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.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

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.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

OPTIMIZING PRINTED CIRCUIT BOARD ASSEMBLY TIMES ON A HIGH-SPEED PICK-AND-PLACE MACHINE

by

Mark Archie 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 0-612-04889-6

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 the Mitel Corporation for letting us use their facilities to test some of our solutions. In addition, we would like to extend a special thanks to 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 twin brother and the rest of my family for their constant moral support and encouragement. Their inspiration was invaluable during this difficult task.

Abstract

In this thesis, we investigate the problem of minimizing printed circuit board assembly time on high-speed pick-and-place machines. As it is considered unlikely that efficient methods will be found for this problem, heuristic methods which give near optimal solutions are sought. Several such methods currently exist, but there are no comparisons made between the different methods. One of the reasons for this is the differences in the models used.

In the thesis, we develop a more general model for the problem which encompasses most other models, and is robust, i.e. it can easily be adapted to different situations. We adapt two of the heuristic methods from the literature to this model, implement and test them and report the resulting assembly times. We also provide improvements to these methods, and improved lower bounding techniques for the problem. Finally, we adapt the best method to a real-world situation, namely the environment at the Mitel Corporation in Ottawa, Canada. We test this method against the sophisticated software tool currently being used at Mitel, with good results.

Table of Contents

CHAPTER 1

Introduction

1.1 Introduction and motivation

A Printed Circuit Board (PCB) is a laminated board which has electronic components inserted or mounted on it at specific locations. In the PCB manufacturing industry, new technologies have been developed to minimize board and chip sizes. Due to these reductions, the required material for a PCB has been significantly reduced which in turn has reduced the cost of PCBs. However, these advances have forced manufacturers of PCBs to place smaller and smaller components on denser designs. In addition to these difficulties, manufacturers must remain competitive and productive in order to survive in today's society.

To address the existing needs of manufacturers of PCBs, high-speed, high-precision pick-and-place machines have been developed. These machines, which are at the leading edge of technology, have been developed with flexibility and speed in mind. These properties are especially important when we consider the diversity of PCBs now being produced, and the fact that 28% of the entire cost of a PCB is due to assembly, most of which is component placements [Mar86].

These machines have three main parts: the feeder tray, the x-y table, and the turret. The feeder tray, which moves laterally under the turret, is used to hold the *component reels*, where a reel contains many components of a single component

type. The x-y table, which moves in the x and y directions under the turret, is used to hold the PCB. The x-y table and the feeder tray are on opposite sides of the turret, which picks up components from the feeder tray, rotates, and places them on the PCB (see Figure 2.2).

With these high-speed pick-and-place machines, the user must specify two things for each PCB assembly: the feeder tray set-up, and the component placement sequence. Determining a feeder tray set-up for a PCB entails deciding where the different component reels required for a particular PCB are to be placed on the feeder tray. Determining a component placement sequence for a PCB entails deciding the order in which the components for the PCB are to be placed onto the PCB by the machine.

These two user controllable elements, which make these machines flexible for the assembly of different PCBs, directly determine the amount of assembly time required by the machine for a particular PCB assembly. Thus, finding solutions to these two problems which result in low assembly times is especially important for manufacturers of high-volume, low-mix productions. An increase of even 3 seconds per board due to a poor feeder tray set-up or sequencing decision could cause a company thousands of dollars in lost production per year. Consequently, the importance of finding good solution techniques for these two decision problems is apparent.

The *PCB assembly problem*, which is the topic of this thesis, is the problem of finding a feeder tray set-up and a component placement sequence for a given PCB. A solution to this problem determines the assembly time for a PCB, and such a solution is called *optimal* if assembly time is minimized by it.

The problem of finding an optimal solution to the PCB assembly problem appears to be very difficult, since two subproblems that arise from it are known to be NP-hard. The first of these is as follows: Given a component placement sequence, find a feeder tray set-up which minimizes assembly time. This problem can be modeled as a quadratic programming problem [BCF94], which is NP-hard [Sah74]. The second of these subproblems is the following: Given a feeder tray set-up, find a component placement sequence which minimizes assembly time. This problem can be modeled as a Traveling Salesman problem [BCF94], which is also known to be NP-hard [L86].

As both of the above subproblems are NP-hard, it is considered unlikely that efficient methods will be found for the PCB assembly problem. As a result, a heuristic approach is taken, i.e. an efficient method is sought which finds a solution that may not be optimal, but is hopefully close to optimal. Many such heuristics have already appeared in the literature. These will be reviewed in the next section.

As well as dealing with the PCB assembly problem in a general form, we are also very interested in this problem as it applies to a particular environment, namely the environment at the Mitel Corporation in Ottawa, Canada. Mitel is a Canadian-based international supplier of telecommunications solutions. They manufacture such items as semi-conductors, telephone sets and a whole list of other telephone products. One step in the manufacturing of many of these products is the assembly of PCBs. Besides manufacturing PCBs for their own products, Mitel obtains contracts from other companies to assemble PCBs. As a result, Mitel assembles approximately 12-13000 PCBs per week. About 6000 of those are

assembled on the Panasonic MVII(LL) NM-2559B machine, which is a high-speed pick-and-place machine used at Mitel specifically for high-volume production of PCBs.

Currently, Mitel is solving the PCB assembly problem using a very sophisticated software package called Panatools, which is commercially distributed by Panasonic for use on their high-speed pick-and-place machines. As it is a commercial package, knowledge of the methods used by this package are not available.

1.2 Overview of currently known methods

As mentioned earlier, a considerable amount of research has taken place on the PCB assembly problem, and many heuristic methods appear in the literature. Given below is a summary of the main approaches that have been used and the papers involved.

One approach that seems to be fairly common when trying to solve the PCB assembly problem is to iterate between using a Hamiltonian path or TSP heuristic for the component placement sequence and a quadratic programming heuristic for the feeder tray set-up problem. This type of approach is used in [C90], [LN89] and [BCF94].

The second most common approach is to place certain restrictions on the component placement sequence to guarantee no feeder delays occur. Using these constraints, one then tries to generate a good component placement sequence using TSP heuristics. This approach is used in [AS92] and [Gro92].

Other methods also exist, such as [BM88], [DLM92], but most of these methods have been designed for very specific machine models with specific properties that are not common properties of current pick-and-place machines. As a result, we cannot use these methods. For a survey of some of these different methods see [M92].

1.3 Contributions

The main contributions of this thesis are the following:

1) A GENERAL MODEL. The methods introduced so far in the literature have been applied to a wide variety of different high-speed pick-and-place machines. By not applying the methods to a general model, the different methods cannot be compared. In this thesis, we give a general model of a high-speed pick-and-place machine that encompasses many of the different models used. This allows the different methods to be compared more easily without having to worry about all the specific details associated with a particular machine.

2) COMPARISON OF TWO METHODS. Surprisingly, while surveying the current research on the PCB assembly problem, we found no performance comparisons of the different heuristic methods. Hence, there is no way to judge which heuristics perform the best. We chose two of the methods from the literature which we felt were most promising and compared them assuming our general machine model. To do this, we had to adapt them to our model, implement them and test them on PCBs.

3) IMPROVEMENTS TO THE METHODS. In studying the two different methods mentioned above, we discovered ways to improve their performance. We implemented and tested our improvements, and report on the results.

4) IMPROVING LOWER BOUNDS. A *lower bound* for the PCB assembly problem is a number k such that the assembly time for any solution to the problem is $\geq$ k. Lower bounds can be used to determine the quality of the solutions we are generating with our heuristic methods. Currently, there is very little in the literature regarding lower bounds for the PCB assembly problem. We propose two new methods for obtaining such lower bounds, one of which upon testing against current methods gave much improved results.

5) ADAPTING METHODS TO THE MITEL ENVIRONMENT. In order to adapt a method for a specific machine, all the machine specific properties must be considered. We indicate how to adapt a method from our general form to the MVII(LL) machine used at Mitel. Many of the complications we had to deal with, such as multiple machine speeds and different component reel sizes, also apply to many of the high-speed pick-and-place machines, and yet have not been addressed previously in the literature.

6) OBTAINING SOLUTIONS CLOSE TO THOSE OF PANATOOLS. Panatools, the very expensive software package currently being used at Mitel for the PCB assembly problem, is commercially distributed by Panasonic, and thus, the source code is not available. As a result, Panatools cannot be adapted by Mitel to handle variations of the general PCB assembly problem which arise in production. As a result, the people at Mitel were very interested to see if we could develop software that could produce solutions for their PCBs whose assembly times were close to

those of Panatools. The software we developed for the Mitel environment accomplished this.

1.4 Outline of thesis

In chapter 2 of the thesis, we formally define the PCB assembly problem and develop a general model for the machine. In Chapter 3, we describe two of the existing heuristic methods for our problem as they apply to our machine model and develop improvements for these methods. We also implement and test these methods, and report on the results. In Chapter 4, we develop and test two new methods for obtaining lower bounds for the PCB assembly problem. In Chapter 5, we describe how to adapt the most promising heuristic we have found to the production environment at Mitel. In addition, we compare the solutions we were able to generate in the Mitel environment against the solutions generated by the software package Panatools currently being used by Mitel. Finally, in Chapter 6 we summarize our results and propose some future research topics.

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

1.5 Graph theoretical background

In this section, we describe graph theory notation and some definitions which we require in the thesis. We also describe two problems that arise frequently as subproblems in the heuristic methods for the PCB assembly problem, namely the minimum cost Hamiltonian path problem and the traveling salesman problem,

and how minimum cost Hamiltonian path problems can be solved as traveling salesman problems.

1.5.1 Definitions

A graph G is a pair (V,E), where V is the set of *nodes* and E consists of unordered pairs of nodes representing the *edges* of G. We say an edge e = uv is *incident* with nodes u and v, and that u and v are *adjacent* in G if they are joined by an edge. The *degree* of a node is the number of edges incident to that node. We say that a graph G is *weighted* if each edge has an associated cost which is a real number. A graph is *complete* if an edge exists between every pair of nodes.

A *path* in a graph G is a sequence $S=n_0e_1n_1e_2n_2...e_kn_k$ that alternates between nodes and edges in G. Often though, the path is simply denoted by the sequence of nodes in that path, i.e. $S=n_0n_1n_2...n_k$. The *cost of a path* in a weighted graph G is the sum of the costs on edges in that path. If all the nodes in the path are distinct, then the path is known as a *simple path*. If this simple path has the additional property that every node in the graph is visited exactly once, then we have a *Hamiltonian path*.

A *cycle* is a path which starts and ends at the same node. The *cost of a cycle* in a weighted graph G is the sum of the costs on edges in that cycle. If all the nodes in the cycle are distinct, then the cycle is known as a *simple cycle*. Furthermore, if a simple cycle contains every node in the graph, then we have a *Hamiltonian cycle*, which is often referred to as a *tour*.

Given a complete weighted graph G, the *minimum cost Hamiltonian path problem* is that of finding a Hamiltonian path in G of minimum cost. The *Traveling Salesman Problem* (TSP) for G is that of finding a minimum cost Hamiltonian cycle in G. For both of these problems, we assume that the edge costs are ≥ 0.

1.6 Solving minimum cost Hamiltonian path problems

Many excellent heuristics, such as the Lin-Kernighan method [LK73], exist for the TSP (see [L86] for a description of these methods). Because of this, other graph problems are often modeled as TSPs and then solved using TSP heuristics. One such graph problem is the minimum cost Hamiltonian path problem. Since three different variations of this problem arise later on in our optimization problems, we will describe how we solve these three different problems as TSPs.

The first problem is how to formulate the problem of finding a minimum cost Hamiltonian path in a graph G as a TSP. We convert this problem to a TSP by adding to the graph G a new node which has an edge of cost zero from it to all other nodes in G, as done in [L86]. Let C be a minimum cost Hamiltonian cycle in this new graph. Then the path obtained by removing the edges in C incident with the new node is a minimum cost Hamiltonian path in G.

The second problem is how to formulate the problem of finding a minimum cost Hamiltonian path with specified endpoints as a TSP. This can be converted to a TSP by adding a new node to the graph G which has an edge of cost zero between it and the two nodes specified as the Hamiltonian endpoints. The costs for the remaining edges from the new node to all other nodes in G are set to a large

value M. This value M is chosen large enough to ensure that no TSP solution will include any of these edges. Let C be a minimum cost Hamiltonian cycle in this new graph. Then the path obtained by removing the new node and the two edges incident with it in C is a minimum weight Hamiltonian path which starts and ends at the required endpoints.

To determine the value M, we consider the worst possible Hamiltonian cycle cost on the new graph which does not use any of the edges with cost M. Let G′ be the new graph we constructed. Let n be the number of nodes in G, and let Max be the maximum edge cost in G. Then any Hamiltonian cycle in G′ which does not use the M edges has cost ≤ (n-1)*Max. By setting the value of M to (n-1)*Max+1, we ensure that an optimal Hamiltonian cycle could not possibly include an edge with a cost of M because other Hamiltonian cycles of cost <(n-1)*Max+1 exist.

The final Hamiltonian path problem which we wanted to solve as a TSP was to find a minimum cost Hamiltonian path which contains a given set of disjoint edges K. To achieve this, we generate a new graph G′ by adding a new node to G which has an edge of cost zero from it to all other nodes in G. We then ensure that the required edges will be included in the TSP solution in the new graph by using a new set of edge costs for the original edges in G. A cost of zero is used for all edges in K. For all remaining edges which were originally in G, the old cost plus some large constant value L is used. This L value is chosen large enough to ensure that the edges in K will be included in the TSP solution for the new graph. Solving the TSP problem on the new graph and removing the new node as well as the two edges incident with it in the optimal cycle produces the desired minimum cost Hamiltonian path.

The value of L is calculated in a similar way as the value for M was calculated above. Let n be the number of nodes in G, k be the number of edges in K, and Max be the maximum edge cost in G. The value of L can now be determined by realizing that there are n-k-1 edges from G-K in any Hamiltonian cycle in the new graph which includes all edges in K. By setting the value of L to be (n-k-1)*Max+1 and adding this value to all edges in G except the edges in K, which have cost zero, causes any Hamiltonian cycle that is generated which does not include all the edges in K to have higher cost than any Hamiltonian cycle that includes all the edges in K. The reason is that any cycle that does not include one of the edges from K must have one extra edge in the tour from G-K. As a result, this extra edge forces any Hamiltonian cycle that does not include all the edges in K to have cost $\geq$(n-k)*L, whereas any Hamiltonian cycle that includes all edges in K can have a cost of at most (n-k-1)*L + (n-k-1)*Max, which is smaller than (n-k)*L. Thus, all edges in K must be included to obtain a minimum cost Hamiltonian cycle.

CHAPTER 2

Problem formulation

In this chapter, we give a general description of a high-speed pick-and-place machine, and describe a general formulation for the problem of minimizing PCB assembly times on such machines. Note that, although particular pick-and-place machines may differ slightly in their specifications, we have attempted to make our formulation general enough that it can easily be adapted to most PCB assembly environments, particularly the Mitel environment.

2.1 Machine and PCB description

A PCB is an electrical board that has wires imprinted into it. These wires connect different locations on the board where components, known as chips, are to be placed. After all the components are placed onto the PCB, the PCB provides some function which will be used in an electrical device. To be able to insert all the necessary components onto a PCB, we need to know the PCB specifications. These specifications include the x-y coordinates for the placement locations, and the component type required at each location. These coordinates are measured in millimeters, and the origin for the coordinates is assumed to lie at the bottom right tooling pin of the PCB. Due to the location of the origin, all component placements will have a negative value for the x-coordinate and a positive value for the y-coordinate.

For illustration purposes, a PCB that is already assembled with components is given in Figure 2.1. We have not shown, however, the imprinted circuitry which exists between the different components.

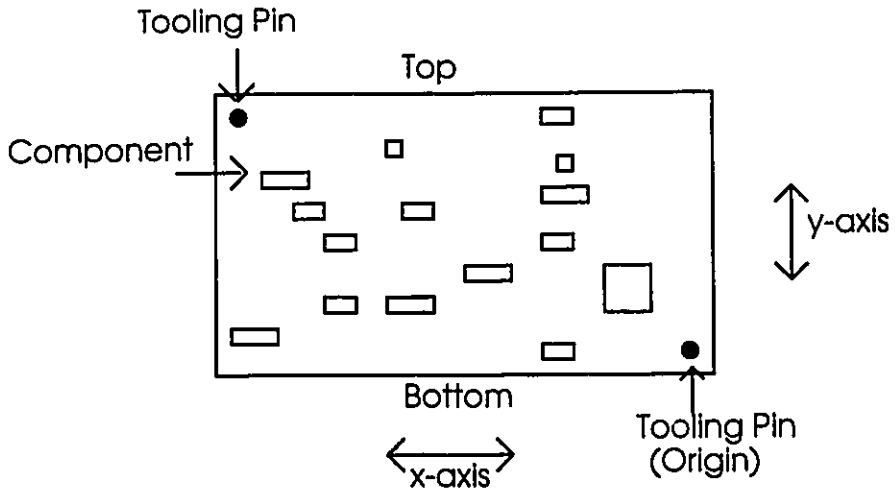


Figure 2.1 A PCB

High-speed pick-and-place machines have three basic parts: a turret, a feeder tray and an x-y table. The PCB to be assembled is moved along a conveyor belt and placed onto the x-y table for assembly. The turret, which is circular, is used to rotate the pick-and-place heads mounted on its circumference to pre-defined locations for the pick-up of a component and the placement of a component. Since these two locations are fixed, the feeder tray is used to align the next component to be picked up to this pick-up location, and the x-y table is used to align the next placement location on the PCB to this placement location. (See Figure 2.2.)

These high-speed pick-and-place machines also have a programmable aspect that allows them to assemble different PCBs. This is accomplished by allowing the user to setup the feeder tray with the required component types,

known as the *feeder tray set-up*, and to specify the order that the components are to be inserted into the PCB. This ordering is known as either the *component placement sequence* or the *pick-and-place sequence*.

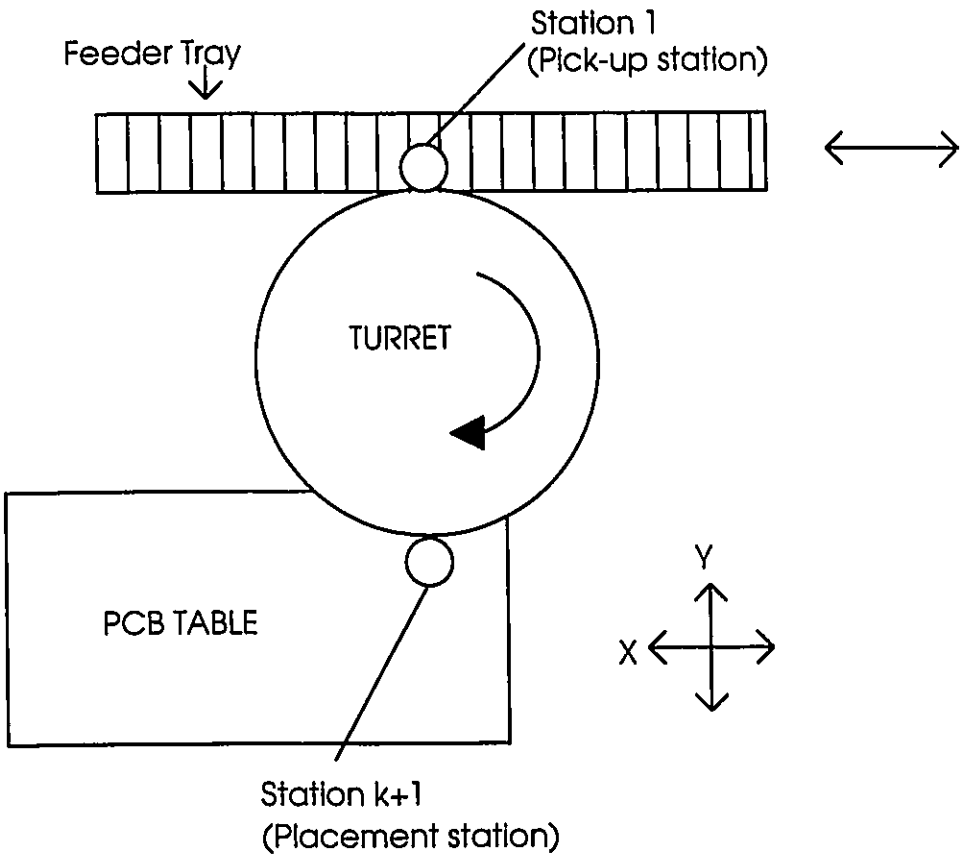


Figure 2.2

We assume that the turret has $2k$ equally spaced pick-and-place heads mounted on its circumference, where $k \geq 1$. Also associated with the turret are pre-defined locations called *stations*. At these stations, a particular operation may be performed by a pick-and-place head. For the $2k$ headed turrets, we assume the same number of stations as heads. The turret now rotates or indexes clockwise at one constant speed through each of these stations to perform the necessary

operations. Two of these stations are reserved for the pick-up and placement operations. Note that these are always mounted across from each other on the turret, and thus are stations 1 and k+1 in a 2k headed turret (see Figure 2.2). The remainder of the stations, if any, are used for operations such as component orientation, component defect detection, component ejection and others. In addition, we assume that these operations occur during the delays caused by the pick-up and placement operations. If these other stations do not exist, then we assume that any other necessary operations between the pick-up and placement station occur during the rotation of the turret.

Note that a turret with 2k heads will be placing the i^{th} component in the component sequence while it is picking up the $(i+k)^{th}$ component. The reason for this is that the turret, after picking up a particular component, must rotate k-1 other heads to the placement station before placing that component.

The second basic part of the pick-and-place machine is the feeder tray. The feeder tray is mounted adjacent to the turret and travels at a constant speed laterally. It has C evenly spaced slots that are used for holding component reels. We assume that each reel holds many components of a single component type and there can be at most one component reel with a particular component type. In addition, we assume each reel requires one feeder tray slot and no more than one reel can be assigned per slot.

During operation, the feeder tray has to position the next required component in a component placement sequence under the pick-up station. In order to determine how long this positioning will take, we need to know how far the feeder tray has to travel. Since the slots are evenly spaced and the feeder tray is

always positioned at a slot, we can measure the feeder tray travel distance by the number of slots jumped. For instance, if the feeder tray is currently at the i^{th} slot and must be repositioned to the j^{th} slot, then the feeder tray travel distance is |j-i| slots.

Note that some machines do allow multiple component reels that have the same component type to be placed onto the feeder tray. With this property, it would be possible to obtain the feeder tray set-up by determining the optimal component placement sequence, and then placing a new component reel on the feeder tray for each component placement in the order determined by the component placement sequence. However, due to the feeder tray capacity restriction, this solution cannot be used.

The third basic part of the pick-and-place machine is the x-y table. The x-y table is located beneath the turret and is able to travel at a constant speed in both the x and y direction simultaneously. Its purpose is to hold a PCB and to position the next placement location on the PCB directly under the placement station.

In order to determine how much time the x-y table requires to position the PCB for the next placement, we need to be able to accurately measure the distance between any two placement locations on the PCB. Since the x-y table traverses in both the x and y direction simultaneously and independently, the distance measurement becomes the maximum of the x and y displacements. Assuming two component locations (x_i,y_i) and (x_j,y_j), this distance measurement is defined as

$$d((x_i,y_i),(x_j,y_j)) = \max (|x_i-x_j|,|y_i-y_j|).$$

This metric is more commonly known as the l_∞ metric [LN89] or the maximum metric [BM88].

We should also mention that sometimes associated with the feeder tray and the x-y table are *home* positions. These home positions are specified positions to which the x-y table and feeder tray return after a PCB is completed and a new one is about to be loaded. As most methods can easily be adapted to accommodate home positions, we decided that there would be no home positions for our machine. Furthermore, we assume that we begin each PCB assembly with the x-y table and the feeder tray correctly positioned for the first placement and pick-up. This allows us to neglect the amount of time that would be required for our machine to position the feeder tray and the x-y table for the first pick-up and placement from home positions.

From the above description, we know that there are five main operations associated with a pick-and-place machine. These operations are the pick-up, placement, x-y table movement, feeder tray movement and turret rotation operations. The specific times required for these operations along with values for k and the feeder tray capacity C make up the *machine specifications* we require for a particular machine. We assume each of these operation rates are measured in particular units. The pick-up and placement operation times are assumed to be measured in seconds. The turret rotation rate is assumed to be measured in the number of seconds required to rotate one station. Finally, we assume that the x-y table and the feeder tray rates are measured in the number of millimeters and the number of slots respectively that can be traversed during the turret rotation rate time.

It is important to note that these five operations can only occur when certain conditions are satisfied. Knowing these conditions and how the operations interact are essential for understanding the operation of the machine, especially the concurrency aspects.

2.2 The pick-and-place cycle

2.2.1 Pick-and-place operations

For a $2k$ headed turret machine, the pick-up and placement operations are performed simultaneously. Thus, the pick-up and the placement operations can only occur if the following three conditions are satisfied: i) the x-y table has positioned the next placement location on the PCB under the placement station, ii) the feeder tray has the next component to be picked-up aligned with the pick-up station, and iii) an empty head must be in position at the pick-up station, and the head which is holding the component to be placed must be at the placement station.

Given the conditions that must be satisfied before a pick-up or placement operation can occur for our 2k headed turret machine, we can now consider the operations that occur concurrently with these operations. For the $2k$ headed turret, we know that the placement and pick-up operations are concurrent and synchronized. In addition, we know that the x-y table and the turret must be stationary during the placement operation, and the feeder tray and the turret must be stationary during the pick-up operation. Consequently, all operations must be stationary for the placement and pick-up operations to take place.

2.2.2 The turret, x-y table and feeder tray

From the above description, we know how the pick-up and placement operations interact with all the other operations. We do not know, however, how the turret, x-y table and feeder tray interact with each other. For the $2k$ headed turret machines, the turret, the x-y table and the feeder tray movements are all concurrent operations. Only during the pick-up and placement operations will any of them be restricted, as indicated earlier. We should also note that we assume that these three concurrent operations are initially synchronized.

Having indicated the main operations and how they can interact, the sequence of machine operations that must occur for a particular pick-up and placement can be given. The x-y table, feeder tray and turret operations simultaneously begin for a particular pick-up and placement. After all three operations complete, the pick-up and placement operations simultaneously occur. It is important to note that this contiguous series of machine operations required for a component pick-up and placement is known as a *pick-and-place cycle*. We should also mention that a pick-and-place cycle may not require all five operations to occur.

There are other operations that are associated with a high-speed pick-and-place machine. These operations, however, generally occur during the operations described above, and thus do not increase the assembly time, or else are overhead costs, such as the checking of PCB marks at the beginning of each assembly to determine if the board is not aligned. Since the overhead costs are constant and all the other operations do not increase assembly time, we have decided to leave out these operations to permit a more basic description of the machine and to focus our

attention on the machine operations which have the largest potential for optimization.

2.2.3 Timeline

To illustrate the interaction of the above operations, we have constructed a timeline of machine operations required for three pick-and-place cycles using a $2k$ headed machine. This timeline is given in Figure 2.3 and is similar to the one given in [BCF94]. Each rectangle on the timeline represents the period and length of time required for each machine operation. The difference between the two timelines is that we also illustrate the three different operations that can cause a delay for any particular placement.

In our timeline, we start at time zero and assume that component $i+k-1$ was just picked-up and component $i-1$ was just placed. From this point, the turret starts its rotation, the feeder tray starts moving to align the $(i+k)^{th}$ component for pick-up, and the x-y table starts moving to position the PCB for the i^{th} component placement. At time unit 1, the x-y table has positioned the PCB for the component placement. This placement, however, can not begin until the turret has finished positioning the pick-and-place heads and the feeder tray is aligned for the pick-up operation. The turret operation is completed at time unit 2 followed by the feeder tray alignment at time unit 4. With the x-y table, feeder tray and turret operations complete, both the pick-up and placement operations can occur simultaneously at time unit 4. These operations are then completed at time unit 5. This now completes the first pick-and-place cycle, which illustrates a delay caused by excess movement of the feeder tray, known as a *feeder tray delay*.

We will not describe the second and third pick-and-place cycles, but they illustrate the other two operations that can cause delays. The second pick-and-place cycle illustrates a delay caused by the x-y table, known as a *table delay*, while the third pick-and-place cycle indicates a delay caused by the turret, known as a *turret delay*.

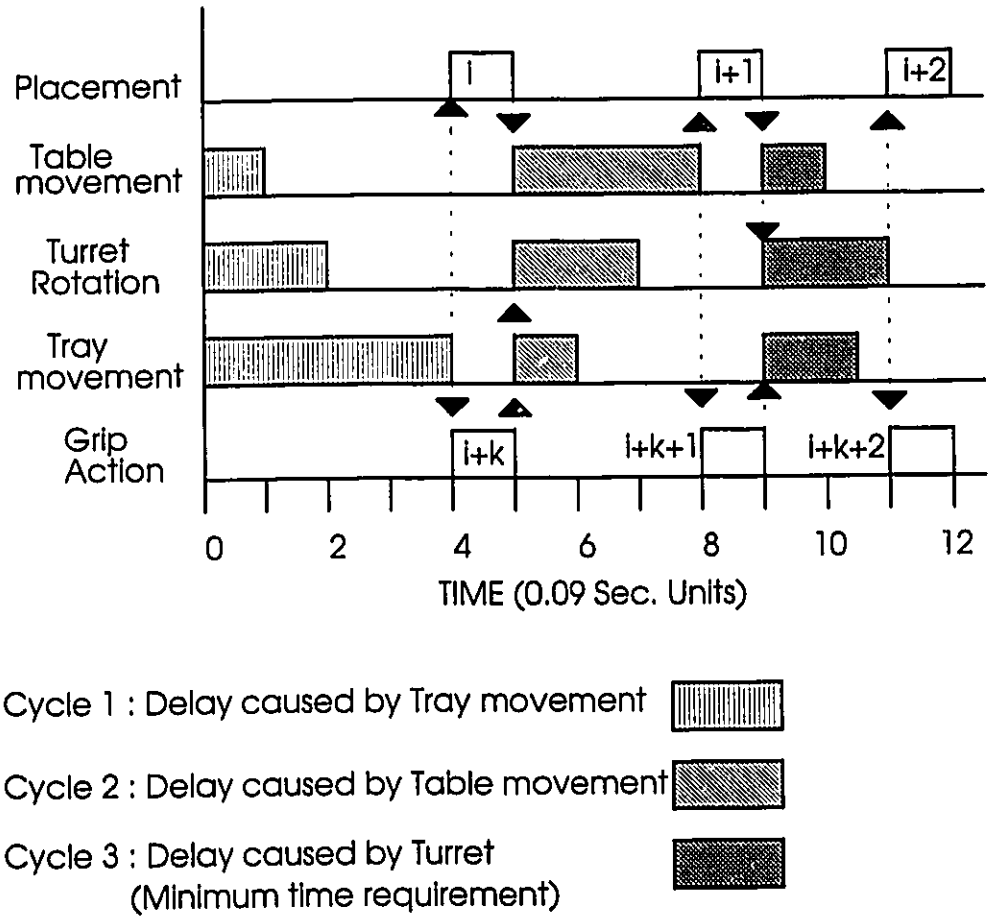


Figure 2.3 Placement timeline

2.3 The PCB assembly problem

The *PCB assembly problem* is defined as follows: Given a set of machine and PCB specifications, find a feeder tray set-up and a component placement sequence. A solution to the PCB assembly problem determines the assembly time required to place all the required components onto a PCB. A solution to this problem is called *optimal* if it provides a minimum assembly time.

2.3.1 Assembly time

In order to address the PCB assembly problem, we must first understand how to calculate the assembly time associated with any given solution, i.e. any feeder tray set-up and component placement sequence. To do this, we can consider the assembly of a PCB to be a sequence of pick-and-place cycles. If we can now determine the time required to complete each pick-and-place cycle and then sum them, we will obtain the assembly time of the PCB, excluding any overhead times. Our task is to now determine how to calculate the time required for a pick-and-place cycle.

The pick-and-place cycle time calculation is dependent upon the properties of the machine. Assuming our general machine with a 2k headed turret, the pick-and-place cycle time can easily be calculated and is based on the five main operations of the machine. With this type of machine, the x-y table, the feeder tray and the turret begin their operations simultaneously at the start of the pick-and-place cycle. Since these operations are concurrent, the time required to complete these three operations is the maximum of these three operation times. Following the completion of these three operations, the pick-up and placement operation can

take place to complete the pick-and-place cycle. Since these two operations start simultaneously and require the same amount of time, we only need to add the placement time to the time required to complete the first three operations in the pick-and-place cycle. By doing this, we have obtained the calculation for the pick-and-place cycle time.

Before proceeding to give the actual calculation for the pick-and-place cycle time, it is important to mention two other points. The first point is that the turret time we are considering in our pick-and-place cycle time is the amount of time required by the turret to align a new pick-and-place head to both the pick-up and placement stations. For our machine, this occurs every time the turret rotates one station. This rotation time between stations is better known as the *turret indexing delay*.

The second point that is of concern on a machine with a $2k$ headed turret, is the fact that a component that is just picked up is not the next component to be placed. In fact, a $2k$ headed turret will be placing the i^{th} component in a component placement sequence while it is picking up the $(i+k)^{th}$ component in the same sequence. The importance of this is that the calculation for the time required by the x-y table and feeder tray in any pick-and-place cycle must take this property into consideration. For example, a component placement sequence that had the i-1^{st} component in this sequence just placed would have the x-y table distance for the next pick-and-place cycle calculated from the PCB location for the i-1^{st} component to the PCB location for the i^{th} component. The feeder tray travel distance for the same pick-and-place cycle, however, would be calculated from the feeder position for the $i+k$-1^{st} component to the $(i+k)^{th}$ component.

More precisely, if we let $c_1, c_2, \ldots, c_n$ represent our component placement sequence, the time for the pick-and-place cycle which places component c_i and picks up component c_{i+k} is calculated as follows:

cycle time = $\max(Turr, F_{c_{i+k}}, T_{c_i}) + P$,

where

Turr = Turret indexing delay.

$F_{c_{i+k}}$ = Time for the feeder tray to travel from the feeder position for component c_{i+k-1} to the feeder position for component c_{i+k}.

T_{c_i} = Time for the x-y table to travel from the PCB location for component c_{i-1} to the PCB location for component c_i.

P = Either the pick-up or placement time.

As previously mentioned, we can calculate the assembly time, neglecting any overhead times, by summing all the pick-and-place cycle times necessary for a particular PCB. Note that the above cycle time calculation is only valid for pick-and-place cycles in which some component is picked up and another is placed, and for some cycles this is not the case. The very first cycle, which picks up c_1, requires no x-y table or feeder tray movement since they are assumed to be correctly positioned for the first component pick-up and placement. In addition, turret rotation is also not needed, as we do not need to align any new heads to the pick-up and placement station. Consequently, the first pick-and-place cycle only requires a pick-up operation to be completed, and we have

cycle time = P.

The next *k*-1 pick-and-place cycles (i.e. cycles 2 to k), only load the turret and require no x-y table movement. Also recalling that we assume that the x-y

table is already aligned for the first placement, there is still no x-y table movement required for the k+1st cycle. Thus for these cycles (2 to k+1st), we have

cycle time = max(Turr, F_{ci}) + P.

Similarly, the last k pick-and-place cycles simply unload the turret, requiring no feeder tray movement. Thus for these cycles, we have

cycle time = max(Turr, T_{ci}) + P.

Taking these facts into consideration, we obtain the following assembly time calculation for a 2k headed machine:

assembly time =

$$P + \sum_{i=2}^{k+1} (\text{Max}(\text{Turr}, F_{ci}) + P)$$

$$+ \sum_{i=2}^{n-k} (\text{Max}(\text{Turr}, F_{ci+k}, T_{ci}) + P)$$

$$+ \sum_{i=n-k+1}^{n} (\text{Max}(\text{Turr}, T_{ci}) + P) . \qquad \text{(Equation 2.1)}$$

Note that during any pick-and-place cycle, the time required for the turret indexing delay is fixed and machine dependent. Since the x-y table and the feeder tray are concurrent operations with the turret rotation, this turret delay can be considered the amount of *free time* available for the x-y table and feeder tray to position themselves without causing any delay. Thus for any pick-and-place cycle in which there is no x-y table or feeder tray delay, the corresponding cycle time is

equal to the turret delay plus the placement time and is optimal. This is known as the *minimum machine pick-and-place cycle time.*

2.3.2 The component placement and feeder tray set-up subproblems

Optimizing this assembly time for the PCB assembly problem requires solutions to both the feeder tray set-up and component placement sequence. A poor solution for either one can significantly affect assembly times. Obtaining good solutions to these two problems simultaneously, however, is not an easy task. They are actually coupled problems, that is the cost of the component placement sequence depends on the feeder tray set-up and vice versa. This property makes the PCB assembly problem very difficult to solve.

To make the PCB assembly problem more manageable, a common approach is to decouple the two problems [M92], solve them and then somehow combine the two solutions to obtain a solution for our PCB assembly problem. Doing this enables the use of quadratic programming methods [Win87] for solving the feeder tray set-up problem given the component placement sequence, which we will call *subproblem 1*, and enables the use of TSP heuristics [L86] for solving the component placement sequence given the feeder tray set-up, which we will call *subproblem 2.*

Before modeling each of these subproblems mathematically, we should mention a problem which arises when trying to model the component placement sequence given the feeder tray set-up as a TSP using our machine model. Recall that our machine retrieves a component several pick-and-place cycles before it can be placed; in fact, the i^{th} component in a component placement sequence is placed

while the $i+k^{th}$ component in the same sequence is being picked up. This means that the cost between any two component placements is dependent on the component placement sequence, i.e. we cannot calculate the feeder tray delay time that will occur between the i^{th} and j^{th} component placements without already knowing the component placement sequence. This property, however, is not consistent with a TSP, and as a result, the component placement sequence problem on these types of machines can not be formulated as a TSP.

In many of the papers, this problem is not even mentioned. However, a common practice in the literature is to simply ignore the entire notion that a component is picked up from the feeder tray k pick-and-place cycles ahead of its placement. By assuming a component is picked up and placed in the same cycle, the cost between any two component placements i and j will be the maximum of three things: the x-y table time required to move between the placement locations i and j, the feeder tray time required to move between the feeder tray locations holding the component types for i and j, and the turret indexing time. Consequently, with this assumption it is possible to model the problem as a TSP, and apply TSP heuristics.

Although modeling the component placement sequence problem on these types of machines as a TSP is not a completely accurate model, this model can still be used to reduce assembly time. The reason is that by solving this model we will be obtaining a component placement sequence that will reduce both the x-y and feeder tray travel distance required between component placements. Since the assembly time uses the x-y and feeder tray distances between component placements, we are still reducing assembly time. (See Section 3.4.1.)

2.3.3 Mathematical formulation for subproblem 1

Recall that subproblem 1 involves finding a solution to the feeder tray set-up problem given a component placement sequence. In this section, we formulate this subproblem as a quadratic programming problem. However, in this thesis, we do not make use of this formulation and thus only include this formulation here for completeness.

In the formulation given below, we have assumed our general machine properties given in Section 2.1, except that we assume that the component picked up is the next component to be placed. Consequently, this formulation assumes that the x-y table and feeder tray are already in position for the retrieving and placing of the first component. In addition, this formulation is similar to that given in [BCF94] except that the feeder tray does not allow multiple slots with the same component type. Thus, we have adapted the formulation from [BCF94] by modifying constraint (1d) below.

Given the component placement sequence, let :

The feeder tray positions be indexed by j=1,2,....,C.

The component placement locations be indexed by i=1,2,....,n.

The unique component types be indexed by p=1,2,....,m.

r_p be the number of placement locations requiring component type p.

a_{ip} = 1 if placement location i requires component type p, and is 0 otherwise.

$t_{jk}(i\text{-}1,i)$ = the time required to retrieve the i^{th} component from feeder tray position k given that the feeder tray was at position j for the $(i\text{-}1)^{st}$ component, (note that this is calculated as the maximum of the turret time, x-y table time and feeder tray time)

Decision Variables :

x_{ij} = 1 if the component type for the i^{th} placement is located at feeder tray position j, and is 0 otherwise.

y_{jp} = 1 if feeder tray position j contains component type p, and is 0 otherwise.

A quadratic programming formulation :

$$\text{Minimize} \sum_{i=2}^{n} \sum_{j=1}^{C} \sum_{k=1}^{C} t_{jk}(i\text{-}1,i)\, x_{i-1,j}\, x_{i,k} \qquad (1a)$$

Subject to

$$\sum_{j=1}^{C} x_{ij} = 1 \quad \text{, for all } i = 1,\ldots, n \qquad (1b)$$

$$\sum_{i=1}^{n} a_{ip}\, x_{ij} \le r_p y_{jp} \quad \text{, for all } j = 1,..,C,\ p = 1,.., m \qquad (1c)$$

$$\sum_{j=1}^{C} y_{jp} = 1 \quad \text{, for all } p = 1,.., m \qquad (1d)$$

$$\sum_{p=1}^{m} y_{jp} \le 1 \quad \text{, for all } j = 1,..,C \qquad (1e)$$

$$x_{ij} = 0,1;\ y_{jp} = 0,1 \quad \text{, for all } i = 1,..,n\ ,\ j = 1,..,C\ ,\ p = 1,..,m \qquad (1f)$$

The objective function (1a) calculates, for any feeder tray set-up, the assembly time required for the given component insertion sequence. Constraint (1b) is used to guarantee that for each placement location, only one feeder tray position is allocated, and constraint (1c) is used to ensure that a component can only be obtained from a feeder tray position with that component type. Besides these two constraints, constraint (1d) is used to ensure that a component type is only once on the feeder tray while constraint (1e) ensures that there can be at most one component type per feeder tray slot.

2.3.4 Mathematical formulation for subproblem 2

Recall that subproblem 2 involves finding a component placement sequence given a feeder tray set-up. In the formulation given below, we assume that the component retrieved is the next component to be placed. By doing this, we can neglect the notion that a component is retrieved several pick-and-place cycles before it is actually placed. We also assume that there is a home position on the PCB which is where the placement sequence starts and stops. We should note that if we assume that the x-y table and feeder tray are already in position for the retrieving and placing of the first component, and that there is no cost incurred by the x-y table and the feeder tray having to go back to their respective origins due to overhead times, then it is actually a minimum cost Hamiltonian path problem we have to solve and not a TSP.

Given the feeder tray set-up, let :

S = Set of component locations (x,y) on the PCB plus the home position.

Let S= {0,1,2,..., n}

t_{ij} = the time required to place component at location j after placing component at location i, (note that this is calculated as the maximum of the turret time, x-y table time and feeder tray time)

Decision Variables :

x_{ij} = 1 if placement location i is followed by placement location j, and is 0 otherwise.

A TSP formulation :

$$\text{Minimize} \quad \sum_{j=0}^{n} \sum_{i=0}^{n} t_{ij} x_{ij} \qquad (2a)$$

Subject to

$$\sum_{i=0}^{n} x_{ij} = 1 \quad , \text{for all } j = 1,2,..,n \qquad (2b)$$

$$\sum_{j=0}^{n} x_{ij} = 1 \quad , \text{for all } i = 1,2,..,n \qquad (2c)$$

$$\sum_{i \in L} \sum_{j \in L} x_{ij} \leq |L|-1 \quad , \text{for all } L \subset S, L \neq \varnothing, L \neq S \qquad (2d)$$

$$x_{ij} = 0 \text{ or } 1 \quad , \text{for all } i = 1,..,n \, , j = 1,..,n. \qquad (2e)$$

The objective function (2a) is to minimize the amount of assembly time required. Constraints (2b) and (2c) guarantee that each component placement position (x,y) is only visited once, while constraint (2d) is a subtour elimination constraint which ensures that the final placement sequence is connected.

2.4 Machine specifications used for testing

Due to making our machine description as general as possible, we avoided having to give machine specific values. For the testing of methods, however, we had to decide on a very specific machine from our general form. The specific values we decided on are as follows:

i) C = 100,

ii) A 2 headed turret which makes k = 1,

and the following speeds for our five main operations. These speeds are that of the Fuji CPII high-speed machine [BCF94].

Operation	Operation Rate
Pick-up operation	0.10 seconds
Placement operation	0.10 seconds
x-y table movement	20 mm / 0.15 seconds
Feeder tray movement	1 slot / 0.15 seconds
Turret rotation.	1 station / 0.15 seconds

Table 2.1

CHAPTER 3

Methods from the literature and proposed improvements

Many different heuristic methods have been proposed for optimizing PCB production on pick-and-place machines. All of these methods must provide solutions to two problems:

i) deciding the feeder tray set-up, which determines the feeder tray movement and feeder tray delay.

ii) deciding the component placement sequence, which determines the x-y table movement and the x-y table delay.

The actual assembly time of a PCB depends on the solutions to both i) and ii), and how these solutions interact. However, due to the complexity of considering both problems simultaneously, many of the methods concentrate on solving only one of the problems first, and then solves the second problem in light of the solution obtained to the first.

In order to obtain an idea of the current approaches that exist for optimizing PCB production time and to see which methods perform the best, we will describe two heuristic methods from the literature. The first method, which we call the *Place-by-Type* method, was introduced by Ahmadi and Sciomachen [AS92], and focuses on the problem of determining a good feeder tray set-up. The second

method, known as the *Pairwise Exchange* method, was introduced by Leipala and Nevalainen [LN89], and concentrates on successively fixing the feeder tray set-up and the component placement sequence. In addition to describing these methods in detail, we will also describe some possible improvements to these methods which 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 four different PCBs for which the data was given to us by Mitel. The machine specifications used are as described in Section 2.4.

3.1 The Place-by-Type method

The place-by-type method for the PCB assembly problem was proposed by J. Ahmadi and A. Sciomachen [AS92]. In this method, all components of the same type are placed consecutively, and the component reels are ordered on the feeder tray to match the order in which the corresponding component types are placed. As a result, the feeder tray movement will consist simply of moving consecutively through the feeder tray positions once. Consequently, this method minimizes feeder tray movement, which affects machine wear and need for calibration. Furthermore, if the feeder tray movement between adjacent positions is masked by the turret indexing delay, such a solution will have no feeder tray delays.

3.1.1 Determining the component placement sequence

To ensure that all placements of a given component type are placed consecutively, all placement locations that require the same component type are grouped together into a *state*. The component placement sequence is then solved

for each state, and then an ordering of the states is decided. The resulting component placement sequence places components one state at a time.

To achieve such a placement sequence, two different procedures were developed. The first is called the *inner-state-first* procedure. This procedure first solves the component placement sequence for each state, and then solves the ordering of the states. The second is called the *inter-state-first* procedure. This procedure first solves the state ordering subproblem and then solves the component placement sequence for each state.

Both the problem of obtaining an optimal component placement sequence for each state and the problem of obtaining an optimal state ordering are formulated as TSP problems, and then TSP heuristics are used to obtain solutions. In the paper, the furthest-insertion heuristic is used to obtain an initial tour for the TSP problem, and then either the 2-opt or 3-opt tour improvement heuristics are used to improve this tour. The 3-opt heuristic produces better results than the 2-opt heuristic, but it requires substantially more time. Consequently, to maintain a fast implementation, the faster 2-opt heuristic is used for the large TSP problems, while the slower 3-opt heuristic is used for the smaller TSP problems.

Before discussing the above procedures in detail, we note that the place-by-type method was originally described for the situation in which the assembly of each PCB begins at a home position. Here we describe the method adapted to our form of the problem in which there is no home position.

3.1.2 The Inner-State-First Procedure

The inner-state-first procedure determines the optimal placement sequence for each state independently. It then determines a state ordering that combines the placement sequences for these states in such a way that an optimal placement sequence is obtained. To use the terminology from the paper, the inner-state sequences are found first and then the inter-state ordering is determined.

Let G=(V,E) be the complete weighted graph for which each node i corresponds to the component placement location (x_i,y_i), and such that, for each edge (i,j) the corresponding cost d_{ij} is the max of ($|y_j-y_i|$,$|x_j-x_i|$). Recall that this distance function is known as the l_∞ metric.

The inner-state-first procedure is as follows:

a) Determine a minimum cost Hamiltonian path for each state in the corresponding subgraph of G.

b) Reduce each Hamiltonian path to a single edge whose cost is the cost of the path. Let K be the set of such edges.

c) Construct a new weighted graph G′ which contains the edges K generated in Step b), as well as all edges from the original graph G which join end nodes of edges in K.

d) Construct a minimum cost Hamiltonian path in G′ which contains K. The order in which the edges K are visited by this path supplies the ordering of the states.

3.1.3 The Inter-State-First Procedure

The inter-state-first procedure also tries to obtain a component placement sequence by obtaining an optimal placement sequence for each state and an optimal state ordering. The difference is that the optimal ordering of the states is determined before the component placement sequence for each state. Using the terminology from the paper, the inter-state sequence is determined first and then the inner-state sequences follow.

Let G be defined as above.

The inter-state-first procedure is as follows:

a) For every pair of states I and J with $I \neq J$, find the shortest distance d_{ij} with $i \in I$ and $j \in J$.

b) With each state represented as a node, determine a minimum cost Hamiltonian path P of the states in the corresponding complete graph, using the distances calculated in Step a). This determines a state order.

c) The edges in P incident with a state node now supply at most two nodes within that state as the entering and departing nodes.

d) For each state, determine a minimum cost inner-state Hamiltonian path in the corresponding subgraph of G with the entering and departing nodes from Step c) as endpoints.

It is important to note that if only one node is found as the entering and departing node for a particular state A, then the starting node from the next state is used as the departing node for A.

With all aspects of the method described, we can now summarize the method into four steps.

Summary of Method :

1) Group all placements requiring the same component type into one state.
2) Determine the component placement sequence by using either the inner-state-first procedure or the inter-state-first procedure.
3) Combine the placement sequences for each state together using the state ordering determined in Step 2.
4) Determine the feeder tray set-up ordering from the state ordering sequence.

3.1.4 An example

Consider the following PCB production problem.

INPUT

1) The (x,y) locations of the components.
2) The type of component required at each location.

Node Number	x location	y location	Component type
1	1	12	A
2	5	11	A
3	6	14	A
4	9	10	A
5	15	27	B
6	15	24	B
7	18	24	B
8	18	21	B
9	20	27	B
10	21	23	B
11	22	20	B
12	13	6	C
13	15	4	C
14	16	6	C

TABLE 3.1

To illustrate the place-by-type method, we will use it twice on the above example: first using the inter-state-first procedure, and second using the inner-state-first procedure.

Using the inter-state-first procedure

Step 1:

Group the placements that require the same component type together.

Node Numbers

State A: {1,2,3,4}.

State B: {5,6,7,8,9,10,11}.

State C: {12,13,14}.

Step 2:

Apply the inter-state-first procedure.

a) Determine the shortest distance between any pair of states assuming the l_∞ metric is being used as the distance function. (See Section 2.1.)

Shortest Distance Between any Two States

	State A	State B	State C
State A	0	10	4
State B	10	0	14
State C	4	14	0

TABLE 3.2

We must also keep track of the nodes that generated these shortest distances.

The nodes which produced these shortest distances are:

State A-B : Node 3 to Node 6

State A-C : Node 4 to Node 12

State B-C : Node 11 to Node 14 .

b) Using the inter-state distances above and considering each state as a node, we must find a minimum cost Hamiltonian path of the states.

The minimum cost Hamiltonian path found for the states results in the following order:

(B,A,C).

c) The state ordering given above now lets us determine at most two nodes in each state as entering and departing nodes.

The resulting interconnection nodes between the states (B,A,C) are as follows:

(11,6)-> (3,4)-> (12,14).

d) Determine each inner-state's minimum cost Hamiltonian path using the restrictions for the end nodes of each path determined in part (c) of this procedure.

From (c), we know the start and end nodes for the Hamiltonian path in each state. By using this information and applying TSP heuristics, we can determine Hamiltonian paths which are hopefully close to optimal for each state.

The Hamiltonian paths found for each state are as follows:

State A: 3,1,2,4

State B: 11,10,8,7,9,5,6

State C: 12,13,14.

Step 3:

Connect the Hamiltonian path from each state in the state order that was determined by the inter-state-first procedure.

The resulting component placement sequence is as follows: 11,10,8,7,9,5,6,3,1,2,4,12,13,14.

Step 4:

With the component placement sequence, determine the feeder tray set-up.

The feeder tray set-up is determined by placing the component reels in the state order that was determined by the inter-state-first procedure.

The resulting feeder tray set-up is: Reel B,A,C.

Using the inner-state-first procedure

INPUT

Use the same input as in the above example.

Step 1:

Group the placements that require the same component type together.

Node Numbers

State A: {1,2,3,4}.

State B: {5,6,7,8,9,10,11}.

State C: {12,13,14}.

Step 2:

Apply the inner-state-first procedure.

Inner-state-first procedure :

a) Determine a minimum cost Hamiltonian path for each state assuming the l_∞ metric is being used as the distance function.

Hamiltonian Paths

State A : 1,2,3,4

State B : 9,7,5,6,8,10,11

State C: 12,13,14

b) Reduce each Hamiltonian path to a single edge. Let K be the set of such edges.

The set of edges K are as follows:

edge 1 : (1,4)

edge 2 : (9,11)

edge 3 : (12,14)

c) Construct a new graph G′ which contains the edges K generated in Step b), as well as all edges from the original graph which join end nodes of edges in K.

The edges that would be included in the new graph G' are:

(1,4),(9,11),(12,14),(1,9),(1,11),(1,12),(1,14),

(4,9),(4,11),(4,12),(4,14),(9,12),(9,14),(11,12),(11,14)

d) For G′, find a minimum cost Hamiltonian path which contains K. The order in which the edges K are visited by this path supplies the ordering of the states.

A minimum cost Hamiltonian path is

1,4,12,14,11,9.

The resulting ordering of the states is

(A,C,B).

Step 3:

Connect the Hamiltonian path from each state by using the state order that was determined in Step 2d).

The resulting component placement sequence is as follows:

1,2,3,4,12,13,14,11,10,8,6,5,7,9.

Step 4:

With the component placement sequence, determine the feeder tray set-up.

The feeder tray set-up is determined by placing the component reels in the state order that was determined in Step 2d)

The resulting feeder tray set-up is:

Reel A,C,B.

3.2 Proposed improvements for the Place-by-Type method

3.2.1 Solution strategies and modifications used in our implementation

In [AS92], the details of how to execute certain steps of the algorithms for the place-by-type method were omitted. Below, we discuss the solution strategies we used to carry out these steps in our implementation. We also discuss alternate solution strategies we used for some other steps which we feel will improve the method's performance.

In both Step a) of the inner-state-first procedure and Step b) of the inter-state-first procedure, a minimum cost Hamiltonian path must be found. However, there was no solution strategy given in the paper to achieve this. In our implementation, we use the method described earlier in Section 1.6 to convert this

to a TSP to obtain these Hamiltonian paths. There is also no solution strategy specified for Step d) of the inter-state-first procedure. Here a minimum cost Hamiltonian path must be found for each state with the restriction that the ends of the Hamiltonian path are given. In our implementation, we convert this problem to a TSP using the method described previously in Section 1.6.

In Step d) of the inner-state-first procedure, a minimum cost Hamiltonian path must be found in the new graph G′ which contains the edges K representing the Hamiltonian paths for each state. In [AS92], this is solved by obtaining a solution to a minimum 2-matching problem on G′ which must contain the edges K. This solution may consist of several subtours. To obtain a single tour, a patching heuristic is used which continues to patch the two largest subtours together until a single tour is obtained. This patching heuristic, however, cannot remove any of the K edges. In our implementation, we instead used the method described previously in Section 1.6 which converts this problem to a TSP for Step d). We believe this method will work as well as, if not better, than the method used in [AS92] due to the excellent heuristics available for the TSP.

Another enhancement we made to the method in our implementation was to use better tour improvement heuristics for solving the related TSP problems. We used the Lin-Kernighan heuristic [LK73] which outperforms both the 2-opt and 3-opt tour improvement heuristics used in [AS92]. One disadvantage, however, is that this heuristic requires substantially more processing time to obtain a solution. We believe that this is the reason that the authors did not propose this heuristic. However, the processing power of computers has grown significantly, allowing for the possibility of using computationally intensive heuristics without the large time requirements previously needed.

Our final enhancement to the method which we did not implement but would at least slightly improve the place-by-type method involved Step c) of the inter-state-first procedure. Recall that in this step, the entering and departing nodes for each state are determined. These nodes are then used as the endpoints for the Hamiltonian path for that state. Often, though, the same node will be found as the entering and departing node. If this occurs, a tour would be generated in the state resulting in one component location being visited twice. As a result, they suggest keeping the single node found for a particular state A as the entering node for that state and then using the entering node of the following state as the new departing node for state A. This solution strategy, however, could be easily improved by a simple enhancement.

Our enhancement is to consider, as another possible solution, using the single node found in a state A as the departing node and to use the departing node of the previous state as the entering node for state A. From their solution and our solution, we now choose for state A the solution which produces the minimum cost path from the departing node of the previous state of A to the entering node of the state that follows state A. This will produce a solution as good as or better than the [AS92] solution.

Another strategy for solving the single node problem which we developed and implemented is called the *second closest node* strategy. This strategy uses the single node found as the entering node and then finds, of the remaining nodes, the one which is closest to any node in the state that follows (hence the name, the second closest node strategy). This node is then used as the departing node of this

state. This solution strategy, however, produced similar results to their method. Consequently, we decided not to use it.

3.2.2 The State-Combining Variation

For many pick-and-place machines, the feeder tray is able to move one slot or more during the free time caused by the turret indexing delay. This provides the possibility of grouping several types of components into a state rather than just one, which can improve the associated component placement sequence. To illustrate this, consider finding a component placement sequence for the graph shown in Figure 3.1. In this graph, each node represents a position on a PCB where a particular component must be inserted. The "o" nodes represent one component type and the "x" nodes represent a second component type.

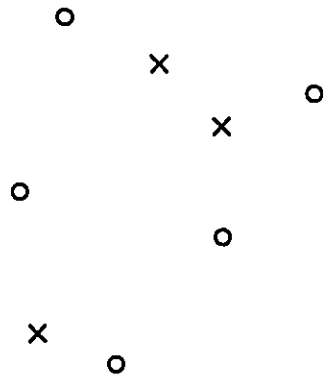
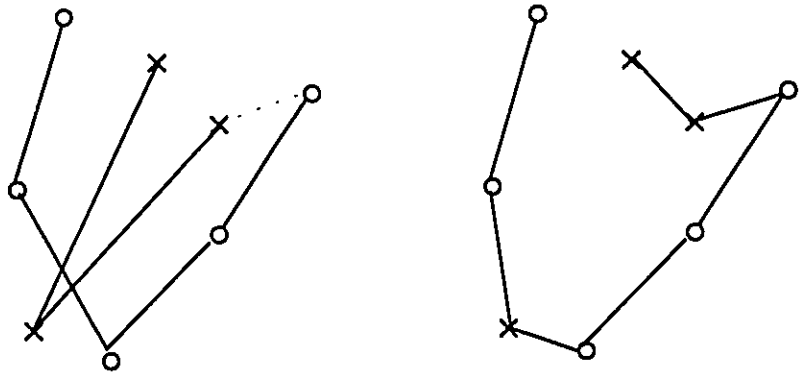


Figure 3.1
Locations On A PCB

If we combine each component type into a state, the component placement sequence consists of the two Hamiltonian paths shown in Figure 3.2 a) plus the dotted edge joining them. If we combine both component types into a single state,

the component placement sequence consists of the Hamiltonian path shown in Figure 3.2 b).



a) Using Two States b) Using one state

Figure 3.2

Clearly, this component placement sequence is shorter than the one shown in Figure 3.2 a). Thus, by combining two Hamiltonian path problems into one, we can improve our resulting solutions.

Taking into consideration the above argument, we propose the *state-combining* variation of the place-by-type method which is outlined below.

The State-Combining Variation

1) Calculate the number k-1 of feeder tray positions that the feeder tray is able to move during the turret indexing delay. The number k indicates the maximum number of component types that can be grouped together without causing any processing delay from the feeder tray to place each group.

2) Combine component types into groups of size at most k. Each of these groups forms a state.

3) With these states, solve the component placement sequence by using the inner-state-first or inter-state-first procedures. These procedures, however, must ensure that the entering and departing nodes for each state are of a different component type.

4) As in the place-by-type method, the component reel order is determined by the state order. Within each state, the feeder order must begin with the first component type placed for that state. The remaining feeder order for that state can be made arbitrarily.

By grouping the maximum k component types together, we ensured that no processing delay from the feeder tray will occur while placing the components from each state. Between states, however, a feeder tray jump of 2k-1 could occur in the worst possible scenario. To avoid any more than k-1 feeder jumps between states, we enforced two restrictions. The first, described in Step 4, ensures the first component placed for each state has its' component type placed in the feeder tray before any other component type for that state. This reduces the maximum number of feeder jumps between states to be k. The second restriction, given in Step 3, ensures the first and last component types placed for a state are different. With these two restrictions, we guarantee a maximum feeder tray jump between states of k-1 as required for no feeder tray delays.

Of these two restrictions, the only one that could be difficult to enforce is the second one. We satisfy this restriction by randomly partitioning the component

types for each state into two sets. We now place a new node into the graph for each of the sets. The distance from each of these new nodes to the nodes in its associated set is zero. All other distances from these new single nodes to any other node are set to a large enough value to ensure that these edges are not used in the Hamiltonian path to be found (see Section 1.6). Now find a minimum cost Hamiltonian path in this graph with the two new nodes as end nodes. The path obtained from this by removing the two new nodes will have the entering and departing nodes for that state from different sets and consequently from different component types.

Note that the state-combining variation guarantees no feeder tray delay, but it does not necessarily minimize the amount of feeder tray movement, as does the place-by-type method. However, it will provide better assembly times for PCB's which in most applications is the primary goal.

For Step 2 of the state-combining procedure, we must decide which component types should be combined. We propose the following two strategies for this:

1) The adjacent reel strategy.

This strategy is based on grouping component types together which are adjacent in the feeder tray set-up produced by the place-by-type method. This feeder tray set-up corresponds to the same order that the component types are placed in by the place-by-type method. By grouping k component types together which are adjacent in the feeder tray set-up, we are guaranteed to improve the placement sequence, and hence the assembly time of the PCB.

2) The close group strategy.

This strategy is based on the idea that it is good to group types together that are "close". In order to do this, we propose forming a complete graph G whose nodes correspond to the different component types. Then, for any pair of component types I and J, calculate the distance d_{IJ} between them in G as follows. For every $i \in I$, let m_i be the distance from i to the closest component in J. Then let $d_{IJ}=MAX(m_i)$ where $i \in I$. This measures the furthest you will ever have to travel from some component in I in order to reach the closest component in J, and thus can be considered a measure of the "closeness" of states I and J. To divide the states into groups of size $\leq k$, we need to heuristically find the minimum weight clique of size k in G, remove these nodes from G, and repeat.

3.2.3 Empirical Results

We implemented the place-by-type method and the state-combining variation (using the adjacent reel strategy) on an IBM-PC using Turbo Pascal version 7.0. The state-combining variation, however, was only implemented for the inner-state-first procedure which seems to produce the best results. In addition, we used $k = 2$ for the state-combining variation, i.e. we combined component types into groups of size ≤ 2. We used these implementations to find assembly times for PCB's from Mitel Industry ranging in size from 129 to 281 components. The results are shown below in Table 3.3.

			Place-by-Type Method		State-Combining Variation
Board #	Number of components	Number of component types	Inter-State-First	Inner-State-First	Adjacent reel strategy
13501201	129	54	42.74s	38.69s	37.0s
13424801	202	48	83.54s	68.97s	66.65s
13424701	177	46	70.55s	61.87s	58.5s
13588801	281	53	96.44s	88.76s	85.46s

TABLE 3.3

From the statistics above, we obtain the general feeling that the place-by-type method performs better when the inner-state-first procedure is used in the method instead of the inter-state-first procedure. This was also the conclusion reported in [AS92]. Furthermore, these statistics support our claim that the state-combining variation using the adjacent reel strategy can improve assembly times by allowing more feeder tray movement during the free time caused by the turret indexing delay.

3.3 The Pairwise Exchange method

The pairwise exchange method was introduced by T. Leipala and O. Nevalainen in [LN89]. This method was created for a one head sequential inserter and is based on the concept of improving a given solution (i.e. a feeder tray set-up and component placement sequence) by pairwise exchanges of the component

reels and using a modified TSP heuristic. In checking pairs of component reels, two component reels i and j are swapped positionally on the feeder tray if such a swap results in a decreased PCB assembly time.

An overview of the pairwise exchange method is as follows.

1) Find an initial feeder tray set-up F.
2) Using F, find a component placement sequence S which minimizes assembly time. Let t be the assembly time obtained.
3) Using S, perform pairwise exchanges of reels in F wherever such an exchange improves production time. Let the final resulting feeder tray set-up be F′.
4) Using F′, find a placement sequence S′ which minimizes assembly time. Let t′ be the current assembly time. If t′<t, then let t=t′, F=F′, S=S′ and go to Step 3. Otherwise, stop.

Below we describe in detail the main steps of the method.

3.3.1 Finding an initial feeder tray set-up

In [LN89], three different methods are proposed for obtaining an initial order for the component reels. These methods are based on completely different solution strategies. The first method uses a minimum cost spanning tree, the second uses a TSP solution, and the third uses a concept based on component usage. Here we only describe the second method, which produced the best initial values in [LN89].

Let G=(V,E), and assume the l_∞ metric for the distance function.

i) Determine a minimum cost Hamiltonian path in G to determine a component placement sequence.

ii) Determine a random feeder tray set-up.

iii) Apply pairwise exchanges on the feeder tray set-up until the assembly time no longer improves. The resulting feeder tray set-up is used as the initial feeder tray order for the component reels.

3.3.2 Finding a component placement sequence

Both steps 2) and 4) of the pairwise exchange method involve finding a component placement sequence which minimizes production time while using a given feeder tray set-up. In [LN89], they formulate this problem as a minimum cost Hamiltonian path problem which they solve by using a modified furthest insertion TSP heuristic. This modified heuristic tries to obtain a minimum cost path instead of a minimum cost Hamiltonian cycle. In addition, they modified the heuristic to start at an origin point zero and end on a particular node *f* instead of starting and ending on two arbitrary nodes which are far apart. By restricting the end node, however, several Hamiltonian paths would not be considered as possible solutions. To avoid this, all alternatives for *f* which are not close to the origin are considered. With these modifications made to the furthest insertion TSP heuristic, Hamiltonian path problems can be solved.

3.3.3 Performing pairwise exchanges on the feeder tray

Step 3 of the method involves a pairwise exchange of the component reels until no further improvement of the production time occurs. The details of this process are as follows:

REPEAT
  i=1 to n (the number of component reels)
    j=i+1 to n
      Swap component reel i with j if the assembly time improves.
UNTIL no swaps occur (i.e. assembly time can no longer be improved by component reel swaps)

This pairwise exchange algorithm checks all possible pairs of component reels as possible swaps.

Note that in the problem formulation in [LN89], they permit multiple reels of the same component type. Thus in their method, their pairwise exchanges are of all possible pairs of component types and not of all possible pairs of component reels. For instance, when component types i and j are exchanged, all feeder tray locations with component types i and j are swapped to feeder tray locations holding component types j and i respectively. However, when multiple reels of the same component type are not permitted as in the general form of our problem, then this reduces to the process described above.

3.3.4 An example

Consider the following PCB production problem.

INPUT

1) The (x,y) locations of the components.

2) The type of component required at each location.

Node Number	x location	y location	Component Type
1	30	60	B
2	60	130	A
3	100	70	A
4	160	90	B
5	160	120	A
6	120	150	C
7	190	60	C
8	240	90	C

TABLE 3.4

3) The x-y table distance matrix obtained by using the l_∞ metric.

(All measurements are in millimeters).

Node	1	2	3	4	5	6	7	8
1	0	70	70	130	130	90	160	210
2		0	60	100	100	60	130	180
3			0	60	60	80	90	140
4				0	30	60	30	80
5					0	40	60	80
6						0	90	120
7							0	50
8								0

TABLE 3.5

4) Table movement rate = 35 units / 1.5 sec.

5) Feeder tray rate = 1 slot / 1.5 sec.

6) Turret rotation rate = 1 station / 1.5 sec.

7) Pick-up and placement time = 0.1 sec.

8) A 12 headed turret which makes k = 6.

STEP 1:

Find an initial feeder tray set-up F.

i) Determine a minimum cost Hamiltonian path for the component placement sequence using the l_∞ metric.

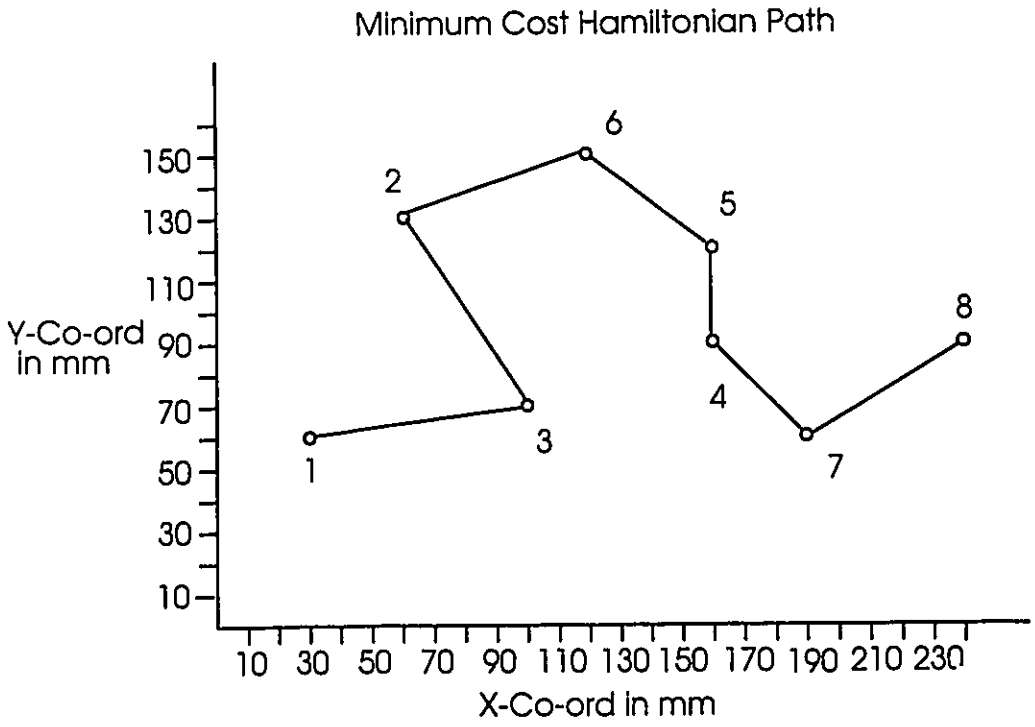


Figure 3.3

The resulting Hamiltonian path using only the board information is the following node order: 1,3,2,6,5,4,7,8 .

ii) Determine a random feeder tray set-up.

A random feeder tray set-up for the reels is

A,B,C .

iii) Apply pairwise exchanges on the feeder tray set-up until the assembly time no longer improves. Assume the assembly time calculation given in Section 2.3.1.

We first calculate the initial assembly time using the feeder tray set-up and the component placement sequence we determined above.

Initial assembly time = 0.1+1.6+1.6+3.1+3.1+1.6+1.6+3.1+
2.671+2.671+1.814+1.6+1.6+2.243
= 28.399 sec.

Test swapping reels 1 and 2:

Try pairwise exchange of component reels 1 and 2, i.e. feeder tray set-up (B,A,C).

New assembly time = 0.1+1.6+1.6+1.6+1.6+1.6+3.1+3.1+
2.671+2.671+1.814+1.6+1.6+2.243
= 26.899 sec.

Assembly time is improved. Continue swapping using the new feeder tray set-up (B,A,C).

Test swapping reels 1 and 3:

The resulting feeder tray set-up is (C,A,B).

New assembly time = 0.1+1.6+1.6+1.6+1.6+1.6+3.1+3.1+
2.671+2.671+1.814+1.6+1.6+2.243
= 26.899 sec.

No improvement in assembly time. Leave feeder tray set-up as (B,A,C) and continue swapping.

Test swapping reels 2 and 3:

The resulting feeder tray set-up is (B,C,A).

New assembly time = 0.1+3.1+1.6+1.6+1.6+3.1+1.6+3.1+
2.671+2.671+1.814+1.6+1.6+2.243
= 28.399 sec.

No improvement in assembly time. Leave feeder tray set-up as (B,A,C).

Reapply the algorithm for the pairwise exchanges of component reels:
Due to an improvement in assembly time, we must reapply the algorithm for the pairwise exchanges of component reels using the feeder tray set-up (B,A,C).

From the above, the initial assembly time = 26.899 sec.

Test swapping reels 1 and 2:
The resulting feeder tray set-up is (A,B,C).
New assembly time = 0.1+1.6+1.6+3.1+3.1+1.6+1.6+3.1+
2.671+2.671+1.814+1.6+1.6+2.243
= 28.399 sec.
No improvement. Leave feeder tray set-up as (B,A,C).

Test swapping reels 1 and 3:
The resulting feeder tray set-up is (C,A,B).
New assembly time = 0.1+1.6+1.6+1.6+1.6+1.6+3.1+3.1+
2.671+2.671+1.814+1.6+1.6+2.243
= 26.899 sec.
No improvement. Leave feeder tray set-up as (B,A,C).

Test swapping reels 2 and 3:

The resulting feeder tray set-up is (B,C,A).

New assembly time = 0.1+3.1+1.6+1.6+1.6+3.1+1.6+3.1+
2.671+2.671+1.814+1.6+1.6+2.243
= 28.399 sec.

No improvement .

The pairwise exchange algorithm is complete and generated no improvements. The resulting feeder tray set-up and thus, the initial feeder tray set-up F is (B,A,C).

STEP 2 :

Using F = (B,A,C) find a component placement sequence S which minimizes assembly time. Let t be this assembly time.

The resulting component placement sequence which minimizes assembly time did not change and is as follows:

S = 1,3,2,6,5,4,7,8.

Thus, the resulting assembly time would not change and is

t = 26.899 sec.

STEP 3 :

Using S, perform pairwise exchanges of reels in F whenever such an exchange improves production time. Let the final resulting feeder tray set-up be F'.

Since the new component placement sequence S did not change from the original component placement sequence, applying the pairwise exchange algorithm again will not improve production time. As a result F′ = (B,A,C).

STEP 4 :

Using F′, find a placement sequence S′ which minimizes production time. Let t′ be the current production time. If t′< t, then let t=t′, F=F′, S=S′, and go to Step 3. Otherwise, stop.

Using F′, we obtain the same placement sequence as S. Thus S′ = S and the production time t′ = 26.899 sec. Since t′ = t, we stop with the following feeder tray set-up and component placement sequence.

Feeder tray set-up = (B,A,C).
Component placement Sequence = 1,3,2,6,5,4,7,8.

3.4 Proposed improvements for the Pairwise Exchange method

3.4.1 Modifications used in our implementation

In our implementation of the pairwise exchange method, we had to modify the method used for finding a good component placement sequence when the feeder tray set-up is given (i.e. Steps 2 and 4 in the pairwise exchange method). The reason is that their method was created for a one headed sequential inserter. With this type of machine, a component is picked up and is immediately inserted into the PCB. Due to this property, the cost to insert a component *j* after inserting

component i will be calculated from the x-y time required to traveled between component location i and component location j, and the feeder tray time required to traveled between the component reel holding component i and the component reel holding component j. Since the cost between any two component placements can be determined and the cost to insert component j after component i is the same as the cost to insert i after j (i.e. symmetric), they could formulate this problem as a minimum cost Hamiltonian path problem and use their modified furthest insertion TSP heuristic to solve it.

For a $2k$ headed turret machine, we know that the feeder tray is always k components ahead of the x-y table. When the i^{th} component in a component placement sequence is being placed the $i+k^{th}$ component in the same sequence will be picked up. Due to this fact, we are not able to determine the cost between any two component placements given the feeder tray set-up unless the component placement sequence is also known, and the problem of finding a cheapest component placement sequence can no longer be formulated as a TSP.

We tried two different approaches for finding a good component placement sequence when the feeder tray set-up is given. The first approach is based on the 2-opt tour improvement heuristic for the TSP (see [L86] for a description of this heuristic). The main step in this approach is to take an existing component placement sequence and remove two non-adjacent edges. Two new edges are then inserted to reconnect the tour in a different fashion. This edge swap is kept if the new resulting tour is cheaper. Usually, the cost of the two edges removed is compared to the cost of the two new edges added to determine if an improvement has been made. For our problem, however, when the tour changes , the cost between component placements changes, as it depends on the new tour.

Consequently, we had to use the assembly time calculation in Section 2.3.1 to see if the new tour is an improvement.

By adapting the 2-opt heuristic to recalculate the tour cost every time two edges were deleted and two new edges were added, and using it in the pairwise exchange method, a considerable amount of CPU time is required to obtain a solution to the pairwise exchange method. From our results, approximately forty minutes of CPU time was required to apply the pairwise exchange method using this 2-opt heuristic on a PCB requiring 202 component placements. This large time requirement is the result of having to recalculate the entire tour cost when any tour improvement is considered in the 2-opt heuristic, and makes this solution strategy undesirable. Note that because of this large time requirement for this adapted heuristic, we stopped our algorithm after running it for over two hours on one of the PCB, and thus obtained no solution for the PCB. (See Tables 3.6 and 3.7.)

The second solution strategy we tried for determining a cheapest component placement sequence involved ignoring the delay between the component being picked up and the one being inserted, thus treating the machine as if it was a one headed sequential inserter. This allows us to model this problem as a TSP, in which case we can use the very successful Lin-Kernighan TSP improvement heuristic to try to minimize the x-y table and feeder tray travel distances. Although we are not minimizing assembly time directly, we should note that minimizing the x-y table and feeder tray travel distances will generally decrease the assembly time of a PCB. The reason for this is that although the feeder tray is k components ahead of the x-y table in a component placement sequence $c_1, c_2, \ldots, c_n$, the actual assembly time calculation uses the time required for the x-y table to move between each of these component placement locations

and the time required for the feeder tray to move between the component reel for each of these components. Realizing this, any method that produces a Hamiltonian path that minimizes the x-y table and feeder tray movement can decrease the assembly time required by a pick-and-place machine and can be considered a heuristic for minimizing assembly time.

Although we are not minimizing assembly time directly, the results we obtained by using the Lin-Kernighan heuristic to minimize the x-y table movement and feeder tray movement for different problems was excellent. Since the assembly time is calculated based on the x-y table and feeder tray movements, we obtained excellent solutions for the assembly time of PCBs. In fact, the assembly times we obtained for different problems by using the pairwise exchange method to minimize the x-y table and feeder tray movement outperformed the results obtained by the very slow pairwise exchange method that used the modified 2-opt heuristic to minimize the assembly time directly. (See Table 3.6). We believe the reason for this is that we were able to use a much better TSP heuristic (Lin-Kernighan) to minimize the x-y table and feeder tray movement than what we could use for minimizing the assembly time problem directly due to the time requirement necessary for using even the adapted 2-opt heuristic. To see just how much CPU time is required to obtain solutions using both solution strategies, consult Table 3.7.

Recall that the machine specifications used for testing is given in Section 2.4 and is the same specifications we used when testing the place-by-type method.

Heuristic

Pairwise Method	13501201 PCB	13424801 PCB	13424701 PCB	13588801 PCB
Minimizing x-y & Feeder mov.	39.88 sec.	60.72 sec.	54.82 sec.	85.0 sec.
Minimizing assembly time	43.4 sec.	71.47 sec.	57.47 sec.	N/A

TABLE 3.6 Assembly Time Required

Heuristic

Pairwise Method	13501201 PCB	13424801 PCB	13424701 PCB	13588801 PCB
Minimizing x-y & Feeder mov.	5 Min.	12 Min.	12 Min.	36 Min.
Minimizing assembly time	7 Min.	39 Min.	27 Min.	2 Hrs. +

TABLE 3.7 CPU Time Required

One final improvement we made to the pairwise exchange method was to update the feeder tray set-up F in Step 3 right after the pairwise exchanges and eliminate the use of a temporary feeder F'. In [LN89], the pairwise exchange of component reels could improve the assembly time using an existing Hamiltonian cycle, but when a new cycle is found using this new feeder tray set-up, the assembly time could actually get worse. The reason is that the furthest insertion heuristic used in [LN89] is a TSP construction method and finds a whole new Hamiltonian cycle based on this new feeder tray set-up. Since it is a heuristic, a

better feeder tray set-up does not mean that the placement sequence found will produce a better assembly time. Consequently, when an improvement is made by pairwise exchanges, we should keep these changes and not wait to see if the new feeder tray set-up and the new placement sequence produces a decreased assembly time.

Although our problem is using TSP improvement methods which cannot make the tour any worse, we still update the feeder tray set-up F after the pairwise exchanges of components and eliminate using a temporary feeder F'. The reason is that our TSP improvement methods are being used on the problem of minimizing x-y table and feeder tray movement and not on minimizing assembly time directly. As a result, our TSP improvement method can only reduce x-y table and feeder tray movement, but it may actually increase assembly time. If the assembly time is increased more than what the pairwise exchanges decreased it then we would have lost the improvements made by the pairwise exchanges by following their algorithm. With our modification this improvement is not lost and only the new component placement sequence would be discarded.

3.5 Comparing the Place-by-Type method and the Pairwise Exchange method

In both the place-by-type method and the pairwise exchange method described earlier, we described and implemented different improvements to these methods. Due to the different versions of each method that we implemented, we will only compare the version of the place-by-type method and the pairwise exchange method that produced the best empirical results.

We can see that the empirical results for the place-by-type method given in Table 3.3 indicates that the version of the place-by-type method that produces the best results is the method that uses the inner-state-first procedure and our state-combining variation that uses the adjacent reel strategy. In addition, the empirical results we obtained for the pairwise exchange method given in Table 3.6 indicates that the version of the pairwise exchange method that produces the best results is the method that minimizes x-y table and feeder tray movement. As a result, we compared these versions of the two methods. (See Table 3.8.)

Board #	Place-by-Type Method (State Combining Var.)	Pairwise Exchange Method (min. x-y & feeder Mov.)
13424701	58.5 s	54.82 s
13501201	37.0 s	39.88 s
13588801	85.46 s	85.0 s
13424801	66.65 s	60.72 s

TABLE 3.8

From Table 3.8, we can see that in all but one of the boards we were able to obtain a better assembly time with the pairwise exchange method that minimizes the x-y and feeder tray movement than we could with the place-by-type method using the state-combining variation. We believe the reason is that the pairwise exchange method makes better use of the concurrency that exists between the x-y table and feeder tray movement. With the place-by-type method, this concurrency is not taken advantage of at all. This method ensures that there are no feeder tray

delays which often causes longer x-y table movements and thus longer delays. The pairwise exchange method, however, tries to balance the x-y table and the feeder tray movement. By doing this, the x-y table delays are decreased and the extra feeder tray delays created can usually take place during the x-y table movement (due to concurrency). The result is often a decreased assembly time.

CHAPTER 4

Lower bounds for the PCB assembly problem

With the use of heuristic methods, it is important to determine the quality of the solutions obtained. If this is not done, we have no way of determining how well our heuristics are performing. One way to determine if a heuristic solution is close to optimal is by finding an appropriate *lower bound*, which is a number k such that the cost of any feasible solution (and in particular, the cost of an optimal solution) for our problem is $\geq k$. If this lower bound value is close to the value obtained by our heuristic solutions then this indicates that the heuristic solutions are performing well. However, it should be mentioned that if the lower bound value and the heuristic solution value are not very close then this does not necessarily indicate the heuristic solution is bad. It may be that the heuristic solution is actually close to the optimal, but the lower bound is poor.

In this chapter, we report a simple lower bounding technique for the PCB assembly problem first mentioned in [LN89], and then suggest three possible ways to improve it.

4.1 A simple lower bound

To obtain a lower bound for the assembly time on high-speed machines, we must first consider the minimum amount of time required by these high-speed machines to complete a pick-and-place cycle. As previously mentioned, this minimum time requirement, known as the minimum pick-and-place cycle time,

can only be obtained when there is no feeder tray or x-y table delays. As first noticed in [LN89], by eliminating these two calculations from the pick-and-place cycle time calculations, and consequently from the assembly time calculation in Section 2.3.1, we obtain a lower bound for the PCB assembly problem.

In more detail, this lower bound for $2k$ headed turret machines is as follows. First we must determine the number of pick-and-place cycles necessary to complete the assembly of a PCB. To obtain this value, we must realize that there are k pick-and-place cycles required to load the turret and, assuming that there are n components to be placed, n more pick-and-place cycles are required to complete the assembly of the PCB. All of these pick-and-place cycles except for the very first one require at least the minimum pick-and-place cycle time. Thus, there are $(n+k-1)$ pick-and-place cycles that require at least the minimum machine pick-and-place cycle time. The initial pick-and-place cycle, however, does not require the minimum pick-and-place cycle time because the turret does not have to rotate to align any head for a component placement or pick-up. Consequently, the initial pick-and-place cycle is reduced to a simple pick-up operation. Knowing this, a lower bound LB1 for our $2k$ headed turret machine can be calculated as follows:

$$LB1 = P + (n+k-1)*(Turr+P),$$

where

P = Either the pick-up or placement time and

Turr = Turret indexing delay.

4.2 Methods for obtaining improved lower bounds

The lower bound LB1 given above assumes that the required x-y table and feeder tray movements can be completed during the turret indexing delays. However, by assuming no delays from the x-y table, we may be losing information that could be used to improve the lower bound. Below we describe three methods that attempt to obtain a better lower bound for the assembly time of a PCB by determining the minimum amount of extra x-y table time that will be required beyond the turret indexing delay for each component placement on a PCB.

4.2.1 The next closest component strategy

The first strategy we thought of for obtaining an improved lower bound is to simply add on to the lower bound LB1 a minimum amount of x-y table delay that will be generated from necessary x-y table movements. To obtain this extra x-y table delays, we use a method called the *next closest component strategy*. We calculate from each component placement the x-y table travel distance to the next closest placement location on the PCB. By determining the minimum x-y travel distance from each component location, we can determine if any of them require more time than the turret delay. For the ones that do, the difference between the turret delay time and the minimum x-y travel time for each of them will be the minimum amount of x-y table time required to position the PCB from each of those particular component locations to some other location. If we now add up all these differences, we will produce the minimum amount of x-y table delay that could not be completed during the turret delay but is still necessary to assemble the PCB. This minimum amount of time can also be considered the *extra x-y table time* (ExyTT).

It is important to mention that the component placement sequence we are finding is a Hamiltonian path. For this reason, the last location in this path does not have the x-y table traveling to any other location on the PCB. We did, however, incorporate the extra x-y table delay, if any, that could not be completed during the turret delay, from this last component to the next closest location. In the worst case scenario, this would be the maximum delay we added. Consequently, we have to subtract out the maximum delay we added to the ExyTT before adding it to the lower bound found earlier.

The improved lower bound LB2 obtained for the PCB assembly problem using the above strategy is as follows:

$$LB2 = P + (n+k-1)*(Turr+P) + ETt$$

Where

n = The number of component placements.

Turr = Turret indexing delay.

P = Either the pick-up or placement time.

ETt = ExyTT - Max x-y table delay that was added to ExyTT.

4.2.2 Two strategies based on TSP lower bounds

Before considering our next two strategies for obtaining improved lower bounds on the assembly time of a PCB, it is important to note that if we ignore feeder tray delay in the PCB assembly problem, then the problem becomes a minimum cost Hamiltonian path problem, where the cost c_{ij} from placement location $i=(x_i,y_i)$ to $j=(x_j,y_j)$ is calculated as follows:

$c_{ij} = \max (T_{|yj-yi|}, T_{|xj-xi|}, Turr)+P,$

where

P = Either the pick-up or placement time

Turr = Turret indexing delay.

$T_{|a-b|}$ = The x-y table time required to travel a distance of |a-b|.

As this minimum cost Hamiltonian path problem can be converted to a TSP using the method described in Section 1.6, we can use lower bounding techniques from the TSP to determine a lower bound on the cost of the optimal Hamiltonian path, which is $\leq$ the optimal PCB assembly cost. Note the assembly time corresponding to a Hamiltonian path must include the time to load the turret and place the first component since these times are not included in the assembly time obtained from a Hamiltonian path. This load time includes the pickup time for the first component and the pickup time and the turret index delay for the k-1 other components needed to finish loading the turret. Finally, we add the time required to place the first component which includes the turret indexing delay and the placement time. Thus, we can add

TurrLoadP = P+k*(Turr+P)

to our lower bound calculation.

The first TSP lower bounding technique that we consider is the one obtained by finding a minimum cost spanning tree [L86], which was also suggested in [LN89] but without any details. By calculating the minimum cost spanning tree using c_{ij} above, its cost plus TurrLoadP will give us a lower bound on the assembly time of a PCB. We call this bound LB3.

Note that since the cost of a minimum spanning tree is also a lower bound for a minimum cost Hamiltonian path, we do not need to convert this to a TSP problem to calculate LB3.

The second TSP lower bounding technique that we consider is the 1-tree relaxation technique described by Held and Karp [HK70]. This method starts off by finding a minimum cost spanning tree. It then finds for each of the nodes that have degree one the second closest node in the tree. The node with the largest cost is then joined to its second closest node forming a unicyclic graph called a 1-tree. The sum of the edge costs in this 1-tree is a lower bound on the length of a minimum cost tour. Following this, costs involving nodes with degree>2 are penalized while costs involving nodes with degree=1 are rewarded. These costs are changed in such a way that the change does not affect the choice of the optimal tour, only its value. The method is then repeated until all nodes have degree 2 or the user stops the algorithm.

If HK is the lower bound calculation using the above method, then our lower bound is HK plus the cost to load the turret. (Note that the time for the first component placement is already included in the lower bound of the 1-tree.) Consequently, our lower bound is

LB4 = HK + TurrLoad

where

TurrLoad = P+(k-1)*(Turr+P).

4.3 Results

In order to determine which of these lower bounding methods seems to produce the best lower bounds for the assembly time of PCBs, we compare the lower bounding methods on ten different PCBs. These lower bounds were obtained assuming the machine specifications given in Section 2.4, except for the x-y table movement rate. Here we assumed that the machine could only move 5mm during the turret indexing delay instead of 20mm. This was done to avoid applying lower bounding techniques that cannot take advantage of calculating extra x-y table time because the components are so densely packed that the x-y table can always get to a different component placement during the turret indexing delay. Consequently, slowing down the x-y table ensures that x-y table delays will occur for a PCB allowing us to emphasize which lower bounding technique described above works best. The results we obtained are as follows:

Board #	# of components	LB1 (Min. machine cycle time)	LB2 (Next closest component)	LB3 (MST)	LB4 (1-Tree)
13689701	112	28.1 s	30.0 s	32.2 s	33.2 s
13738401	82	20.6 s	21.8 s	24.0 s	24.9 s
13813501	127	31.9 s	32.8 s	35.5 s	36.7 s
13813601	146	36.6 s	38.3 s	41.5 s	43.1 s
13742301	131	32.9 s	34.9 s	38.0 s	39.1 s
136725R1	43	10.9 s	12.2 s	15.8 s	16.3 s
13424701	177	44.4 s	46.8 s	51.5 s	53.7 s
13501201	129	32.4 s	34.1 s	37.7 s	38.6 s
13588801	281	70.4 s	75.8 s	83.0 s	N/A
13424801	202	50.6 s	52.9 s	58.4 s	N/A

Table 4.1

From this table, we can see that the 1-tree relaxation lower bound performs the best. As described above, however, if a PCB has the property that the components are densely packed and the x-y table is able to get to a different component placement during the turret indexing delay then it is likely that the lower bound will not be increased by considering extra table time. The minimum amount of table time required from each component placement will likely occur during the turret indexing delay. Only if the components on the PCB are sparse or the x-y table has very little time during the turret indexing delay to position itself will the lower bound be likely to be increased by calculating extra x-y table time, as in LB2, LB3, and LB4.

It is important to note that although many of the PCBs being produced are densely packed, the new high-speed pick-and-place machines currently being used often have different head speeds at which the turret operates. For reasons described in Section 5.2.1, these machines must place all the components at the fastest head speed first and then all components at the next fastest speed and so on. For this reason, the component placement sequence for each speed group is solved individually creating the property of a sparse board. Consequently, the calculation of extra table time on dense boards when using these types of high-speed machines is still advantageous.

4.4 The lower bound gap

Having determined that the 1-tree relaxation lower bounding method performs the best, we can now determine how good our heuristic solutions performed in Chapter 3 by comparing them against the lower bounds we obtain by using the 1-tree method. In this comparison, we only compare the assembly times

we obtained for four different PCBs using our best two heuristics from Chapter 3 against the lower bounds we obtained for the assembly time for each PCB using the 1-tree relaxation method.

Board #	Pairwise Exchange Method (min. x-y & feeder Mov.)	Lower Bound (1-Tree)	% Gap
13424701	54.82 s	44.35 s	19 %
13501201	39.88 s	32.35 s	19 %
13588801	85.0 s	70.35 s	17 %
13424801	60.72 s	50.6 s	17 %

TABLE 4.1

Also reported in Table 4.1 is the % Gap, which represents how far the heuristic solutions are from the lower bound as a percentage, which can be calculated using the following formula:

$$\%\ Gap = \frac{Heuristic\ Solution - LowerBound}{Heuristic\ Solution} * 100$$

From the empirical results given above, we see that on average the lower bound gap for the four boards is 18%. This means that the pairwise exchange method that minimizes x-y table and feeder tray movement is on average within at least 18% of the optimal solution when we assume the same machine specifications we used for testing (see Section 2.4). Without this lower bounding concept, we would have no way of determining how good our heuristic methods are actually performing.

CHAPTER 5

Adapting the Pairwise Exchange Method to the Mitel environment

One of the high-speed pick-and-place machines currently being used by Mitel for high-volume PCB assembly is the Panasonic MVII(LL) NM2559B machine. To solve the feeder tray set-up and component placement sequence problem associated with the assembly of a PCB, Mitel uses a very sophisticated software tool called Panatools, which is commercially distributed by Panasonic for use on Panasonic high-speed pick-and-place machines.

Since the Panasonic MVII machine used at Mitel differs from our general model of a high-speed pick-and-place machine in several ways, the methods discussed in Chapter 3 cannot be applied directly to the Mitel environment. In this chapter, we discuss ways to adapt the best method from Chapter 3, the pairwise exchange method, to the MVII machine. We also report on how this adapted version of the pairwise exchange method faired against Panatools in assembly time.

5.1 Differences between the MVII machine and our general model

There are three main differences between the MVII machine and our general model of a high-speed pick-and-place machine described in Section 2.1.

They are:

i) Multiple head speeds.

ii) Multiple component reel sizes and types, and dead space.

iii) Feeder tray restrictions.

5.1.1 Multiple head speeds

The MVII machine has eight different head speeds at which the turret can operate. The fastest head speed is head speed 1 and the slowest is head speed 8. Each component to be placed by the machine has a particular speed at which it must be placed. This head speed that is associated with each component is essentially determined by the component's size and shape (volume).

In order to understand why this high-speed pick-and-place machine and many like it require different head speeds, we must realize that these machines are used to place very small to very large components, and that they often use air suction to pick-up and hold the component to be placed against the pick-and-place head. When a small component is picked up, the turret can index one station very quickly without having the component fly off. With larger components, however, this air suction used to hold the component against the pick-and-place head is not sufficient to allow the turret to index quickly. Consequently, the larger the component, the slower the turret must index. The placement operation, which also slows down with the head speed, is another reason that different head speeds are required. If a larger component is placed at a fast head speed, that component will have a fair amount of inertia. This inertia caused by the component weight could possibly damage the component or the board itself on placement.

5.1.2 Multiple component reel sizes, types and dead space

In our model of a high-speed pick-and-place machine, we never make any distinction between the component reels for the different component types. However, different components have different sizes and shapes. These different component sizes are stuck to tapes of different widths. These different sized tapes are then placed onto appropriately sized component reels. Since there are different sizes of tapes, a different sized component reel exists for each one. At Mitel, there are six different sizes of component reels.

Besides the different sizes of tapes, it is important to mention that there are two types of tapes to which the components can be attached. The first is an embossed tape, and the second is a paper tape. A different type of component reel exists for each. If we now give the size and type of tape to which a component is attached, we will be indicating how the component is being supplied, known as the *supply method.*

With the property that the component reels have different sizes, we can no longer assume that a component reel will fit into a single feeder tray slot, as we did in our general case. In fact, the idea that a component reel is placed directly in a feeder tray slot is not quite valid. In reality, the component reels are placed into component feeders, and then it is these component feeders that are placed onto the feeder tray. Since the MVII has six different sized component reels and two types of feeder tapes, it requires twelve different component feeders.

In addition to having different sizes of component feeders and component feeders for the different tape types, the MVII machine also has two different types

of component feeders that it uses, which are known as *single feeders* and *double component feeders*.

The single feeders have different sizes to hold the different sizes of component reels. However, they all have one common property. This property is that they are only able to hold one component reel. On the other hand, the double component feeders are able to hold two component reels, one mounted on each side of the double component feeder. The two component reels mounted on a double component feeder will be equally spaced and will each have their own feeder number.

It is obvious that double component feeders are the best possible feeders to use on the MVII machine. Besides doubling your feeder tray capacity, the feeder tray distance required to travel between two adjacent component reels on a double component feeder is approximately half the distance required by the feeder tray to travel to an adjacent component reel on single feeders. Thus, using double component feeders reduces the amount of feeder tray travel distance and possibly assembly time. However, double component feeders can only be used under certain circumstances and have certain restrictions. First of all, only the component reels in the smallest size class can be placed on a double component feeder. Secondly, both sides of a double component feeder must have the same supply method. Thirdly, a double component feeder that is being used on a feeder tray must have component reels on both sides before another double component feeder with the same supply method can be placed onto the feeder tray. Lastly, there is a limited number of double component feeders which typically, at Mitel, do not provide enough feeder tray capacity to accommodate all of the small-sized components for a PCB.

As a consequence of having different sizes and types of feeders, it is not always possible to put two feeders directly beside one another, as in the general model. It is sometimes necessary to leave *dead space* between adjacent feeders. Dead space is the number of slots which must be left empty to permit two particular feeders to be placed side-by-side.

By describing the different types and sizes of feeders, it is easy to visualize that the different sized feeders and types can take up different amounts of room on the feeder tray. As a result, different dead space requirements are necessary between the different sizes and types of feeders. For instance, a double component feeder that is going to be placed to the left of another double component feeder may require 1 slot between them. However, if the same sized single feeder is placed to the left of the same sized double component feeder, two slots may be required between them. The dead space requirements between all the different pairs of feeders for the Mitel machine can be found in [Pan92a].

5.1.3 Feeder tray restrictions

The complexity of the feeder tray for the MVII machine made it necessary to make a few adaptations to our implementation of the pairwise exchange method. The MVII machine actually has two feeder trays, a left and a right, each with exactly 75 slots. These feeder trays can be used in different modes. The mode that is currently being used at Mitel is the *connection mode*, in which the two feeder trays are connected to form one feeder tray with 150 slots. This method is used whenever a feeder tray capacity of greater than 75 slots is required.

One problem that arises in the connection mode is that a considerable amount of space is lost where the two feeder tray are connected. As a result, our implementation of the pairwise exchange method must take into consideration the amount of extra time required to jump from one feeder tray to another.

Another complication that arises from the feeder tray on the MVII machine is that there are certain locations on the feeder tray that have reinforcement ribs. These reinforcement ribs create a certain amount of dead space on the feeder trays that prevents some of the feeders from using certain slots close to these reinforcement ribs. The exact slots that cannot be used by these feeders due to the reinforcement ribs are given in [Pan92a]. As a result, we again had to adapt our implementation to ensure these new restrictions were not violated.

5.2 Adapting the Pairwise Exchange Method to the MVII machine

In order to apply the pairwise exchange method to PCBs being built on the MVII, we must first investigate strategies for adapting this method to handle each of the previously mentioned differences between the MVII and our general machine model.

5.2.1 Adapting to multiple operation speeds

The first adaptation we had to make to the pairwise exchange method was caused by the MVII machine having eight different speeds for the turret instead of just one. With these eight different speeds, the MVII machine must place all the components at the fastest head speed first and then all the components at the next fastest speed and so on in order to be as efficient as possible. To understand this,

consider what happens when one of the components on the turret must rotate at a slow head speed, while all the other components on the turret can rotate at a fast head speed. In this scenario, the turret is forced to rotate at the slowest speed required by any component on the turret.

By having to place the components at the fastest head speed first and then the components at the slower head speeds, we had to adapt our method to solve eight separate problems instead of one large problem. To do this, we group all the component locations for each speed group, and then we group all the component types required for each of these groups on the feeder tray in the order of fastest speed group to slowest speed group. We can then apply the improved pairwise exchange algorithm given in Chapter 3 to each of the eight different groups of component locations and their corresponding group of component types on the feeder tray. Following this, we will obtain a component insertion sequence for each speed group and an associated feeder tray set-up for that group. We then combine these solutions by using the solution to the fastest speed group followed by the next fastest speed group and so on.

There were, however, certain problems that arose with this adaptation of the method. The first problem was that the solution to the component insertion sequence and the feeder tray set-up for one group does not take into consideration where we ended on the x-y table and the feeder tray in the preceding group. By not taking this into consideration, the x-y table and feeder tray movement required from the last component placement and pick-up of one group to the first component placement and pick-up of the next group can both be extremely long, causing increased assembly times.

To address this problem, we include the last x-y and feeder tray position used in the solution for the previous speed group as the starting location for the following speed group. By incorporating this information, the first component placement and pick-up will be based on where the feeder tray left off in the previous speed group, which will reduce assembly times.

The second problem we had to address deals further with the feeder tray movement. When the feeder tray completes its last component pick-up for a particular speed group, it has to move to the next group of component types which immediately follows the previous group of component types. If, however, the feeder tray has its last component pickup near the start of the feeder tray set-up for that group of component types, then the feeder tray has to travel over all the feeder slots assigned for that group of component types before getting to the next group. To try to avoid this situation, which can cause increased assembly times, we can incorporate into the assembly time used for the pairwise method the cost for the feeder tray to travel to the last feeder tray slot for the particular group of component types on which it is working. By doing this, solutions will be generated that take into consideration the amount of feeder tray movement that will be required to get to the next group of component types.

In addition to the above problems, it is important to mention that certain pick-and-place machines are not able to position the x-y table and the feeder tray for the first component placement and pick-up during the overhead operations required before the assembly of a PCB. If this is the case, then it is important to generate a solution for the initial speed group that takes into consideration the location of where the x-y table and the feeder tray are starting off, known as the *home positions*. By doing this, you can possibly avoid a long x-y table and feeder

tray movement for the first component placement and pick-up. Sometimes the home position for the x-y table is a PCB mark, which is a mark that is checked in order to determine if the PCB is perfectly aligned before assembly.

The problem of a home position, however, is not an issue for the MVII machine, since the x-y table and the feeder tray are able to position themselves for the first component placement and pick-up during the overhead operations required before the assembly of a PCB. Consequently, our implementation of the pairwise exchange method did not incorporate home positions for the x-y table and feeder tray when generating a solution.

5.2.2 Adapting to multiple component reel sizes and types, and dead space

Due to differences in component reel sizes, types and the different dead space requirements between adjacent feeders, the pairwise exchange algorithm can no longer simply swap any chosen pair of component reels to try to obtain a better feeder tray set-up. For instance, a 12mm component reel cannot be swapped from its 12mm feeder onto a 16mm feeder. Nor can a 16mm feeder be swapped with the 12mm feeder due to the differences in space required between the adjacent feeders. Consequently, we had to modify our concept of swapping.

In order to obtain the best possible feeder tray set-up, we have to try to swap as many component types as possible on the feeder tray without violating any of the feeder tray restrictions described so far. To do this, we must first consider which component reels and feeders can be swapped without violating these restrictions. Certainly, we know that any single feeder can be exchanged with another single feeder if both feeders are of the same size. In addition to this, we

can conclude from the table given in [Pan92a] which indicates the dead space requirement between different feeders, that certain different sizes of single feeders can be exchanged with each other without violating the dead space requirements, since they have the same dead space requirements when placed beside any of the other sized feeders, i.e. they require the same number of slots between the different sized feeders.

When we consider what can be swapped in and out of the double component feeders, we must remember that both sides of the double component feeder must have the same supply method. Keeping this in mind, we can swap any sized component reel, whether it is on a single feeder or a double component feeder, with the same sized component reel on a double component feeder if it has the same tape type. To further increase the swaps that can occur, we also consider swapping entire double component feeders that have the same size tape reels but differing tape types.

Besides having to adapt our method to ensure that the correct amount of dead space exists between any two adjacent feeders, we also want to generate an initial feeder tray set-up which takes the least amount of feeder tray capacity so as to reduce the amount of feeder tray movement. To do this, we realized from looking at the dead space requirements given in [Pan92a] that by placing the feeders within each speed group in increasing size, the amount of feeder tray capacity required is reduced. The reason for this is that the design of the feeders allow feeders of the same size to be more closely packed then two adjacent feeders that are of different sizes. Thus, the mixing of feeder sizes in a feeder tray causes extra feeder slots to be lost. As a result, we adapted our method to place all the required feeders for a particular speed group in increasing order of size in the

feeder tray. In addition, we should mention that we placed all the double component feeders before all the single component feeders of the same size.

5.2.3 Adapting to feeder tray restrictions

Two differences in the MVII machine's feeder tray and the feeder tray for our model are the middle connection of the feeder trays and the reinforcement ribs that exist on the MVII machine's feeder tray.

To address the problem of the middle connection, we had to first determine how long it takes the feeder tray to jump between the two feeder trays. Since this value was not indicated in any of the manuals for the MVII machine, we had to estimate this time using the method we describe in Section 5.4. After determining this value, our implementation simply adds this additional amount of time to any feeder jumps required between the two feeder trays.

In our implementation, we found two instances where we had to consider reinforcement rib violations. The first was when we built the initial feeder tray set-up. The second place, which was not so obvious at the time, was when certain sized component feeders are swapped with certain other sized component feeders. In some instances, two feeders cannot be swapped due to reinforcement rib restrictions.

5.2.4 Overview of the adapted Pairwise Exchange Method for the MVII machine

1) For each speed group i=1,2,..,8, group together all the component placements for that speed group. Let each component placement group be C_i.

2) Find an initial feeder tray set-up F by grouping all the component types required for head speed one followed by all the component types required for head speed two and so on. For each group, place the required feeders in increasing size. Let F_i be the section of the feeder tray holding the component types for head speed i, i=1,..,8.

3) For each of the speed groups i=1,..,8 do the following:

   a) Using the feeder tray section F_i and component group C_i, obtain a component placement sequence S_i by using the method described in Section 3.4.1 and our adaptations in Section 5.2.1. Let t_i be the time required to complete the assembly of all the component placements at head speed i by using F_i, S_i and equation 2.1.

   b) Using S_i, perform pairwise exchanges of reels in F_i (as outlined in Section 3.3.3 and by incorporating our adaptations in Sections 5.2.2 and 5.2.3), wherever such an exchange improves the assembly time required to place the components in head speed i. Let the resulting feeder tray set-up be placed back into F_i.

   c) Using F_i and C_i, find a placement sequence S_i' which minimizes production time for that head speed. Again use the methods described in Section 3.4.1 and our adaptations in Section 5.2.1. Let t_i' be this production time. If $t_i'<t_i$, then let $t_i=t_i'$, $S_i=S_i'$, and go to Step 3b). Otherwise stop.

4) Obtain the solutions to the feeder tray set-up and component placement sequence by combining all the feeder groups F_i in the order i=1,..,8 and the component placement sequences S_i in the same order.

5.3 Lower bound adaptations

The lower bound calculation LB1 given in Section 4.1 for our machine model, which is based on the minimum amount of time required for the machine to complete a pick-and-place cycle, is not valid for the MVII machine, since it has multiple head speeds. There is a minimum pick-and-place cycle time for the MVII machine for each of the eight head speeds. These minimum pick-and-place cycle times are given in [Pan92b] and are referred to as the *tact time* for each head speed.

To be able to calculate a lower bound for the assembly of a PCB, we must now determine the number of pick-and-place cycles required at each head speed. For every head speed except the last one, the number of pick-and-place cycles required is equal to the number of components that have to be placed at that head speed. For the last head speed, however, we must realize that if we make the number of pick-and-place cycles for this head speed equal to the number of components to be placed at that head speed the turret will still be fully loaded with components. To unload the remaining components from the turret, an additional k pick-and-place cycles are required at that head speed.

Besides the above information, only one other point must be considered before we can give our lower bound calculation for the MVII machine. As in our general case, the very first pick-and-place cycle does not require the minimum

pick-and-place cycle. The reason is that the turret should be empty, and as a result, the turret will not need to rotate before picking up the first component. Consequently, the initial pick-and-place cycle is reduced to a simple pick-up operation.

Given the above information, the lower bound calculation is obtained by multiplying the number of components in each head speed by the minimum pick-and-place cycle time for that head speed. The only exceptions are that the very first pick-and-place cycle is reduced to a simple pick-up operation, and an addition k pick-and-place cycles are required to unload the turret. Knowing this, a good lower bound LB5 for our MVII machine can be calculated as follows:

$$\text{LB5} = P1 + ((N1 - 1) * T1) + \sum_{i=2}^{8} (Ni * Ti) + (k * T8)$$

Where :

$P1$ = Pick-up time at the first head speed (estimated as half the tact time for that head speed)

Ni = The number of components to be placed at head speed i, i=1,2,3,4,5,6,7,8.

Ti = The tact time required for head speed i, i=1,2,3,4,5,6,7,8.

k = The number of pick-and-place cycles required to unload the turret.

Finally, the lower bound LB5 for the MVII machine can be improved in the same way the lower bound LB1 was improved for our general machine. This improvement is to include in the calculation of the lower bound the minimum amount of x-y table delay that could not be completed during the turret delays for the assembly of a PCB. (See Section 4.2.) However, several complications arise

when calculating the extra table time (ETt) required for a particular PCB being assembled on the MVII machine.

The first complication is that the MVII machine places all the components one speed group at a time. As a result, the minimum x-y table distance required from each component must be calculated by only considering the other component locations in the same speed group and not from all possible component placement locations as described in Section 4.2. By doing this, we can obtain the extra table time required for each speed group remembering that because we require a Hamiltonian path in each speed group and not a tour, we have to throw out the maximum delay we added to the extra table time for that speed group.

The second problem is that up to six of the components to be placed in each speed group will be placed at a slower head speed than the speed group they are in. The reason being is that the turret on the MVII machine can pick-up a component in the next speed group while it still has six components from the previous speed group to be placed. As a result, the remaining six placements in a speed group could be placed at a number of different head speeds. To deal with this problem, you could subtract from your extra table time the six largest delays that were added from each speed group, or assume that the six largest minimum table movements in each speed group are placed at the slower speed which can be determined. Note that for the last speed group all the components will be placed at that head speed.

Although we did not implement this improvement, our new lower bound calculation LB6 for our MVII machine is as follows:

LB6 = LB5 + ETt

Where :

ETt = Extra table time, calculated as described above.

5.4 Estimating machine specification times

In order to implement the pairwise exchange method for the MVII machine, we had to first overcome the hurdle of obtaining certain operation rates for the MVII machine. Some of these operation rates were available in the MVII manual [Pan92b], however many of them were not. We also requested these operation rates from Panasonic, who manufacture the MVII. They willingly gave us rates for other older machines, but would not give us the rates for the MVII machine. As a result, we had to determine some method for estimating the operation rates which we required.

Our first proposal for obtaining these operation rates was to go to Mitel and actually time the operations we needed. This would require setting up PCB scenarios and running them on the MVII machine at Mitel to obtain the required operation rates. With the eight different speeds, however, there were almost twenty different operation rates we needed, each of which may require the running of several PCB scenarios on the MVII machine. This solution was unacceptable due to the time we would have needed on the MVII machine, which is in very high demand at Mitel. In addition to these problems, the high-speed of the machine would have made it virtually impossible to obtain accurate operation rates.

Instead of timing operation rates on the MVII machine, we were able to derive operation times by using the software package called Panatools mentioned

earlier. This software package is able to generate solutions for the assembly of a PCB. By giving Panatools contrived PCBs and particular feeder tray set-ups for these PCBs, we were able to compare the simulated assembly times provided by Panatools for the different PCBs and extract estimations of the operation rates we required. Since Panatools was developed by Panasonic, we believe that the operation rates obtained in this way should be fairly close to the actual operation rates of the machine. We should mention, however, that a considerable amount of time and effort was required to generate and run the different PCB scenarios that would isolate the different operation rates.

5.5 Machine specifications

Before we can apply the methods we described earlier for minimizing the assembly time on a particular machine, certain machine specifications are required for that machine. These machine specifications include the number of pick-and-place heads, the feeder tray capacity and the operation times for the pick-up of a component, the placement of a component, the x-y table speed, the feeder tray speed and the turret rotation speed.

On the MVII machine, there are twelve pick-and-place heads mounted on the circumference of the turret. Since our general form of the machine assumed a 2k-headed turret, this makes k=6 for the MVII machine. In addition, the feeder tray capacity of the MVII machine can reach 300 component types when the feeder tray is in connection mode and only double component feeders are used. Since C represents the feeder tray capacity of our general form of the machine, and we will be using the connection mode and double component feeders, the MVII machine has a value of C=300. However, if the feeder tray was in connection mode but no

double component feeders could be used, then the maximum feeder tray capacity would only be 150 which is equal to the number of feeder slots.

For the MVII machine, the operation times for the pick-up of a component, the placement of a component and the turret rotation rate are all combined into one operation rate called the *tact* time. This tact time represents the time required for the turret to index one station and then perform the pick-up and placement operation which occur concurrently. Since the MVII has eight different head speeds, there are eight different tact times that exist for this machine. These tact times can be found in the Panasonic MVII Programming manual [Pan92b].

The only other machine specifications not yet discussed are the operation rates for the x-y table and the feeder tray. As in our general form, the x-y table and the feeder tray each travel at a constant speed. The feeder tray travels at one constant speed laterally while the x-y table can travel at one constant speed in both the x and y direction simultaneously. These two operation rates, however, were not given out in any of the manuals we could locate on the MVII machine. Consequently, we had to actually estimate these times using the method described in Section 5.4.

In order to calculate the assembly time required by a high-speed machine for a particular PCB, all the above operation rates are required. There is, however, one problem that is generated when the pick-up, placement and turret rotation rate is given as one tact time instead of individual rates. This problem is that we do not know how long it takes the turret to index one station without the pick-up and placement time included. This turret indexing time is the amount of free time that the x-y table and the feeder tray have to position themselves without costing any

assembly time since the turret, x-y table and feeder tray are all concurrent. Without this information, we cannot determine the amount of extra time, if any, that will be required by the x-y table and feeder tray to position themselves. As a result, we had to estimate how many millimeters the x-y table could move and how many slots the feeder tray could jump at each speed during the turret indexing delay to allow us to determine the extra x-y or feeder tray time required for a particular component placement. To do this, we again used the approach described in Section 5.4.

5.6 Empirical results

After adapting the pairwise exchange method with the necessary modifications indicated above, we were able to obtain solutions for the component insertion sequence and the feeder tray set-up for PCBs being assembled in the Mitel environment on the MVII machine. To be able to compare how well our adapted pairwise exchange method performs, we compared our simulated assembly times for our solutions for ten different PCBs with Panatools simulated assembly times for their solutions for the same ten boards. In addition to these statistics, we included the lower bound for the assembly time of these ten boards obtained by our new lower bound calculation LB5 and the lower bound calculation obtained by Panatools. (See Table 5.1.)

Simulated Assembly Times

Board #	Our Lower Bound	Our Assembly Time	Our % Gap	Panatools Assembly Time	Panatools Lower Bound	Panatools % Gap
13424701	38 37 s	40.17s	4 %	42 s	38 s	10 %
13501201	29.41 s	30.54s	4 %	32 s	30 s	6 %
13588801	59.28 s	61.14s	3 %	63 s	59 s	6 %
13424801	43.0 s	45.24s	5 %	46 s	43 s	7 %
13813501	29.0 s	29.8 s	3 %	31 s	29 s	6 %
13813601	32.8 s	33.7 s	3 %	35 s	33 s	6 %
13742301	29.77 s	30.83 s	3 %	32 s	30 s	6 %
13527501	35.08 s	36.26 s	3 %	38 s	35 s	8 %
13587901	42.45 s	43.46 s	2 %	44 s	43 s	2 %
13729001	38.6 s	41.1 s	6 %	44 s	40 s	9 %

Table 5.1

From the statistics above, we were very enthusiastic about the results we obtained. First of all, the simulated assembly times for our solutions were slightly better than the simulated assembly times the software package Panatools predicted for its solutions. Secondly, our simulated assembly times were also fairly close to the lower bounds obtained by us and Panatools. In fact, the percentage gap on average between our simulated assembly times and our lower bounds is about 4%, whereas the percentage gap on average for the Panatools solutions is about 7%. This indicates that our solutions are very close to optimal.

It is also important to mention the amount of CPU time that was required to obtain solutions for the above boards by our improved pairwise exchange algorithm and the Panatools software. When using a 486 66 MHz machine, the average amount of CPU time required by Panatools to obtain solutions for each of the ten boards given in Table 5.1 was one minute and forty five seconds. Using our improved pairwise exchange method which is implemented in Turbo Pascal Version 7.0, the average amount of CPU time required for the same ten boards was six minutes and forty five seconds. However, in our implementation, we were not concerned with reducing CPU time.

As well as comparing simulated times, we also wanted to compare the actual build times obtained by using the Panatools and the pairwise exchange solutions directly on the MVII machine at Mitel. Unfortunately, it was very difficult for us to obtain time on the machine for testing because the MVII machine is heavily used at Mitel and expensive to run. As a result, we were only able to test our solutions on four boards. Moreover, we did not have time to actually load the components on the feeder tray (which would take approximately 2.5 hours for each board), but instead ran the MVII in a simulated build mode. In this mode, the machine goes through all the motions of building the PCB, however no components are actually present. As the MVII machine does not report the build time of a PCB while it is in simulated build mode, it was necessary for us to use a stopwatch for timing. It was very difficult to accurately judge when the build time of a board started and stopped while on the machine, so we decided to time the building of three consecutive boards for three of the PCBs and five consecutive boards for the fourth PCB. We then divided each assembly time by the number of PCBs assembled during that time to obtain the associated time for building one board. These results are reported in Tables 5.2 and 5.3.

Ideally, we wanted to obtain the assembly time required to build five consecutive boards for each of the four PCBs. However, the first time we went to Mitel to obtain these times on the MVII machine, we were only able to obtain the times for one PCB (13588801.PCB) before Mitel required the machine. As a result, during our second visit to Mitel to obtain the assembly times for the remaining three PCBs, we only timed the building of three consecutive boards of each to ensure we had enough time to get our results.

You will note that the actual build times given in Tables 5.2 and 5.3 are higher than the predicted assembly times given in Table 5.1 for both pairwise exchange and Panatools. The reason is that, because we timed the building of a certain number of consecutive boards, the actual build times we obtained include the *PC board exchange time*, which is the time required for the PCB being assembled to be removed and a new board to be loaded.

Actual machine build times

Board #	Our Solutions		Panatools Solutions	
	Time for three boards	Calculated time for one board	Time for three boards	Calculated time for one board
13424701	2:22 s	47.3 s	2:20 s	46.7 s
13501201	1:51 s	37 s	1:48 s	36 s
13424801	2:40 s	53.3 s	2:35 s	51.7 s

Table 5.2

Board #	Our Solutions		Panatools Solutions	
	Time for five boards	Calculated time for one board	Time for five boards	Calculated time for one board
13588801	5:59 s	71.8 s	5:52 s	70.4 s

Table 5.3

From the reported results in Tables 5.2 and 5.3, we can see that the actual assemble times for our solutions on the MVII machine are approximately one second slower per board than the solutions Panatools generates. From our simulated assembly times, however, our solutions should have had better assemble times than the Panatools solutions. This then indicates that the operation rates we had estimated for the MVII machine may not have been accurate enough, or our simulation of the MVII machine is not entirely accurate. In either case, our heuristic algorithm is obtaining its solutions based on this information and could possibly perform better with exact operation rates and more information about the machine.

Although our actual machine build times did not beat those of Panatools, we were extremely pleased with how well we did compared to the solutions generated by this very expensive software package. This is especially true when we consider that our algorithm could perform even better with more detailed and accurate information on the MVII machine.

5.7 Using the Pairwise Exchange Algorithm for a set of PCBs requiring one common feeder tray set-up

Currently at Mitel, the MVII machine is being used to assemble a high volume of three to five different PCBs. Each week Mitel decides on a weekly and daily quota for each type of PCB based on demand. For the MVII machine to assemble these boards as quickly as possible, we would have to produce an optimal feeder tray set-up for each of the different boards. With this approach, however, when the quota for one type of PCB is reached for the day, the entire feeder tray set-up on the MVII machine would need to be taken off and the feeder tray set-up for the next type of PCB would need to be placed on. This tear down and set-up of the feeder tray is of importance when we consider the fact that the time required to change a feeder is approximately three minutes. If the feeder tray has to be set-up for a PCB requiring fifty different component types, then two and a half hours is required to set-up the feeder tray. Assuming three different types of PCBs with fifty component types each must be produced every day, Mitel would require up to seven and a half hours of feeder tray set-up time per day.

In order to avoid so much down time on the MVII machine due to feeder tray set-ups, Mitel uses Panatools to generate one feeder tray set-up for the different PCBs that the MVII machine has to produce during the week. Obviously, the best feeder tray set-up for the different boards would be to place the optimal feeder tray set-up for one board on the feeder tray followed by the optimal feeder tray set-up for the next type of PCB to be assembled and so on. Unfortunately, the feeder tray does not have enough space for this, and thus, it is necessary to share a common feeder tray set-up for the different PCB assemblies. Consequently, we

will be avoiding the down time due to feeder tray set-ups at the expense of taking slightly longer to assemble each board.

With this in mind, we wanted to consider how we could adapt our modified pairwise exchange method to solve the problem of finding a common feeder tray set-up for a set of PCBs, rather than just one. In [LN89], they discuss briefly how the pairwise exchange method could be adapted to this type of situation. Using this general idea, we give a detailed method on how to adapt our modified pairwise exchange algorithm to address this new problem. Letting $J_1, J_2, \ldots, J_n$ represent the set of PCBs to be produced, an overview of the modified method is as follows.

1) Find an initial feeder tray set-up F using the same method described in Section 5.2.4 Step 2) for the adapted pairwise exchange method, however apply the method to the component set obtained by taking the union of the components in $J_1, J_2, \ldots, J_n$.

2) Using F, find a component placement sequence for each PCB using the methods described in Section 5.2.4 Step 3a) and Step 4 for the adapted pairwise exchange method. Let S_i be the component sequence for PCB J_i, $i=1,2, \ldots, n$. Also, let t be the time required to complete the production schedule using F and $S_1, S_2, \ldots, S_n$, which is calculated using the following:

Assembly time required to complete a production schedule $= \sum_{i=1}^{n} W_i A_i$, (Equation 5.1)

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

where,

W_i is the number of PCB J_i that must be produced,

and A_i is the assembly time required to assemble PCB J_i.

3) Using $S_1, S_2, \ldots, S_n$, perform pairwise exchanges of reels in F (as outlined in Section 5.2.4 Step 3b)), wherever such an exchange improves the assembly time required to complete the production schedule, calculated using equation 5.1. Let the final resulting feeder tray set-up be placed back into F.

4) Using F, find a new placement sequence for each of the different PCBs that minimizes the assembly time for that PCB. Let $S_1', \ldots, S_n'$ be the new placement sequences and t′ be the new assembly time required to complete the production schedule. If t′<t, then let t=t′, $S_1=S_1', \ldots, S_n=S_n'$ and go to Step3. Otherwise, stop.

Although we did not have an opportunity to implement and test the above method, we believe that it would perform well, based on the performance of the adapted pairwise exchange method described in Section 5.2

CHAPTER 6

Conclusion and future research

6.1 Conclusion

In this thesis, we have addressed the two main problems associated with high-speed pick-and-place machines. These two problems are how to determine a good component placement sequence and feeder tray set-up for a particular PCB. Without good solutions to these two problems, the efficiency of these high-speed pick-and-place machines is greatly reduced.

We considered different combinatorial optimization methods that currently exist for obtaining good solutions to these two problems. From these different methods, we chose two of the more sophisticated and most promising methods to implement and compare on a general form of a high-speed pick-and-place machine. These methods were the place-by-type method [AS92] and the pairwise exchange method [LN89]. We adapted each of these methods to our general machine model, and implemented them. We also suggested several improvements for each of the methods.

From our empirical results, we found that the pairwise exchange method generally produces better results than does the place-by-type method, in fact on average it outperformed the place-by-type method by 2%. Through lower bounding procedures, we were also able to establish that the pairwise exchange method solutions were, on average, within 18% of the optimal on our test problems. We then adapted this pairwise exchange method with our improvements

to a very specific machine. This machine was the MVII(LL) NM-2559B high-speed pick-and-place machine which is currently being used at the Mitel Corporation.

By adapting the pairwise exchange method to the Mitel environment, we were able to compare the solutions we obtained for four different PCBs for the MVII machine against the solutions Mitel would have generated for the same PCBs by using a very sophisticated and costly program called Panatools. By running our solutions and the solutions generated by Panatools for the four different PCBs on the MVII machine, we found that the assembly times for our solutions were very close to those produced by Panatools (within, on average, 2%).

We were extremely pleased with how well our adapted pairwise exchange method performed at Mitel compared to Panatools. This is especially true when we consider the fact that our results could possibly be improved with more detailed information on the MVII machine.

6.2 Future research

In general, we feel that more testing of the different methods for the PCB assembly problem is needed. We have studied and compared heuristic methods that are based on combinatorial optimization techniques. It would be interesting to see how methods based on different techniques, such as non-linear integer programming, would compare to the methods we have studied. We feel, however, that such methods would be very difficult to adapt to specific environments, such as Mitel.

In Section 5.7, we describe an algorithm that adapts our modified pairwise exchange method to solve the problem of finding a common feeder tray set-up for a set of PCBs. We are also interested in implementing and testing our algorithm against the solutions currently being generated by Panatools in the Mitel environment. Although Panatools is able to solve the problem of finding a common feeder tray set-up for a set of PCBs, the people at Mitel believe that Panatools places the most emphasis or weight on the first PCB of the set, and ignores the weighting of the rest of the set. Our algorithm, however, places a weight on each PCB based on the number of boards that have to be produced. With this more accurate weighting of the set of PCBs and the success of our adapted pairwise exchange method, we believe that our algorithm for finding a common feeder tray set-up for a set of PCBs will do well, and perhaps outperform Panatools.

Finally, we would like to investigate adapting some of our improvements for lower bounds for our general model to the Mitel environment. Although the eight different head speeds on the MVII machine complicate this task, we believe that these methods could be incorporated. From our research, the first speed group incorporates between 80-90% of all the component placements for a PCB. Keeping this in mind, we propose applying our improvements for lower bounds only on this speed group on the MVII machine.

BIBLIOGRAPHY

[AGJ88] Ahmadi, J., Grotzinger, S., Johnson, D. (1988), Component Allocation and Partitioning For a Dual Delivery Placement Machine, *Operations esearch*, Vol. 36, No. 2, 176-191.

[AS92] Ahmadi, J., & Sciomachen, A. (1992), Modeling and Optimization Environment for Component Placement Machine, *International Journal of Systems Automation: Research and Applications (SARA)*, Vol. 2, 115-125.

[BCF94] Bard, J. F., Clayton, R. W., & Feo, T. A. (1994), Machine Setup and Component Placement in Printed Circuit Board Assembly, *International Journal of Flexible Manufacturing Systems*. (To appear)

[BM88] Ball, M. O., & Magazine, M. J. (1988), Sequencing of Insertions in Printed Circuit Board Assembly, *Operations Research*, Vol. 36, No.2, 192-201.

[Bow85] Bowlby, R. (1985), The Dip May Take Its Final Bows, *IEEE Spectrum*, 37-42.

[Bur84] Burkard, R. E. (1984), Quadratic Assignment Problems, *European Journal of Operational Research*, Vol. 15, 283-289.

[C90] Crama, Y., Kolen, A.W.J., Oerlemans, A.G., Spieksma, F.C.R. (1990), ThroughPut Rate Optimization in the Automated Assembly of Printed Circuit Boards, *Annals Of Operations Research*, Vol. 26, 455-480.

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

[CM89] Chan, D., Mercier, D. (1989), IC Insertion: An Application of the Travelling Salesman Problem, *International Journal of Production Research*, Vol. 27, No. 10, 1837-1841.

[DLM92] Downs, K. A., Linn, R. J., & Martinson, N. L. (1992), Sequencing Methods for Single In-Line Packages in PCB Assembly, *Manufacturing Review*, Vol. 5, No. 3, 175-183.

[Gro92] Grotzinger, S. (1992), Feeder Assignment Models for Concurrent Placement Machines, *IIE Transactions*, Vol. 24, No. 4, 31-46.

[GS88] Grotzinger, S., Sciomachen, A. (1988), A Petri Net Characterization of a High Speed Placement Machine, *Proceedings 38th Electronic Components Conference, IEEE*, 64-68.

[HK70] Held, M., Karp, R. M. (1970), The traveling-salesman problem and minimum spanning trees, *Operations Research*, Vol. 18, 1138-1162.

[Jon92] Jones, H. (1992), Basic Overview of PCB Manufacturing, *Proceedings of the Technical Program, Nepcon West*, 434-440.

[L86] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B. (1986), *The Traveling Salesman Problem*, John Wiley & Sons, New York.

[Lig81] Liggett, R. S., (1981), The Quadratic Assignment Problem: An Experimental Evaluation of Solution Strategies, *Management Science*, Vol. 27, No. 4, 442-448.

[LK73] Lin, S., Kernighan, B. W., (1973), An Effective Heuristic Algorithm for the Traveling Salesman Problem, *Operations Research*, Vol. 21, 498-516.

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

[M92] 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 Ciruit Card Assembly, *IIE Transactions*, Vol.24, No.2, 18-29.

[Mar86] Marcoux, P. P. (1986), Putting on The Chips, *Printed Circuit Fabrication*, Vol. 9, No.3, 81-84.

[Pan92a] Panasonic (1992), *Specifications : Panasert MV-II (LL) NM-2559 BA,BB*, Matsushita Electic Industrial Co., Ltd. Manufacturing Equipment Division.

[Pan92b] Panasonic (1992), *MV-II Programming Manual*, Matsushita Electic Industrial Co., Ltd. Manufacturing Equipment Division.

[Pan92c] Panasonic (1992), *Panasert Software Panatools II Getting Started*, Panasonic Factory Automation Company.

[Sah74] Sahni, S. (1974), Computationally related problems, *SIAM J. Comput.*, Vol. 3, 262-279.

[SP91] Sanchez, J. M., Priest, J. W. (1991), Optimal Component-Insertions Sequence-Planning Methodology for the Semi-Automatic Assembly of Printed Circuit Boards, *Journal of Intelligent Manufacturing*, Vol. 2, 177-188.

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