tries to improve on this job sequence.

Before discussing these three steps, it should be mentioned that any list of jobs to be processed may contain redundant jobs. A *redundant job* is a job that can be ignored when generating a job sequence because its required components are a subset of those required for another job. For example, if all the components required by job j are also required by job i, then we know that an optimal position for job j in the job sequence is right after job i. This will ensure that job j does not necessitate any component switches. As a result, all redundant jobs should be ignored while generating a job sequence.

Step 1: Finding a good job sequence.

In this step, we first create a weighted complete graph G=(V,E) in which each node represents a job. The weight on an edge is defined to be the minimum number of component switches required to process the corresponding node pair sequentially. Since this minimum number cannot be negative, we must ensure that it is greater than or equal to zero. More formally, the weight LB(i,j) for an edge ij is defined as follows:

$$LB(i,j) = [(\text{The number of components needed by job } j, \text{ but not job } i) - (\text{ The number of slots not required by job } i)]^{+}.$$

This weight function generates a lower bound on the total number of component switches that must occur between any two jobs in sequence. It calculates this lower bound by assuming that any component reels on the feeder tray not required by job i are required by job j. If we let S_i represent the set of components required by a job i, this function can be written more concisely as follows (recall C = the feeder tray capacity):

$$\begin{aligned} LB(i,j) &= [(|\, S_j \,| - (\, |\, S_j \cap S_i \,|\,)) - (\, C - |\, S_i \,|\,)]^{+} \\ &= [|\, S_j \,| + |\, S_i \,| - (\, |\, S_j \cap S_i \,|\,) - \, C \,]^{+}. \end{aligned}$$

Note that $LB(i,j) = LB(j,i)$.

If we find a minimum weight Hamiltonian path in the graph G with the edge weights defined by the function $LB(i,j)$, then we will have found a job sequence (J) which requires the fewest number of component switches in G. As a result, the job sequence will have jobs which have similar component requirements in sequence. Intuitively, this should result in a good initial job sequence.

The heuristic method used in [TD88a] to generate this Hamiltonian path in G is as follows:

Step i) Let G'=(V',E'), where V'=V, E'=∅. Let i=1.

Step ii) Sort the edges of E into non decreasing order by weight.

Let $e_1, e_2, \ldots, e_{|E|}$ be the resulting order.

Step iii) While |E'| < |V|-1 do the following:

Check to see if adding e_i to G' creates a node of degree 3 or a cycle in G'. If it does not, add e_i to E'. Let i = i+1.

Step 2: Determining the component switches.

Given the job sequence (J) obtained from Step 1, we find an initial component mix and the component switches that must occur in order to process each job in the job sequence. This initial component mix will contain the *C* component reels that will be required the soonest, and the component switches will be determined by the procedure called the *Keep Component Needed Soonest* (henceforth KCNS) procedure which is based on the following two criteria:

i) No component reel is added unless it is required by the next job;

ii) If a component reel must be added, the component reel required furthest in the future is the one which is switched off the feeder tray to make room for the new reel.

This procedure ensures that before each job is processed the required component reels have been loaded onto the feeder tray, and the component reels needed

furthest in the future are the ones which are to be removed from the feeder tray if a component switch must occur.

Note that, given a job sequence, the KCNS procedure finds an optimal series of component switches, i.e. a minimum number of component switches for that particular job sequence. Thus, if we could find an optimal job sequence for a group of jobs, we could then use the KCNS procedure on this sequence to determine the minimum number of component switches that must occur to process all the jobs in the production list.

Step 3: Perturbation

Now that we have determined the set-up in Step 2 for the job sequence (J) obtained from Step 1, we may be able to find other job sequences that can be processed given this same set-up. We accomplish this by obtaining sets of jobs which can be processed by each of the different component mixes in the set-up. We then enumerate all job sequences that are possible from these sets and then reapply the KCNS procedure on each job sequence to check if the number of component switches decreases. If so, we take this new improved job sequence as J and repeat the perturbation step. Otherwise, we try the next possible job sequence. Notice that the number of component switches cannot possibly increase since the original set-up for J can be used for each of the new job sequences.

3.2.1 Example

To illustrate the greedy perturbation method, we will use a simple example. Let the following matrix represent the component reels required by each job, and let the feeder tray capacity (C) equal 4.

		Jobs			
		1	**2**	**3**	**4**
Component Reels	**a**	1	1	1	1
	b	1	0	0	1
	c	0	0	1	0
	d	1	0	0	0
	e	0	1	1	0
	f	1	1	0	0

Before starting the procedure, we must find all the redundant jobs. By examining the above matrix, we can see that the only redundant job is Job 4. Job 4 is redundant because it requires component reels (a, b) which are a subset of Job 1's required component reels (a, b, d and f). As a result, Job 4 can be processed after Job 1 without causing any component switches. Thus, we simply ensure that Job 4 follows Job 1 in the final solution.

Step 1: Finding a good job sequence.

In this step, we must calculate $LB(i,j) = [\,|\,S_j\,| + |\,S_i\,| - (\,|\,S_j \cap S_i\,|\,) - C\,]^+$ for all pairs of non redundant jobs:

$LB(1,2) = 3 + 4 - 2 - 4 = 1$

$LB(1,3) = 3 + 4 - 1 - 4 = 2$

$LB(2,3) = 3 + 3 - 2 - 4 = 0.$

The corresponding weighted graph G is shown in Figure 3.1.

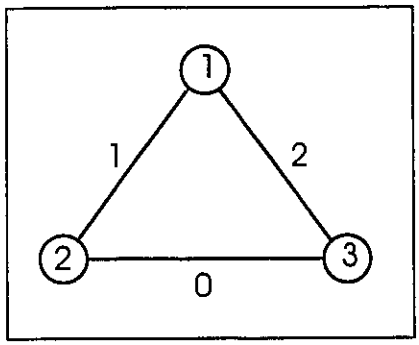


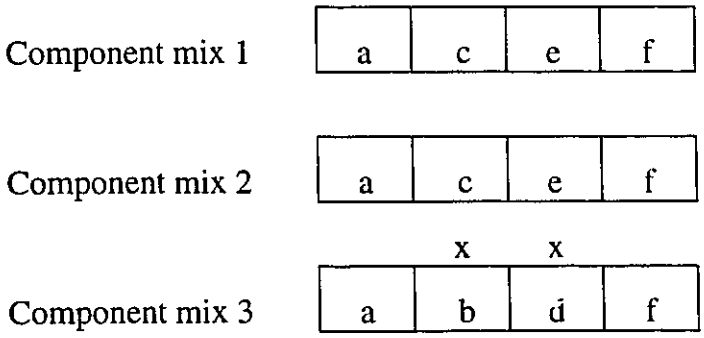
FIGURE 3.1

Next we find a minimum weight Hamiltonian path on G. Normally a heuristic is used for this, but for this small example we can see that job sequence 3, 2, 1 with weight one is a minimum weight Hamiltonian path for G.

Step 2: Determining the component switches.

In this step, we generate the sequence of component mixes required to process all the jobs in the job sequence. The initial component mix contains the component reels that are required the soonest, given the job sequence 3, 2, 1. As a result, the component reels *a*, *c*, *e*, and *f* are initially placed onto the feeder tray. Once we have generated the initial component mix,

we apply the KCNS procedure to determine all the required component switches which in turn gives us the remaining component mixes. The component mixes obtained are as follows:



These are the component mixes that have been obtained for the job sequence 3, 2, 1. We can see that the

# of component switches = 2 (component reel *b* replaced reel *c* and component reel *d* replaced reel *e)*

# of switching instants = 1 (We never include the initial tray. Since the second mix is the same as the first, no switches are required. Switches are required for the third mix.)

Step 3: Perturbation

In this step, we find all the jobs which can be processed by each component mix. Then we enumerate all possible job sequences for that set-up.

					Jobs
Component mix 1	a	c	e	f	3, 2
Component mix 2	a	c	e	f	2, 3
Component mix 3	a	b	d	f	1

As can be seen from the above table, the component mix for Job 3 and Job 2 are the same. As a result, the following two job sequences are possible:

3, 2, 1 OR 2, 3, 1.

Since we have already tried job sequence 3, 2, 1, we will now try applying KCNS to job sequence 2, 3, 1. By doing so, we obtain the following:

Component mix 1	a	e	f	c
Component mix 2	a	e	f	c
		x		x
Component mix 3	a	b	f	d

# of component switches = 2 (component reel *b* replaced reel *e* & component reel *d* replaced reel *c*)

# of switching instants = 1

In both cases, we obtain the same results. Hence, we do not obtain an improved solution.

As a final step, we must remember to insert redundant Job 4 into the job sequence after Job 1. Thus, our final job sequence is

3, 2, 1, 4.

3.2.2 Our improvements

In studying the greedy perturbation method, we noticed two modifications which could possibly improve the results obtained. The first modification that we tried was to alter the way in which the edge weights are calculated for G in Step 1. Recall that the calculation used for LB(i,j) is as follows:

$$LB(i,j) = [\,|\,S_j\,| + |\,S_i\,| - (\,|\,S_j \cap S_i\,|\,) - C\,]^+.$$

Notice that C is a constant that is always subtracted. As a result, we can consider simply ignoring this constant when calculating LB(i,j). Hence, we obtain a new formula which can be described as follows:

$$RC(i,j) = |\,S_j\,| + |\,S_i\,| - (\,|\,S_j \cap S_i\,|\,).$$

This new definition states that the weight given to an edge ij will be the total number of component types required to process both jobs i and j sequentially. Therefore, when we generate a minimum weight Hamiltonian path on a graph with

such edge weights, the cost of this Hamiltonian path will indicate the total number of component types required to process each sequential pair of jobs in the path.

Using function RC(i,j) rather than LB(i,j) seems to be a good idea because we are able to keep information that is lost by function LB(i,j). Because LB(i,j) calculates a lower bound on the number of component switches required by node pair ij, its resulting value must be positive. If this were nct true, we would have to be able to talk about node pairs that require a negative number of component switches to process them. Obviously, this is impossible. Hence, all negative values are changed to zero. As a result, an edge with weight zero supplies us with very little information when we are generating a job sequence. On the other hand, all values generated by the function RC(i,j) supply us with some information. Consequently, we feel that we will be able to obtain better job sequences when using function RC(i,j).

The second modification that we tried was to change the method used to heuristically generate a minimum weight Hamiltonian path. Instead of using the process described in [TD88a], we transformed the problem into a TSP problem using the procedure outlined in Section 1.4.3. We then solve this TSP problem using the nearest neighbour heuristic followed by the 2-opt and Lin-Kernighan tour improvement heuristics. We feel that solving the Hamiltonian path problem in this way will improve the overall performance of the algorithm because the Lin-Kernighan heuristic performs so well for TSP problems [LLRS86].

3.2.3 Implementation results

To check what kind of effect our modifications had on the greedy perturbation method, we executed this algorithm with and without the modifications on our test set. The results are tabularized in Table 3.1. Note that the number in each array element is the number of component switches necessary, and the notation used in this table is defined as follows:

T Equa - Their Equation T Heur - Their Heuristic

O Equa - Our Equation O Heur - Our Heuristic

Heuristic	*10 Groups of 15 PCBs*									
Perturb	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
T Equa O Heur	164	135	157	172	152	142	156	146	156	150
O Equa O Heur	159	114	144	151	138	138	156	139	131	149
O Equa T Heur	160	112	142	151	155	139	154	143	141	159
T Equa T Heur	182	114	144	171	143	151	157	157	147	165

TABLE 3.1

We need to compare the results in the top three rows of Table 3.1 to those in the bottom row to see what kind of effect our modifications have on the greedy perturbation method. First, we consider the case where we use our equation, and our heuristic (row 2) to implement the greedy perturbation method. In comparing these results to those in row 4, our equation and our heuristic decreased the number of component switches eighty percent of the time, and in the other twenty percent, the number of component switches did not change. Secondly, we consider

the case where we use our equation, but their heuristic (row 3). In this case, our equation gave better results than those in row 4 ninety percent of the time. Lastly, we consider the case where we use their equation, but our heuristic (row 1). In this case, our heuristic reduces the number of component switches only fifty percent of the time. We feel that the Lin-Kernighan heuristic does not do well in this case because some information is lost when using their equation (see Section 3.2.2). In any case, we can conclude that, in general, better results will be achieved if our equation is used in the greedy perturbation method, and that our heuristic will do as well or better than their heuristic.

3.3 The Labelling Method

The labelling method for the MCS criterion was proposed by Lofgren and McGinnis in [LM86a]. This method generates a job sequence by the use of labels and determines an optimal set-up for this sequence by using the KCNS procedure in [TD88a], which is described in Section 3.2.

In this method each component reel has a label, where each label is an array indexed from 0 to C (the number of feeder slots). The i^{th} element in the label for component reel j represents the number of currently unsequenced jobs which use component type j and require exactly i additional component reels to be added to the feeder tray before they can be assembled. These labels are calculated using:

i) the current component mix;

ii) the assumption that each component reel is on the feeder tray while calculating its label.

Take for example the following label, where the feeder tray can hold 4 component reels:

	0	1	2	3	4
L_a	0	0	3	0	0

This is the label for component reel *a*. It indicates that there are 3 unsequenced jobs that use component type *a* and require exactly 2 additional component reels to be added to the feeder tray before they can be assembled. These labels are then used to determine which component reels are to be added to and removed from the feeder tray. Component reels to be added to the feeder tray have lexicographically maximum labels, while component reels to be removed have lexicographically minimum labels. In other words, component reels that will allow us to sequence jobs the soonest are placed onto the feeder tray, while component reels that are not required, or at least not for a while, are removed.

Lets assume we have the following component/job matrix, and the feeder tray capacity is two.

		Jobs		
		1	**2**	**3**
	a	0	0	1
Component	**b***	0	1	1
Reels	**c***	1	0	0
	d	1	1	0

Note: An asterisk (*) beside a component reel indicates that it is on the feeder tray.

If component reels *b* and *c* are on the feeder tray, the four labels (L_a, L_b, L_c, L_d) are as follows.

L_a

0	1	2
1	0	0

L_b

0	1	2
0	2	0

L_c

0	1	2
0	1	0

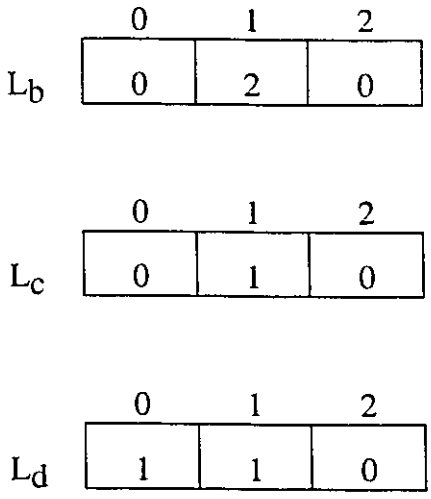
L_d

0	1	2
2	0	0

The ordering of these labels from lexicographically minimum to lexicographically maximum is L_c, L_b, L_a, L_d. Since component reel *c* has the lexicographically minimum label of all component reels on the feeder tray, it is removed. The labels are now updated and are shown below:

L_a

0	1	2
1	0	0



The ordering of these new labels from lexicographically maximum to lexicographically minimum is L_d, L_a, L_b, L_c. Since component reel *d* has the lexicographically maximum label of all component reels not on the feeder tray, it is added to the feeder tray. This single component switch allowed job 2 to be processed.

This sequencing algorithm (SEQ) can be stated as follows:

Algorithm SEQ:

Step 1: If an initial component mix does not exist, generate a component mix such that all component reels are not required by any job;
This is accomplished by taking the component reels not used by any job and adding them to the feeder tray.
If the feeder tray is still not full, then generate dummy component reels for the rest of the feeder tray;

Step 2: Initialize all jobs as unsequenced;

Based on the current component mix, determine the number of component reels that must be added to the machine for each job to be processed (Job Cost);

Sequence unprocessed jobs that can now be processed with the current component mix; Compute the labels;

Step 3: **While there are unsequenced jobs do the following**

Of the component reels currently on the feeder tray, remove the one which has the lexicographically minimum label;

Update the labels;

Of the component reels currently not on the feeder tray, add the one which has the lexicographically maximum label.

Add to the current job sequence unprocessed jobs that can now be processed with the current component mix;

Update the labels;

End(*While*).

This algorithm will result in a job sequence and a component mix for each job. These component mixes, however, may not be optimal. Therefore, the KCNS procedure in [TD88a] is used to generate component mixes that minimize the number of component switches required to process all the jobs in the job sequence.

3.3.1 Example

To illustrate this procedure, we will use the following simple example. Let the following matrix represent the component reels required by each job and let the feeder tray capacity equal 4.

		Jobs			
		1	**2**	**3**	**4**
Component Reels	**a**	1	1	1	1
	b	1	0	0	1
	c	0	0	1	0
	d	1	0	0	0
	e	0	1	1	0
	f	1	1	0	0

Step 1: Assume our initial component mix contains component reels (*a*, *b*, *c*, *d*).

Step 2: Initialize all jobs as unsequenced

Job 1, Job 2, Job 3, Job 4 = 'U' (unsequenced).

Compute the number of component reels that must be added to the machine for each job to be processed (Job Cost).

Job Number	Job Cost
Job 1	1
Job 2	2
Job 3	1
Job 4	0

Sequence all jobs which can be processed, given the current component mix.

Sequence - { Job 4 }

Compute the labels.

Component Reels Required

Labels	**0**	**1**	**2**	**3**	**4**
a*	0	2	1	0	0
b*	0	1	0	0	0
c*	0	1	0	0	0
d*	0	1	0	0	0
e	1	1	0	0	0
f	1	1	0	0	0

Step 3: There are three unsequenced jobs.

Remove component reel with the lexicographically minimum label.

Remove component reel *d* from the feeder tray

Update labels

Labels	Component Reels Required				
	0	**1**	**2**	**3**	**4**
a*	0	1	2	0	0
b*	0	0	1	0	0
c*	0	1	0	0	0
d	0	1	0	0	0
e	1	1	0	0	0
f	0	2	0	0	0

Add component reel with the lexicographically maximum label.

Add component reel *e* to the feeder tray

Sequence all jobs which can be processed, given the current component mix

Sequence - { Job 4, Job 3 }

Update Labels

Labels	Component Reels Required				
	0	**1**	**2**	**3**	**4**
a*	0	1	1	0	0
b*	0	0	1	0	0
c*	0	0	0	0	0
d	0	1	0	0	0
e*	0	1	0	0	0
f	1	1	0	0	0

Step 3: There are two unsequenced jobs.

Remove component reel with the lexicographically minimum label.

Remove component reel *c* from the feeder tray

Update labels

	Component Reels Required				
Labels	**0**	**1**	**2**	**3**	**4**
a*	0	1	1	0	0
b*	0	0	1	0	0
c	0	0	0	0	0
d	0	1	0	0	0
e*	0	1	0	0	0
f	1	1	0	0	0

Add component reel with the lexicographically maximum label.

Add component reel *f* to the feeder tray.

Sequence all jobs which can be processed, given the current component mix.

Sequence - { Job 4, Job 3, Job 2 }

Update labels

	Component Reels Required				
Labels	**0**	**1**	**2**	**3**	**4**
a*	0	1	0	0	0
b*	0	1	0	0	0
c	0	0	0	0	0
d	1	0	0	0	0
e*	0	0	0	0	0
f*	0	1	0	0	0

Step 3: There is one unsequenced job.

Remove component reel with the lexicographically minimum label.

Remove component reel *e* from the feeder tray

Update labels

	Component Reels Required				
Labels	**0**	**1**	**2**	**3**	**4**
a*	0	1	0	0	0
b*	0	1	0	0	0
c	0	0	0	0	0
d	1	0	0	0	0
e	0	0	0	0	0
f*	0	1	0	0	0

Add component reel with the lexicographically maximum label.

Add component reel *d* to the feeder tray

Sequence all jobs which can be processed, given the current component mix.

Sequence - { Job 4, Job 3, Job 2, Job 1 }

Update labels and stop.

The final job sequence obtained by this procedure is

Sequence - { Job 4, Job 3, Job 2, Job 1 }

Since we now have a job sequence, we can apply KCNS, as described in Section 3.2, to generate component mixes which minimize the total number of component switches required to process each job in the job sequence.

3.3.2 A Modification used in our implementation

If we examine the assumption that each component reel is on the feeder tray while calculating its label, we will see that by ignoring this assumption it has no affect on the solutions. Obviously, the only labels that will be affected by this assumption are the ones for component reels that are not on the feeder tray. Take for example the following label for component type a which is not on the feeder tray:

	0	1	2	3	4
$\mathbf{L_a}$	0	0	0	1	0

This label, L_a, indicates that there is one job that uses component type a and requires exactly three additional component reels to be added to the feeder tray

before it can be processed. If we drop the assumption that each component reel is on the feeder tray while calculating its label, label L_a now becomes the following:

	0	1	2	3	4
L_a	0	0	0	0	1

This label now states that there is one job which requires four additional component reels to be added to the feeder tray. As can be seen, dropping this assumption only shifts the numbers in the label to the right one. This right shift would occur to all labels whose components were not on the feeder tray. Hence, we would not change the order in which component reels are added to the feeder tray. As a result, our solutions do not change.

3.3.3 Implementation results

We executed this algorithm on our test set and tabularized our results in Table 3.2. Note that the number in each array element is the number of component switches necessary.

Heuristic	***10 Groups of 15 PCBs***									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Labelling Method	187	119	160	163	157	171	152	152	146	165

Table 3.2

A comparison between the different methods will be given in Section 3.6.

3.4 The Matrix Diagonalization Method

The matrix diagonalization method was proposed by Maimon and Shtub in [MS91]. This method also deals with reducing the amount of set-up time required. However, it uses the MSI criterion rather than the MCS criterion. In other words, the goal of this method is to minimize the number of times the machine must be stopped to perform component switches for a set of jobs. As mentioned earlier, this MSI criterion is useful whenever component switches can be accomplished in parallel. In order to use this criterion, the jobs are partitioned into groups, where a *group* is a set of jobs that can be assembled using a single component mix. Thus, minimizing the number of times at which component switches occur corresponds to minimizing the number of groups.

To try to minimize the number of groups, the matrix diagonalization method attempts to place as many jobs in a group as possible without exceeding the feeder tray capacity. This is achieved by grouping similar jobs together. Similar jobs are jobs that require nearly the same components, and thus, this grouping enables the addition of a job to a group with little additional feeder tray capacity consumption. Note that the procedure which performs this grouping also allows the user to choose the degree to which split boards will be present in the final solution. As our problem formulation does not allow split boards, we describe the corresponding simplified version of this procedure adapted to our situation.

The matrix diagonalization method consists of the following three steps. The first step is to take the component/job matrix and reorganize it in a matrix diagonalized form so that the columns corresponding to similar boards are adjacent. The algorithm used to perform this task was developed by King in

[Kin80]. The second step determines which of the jobs can be placed in the group. Finally, the third step removes all the jobs from the matrix that are in the group. If no jobs are left in the matrix, the procedure is complete. Otherwise, the first step is repeated. In more detail, the matrix diagonalization method is as follows:

Step 1: Diagonalize the matrix.

Diagonalization of the component/job matrix is accomplished using the Rank Order Clustering (ROC) algorithm described in [Kin80]. This algorithm takes the rows of the matrix as binary words and then ranks them in reducing binary value. It then uses this ranking to rearrange the rows. Similarly, the columns are rearranged. This process is repeated until either the matrix row order or the matrix column order does not change. The result of this procedure is a matrix in which columns corresponding to similar jobs are adjacent.

Note that a problem may occur with this method if the binary words are too large for the computer. However, a simple solution to this problem is to compare two rows (or two columns) digit by digit until you can conclude which of the two is larger. This avoids the problem of trying to represent integers larger than the computer can handle.

Step 2: Generate a Group.

Now that we have rearranged the matrix so that similar jobs are adjacent, we can generate a group. To do this a greedy approach is used. We consider the jobs in the same order as their corresponding columns. If adding the next job to the

group can be done without exceeding the capacity of the feeder tray, this job is included in the group. Otherwise, we try the next job in the column order.

Step 3: Check to see if all jobs are grouped.

Remove all jobs from the matrix that were placed in the group. If no more jobs exist in the matrix, the procedure is complete. Otherwise, return to Step 1.

After the completion of this procedure, we are hopefully left with a small number of groups. The number of groups minus one will equal the number of times component switches occur (recall that we ignore the initial component mix, Section 2.1.3).

3.4.1 Example

To illustrate this matrix diagonalization procedure, we will use the following simple example. Let the following matrix represent the component reels required by each job and let the feeder tray capacity (C) equal 4.

		Jobs				Ranking
		1	2	3	4	
Component Reels	a	1	1	0	1	1
	b	1	0	0	1	3
	c	0	0	1	0	5
	d	0	1	0	0	4
	e	0	0	1	0	6
	f	1	1	0	0	2

Step 1: Diagonalizing the matrix

Using the ROC procedure, we find the row ranking shown above. After rearranging the rows in this order, we obtain the following matrix:

		Jobs			
		1	2	3	4
Component Reels	a	1	1	0	1
	f	1	1	0	0
	b	1	0	0	1
	d	0	1	0	0
	c	0	0	1	0
	e	0	0	1	0
Ranking		1	2	4	3

Next we find the column ranking shown above. After rearranging the columns accordingly, we obtain the following matrix:

Component Reels	Jobs 1	2	4	3	Ranking
a	1	1	1	0	1
f	1	1	0	0	2
b	1	0	1	0	3
d	0	1	0	0	4
c	0	0	0	1	5
e	0	0	0	1	6

Since the matrix row order does not change with this new ranking, Step 1 is complete.

Step 2: Generate a group

Considering the jobs in the same order as the columns (i.e. 1, 2, 4, 3), we add jobs to the group in a greedy fashion until all the jobs are scanned.

- Job 1 needs component reels *a*, *f*, and *b*. (feeder tray 3/4 full)

  Group 1 = {Job 1}.

- Job 2 needs component reels *a*, *f*, and *d*. Component reels *a* and *f* are already on the feeder tray, so only component reel *d* must be placed onto the feeder tray. (feeder tray now full)

  Group 1 = {Job 1, Job 2}.

- Job 4 needs component reels *a* and *b*. They are already on the feeder tray. Therefore, Job 4 can also be included in the group.

  Group 1 = {Job 1, Job 2, Job 4}.

- Job 3 needs component reels *c* and *e*. As the feeder tray is already full, Job 3 is not included in Group 1.

Step 3: Check to see if all jobs are grouped.

We remove all jobs from the matrix that were placed in group 1. We are now left with the following matrix.

		Jobs	**Ranking**
		3	
Component Reels	**a**	0	3
	f	0	4
	b	0	5
	d	0	6
	c	1	1
	e	1	2

As job 3 is still not in a group, we return to Step 1.

Step 1: Diagonalizing the matrix

Rearranging the rows using the row ranking shown above, we obtain the following matrix.

		Jobs
		3
Component Reels	c	1
	e	1
	a	0
	f	0
	b	0
	d	0

As there is only one column, no column ranking is necessary.

Step 2: Generate a group.

Group 2 = {Job 3}

Step 3: Check to see if all jobs are grouped.

As all jobs are grouped, the procedure terminates with the following grouping:

Group 1 = {Job 1, Job 2, Job 4}

and Group 2 = {Job 3}.

As a result, we have two groups, and thus the number of times at which component switches must occur is one.

3.4.2 Modifications used in our implementation

The first modification we made to this algorithm was to add an additional step. This step which precedes the diagonalization of the matrix ensures that all redundant jobs, as described in Section 1.2, are ignored when generating the groups. After these groups are generated, these redundant jobs are placed in the proper groups.

We also had to determine how we wanted to handle exceptional elements. An *exceptional element* is an entry of the matrix that does not allow the division of data into mutually exclusive groups. These exceptional elements can be dealt with in the ROC algorithm by overwriting them with the asterisk symbol, *, which is equivalent to a blank. The diagonalization procedure is then reapplied to this new matrix. Any operation represented by an element with an asterisk, *, in the resulting matrix must be performed after all the groups have been processed. This, of course, would result in split boards which we do not allow. Hence, we diagonalize the matrix and simply ignore exceptional elements. An illustration of an exceptional element is shown in the following component/job matrix:

		Jobs			
		1	**2**	**3**	**4**
	a	1	1	0	0
Component	**b**	1	0	0	0
Reels	**c**	0	1	1	0
	d	0	0	1	1

Intuitively, two job groups (1, 2) and (3, 4) are suggested by the matrix above. However, component reel c is required by both groups and thus does not allow the mutual exclusion between groups. As a result, matrix element (c,2) is an exceptional element. In our implementation, we leave exceptional elements alone.

3.4.3 Implementation results

We applied this algorithm to our test set and tabularized our results in Table 3.3. Note that the number in each array element is the number of times component switches must occur. Furthermore, Matrix Diag stands for the matrix diagonalization method.

Heuristic	***10 Groups of 15 PCBs***									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Matrix Diag	9	6	8	9	8	8	9	8	8	11

TABLE 3.3

3.5 The Maximal Intersection Minimal Union Method

The Maximal Intersection Minimal Union (henceforth MIMU) procedure was proposed by Tang and Denardo in [TD88b]. In this paper, they use this MIMU procedure in conjunction with another procedure to generate a branch and bound algorithm which uses the MSI criterion. As in their paper on the greedy

perturbation method, the authors do not deal with PCB assembly, but instead deal with machines that require tools rather than component reels to perform their operations. Also, they talk about producing parts rather than assembling PCBs. Because we wish to minimize the number of times at which component switches occur for a group of PCBs rather than the number of times at which tool switches occur for a group of parts, we must again consider a part to be a PCB and a tool to be a component reel. This branch and bound algorithm can be used to generate an optimal solution to the set-up/sequencing decision problem using the MSI criterion. However, as the number of jobs and required component types increase, computational space and time required by their branch and bound algorithm increase exponentially. As a result, we examine and implement only their MIMU procedure which provides a heuristic method for the set-up/sequencing decision problem using the MSI criterion.

Before specifying the steps required for the MIMU procedure, we will discuss how it works. This procedure is based on the concept of *classes*, where a class S is a set of jobs that require at most C components. As a result, the number of classes minus one determines the number of switching instants (recall that we do not include the initial component mix, see Section 2.1.3). Thus, the procedure's task is to try to minimize the number of classes. In doing this, there are two basic selection rules used when trying to expand a class S. These rules are as follows:

i) Select a job j not yet in any class for which the total number of component reels in common between job j and the class S is maximized;

ii) If there is a tie in i), select the job j from i) for which the number of component reels required by job j but not by the jobs in the class S is minimized.

Let us assume that J is the set of jobs we have to place in classes, and for each job $j \in J$, let T_j represent the set of component reels required by job j. Assuming R(S) represents the set of component reels required by the jobs in class S, and L(S) represents the set of jobs i not yet in any class such that $S \cup \{i\}$ is a class, we can state the rules as follows:

i) Select a job i from L(S) which maximizes $| R(S) \cap T_i |$;
ii) Break ties by selecting the job i from i) that minimizes $| T_i \setminus R(S) |$.

Because we are minimizing $| T_i \setminus R(S) | = | T_i \cup R(S) | - | R(S) |$, we can ignore the constant $| R(S) |$. As a result, we can either minimize $| T_i \cup R(S) |$ or maximize $- | T_i \cup R(S) |$ and still obtain the same results.

Given these rules that the MIMU procedure use, we can now describe the steps involved in this procedure.

Step 1: Set n = 0 and J = 1,...,N.

Step 2: **While |J| > 0 do the following**

Set n = n+1;

Pick a job $i \in J$ such that $|T_i|$ is maximized;

Set $S_n = \{i\}$ and $J = J \setminus \{i\}$;

Step 2a: **While $L(S_n) <> \varnothing$ do the following**

Select a job $i \in L(S_n)$ that maximizes $[\, |R(S_n) \cap T_i| \,,$

$-|R(S_n) \cup T_i| \,]$ lexicographically;

Set $S_n = S_n \cup \{i\}$ and $J = J \setminus \{i\}$.

End(*While*).

End(*While*).

Once this procedure has terminated, we are left with the jobs partitioned into classes where each class requires at most *C* component reels.

Note that the size of a problem can be reduced by simply ignoring redundant jobs (see Section 3.2) while generating the different classes. Once these classes are formed, the redundant jobs can be placed in the proper classes. By performing this simple task, computational time required to solve a problem can be reduced.

3.5.1 Example

To illustrate this MIMU procedure, we will use the following component/job matrix and assume that the feeder tray capacity (*C*) equals 4.

	Jobs				
Component Reels	**1**	**2**	**3**	**4**	**5**
a	1	0	0	1	1
b	1	0	1	0	0
c	1	0	1	1	0
d	0	1	0	0	0
e	0	0	0	1	0
f	0	1	1	0	1

Step 1: n = 0 and J = {1, 2, 3, 4, 5}.

Step 2: |J| = 5. Therefore, |J| > 0 is true.

Set n = 1.

Pick a job *i* that maximizes | T_i | over the set J.

	Job				
	1	**2**	**3**	**4**	**5**
$\|T_i\|$	3	2	3	3	2

In this example, there are three jobs (1,3,4) that could be chosen for job *i*. We will choose Job 1.

S_1 = {1} and J = J \ {1}. Therefore, J = {2, 3, 4, 5}.

Step 2a: $L(S_1)$ = {3, 4, 5}. Therefore, $L(S_1) <> \varnothing$ is true.

Maximize | $R(S_1) \cap T_i$ | over $L(S_1)$.

	Job		
	3	**4**	**5**
$\|R(S_1) \cap T_i\|$	2	2	1

Because we have a tie between Job 3 and 4, we maximize

$-|R(S_1) \cup T_i|$ over Job 3 and 4.

	Job 3	Job 4
$-\|R(S_1) \cup T_i\|$	-4	-4

Again, we are left with a tie. As a result, we can pick either job.

We will pick Job 3.

$S_1 = S_1 \cup \{3\}$. Therefore $S_1 = \{1, 3\}$

$J = J \setminus \{3\}$. Therefore $J = \{2, 4, 5\}$.

Step 2a: $L(S_1) = \{5\}$. Therefore, $L(S_1) <> \varnothing$ is true.

Maximize $|R(S_1) \cap T_i|$ over $L(S_1)$.

We only have one job to choose from, so we choose it.

We pick Job 5.

$S_1 = S_1 \cup \{5\}$. Therefore $S_1 = \{1, 3, 5\}$

$J = J \setminus \{5\}$. Therefore $J = \{2, 4\}$.

Step 2a: $L(S_1) = \varnothing$.

Step 2: $|J| = 2$. Therefore, $|J| > 0$ is true.

Set n = 2.

Pick a job i that maximizes $|T_i|$ over the set J.

	Job 2	Job 4
$\|T_i\|$	2	3

Job 4 maximizes $|T_i|$.

$S_2 = \{4\}$ and $J = J \setminus \{4\}$. Therefore $J = \{2\}$.

Step 2a: $L(S_2) = \varnothing$.

Step 2: $|J| = 1$. Therefore, $|J| > 0$ is true.

Set n = 3.

Pick a job *i* that maximizes $|T_i|$ over the set J.

	Job
	2
$\|T_i\|$	2

Job 2 maximizes $|T_i|$.

$S_3 = \{2\}$ and $J = J \setminus \{2\}$. Therefore $J = \varnothing$.

Step 2a: $L(S_3) = \varnothing$. Therefore, $L(S_3) <> \varnothing$ is false.

Step 2: $|J| = 0$. Therefore, $|J| > 0$ is false.

Algorithm terminates.

We have finished placing each of the jobs in J into classes. The resulting classes are as follows:

Class 1 contains Jobs 1, 3, 5, and uses component reels *a*, *b*, *c*, and *f*.

Class 2 contains Job 4, and uses component reels *a*, *c*, and *e*.

Class 3 contains Job 2, and uses component reels *d* and *f*.

3.5.2 A similar heuristic based on Bin-Packing

In the paper [TD88b], there were two references to the concept of bin-packing. The first reference was used to show that the problem we are trying to

solve is NP-hard, and the second stated that the MIMU selection rule is similar to the first fit decreasing (FFD) rule proposed in [GJ78] for the bin-packing problem. Because there were no comparisons made between the MIMU selection rule or the FFD rule, we decided to try modelling the problem as a bin-packing problem and use some of the different solution strategies for bin-packing to solve the problem. Before discussing how we modelled our problem as a bin-packing problem, we describe the bin-packing problem and some heuristic methods which have been developed for it [Baa88].

Given a set of object sizes $S = (s_1,...,s_n)$ where $0< s_i \leq 1$ for $1\leq i \leq n$, the *bin-packing problem* is to pack $s_1,...,s_n$ into as few bins as possible, where each bin has capacity 1. Three heuristic methods used for solving this problem are the first-fit, best-fit, and next-fit strategies.

The *first-fit* or *greedy* strategy places the next object in the set into the first bin into which it will fit, considering the bins from left to right. To illustrate this procedure, we have the following example. Let S = (0.6, 0.7, 0.2, 0.4). Then the first-fit strategy packs the objects into three bins, as shown in Figure 3.2.

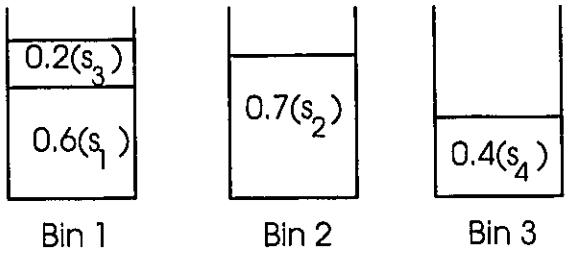


FIGURE 3.2

The *best-fit* strategy places the next object in the set into the bin which is the fullest of those bins into which the object will fit. To illustrate this procedure, we will use the same example as above. The best-fit strategy gives the solution shown in Figure 3.3. As can be seen, the best-fit strategy only uses two bins, and thus out-performs the first-fit strategy in this example.

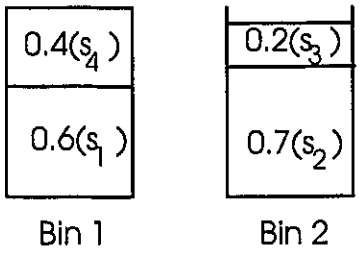


FIGURE 3.3

Finally, the next-fit heuristic places the next object in the set into the last bin generated. If it does not fit into this bin, a new bin is generated and the object is placed into this bin. To illustrate this procedure, we will use the same example as above. The next-fit heuristic packs the objects into three bins, as shown in Figure 3.4.

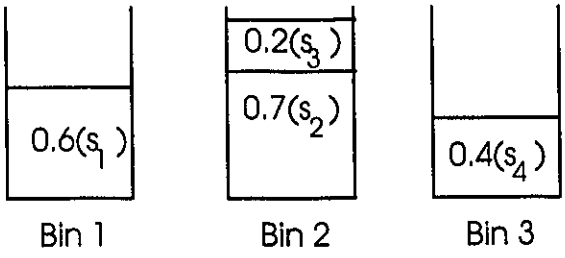


FIGURE 3.4

In order to model our problem as a bin-packing problem, we consider the bins to be feeder trays and the objects to be jobs. Placing an object into a bin then corresponds to placing the component reels needed for that job onto the feeder tray. We then let the bin capacity equal the feeder tray capacity C, and let S_i for an object be the number of component reels required for the corresponding job, where $0< S_i \leq C$, $1\leq i \leq n$. If there is no overlap of the components needed by the different jobs, we now have a modified bin-packing problem which can be solved using either of the three heuristic methods mentioned above. However, in reality, many jobs will have some component requirements in common. As a result, the amount of additional space required to add a job to a bin (thinking of a bin as a feeder tray) may be less than S_i, as some of the needed component reels may already be in the bin. Thus, we must keep a cost array for each bin which indicates the number of additional spaces each job will require given the current bin contents.

Example

To illustrate this procedure, let us assume that we have the following component/job matrix with feeder tray capacity (C) equal to 4. We will apply the best-fit heuristic strategy to solve the corresponding modified bin-packing problem.

Component Reels	Jobs 1	2	3	4	5
a	1	0	0	1	1
b	1	0	1	0	0
c	1	0	1	1	0
d	0	1	0	0	0
e	0	0	0	1	0
f	0	1	1	0	1

To begin, we create a bin called bin 1. The space required or "cost" of adding each of the five jobs to this empty bin is indicated in the following table.

Bin 1

JOB	COST
Job 1	3
Job 2	2
Job 3	3
Job 4	3
Job 5	2

Suppose that we add Job 1 to this bin. This means bin 1 contains component reels *a*, *b*, *c* and has one free slot left. We now have to recalculate the cost of adding the remaining jobs to this bin. Because component reels *a*, *b*, and *c* are already contained in the bin, the new costs are as follows:

Bin 1

JOB	COST
Job 2	2
Job 3	1
Job 4	1
Job 5	1

Next, we must place Job 2 into a bin. Job 2 requires two free slots and thus cannot fit into bin 1 which has only one free slot. Thus, we must create a new bin, bin 2, and place Job 2 into it. Bin 2 now contains component reels *d* and *f*, and has two free slots left. The cost of adding each of the remaining jobs to either bin is indicated below.

Bin 1

JOB	COST
Job 3	1
Job 4	1
Job 5	1

Bin 2

JOB	COST
Job 3	2
Job 4	3
Job 5	1

The best-fit heuristic states that we must look for the fullest bin into which Job 3 will fit. Bin 1 is the fullest bin into which Job 3 will fit, so we now place Job 3 into it. Therefore, Bin 1 contains component reels *a*, *b*, *c* and *f*, and is now full. The new cost for adding the remaining jobs to the bins are:

Bin 1

JOB	COST
Job 4	1
Job 5	0

Bin 2

JOB	COST
Job 4	3
Job 5	1

Again, we find the fullest bin into which Job 4 will fit. Job 4 will not fit into either bin, and so we create bin 3, and place Job 4 into it. Thus, Bin 3 contains component reels *a*, *c*, and *e*. The only remaining job to be placed in a bin is Job 5. The cost of adding this job to any of the bins is indicated below:

Bin 1

JOB	COST
Job 5	0

Bin 2

JOB	COST
Job 5	1

Bin 3

JOB	COST
Job 5	1

As can be seen from the above costs, Job 5 can be added to bin 1 with 0 cost. Therefore, Job 5 is added to bin 1. We are finished placing all the jobs into bins. The final results, using three bins, are as follows:

Bin 1 contains Jobs 1, 3, 5, and component reels *a*, *b*, *c*, and *f*.

Bin 2 contains Job 2, and component reels *d* and *f*.

Bin 3 contains Job 4, and component reels *a*, *c*, and *e*.

3.5.3 Our improvements to the Bin-Packing Heuristics

We have mentioned three different heuristics which can be used to solve the bin-packing problem. However, the next-fit heuristic can do much worse than the other two heuristics [Baa88]. As a result, we implement only the first-fit and best-fit heuristics.

To improve the performance of the first-fit and best-fit heuristics mentioned in Section 3.5.2, we tried varying the order in which the jobs were processed. Intuitively, the jobs to be placed in bins early should be the ones that will take up a lot of the space in the bins, i.e. the big jobs. This allows the jobs that take up smaller amounts of space, i.e. the small jobs, to fit into the space left over by the big jobs. If this were not the case and the big jobs were placed last, then each big job would probably not fit into any existing bins and the creation of new bins would be necessary. To incorporate this improvement into our implementation, we needed a criterion to determine which jobs were considered big and which were considered small. The criterion we used when determining the processing sequence for the jobs was based on a counting technique. This counting technique is very simple. For each job, we consider its size to be the number of component types it requires. We then process the jobs from largest to smallest according to their size.

In addition to the counting technique, we thought of an alternate way to determine the order in which to process the jobs. Considering the function of each bin, our goal is to place as many jobs in it as possible. To reach our goal, we must group in the same bin the jobs that have high overlaps of component types. By doing this, we hope to maximize the number of jobs in each bin and thus

minimizing the number of bins. Consequently, we calculate the overlap between each job and all other jobs. Then we process the jobs from largest to smallest according to their overlap values.

3.5.4 Implementation results

As mentioned earlier, one of the reasons we examined the bin-packing strategies was to compare the first-fit decreasing heuristic with the MIMU selection rule. Because no comparisons had been made between any bin-packing heuristic and the MIMU heuristic, we decided to implement both the first-fit and best-fit heuristics for the modified bin-packing problem. The results of the first-fit and best-fit heuristics were very similar, so we only compare the first-fit and MIMU heuristic. We applied these algorithms to our test set and tabularized our results as shown in Table 3.4. Note that the number in each array element is the number of times component switches must occur. Furthermore, F-Fit, and MIMU represent the first-fit, and the maximal intersection minimal union heuristic, respectively. Overlap and counting identify which order the jobs are processed in, as described in Section 3.5.3.

Heuristic	***10 Groups of 15 PCBs***									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
F-Fit overlap	9	6	8	9	8	8	8	8	8	11
F-Fit counting	9	6	8	9	8	8	8	8	8	11
MIMU	9	6	8	9	8	8	8	8	8	11

TABLE 3.4

As can be seen, all the techniques gave the same results for the MSI criterion. In any case, the computational time required for these techniques is minimal, and thus the execution of all these methods is a practical solution.

3.6 Summary of results

Tables 3.5 and 3.6 below give a summary of our results for the MCS and MSI criteria, respectively. As can be seen from these tables, the method which gave the best results for our test set was the improved greedy perturbation method for the MCS criterion, and each of the grouping techniques for the MSI criterion performed equally well.

Heuristic	*10 Groups of 15 PCBs*									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Greedy Perturb	159	114	144	151	138	138	156	139	131	149
Labelling Method	187	119	160	163	157	171	152	152	146	165

TABLE 3.5

Heuristic	*10 Groups of 15 PCBs*									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Matrix Diag	9	6	8	9	8	8	9	8	8	11
F-Fit overlap	9	6	8	9	8	8	8	8	8	11
F-Fit counting	9	6	8	9	8	8	8	8	8	11
MIMU	9	6	8	9	8	8	8	8	8	11

TABLE 3.6

Chapter 4

Minimizing both the MCS and MSI criteria

4.1 Introduction

In Chapter 3, we looked at different methods for solving the set-up/sequencing decision problem using either the MCS (number of component switches) performance criterion or the MSI (number of switching instants) performance criterion. For each of these methods, we can calculate the performance for the other criterion not being used. If we are using the MCS criterion, we can simply count the switching instants in the resulting solution to obtain a solution to the MSI criterion. If, on the other hand, we are using the MSI criterion, we can consider each group (or class) in the solution to the MSI criterion as a job, where the component reels for that job are the component reels required by the group. We can then use the order in which the groups were formed as the job sequence, and apply the KCNS procedure, as described in Section 3.2, on this job sequence. This will determine a solution for the MCS criterion. Using our test set, we report on the results of both criteria for each method. Tables 4.1 and 4.2 tabulates the results for the MCS and MSI heuristics, respectively. Note that the top number in each array element represents the number of times component switches must occur, while the number below it represents the number of component switches necessary.

Heuristic	*10 Groups of 15 PCBs*									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Greedy Pert	14 159	12 114	13 144	14 151	12 138	14 138	13 156	13 139	13 131	14 149
Labelling	12 187	11 119	10 160	11 163	12 157	12 171	10 152	10 152	12 146	12 165

TABLE 4.1

Heuristic	*10 Groups of 15 PCBs*									
MatD (KCNS)	9 198	6 149	8 185	9 185	8 156	8 175	9 178	8 173	8 162	11 177
F-Fit overlap (KCNS)	9 207	6 156	8 170	9 176	8 180	8 162	8 180	8 167	8 150	11 189
F-Fit counting (KCNS)	9 222	6 163	8 177	9 215	8 196	8 196	8 186	8 167	8 187	11 199
MIMU (KCNS)	9 222	6 162	8 177	9 215	8 196	8 196	8 183	8 167	8 187	11 199

TABLE 4.2

As can be seen from the tables above, the methods typically performed badly with respect to the other criterion.

In some situations, set-up time is actually determined by both the number of component switches and the number of times at which component switches occur. For example, consider the case where four component switches can occur in parallel, and there are eight switches that must occur. Obviously, a solution where the machine is stopped only twice to perform the required component switches is better than one that stops the machine eight times. Furthermore, if we minimize the

number of component switches that must occur each time the machine is stopped, then set-up time will be further reduced. As a result, both the MSI and MCS criteria become important. In such situations, it is desirable to have a solution for the set-up/sequencing decision problem which performs well with respect to both the MCS and the MSI criteria. In this chapter, we discuss two methods which we have developed for doing this. We also report the results of implementing and comparing these methods on our test set.

4.2 Method 1: Begin with a solution for the MSI criterion, and reorder the groups

A solution to the set-up/sequencing decision problem using the MSI criterion partitions the set of jobs to be sequenced into groups (or classes), where all the jobs in one group are built using a common feeder tray assignment. Thus, the number of component switches necessary to complete all jobs is completely dependent upon the order the groups are processed. To find an ordering which requires a low number of component switches, we can consider each group (or class) in the solution for the MSI criterion as a job, where the component reels for that job are the component reels required by the group. We then apply the greedy perturbation method, as described in Section 3.2, on these new jobs. The ordering of these jobs then corresponds to an ordering of the groups. By processing the groups in this order, we will have a job sequence with the same solution for the MSI criterion, but a hopefully much improved solution for the MCS criterion.

4.3 Method 2: Begin with a solution for the MCS criterion, then group according to this job ordering

A solution to our problem using the MCS criterion gives us a job ordering with a hopefully low number of component switches. If we maintain this ordering while grouping jobs for the MSI criterion, we can hopefully obtain a solution which performs well relative to both the MCS and MSI performance criteria. To do this, we can use bin-packing with the next-fit heuristic, as described in Section 3.5.2, where the initial ordering of the jobs is given by some MCS criterion heuristic, such as the greedy perturbation method. Recall that the next-fit heuristic places the next object in the ordering into the last bin generated. If it does not fit into this bin, a new bin is generated and the object is placed into this bin. By using the next fit heuristic, we keep the general ordering of the jobs so that component switching results are not affected. Of course, we may not obtain the best results for the number of times component switches must occur, but we are trying to minimize both the MCS and MSI criteria.

Once we have generated the groups, we can further reduce the number of component switches by applying the method described in Section 4.2, where each group is considered as a job and then the greedy perturbation method is applied to these new jobs.

4.4 Comparison of methods

To see how the different methods compare, we ran them on our test set and tabularized our results for each method in Table 4.3. In this table, we show the results for method 1 when bin-packing, matrix diagonalization, and maximal

intersection minimal union are used to generate the groups, and we show the results for method 2 when greedy perturbation is used to generate the initial job ordering. The top number in each array element represents the number of times component switches must occur, and the number below it represents the number of component switches necessary.

Heuristic	*10 Groups of 15 PCBs*									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
F-Fit overlap	9 174	6 135	8 163	9 161	8 146	8 148	8 161	8 149	8 143	11 162
F-Fit counting	9 168	6 134	8 155	9 164	8 146	8 147	8 162	8 147	8 137	11 162
Matrix Diag	9 172	6 134	8 157	9 161	8 146	8 154	9 159	8 150	8 138	11 162
MIMU	9 168	6 137	8 155	9 163	8 146	8 147	8 165	8 147	8 137	11 162
Method 2 GP	10 169	6 125	8 151	10 146	9 138	9 141	8 155	8 150	8 136	11 152

TABLE 4.3

As can be seen from Table 4.3, method 2 gave the best overall results with respect to both criteria. When considering the minimization of both performance criteria simultaneously, we do not expect to obtain optimal results for each of the performance criterion, however, we do hope to obtain results that are not too far off. If we can accomplish this, our new methods would be very useful when the set-up time is dependent on both performance criteria. If we examine the results for method 2 for the MSI or MCS criterion, we can see that the solutions obtained perform almost as well as our best results for these criteria found in Chapter 3. In fact, on average we are only 5% away from our best solutions for the MSI criterion, and 3% away from our best results for the MCS criterion.

Chapter 5

Lower bounds for the set-up/sequencing decision problem

5.1 Introduction

If we can show that the set-up time corresponding to any solution for the set-up/sequencing decision problem is at least some value k, then k is called a *lower bound* for the problem. Good lower bounds for a problem play an important role when trying to determine how well heuristic solution strategies perform. If the lower bound and the solution obtained for a particular problem are very close, then it follows that the solution is near optimal, i.e. "good". Note, however, that if the lower bound is much smaller than the solution obtained, then we cannot conclude anything about the "goodness" of the solution. It may be that either the solution is poor, or the lower bound is poor.

In the following sections, we examine methods for obtaining lower bounds which have already been developed for the set-up/sequencing decision problem for both the MCS and MSI criteria, and we discuss ways of strengthening those bounds. We also use these bounds to assess the performance of our best results from Chapter 3 and Chapter 4.

5.2 Lower bounds for the MCS criterion

Given a list J of PCB jobs, let S be the set of all component types which are required to process the list J. Clearly, in any solution for the set-up/sequencing decision problem for J, any component type in S which is not in the initial component mix must be switched onto the feeder tray at least once during the processing of the jobs in J. Thus, an obvious lower bound [BS93] for the number of component switches which must occur in any solution is $[\,|S| - C\,]^+$.

Another, not so obvious, lower bound can be obtained by using Tang and Denardo's method for generating a job sequence. As described in Section 3.2, a job sequence is generated by creating a graph and obtaining a minimum weight Hamiltonian path on this graph, where each node in the graph represents a job, and the weight on an edge *ij* represents the minimum number of component switches required to process the node pair *ij* sequentially. It follows that the length of this minimum weight Hamiltonian path is a lower bound on the number of component switches.

To see how these lower bounds compare relative to each other, we generate the lower bounds on our test set. To find the minimum weight Hamiltonian paths, we change each problem into a TSP problem, as described in Section 1.4.3, and then use the Lin-Kernighan heuristic for TSPs. The results are shown in Table 5.1. Note that for Tang and Denardo's method, we are assuming that our solution technique for finding a minimum weight Hamiltonian path is in fact finding optimal solutions, otherwise, we could not consider the weights of these Hamiltonian paths as lower bounds. Luckily, the performance of the Lin-Kernighan method has been shown empirically that it is very good [LLRS86].

Heuristic	*10 Groups of 15 PCBs*									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
LB on switches ($[\|S\| - C]^+$)	125	110	110	119	121	114	115	112	110	123
LB on switches ([TD88a])	40	7	26	22	47	42	32	31	17	32

TABLE 5.1

As can be seen from Table 5.1, the simpler lower bounding technique, i.e. using $[\ |S| - C\ ]^+$, obtained much better lower bounds for these examples than the ones generated as a consequence to Tang and Denardo's method for generating a job sequence.

5.2.1 Improving the lower bound

So far, the lower bound which seems to work the best for the number of component switches is achieved when using the formulation:

$$LB = [\ |S| - C\ ]^+.$$

Note that this lower bound LB assumes that each component type i is switched on at least w times, where w is 0 if i is in the original component mix, and w is 1 otherwise. If it is possible to show that for some group of component types, at least y of them must be switched on and off at least $w+1$ times, then we can improve our lower bound to LB+y. For example, given a list J of jobs to be processed, consider the situation in which three of the jobs, call them jobs i, j, and

k, have the number of component types required and component overlaps as illustrated in Figure 5.1.

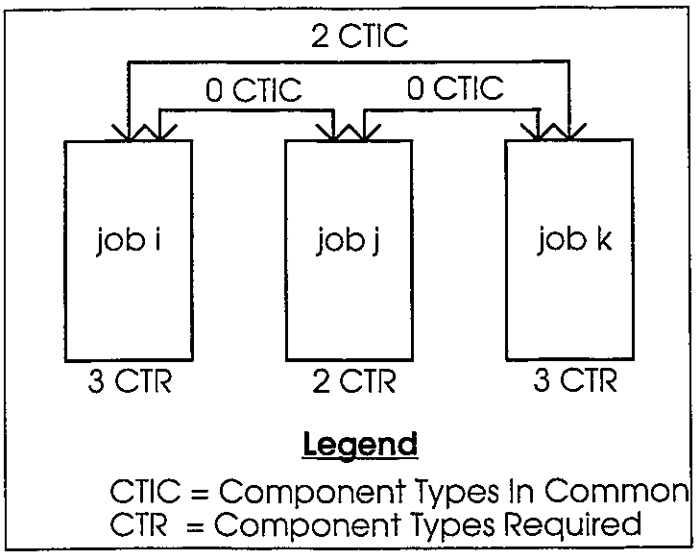


Figure 5.1

Let us assume the feeder tray capacity is three. As can be seen above, job i requires three component types: two of these are also required by job k, and none are required by job j. If we consider a job sequence in which these three jobs appear in the order i, j, k (perhaps with other jobs in between), we know that two component types used by job i will also be required by job k. However, job j which is assembled between job i and job k requires two unique component types. As a result, there is only one component type from job i that can be left on the feeder tray while processing job j. This means that at least one component type that is common to both job i and job k must be switched onto the feeder tray when job k is processed. If this component type had been in the original component mix, then we see that it must be switched on at least once, otherwise it must be switched on at least twice. In either case, this means an extra switch beyond what was

calculated for LB is needed. Thus, we know that if the jobs occur in the order i, j, k we can strengthen the lower bound to be LB+1.

Of course, when calculating a lower bound, we cannot impose a particular ordering on any three jobs i, j, and k. Thus, we must calculate the number of extra component switches required for the six possible job sequences: i,j,k, k,j,i, i,k,j, j,k,i, j,i,k, k,i,j. Notice that the number of extra component switches required for job sequences k,j,i, j,k,i, and k,i,j are the same as the number of extra component switches required for job sequences i,j,k, i,k,j, and j,i,k, respectively. As a result, we perform the calculation on only the three job sequences i,j,k, i,k,j, and j,i,k. By performing these three calculations, we can determine the minimum number of extra component switches necessary to assemble jobs i, j, k in any sequencing order. Once this minimum value has been determined, we can increase our lower bound (LB) by this value.

Recall that S_i represents the set of components required by job i. Assuming that we are processing the job sequence i,j,k, we calculate the number of extra component switches required by doing the following. First, we must ensure that the number of component types required by the pairs of jobs i,j, i,k, and j,k are all greater than the feeder capacity (i.e. $|S_i \cup S_j| > C$; $|S_i \cup S_k| > C$; $|S_j \cup S_k| > C$). If this were not the case, it would be impossible with these three jobs to show that a component type has to be switched on the feeder tray an extra time since two component mixes could be used to process all three jobs. Once we have done this, we must calculate the number of component types in common between jobs i and k, but not required by job j, and from this we must subtract the number of feeder tray slots not required by job j (i.e. $|(S_i \cap S_k) \setminus S_j| - (C - |S_j|)$). The resulting value tells us the number of component reels which must be switched off to process job

j, then switched back on to process job k. This is the number of extra component switches that has to occur when using the job sequence i,j,k.

For one of the three job sequences (i,j,k, i,k,j, and j,i,k) with this minimum value, we must ensure that all the component types that could be switched onto the feeder tray an extra time are not used again to calculate extra switches. This step is necessary because we cannot determine in all cases which component types will be switched the extra time (i.e. if one of two component types has to be switched onto the feeder tray an extra time, we do not know which one it will be). Following this, we can search for another three job combination and repeat the procedure.

More formally, the algorithm which we developed to find such extra component switches can be described as follows:

Component Seek

Step i) Mark all component types as unused.

Step ii) Compute the lower bound using the formulation :

$$LB = [\, |S| - C \,]^{+}.$$

Step iii) For each three job combination (i,j,k) do the following:

Check to see if $|S_i \cup S_j| > C$; $|S_i \cup S_k| > C$; $|S_j \cup S_k| > C$.

If so

Step iiia) Calculate the minimum number of extra component switches necessary for the three job sequences i,j,k, i,k,j, and j,i,k

$$|(S_i \cap S_k) \setminus S_j| - (C - |S_j|);$$

$$|(S_i \cap S_j) \setminus S_k| - (C - |S_k|);$$

$$|(S_j \cap S_k) \setminus S_i| - (C - |S_i|);$$

Step iiib) Increase the lower bound by the minimum value obtained from step iiia).

Step iiic) For one of the job sequences that achieved the minimum value in step iiia), mark all the component types that were used to achieve this minimum value.

Of course, in the above method, the order in which the three job combinations are considered matters. To further improve our lower bound, we compare the lower bound from the above method with a lower bound obtained from a different order, and then take the maximum of the two as the lower bound. This second ordering is found as follows: In each iteration of the above algorithm, we search for the three job combination which has the largest minimum value in Step iiia). By taking this combination, we attempt to keep our minimum value equal to the number of component types to be marked as used. Hence, we try to avoid the situation where we increase our lower bound by a small amount but have to mark a large number of component types as used. Following this selection, we perform Steps iiib) and iiic) on these three jobs. The entire process is then repeated until no job combination can improve our lower bound.

5.2.2 Complexity of the algorithm Component Seek

Without the improvement

Both Step i) and Step ii) can be done in constant time, while Step iii) can be done in $O(n^3)$ time, where n is the number of jobs. Each substep within Step iii) can be done in constant time. As a result, the algorithm can be run in $O(n^3)$ time.

With the improvement

If we add the improvement to always search for the three job combination which has the largest minimal value, the complexity of the algorithm changes. Steps i) and ii) can still be done in constant time. However, Step iii) must loop n^3 times while each pass through the loop takes $O(n^3)$ time. Consequently, the complexity of the algorithm now becomes $O(n^6)$.

5.2.3 Implementation results

In Table 5.2, we can compare our lower bounds with the best solutions obtained for the MCS criterion on our test set. This will give us an indication on how close we are to the optimal solution.

Heuristic	***10 Groups of 15 PCBs***									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Best Results	159	114	144	146	138	138	152	139	131	149
LB on MCS (types-C)	125	110	110	119	121	114	115	112	110	123
Component Seek	127	110	115	120	122	116	116	117	111	124
Component Seek (+ order)	133	110	115	120	122	122	122	117	111	129
Percentage Gap	16%	4%	21%	18%	12%	12%	20%	16%	16%	14%

TABLE 5.2

As can be seen from Table 5.2, the algorithm Component Seek improves the simple lower bound LB in nine of the ten groups. If we use the second improvement where we always choose the three job combination which generates the largest minimal value, we further improve our lower bound in four of the ten groups. Also given above are percentage gaps between our best lower bounds and our best solutions obtained for each of the ten groups. This gap is calculated using the following formula:

$$\% \text{ Gap} = \frac{\text{Best solution - Best lower bound}}{\text{Best solution}} * 100 .$$

If we now calculate the average percentage gap for all ten groups, we obtain a 15% gap. As a result, we can say that we are on average no more than 15% away from the optimal solution.

5.3 Lower bounds for the MSI criterion

An obvious lower bound for the number of times component switches must occur is to take the ceiling of the total number of component types required in the production list divided by the feeder tray capacity [TD88b]. Because we do not include the loading of the initial feeder tray as a switching instant (see Section 2.1.3), our lower bound becomes the following:

$$LB = \left\lceil \frac{\text{\# of component types required in production list}}{\text{C (the feeder tray capacity)}} \right\rceil - 1$$

A lower bound that generally does better than the one above was proposed by Tang and Denardo in their paper [TD88b]. In their paper, they talk about jobs

which are compatible. Two jobs *i,j* are *compatible* if all the required component reels for both jobs can be placed on the feeder tray simultaneously (i.e. $|S_i \cup S_j| \leq C$). Obviously, each job is compatible with itself since no job requires more than C component types. A *compatibility matrix* can now be created to specify which jobs are compatible. For example, if job i is compatible with job j then the $(i,j)^{th}$ array element in the compatibility matrix will be one, otherwise it will be zero. Furthermore, *B(i)* will represent the set of jobs which are compatible with job i. Given these definitions, the *sweeping procedure* which generates the lower bound is as follows:

k = 1; J = {1,...,N};
While |J| <> 0 do the following:
    Set S[k]=∅;
    Compute the compatibility matrix;
    Select a job $i \in J$ that minimizes $[|B(i)|, |\cup_{j \in B(i)} S_j|]$
        lexicographically over J
    J = J \ B(i); S[k] = B(i); k = k+1;
LB = k-1

Note that the lower bound (LB) is equal to the number of groups-1 because the loading of the initial feeder tray is not considered a switching instant.

5.3.1 Implementation results

In this section, we compare the two lower bounding techniques with the best results found in Chapter 3 and Chapter 4 for the set-up/sequencing decision problem using the MSI criterion on the test set.

10 Groups of 15 PCBs

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Best Results	9	6	8	9	8	8	8	8	8	11
LB on MSI (Ceiling)	3	3	3	3	3	3	3	3	3	3
LB on MSI (Compat)	9	6	8	9	8	8	8	8	8	11

Table 5.3

As can be seen in Table 5.3, the lower bounding technique which uses a compatibility matrix (Compat) generated better lower bounds in all ten groups and in fact proves our best solutions are optimal for the entire test set. However, the simpler lower bounding technique (Ceiling) can do better in certain situations. Take for example, a situation where every job is compatible with every other job (i.e. C is very large). In this case, the lower bound obtained from using the compatibility matrix would be zero. On the other hand, the simpler lower bounding technique could not do any worse and would perform better if there were at least C+1 component types required in the production list. For example, if we double our C (feeder tray capacity) value and rerun a grouping heuristic like best-fit and our lower bounding techniques for the MSI criterion on group 1, the results we obtain are as follows:

Heuristic	*1 Group of 15 PCBs*
	Group 1
Best-Fit (Overlap)	2
LB on MSI (Ceiling)	1
LB on MSI (Compat)	0

Table 5.4

As can be seen, the method which uses the compatibility matrix (Compat) does worse than the simpler lower bounding technique (Ceiling). As a result, Tang and Denardo take the maximum of the simpler lower bounding technique and the lower bound computed by the sweeping procedure.

5.4 Summary of results

To see how good our solutions from Chapter 3 and Chapter 4 are, we compared our best solutions with our best lower bounds. From Table 5.2, we can conclude that for our test set we were on average no more than 15% away from the optimal solution to the set-up/sequencing decision problem using the MCS criterion. Furthermore, from Table 5.3, we can conclude that for our test set optimal solutions for the set-up/sequencing decision problem using the MSI criterion were obtained. Optimal solutions for the set-up/sequencing decision problem using the MSI criterion are not guaranteed when using the grouping heuristics we studied in Chapter 3 and Chapter 4, but they all seemed to work well

even when we manipulated the size of C. However, the solutions obtained for the MCS criterion differ substantially from one heuristic to another.

Chapter 6

Adapting heuristics to fit particular manufacturing environments

6.1 Introduction

The solution strategies that we have described so far for the set-up/sequencing decision problem are for the idealized model, as described in Chapter 2. If we consider applying these strategies to a particular machine, it is likely that some adaptations will be needed. In this chapter, we investigate adapting these methods to other manufacturing environments, and in particular to the environment at the Mitel Corporation in Ottawa. At Mitel, there are some component placement machines which are specially allocated for building numerous small volume jobs. One such machine is the Panasonic Mv-II (LL) model NM-2559B. This machine differs from our idealized model in several important ways, with the consequence that we can no longer switch any chosen pair of component reels, which has been assumed previously. Here, we give an overview of these differences, and then describe possible strategies for adapting our methods to overcome these differences.

6.2 An overview of the differences between the idealized model and the Panasonic Mv-II

6.2.1 Multiple component reel sizes, and feeder complications

In our idealized model, we assumed that every component reel occupies a single feeder tray slot. However, due to the different component sizes that exist, component reels may require more than one slot in the Mv-II (LL) feeder tray. The reel size is determined by the tape width required to hold that component type. Some of the more common reel sizes (taping widths) are as follows: 8, 12, 16, 24, 32, and 44mm. For each reel size, there is a corresponding feeder that is required to hold it, and it is these feeders which are placed into the feeder tray slots. An 8mm reel size requires an 8mm feeder, and so on.

In some cases, a single feeder tray slot can be used to hold two of the smallest sized (8mm) reels by using a *double component feeder*, which is a feeder that can hold two 8mm component reels. Using double component feeders is very beneficial because they can double the capacity of the feeder tray without increasing its size. Using these double feeders also reduces feeder tray travel distance. The actual number of double component feeders which are used in a solution depends on the hardware resources available. Typically, there are not enough double component feeders to hold all the 8mm components required, in which case single component 8mm feeders must be used for the rest.

To further complicate matters, there are two different component tape types that can be used for a component reel: paper and embossed. Because of these two tape types, we have two different component reel types, each requiring its own

special feeder type. As a result, each feeder size has feeders for both the paper and embossed component reels. Note that there are also two types of double feeders, one for embossed reels and one for paper reels, and thus both reels on a double feeder must be of the same tape type.

6.2.2 Dead space requirements between feeders

Due to the different feeder sizes, the feeders sometimes cannot be placed directly beside one another on the feeder tray. The space required between two feeders on the feeder tray is called the *dead space*, and is dependent on the sizes of the two feeders. We refer to this dead space as the number of unused slots. A listing of the dead space requirements between one feeder size and all other feeder sizes is called the *spacing requirements* for that particular feeder size. Note that feeder type does not affect the spacing requirements. However, double component feeders do have different spacing requirements than the single component feeders of identical size.

6.2.3 Rib restrictions on the feeder tray

In addition to the problems caused by the different feeder sizes and types, the feeder tray requires numerous reinforcement ribs to support it. These ribs place additional restrictions on where certain sized feeders can be placed.

6.3 Strategies for adapting our methods to the Panasonic Mv-II machine

In this section, we will discuss how we would overcome the problems described in Section 6.2.

6.3.1 Adapting to different component reel sizes and feeder complications

Due to the different sizes and tape types of component reels used, we can no longer switch any chosen pair of component reels with each other. Thus, in order to apply the heuristics from Chapter 3 and Chapter 4, we must determine how we will perform component switches. Obviously, if we could always switch an 8mm paper reel with an 8mm paper reel, or a 12mm embossed reel with a 12mm embossed reel, etc., we would have no problems when performing component switches. Thus, one possible strategy for adapting our current methods to multiple reel sizes and types would be to ensure that if a component switch is required, the component reel which is removed from the feeder tray is the same size and tape type as the one to be added. To ensure that this is always possible, we could calculate the maximum number of feeders of each size and type required by any single job in the production list. Once this maximum number is determined for a particular feeder size and type, we could ensure that this number is placed into the feeder tray. These feeders which must be placed into the feeder tray to avoid component switching problems will be called the *required feeders* for a production list.

One problem that arises with the above solution strategy is that it greatly restricts the selection of component reels which can be switched off the feeder tray at each step of the KCNS procedure, which may result in a high number of

component switches in the corresponding solution. Another problem that arises is that there may not be enough feeder tray capacity to accommodate all the required feeders for a production list, due to the fact that we are including the maximum number required for so many different sizes and types of component reels.

To alleviate the above problems, we can improve our solution strategy by observing that it is possible to group certain component reel sizes and types together. First note that instead of switching pairs of component reels, we can switch pairs of feeders in the case of single component feeders. Thus, a component reel which must be placed into the feeder tray can first be placed onto its feeder, then this feeder can be switched with any feeder currently on the feeder tray which has the same spacing requirements, regardless of the size or tape type of the component reel it holds. (Of course, in the case of an 8mm reel, we can also switch the reel with any 8mm reel on a double feeder which uses the same tape type.) Since several feeder sizes may have the same spacing requirements, as do the two types (paper and embossed) of feeders of each size, we now have a much larger choice for switching during the KCNS procedure, which will reduce the number of switches in the corresponding solution. Moreover, it may be possible to perform some switches more quickly than before, since a component reel to be switched onto the feeder tray can be loaded onto its feeder before the switch is to occur.

More precisely, let a *feeder set* for a particular component reel be the set of all single component feeders which have the same spacing requirements as the single component feeder which holds that component reel. Then the adapted form of KCNS is as follows:

For the given component reel to be placed onto the feeder tray, calculate the components currently on the feeder tray which are either

1) on a double component feeder and on a reel of the same tape type and size, or
2) on a single component feeder in the same feeder set as the feeder required by the component coming on.

Of these components, KCNS will choose to switch the component in that set which is required furthest in the future. The component being switched on is loaded onto its feeder and the feeders are switched, unless the component being switched off is on a double feeder, in which case the reels are switched.

By grouping the component reels into feeder sets, we can also reduce the number of required feeders for a production list. These required feeders can now be calculated as follows:

1) For 8mm component reels, it is advantageous (capacity-wise) to use as many double feeders as possible. So we begin by calculating
   MaxP:= the maximum number of 8mm paper component reels required by any job in the production list, and
   MaxE:= the maximum number of 8mm embossed component reels required by any job in the production list.
   Let PD be the number of 8mm paper double feeders available.
   Let ED be the number of 8mm embossed double feeders available.
   Now place MP:=Min(PD, $\lfloor$(MaxP/2)$\rfloor$) paper double feeders into the feeder tray and ME:=Min(ED, $\lfloor$(MaxE/2)$\rfloor$) embossed double feeders into the feeder tray.

2) Reduce the number of 8mm paper reels and 8mm embossed reels required by each job by 2*MP, and 2*ME, respectively (of course, not going below zero).

3) For each job, partition the components into their respective feeder sets, and find the total number of required components for each feeder set.

4) For each feeder set, find its maximum requirement over all the jobs. Make sure there is space allocated for that many feeders for each set in the initial component mix.

To see if this solution to the component switching problem would work on the 26 jobs given to us by Mitel, we calculated the required feeders for the 26 jobs. Recall that these are the feeders which must be placed into the feeder tray to avoid component switching problems. Once this was done, we placed all the required feeders on the feeder tray so that they would be arranged in order of increasing size. When we were done, just over half of the feeder tray capacity was used. Hence, there was plenty of feeder tray capacity for the required feeders. If this were not the case and all the required feeders could not fit into the feeder tray, we would have to split our production list into groups so that all the required feeders for each group could fit into the feeder tray. A method for doing this is discussed in Section 6.5.

6.3.2 Adapting to dead space requirements

Whenever we are switching feeders, the dead space requirements could become a problem. However, in Section 6.3.1, we have discussed a method which ensures that only feeders for component reels in the same feeder set are exchanged. Since only feeders with the same spacing requirements are switched, no feeder switching problems can occur.

To try to minimize dead space, which is the number of slots lost due to the spacing required between different feeder sizes, we must place the feeders in order of increasing or decreasing size. This will maximize the number of feeders (component reels) which can fit into the feeder tray. For example, if we have two 8mm and one 44mm feeder to be placed into the feeder tray, we could place the two 8mm feeders adjacent to one another and the 44mm feeder beside these 8mm feeders (increasing order). Taking into consideration the spacing requirements, this arrangement loses one slot to dead space and requires a feeder tray capacity of four. However, if we mix the feeder sizes on the feeder tray so that the 44 mm feeder separates the two 8mm feeders, one additional slot is lost. As a result, this feeder tray arrangement requires a feeder tray capacity of five. See Table 6.1 for the feeder tray arrangements.

Slot Number	Feeder Size	Feeder Size
1	8mm	8mm
2	8mm	dead space
3	dead space	44mm
4	44mm	dead space
5	unused	8mm

TABLE 6.1

In the case of the Panasonic Mv-II (LL) machine, if we place feeders in order of increasing size, we not only maximize the number of feeders which can be placed into the feeder tray but also reduce the feeder tray travel distance required when assembling a PCB. The reduction in travel distance is achieved by this simple ordering because of how the Mv-II machine operates. It assigns each component type to one of eight speed groups and then places all the component types in the fastest speed group onto the PCB first, followed by the next fastest group, etc. Obviously, if we wish to minimize feeder tray travel distance, feeders containing component types from the first speed group should be placed into the feeder tray first, followed by the second speed group, and so on. Because each speed group contains components which are close in size, shape, and volume, feeders within each speed group are generally one size. In addition, as we go from speed group one to speed group eight, component types within each speed group tend to become larger. Consequently, placing the feeders in order of increasing size tends to reduce the feeder tray travel distance required for a PCB.

6.3.3 Adapting to handle rib restrictions

Avoiding the problem of placing feeders where they cannot go because of reinforcement ribs is easy to solve. We simply place restrictions on where certain feeders cannot be placed when applying the different heuristics from Chapter 3 and Chapter 4.

6.3.4 Generating an initial component mix and the required component switches for a particular job sequence

From Section 6.3.1, we can determined the number of 8mm double component feeders for each tape type to be placed into the feeder tray, and the space to be allocated in the feeder tray for each feeder set. Using the job sequence, we can determine which feeders in each feeder set are to be placed into the feeder tray and which component reels will be used to fill these and the double component feeders. Component types required the soonest are placed onto the feeders first. If we run out of a particular feeder in the feeder tray, we recalculate, according to the order of increasing feeder size, the slots required by all the feeders in the tray plus the new one. If the new feeder fits, it is added to the feeder tray. Otherwise, we simply continue to try to add feeders with the component reels required the soonest until the feeder tray is full (i.e. no new feeder can be placed into the feeder tray). This feeder tray will now contain our initial component mix. Although this mix is unlikely to have C component reels, we will still assume that the time required to place the initial component reels onto the feeder tray is constant for all solutions. As a result, this time will not be included into the set-up time.

Once we have generated our initial component mix, we can use the modified KCNS method, as described in Section 6.3.1, to determine which component switches are required to assembled the jobs in the job sequence. This will ensure no component switching problems occur. Following the application of the KCNS procedure, we will have generated the entire set-up for the production list.

6.4 Verification of the proposed adaptations

In this section, we will see if the adaptations from Section 6.3 will enable us to apply the heuristics from Chapter 3, and the new heuristics from Chapter 4 to the Panasonic Mv-II (LL) machine. As mentioned in Section 2.1.3, we assume that the feeder tray is initially empty. If this were not the case, we would still have to ensure that all the required feeders were on the feeder tray so that no component switching problems occur. This could entail the removal and addition of feeders.

6.4.1 The Greedy Perturbation Method

This method, as described in Section 3.2, consists of three major steps. The first generates a good job sequence, the second determines an initial component mix and the component switches that must occur (using KCNS), and the third step tries to generate a better job sequence. Generating a good job sequence with the function RC(i,j) can still be accomplished with no required changes. Determining an initial component mix and the required component switches for a job sequence has already been discussed in Section 6.3.4. Finally, the third step, or perturbation step, can remain the same. However, for each new job sequence, you have to

generate an initial component mix and the component switches according to the method discussed in Section 6.3.4.

6.4.2 The Labelling Method

This method, as described in Section 3.3, has two main steps. The first generates a job sequence by using the algorithm SEQ, and the second step generates an initial component mix and all the required component switches for that job sequence. To generate a job sequence, a few steps must change in the algorithm SEQ. In Step 1 of the algorithm, we must ensure that all the required feeders for the production list are placed into the feeder tray so that no component switching problems arise. As a result, Step 1 becomes the following:

Step 1: Determine the required feeders, as described in Section 6.3.1, for the production list. Place component reels that are not required by any job onto these feeders. If some feeders are empty, generate dummy component reels for these empty feeders.

Step 2 in the algorithm SEQ remains the same. However, Step 3 must be changed so that we ensure that the component reel to be removed from the feeder tray can be replaced with the component reel to be added. Otherwise, a component switching problem could arise. As a result, the component reel to be removed from the feeder tray is the one which has the lexicographic minimum label of all component reels on the feeder tray that could switch with the component reel to be added. The set of component reels that could be switched with another component reel has been discussed in Section 6.3.1.

Once we have obtained the job sequence, we can use the method discussed in Section 6.3.4 to generate an initial component mix and all the required component switches required for that job sequence.

6.4.3 The Matrix Diagonalization Method

This method, as described in Section 3.4, is broken down into three steps. The first diagonalizes the matrix, the second generates a group, and the third updates the matrix. Steps 1 and 3 can be implemented with no required changes. However, Step 2 must consider the spacing requirements between feeders when generating the groups. This makes the task more difficult, but not impossible. To check if a job can be placed into a group, we recalculate, according to the order of increasing feeder size, the slots required by all the feeders in the feeder tray plus the new ones required by the job to be added to the group. If there is enough feeder tray capacity, this new job is added to the group and we update the feeder tray.

6.4.4 The Maximal Intersection Minimal Union Method

This method, as described in Section 3.5, generates classes where each class requires less than the capacity of the feeder tray. The steps involved in this method remain the same. However, generating the set of jobs, L(S), that can be added to a class becomes more difficult because of the spacing requirements between feeders. As described in Section 6.4.3, we have to recalculate for each job the feeder tray capacity required by the feeders in the feeder tray plus the additional ones required by the new job. If there is enough feeder tray capacity, this job is added to the list L(S).

6.4.5 The Bin-Packing Method

This method, as described in Section 3.5.2, uses either the best-fit, first-fit, or next-fit heuristic to add jobs to bins. All three heuristics use cost arrays to determine into which bin a job is to be placed. This cost array becomes complicated to calculate because of the spacing requirements needed between feeders. As a result, we can no longer simply count the additional space each job will require in a bin given its contents. Having such a cost array would be meaningless as feeders may require numerous slots. However, a meaningful cost array for each bin could indicate the last slot that would be used in the feeder tray by each job if they were added to the bin, assuming the feeders are placed into the feeder tray in order of increasing size. With such a cost array, the bin-packing heuristics could be used.

Now, we must see if the improvement in Section 3.5.3 can still be used. This improvement was to vary the order in which jobs were processed by using the counting and overlap techniques. Both of these can be used without any modifications.

6.4.6 Method 1 for solving for both the MCS and MSI criteria

In Section 4.2, we discuss a method which takes the solution to the MSI criterion and reduces the number of component switches. It accomplishes this task by taking each group in the solution to the MSI criterion as a job, where the component reels for that job are the component reels required by the group, and then applies the greedy perturbation method on them. Taking each group as a job

requires no changes, and we have already discussed in Section 6.4.1 how to apply the greedy perturbation method.

6.4.7 Method 2 for solving for both the MCS and MSI criteria

This method, discussed in Section 4.3, uses the greedy perturbation method to generate a job sequence and then the next-fit heuristic to group these jobs. Following this, we apply the greedy perturbation method on these groups. The generation of the initial job sequence with function RC(i,j) requires no changes. However, when using the next-fit heuristic, we must modify the cost array as discussed in Section 6.4.5. Finally, if we take each group as a new job and then apply the greedy perturbation method, we must make the same modifications that were described in Section 6.4.1.

6.5 Splitting our production list

It is possible that all the required feeders for a production list may not fit into the feeder tray. To solve this problem, we would have to split our production list into groups so that all the required feeders for each group could fit into the feeder tray. This could be accomplished by using a modified bin-packing strategy that works with required feeders (the feeders required to avoid component switching problems) rather than components reels. For each bin, the cost array would indicate for each job, the last slot required to hold all the required feeders for that job and the jobs already assigned to that bin. Once this cost array is generated, we can use either the first-fit or best-fit heuristics to generate the groups.

6.6 A solution to the component insertion sequence / feeder tray assignment problem

As described in Section 2.1.2, the component insertion sequence / feeder tray assignment problem is to find for each job in the job sequence an order for the placement of components on the PCB and an assignment of component reels (feeders) to feeder tray slots. Since our main goal is to reduce set-up time, we must try to generate solutions to these problems without affecting the set-up time. For the feeder tray assignment problem, we will simply use the component mixes for each PCB generated by Section 6.3.4 as feeder tray assignments

Given a solution to the feeder tray assignment problem, we must try to generate a component insertion sequence such that assembly time is minimized. Reducing assembly time can be achieved by minimizing both feeder tray delays and *xy* table delays. *Feeder tray delays* occur whenever the PCB table is in position and the placement head is waiting for the feeder tray to get into position. *Table delays* are similar to feeder tray delays except the feeder tray is in position and the placement head is waiting for the PCB table to move into position. One heuristic, similar to the one given in [LN89], which tries to minimize these delays while generating a component insertion sequence can be described as follows:

Generate a graph G where each node in the graph represents an *xy* position on a PCB where a component is to be placed, and let the distance between these nodes represent the maximum time delay required by either the feeder tray or the PCB table. Following this, we try to find for each speed group the minimum weight Hamiltonian path in the graph. We then connect these paths so that all the

components from speed group one are placed first, followed by speed group 2, and so on. The resulting path is a component insertion sequence.

Chapter 7

Conclusion and future research

7.1 Conclusions

In this thesis, we have examined and compared different heuristic methods for the set-up/sequencing decision problem for both the MCS and MSI performance criteria.

For the MCS criterion, we implemented and tested the following heuristics: greedy perturbation [TD88a] and labelling [LM86a]. In our testing, we found that the greedy perturbation and labelling method gave similar results for our test set. However, we also developed and implemented our improved version of the greedy perturbation method, which, on average for our test set, improved the performance of the greedy perturbation method by 8%. This new improved greedy perturbation method out performed the labelling method in 90% of our test cases. Thus, overall, this improved greedy perturbation method gave us the best results for the MCS criterion. Using our improved lower bounding methods, we are able to show that the number of switches found by our improved greedy perturbation method is, on average, within 15% of optimality. We feel that these solutions are likely even closer to optimal than these lower bounds indicate, as we believe there is still room for improvement in the lower bounding techniques.

For the MSI criterion, we implemented and tested the following heuristics: matrix diagonalization [MS91], maximal intersection minimal union [TD88b], and some bin-packing heuristics. In our testing, we found that no one method performed better than any other, and that all the methods gave optimal or near optimal results for our test set.

We also discuss in the thesis situations in which both performance criteria actually affect set-up time. We develop two methods which attempt to give good solutions for both criteria, rather than just one. Our best of these methods obtained results that were, on average, 5% and 3% away from our best results for the MSI and MCS criteria, respectively. This could translate into time savings in certain situations in industry, for example, when several component reels can be switched in parallel at each switching instant.

Lastly, we discussed how to adapt the methods we looked at to different manufacturing environments. After all, our goal is not only to generate methods which obtain good solutions to the set-up/sequencing decision problem, but also to adapt these methods so that they can be used to solve problems in industry. Although the problem becomes much more complicated in these situations, we were successful in developing schemes for adapting all of the heuristics.

7.2 Future research

We have compared heuristic methods for the set-up/sequencing decision problem that are based on combinatorial optimization. However, we have not yet examined or compared methods outside this domain, for example, methods which are based on non-linear integer programming techniques. It would be interesting to

see how methods based on different techniques would compare to the ones we have studied.

In studying the lower bounding techniques for the MCS criterion, we were surprised to find virtually no previous results in this area. We feel that, with more research, the methods we develop in the thesis could possibly be improved further.

In this thesis, we have also discussed how to adapt the methods that we have studied for the idealized model to different manufacturing environments, and how both the MCS and MSI criteria can be used together to more accurately reflect set-up time. It would be interesting to investigate how well our adaptations works in practice, i.e. how much set-up time would be saved by a company like Mitel if they were to use the software which generated the best solutions for us.

Finally, we have focused on the single machine case for the set-up/sequencing decision problem. It would be interesting to investigate the situation where the PCBs are assembled on two or more machines in sequence. In this situation, we would have to consider such things as work load balancing, and machine restrictions (i.e. which components can be placed by each machine.).

Bibliography

[Baa88] Baase, S. (1988), *Computer Algorithms, Introduction to Design and Analysis, Second Edition*, Addison-Wesley, California.

[Bar88] Bard, J.F. (1988), A Heuristic for Minimizing the Number of Tool Switches on a Flexible Machine, *IIE Transactions*, Vol. 20, No. 4, pp. 382-391.

[BCF] Bard, J.F., Clayton, R.W., & Feo, T.A. (To appear), Machine Setup and Component Placement in Printed Circuit Board Assembly, *International Journal of Flexible Manufacturing Systems*.

[BS93] Barnea, A. & Sipper, D. (1993), Set-Up Reduction in PCB Automated Assembly, *Computer Integrated Manufacturing Systems*, Vol. 6, No. 1, pp. 18-26.

[BM76] Bondy, J.A. & Murty, U.S.R. (1976), *Graphy Theory with Applications*, American Elsevier Publishing Co., Inc.

[CB86] Cunningham, P. & Browne, J. (1986), A LISP-Based Heuristic Scheduler for Automatic Insertion in Electronics Assembly, *Int. J. Prod. Res.*, Vol. 24, No. 6, pp. 1395-1408.

[CKOS90] Crama, Y., Kolen, A., Oerlemans, A. & Spieksma, F. (1990), Throughput Rate Optimization in the Automated Assembly of Printed Circuit Boards, *Annals of Operations Research*, Vol. 26, pp. 455-480.

[CLR90] Cormen, T.H., Leiserson, C.E., & Rivest, R.L. (1990). *Introduction to Algorithms*, McGraw-Hill Book Company, New York.

[GJ78] Garey, M. R., and Johnson, D. S. (1978), "Strong" NP-Completeness Results: Motivation, Example and Implication, *J. Assoc. Comput. Mach.*, Vol 25, pp. 499-508.

[Kin80] King, J. R. (1980), Machine-Component Grouping in Production Flow Analysis: An Approach using a Rank Order Clustering Algorithm, *Int. J. Prod. Res.*, Vol. 18, No. 2, pp. 213-232.

[KN82] King, J. R. & Nakornchai, V. (1982), Machine-Component Group Formation in Group Technology: Review and Extension, *Int. J. Prod. Res.*, Vol. 20, No. 2, pp. 117-133.

[LLRS86] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B. (1986). *The Travelling Salesman Problem.* John Wiley & Sons, New York.

[LM86a] Lofgren, C.B. & McGinnis, L.F. (1986), Soft Configuration in Automated Insertion, *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, San Francisco, pp. 138-142.

[LM86b] Lofgren, C.B. & McGinnis, L.F. (1986), Dynamic Scheduling for Flexible Printed Circuit Card Assembly, *Proceedings IEEE Systems, Man, and Cybernetics*, Atlanta, GA. pp. 1294-1297.

[LN89] Leipala, T., & Nevalainen, O. (1989), Optimization of the Movements of a Component Placement Machine, *European Journal of Operational Research*, Vol. 38, 167-177.

[Lof86] Lofgren, C.B. (1986), Machine Configuration of Flexible Printed Circuit Board Assembly Systems, Unpublished PhD dissertation, Georgia Institute of Technology.

[MAC92] McGinnis, L.F., Ammons, J.C., Carlyle, M., Cranmer, L., Depuy, G.W., Ellis, K.P., Tovey, C.A., & Xu, H. (1992), Automated Process Planning for Printed Circuit Card Assembly, *IIE Transactions*, Vol. 24, No. 4, pp. 18-29.

[MS91] Maimon, O. & Shtub, A. (1991), Grouping Methods for Printed Circuit Board Assembly, *Int. J. Prod. Res.*, Vol. 29, No. 7, pp.1379-1390.

[Pan92] Panasonic (1992), Specifications : Panasert Mv-II (LL) NM-2559BA, BB. Matsushita Electric Industrial Co., Ltd. Manufacturing Equipment Division.

[SL92] Sadiq, M. & Landers, T.L. (1992), Decision Support System For Intelligent Part/Slot Assignment on a SMT Placement Machine, *Proceedings of the Technical Program, Nepcon West '92.*,Vol. 2, pp. 565-574.

[Tan86] Tang, C.S. (1986), A Job Scheduling Model for a Flexible Manufacturing Machine, *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, San Francisco, pp. 152-155.

[TD88a] Tang, C.S., & Denardo E.V. (1988), Models Arising From a Flexible Manufacturing Machine, Part 1: Minimization of the Number of Tool Switches, *Operations Research*, Vol. 36, No. 5, pp. 767-777.

[TD88b] Tang, C.S., & Denardo E.V. (1988), Models Arising From a Flexible Manufacturing Machine, Part 2: Minimization of the Number of Switching Instants. *Operations Research*, Vol. 36, No. 5, pp. 778-784.

[Win87] Winston, W.L. (1987). *Operations Research: Applications and Algorithms*, PWS-KENT Publishing Company, Boston.