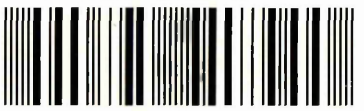


INTELLIGENT PLANNING AND CONTROL

OF MULTI-ASSEMBLY SYSTEMS

Eiad Khalil

A thesis submitted in partial fulfilment of the requirements of
Sheffield Hallam University
for the degree of Doctor of Philosophy

May 2008

Abstract

The global trend towards cost minimisation in manufacturing has intensified during the last two decades. Cost reduction can be achieved either directly, through elimination of waste, or indirectly, through optimisation of production processes and generating more reliable information regarding the costs incurred. The research presented in this thesis considers cost reduction in three aspects: optimisation of production processes, accurate cost estimation and accounting. Due to the increasing number of combinatorial optimisation problems associated with the production of Printed Circuit Boards (PCB), it has attracted the attention of many researchers who tried to solve these problems with the aim of minimising the production cost. Therefore, PCB production is used in this research as a test-bed for the three aspects mentioned above.

Regarding cost reduction in PCB manufacturing, three interrelated combinatorial optimisation problems are considered: the component placement sequencing problem, the feeder assignment problem and the board type sequencing problem. Solving these problems ensures cost reduction by reducing the time required for manufacturing PCBs. As for cost reduction in the costing and accounting aspects, the traditional standard costing and standard accounting have some problems that make them unsuitable for today's manufacturing. Standard costing allocates overhead to labour or machine hours, which leads to a distortion of product costs due to the fact that today's manufacturing relies more on technology and less on human power. As for standard accounting, it has some features and characteristics that contradict with the widely spread lean manufacturing. The deficiencies in standard costing and standard accounting may create more waste and lead to the wrong decisions being taken.

A framework is developed to provide solution to the above-mentioned problems in an integrated environment. A mathematical formulation for the three PCB manufacturing-related problems is developed and solved using a metaheuristic-based algorithm. In order to deal with the costing and accounting part of the framework developed, Activity Based-Costing (ABC) and Lean Accounting (LA) are implemented on a PCB manufacturing facility using a case study. ABC is used to estimate the costs of manufacturing PCBs and provide detailed information on how the costs are incurred. As for LA, it is used to reduce the costs associated with the accounting system, which is achieved by eliminating and/or replacing accounting transactions and promoting lean measures.

Simulation results obtained show an average reduction in total assembly time of 5.96% and 5.43% when Taboo Search (TS) and Genetic Algorithms (GA) meta-heuristics are used respectively. The results also show how ABC can be used to identify the activities used in PCB manufacturing and calculate their costs. By targeting the most costly activities identified by ABC, the production costs can be reduced. Regarding LA, the results indicate how the accounting system costs can be reduced by eliminating some accounting transactions and processes or replacing them with less costly alternatives.

Acknowledgments

Great thanks go to:

- My supervisory team:
  - Prof. Sameh Saad without whom this work would not have come to light. He has been very helpful and supportive throughout the years and would not hesitate to help whenever possible.
  - Dr Ivan Basarab-Horwath for his continuous support and offer to help.

- My sponsor:
  - Al-Baath University for their financial support

- The staff at the Faculty of Chemical and Petroleum Engineering, Al-Baath University for their material and moral support, especially:
  - Dr Ahmad Al-Mahmoud
  - Dr Ahmad Kasser Ibrahim
  - Dr Hassan Al-Khalaf

Table of contents

AAT:	Acyclic Assembly Time
ABC:	Activity-based Costing
ABCM:	Activity-based Cost Management
ACD:	Activity Cost Driver
ACDR:	Activity Cost Driver Rate
CDPP:	Chebychev Dynamic Pick-and-Place
CCD:	Cost Centres Driver
CCR:	Cost Centre Rate
DPP:	Dynamic Pick-and-Place
EDPP:	Extended Dynamic Pick-and-Place
F:	Feeder
GA:	Genetic Algorithms
HR:	Human Resources
IR:	Infra Red
IT:	Information Technology
IT:	Insertion Time
JIT:	Just In Time
LA:	Lean Accounting
MPS:	Master Production Scheduling
MRP:	Material Requirements Planning
OEE:	Overall Equipment Effectiveness
PCB:	Printed Circuit Board
PPT:	Pick and Place Time
PT:	Pick Time
QAP:	Quadratic Assignment Problem
RD:	Resource Driver
RR:	Resource Rate
SA:	Simulated Annealing
SMC:	Surface Mount Component
SMT:	Surface Mount Technology
ST:	Set-up Time

TS:	Taboo Search
TSP:	Travelling Salesman Problem
TPT:	Total Processing Time
TT:	Travel Time
UV:	Ultra Violet
VNS:	Variable Neighbourhood Search

List of figures

CHAPTER ONE

1. INTRODUCTION

The diversity of product types required by today's customers has forced manufacturing companies to introduce multi-assembly systems. This has led to some production optimisation problems one of which is the sequence at which the product types should pass through the assembly lines. In addition to these production problems, there are also cost estimation and accounting issues that are associated with manufacturing and need to be addressed. Due to the intense competition in modern-day manufacturing, most manufacturers aim to reduce the production costs of their products. Taking this into account and considering the widespread use of multi-assembly systems, this research focuses on cost reduction within three aspects of manufacturing: optimisation of production processes, cost estimation and accounting.

The electronics industry has grown rapidly in the last two decades. The increase in PC production is, amongst others, one reason for this growth. Due to this growth, a global competition has emerged creating lower profit margins. As an important segment of the electronics industry, the production of Printed Circuit Boards (PCBs) has been paid considerable attention because, firstly, PCBs are found in almost all electronic devices and, secondly, because PCB production is associated with many problems that have the potential for optimisation. For these two reasons and for the fact that some of the PCB production problems are more general, PCB manufacturing is the area of industry that will be considered in this research. Given the highly automated production processes used in response to the high demand for PCBs, the problems associated with PCB production have become more complex and interrelated. Most of these problems are combinatorial optimisation problems (problems that involve identifying the best possible solution amongst a finite set of possible solutions), which, in some cases, require metaheuristics (e.g. Taboo Search, Genetic Algorithms, Simulated Annealing, etc.) to solve them. A metaheuristic is a strategy or a framework that guides heuristics to search for solutions for hard problems. Heuristics can be defined as a methodology, tool or a problem-solving technique that uses specific solutions, of several found, in the successive steps to obtain more feasible solutions. They have been created and

developed throughout years of experience in solving mathematical problems. For example, the following can be considered as heuristics (Sickafus 2004):

- Simplification: divide complex problems into small ones, take small steps, combine functions, etc.
- Extremes: vary attributes to their extremes, multiply and divide objects to extremes.
- Focus: search root causes for solution concepts, search technological contradictions, etc.

Although metaheuristics are very successfully widespread, the way they work is still not widely understood and there is very little in depth research about the theory behind them. The reason for this might be due to the fact that it is not that important to know how they work as much as it is important to know how to use them and whether using them gives satisfactory results or not. However, Watson (2003) argues that the limitation of the theoretical understanding of metaheuristics obstructs researchers from developing more effective ones. That is why he has developed theoretical behavioural models to some of well-known metaheuristics (or local search algorithms as he calls them).

As for the other two aspects of cost reduction, two well-known techniques will be considered: Activity-Based Costing (ABC) and Lean Accounting (LA). Activity-Based Costing has emerged as a method for estimating the costs of products (or services) in order to overcome the limitation of the conventional cost allocation method (allocating overhead to labour or machine hours). Examples for these limitations are the distortion of product costs resulted from the volume-based allocation of overheads to products (Cooper 1987 (from: Innes & Mitchell 1995)), the lack of cost information for decision making (Johnson & Kaplan 1987) and the lack of the availability of costing data at the design stage of the product life cycle (Berliner & Brimson 1988).

Although ABC is not a method designed to directly minimise the cost, the need for it was driven by the development of new technologies in manufacturing systems and by the introduction of automation in the early 1980s in order to provide accurate cost estimation that may lead to better decision making and eventually to cost reduction. In the conventional cost allocation method, the overhead is allocated to products, or services, depending on direct labour or on volume-related factors such as machine hours (Bellis-Jones & Develin 1999). The introduction of automation and new technologies has meant that a higher percentage of overheads cannot be volume related. ABC method follows a different approach in cost allocation, an approach that overcomes the

limitations of the conventional approach. The basic concept of ABC is that products and services consume activities, which in turn consume resources that have specific costs. For example, in a PCB manufacturing facility, producing a PCB requires performing many activities (e.g. sequencing parts, screen printing, placing components, etc.); the 'placing components' activity, in turn, consumes many resources (e.g. manufacturing engineer, operator, pick-and-place machine, etc.). This means that the costs can be traced to products and services through the activities needed to produce them.

LA, in contrast to ABC, is directly connected to cost minimisation. Its scope is much wider than the scope of ABC and it is not a stand-alone system, it should always be implemented by companies already implementing lean manufacturing. The implementation of lean manufacturing principles by most companies has been undermined by the traditional practice in the accounting departments of these companies. The principles of standard accounting are based on the principles of mass production. They are not wrong as such but they are not suitable for lean manufacturing, which has principles and rules (e.g. low and consistent inventory, small batches, small orders of raw materials and other supplies, etc.) that are at odds with mass production (Maskell 2004). In order to overcome this obstacle and show the full potential of the implementation of lean manufacturing, a new accounting system that takes into account the principles of lean has been developed. This new accounting system is called "Lean Accounting". The basic idea of LA is to change the way the standard accounting system handles the accounting, control, measurement and management of production processes into a way that supports lean manufacturing by applying the principles of lean thinking. LA is a necessary tool for lean manufacturing to survive; it provides better information for the management, which helps them understand the financial impact of lean improvements. This, in turn, helps them achieve better decision-making and save money by reducing costs, eliminating waste and providing more control over production processes.

The goal of this thesis is to develop a framework that can be used for process optimisation and cost minimisation in multi-assembly systems in general and in the PCB industry in particular. In order to fulfil this goal and as mentioned earlier, this research will consider three aspects in manufacturing: optimisation of production processes, cost estimation and accounting. As for the optimisation of production processes, three problems associated with PCB assembly systems will be considered: component placement sequencing, feeder assignment and board type sequencing. An

integrated approach will be developed to simultaneously solve these problems by finding the optimal component placement sequence, the optimal component to feeder assignment and the optimal board type sequence when working in multi-assembly systems. In order to achieve that, the problems will be studied in detail, a mathematical model will be developed for them and the model will be solved using metaheuristics. Since this kind of problems cannot be practically solved using conventional mathematics as it requires years of computational time for present-time personal computers to solve a medium-sized problem of this kind, two metaheuristics will be used to solve them. The metaheuristics, Taboo Search (TS) and Genetic Algorithms (GA), will be used to find the optimal (or near optimal) solution to the developed mathematical model. TS will be used because it has been used widely in the literature to solve combinatorial optimisation problems but not widely used to specifically solve PCB related problems as will be seen later in the literature review. Since GA has been widely used in the literature to solve combinatorial problems in general and PCB related problems in particular, it will be used here mainly for comparison purposes.

As mentioned earlier, regarding the cost estimation and the accounting aspects, Activity-Based Costing and Lean Accounting will be considered in this research and they will be implemented on a PCB manufacturing facility. This will allow for establishing a relationship between the three aspects under consideration in this research. A case study will be used in this research to help the reader easily understand and follow the proposed framework of this thesis. In order to validate this research, the results obtained for the three aspects considered will be compared to the results of similar studies from the literature.

The main objectives of this research are as follows:

- Development of a mathematical model for the combined problem of component sequencing, feeder assignment and board type sequencing.
- Formation of suitable TS- and GA-based algorithms to find the optimum or near optimum solution to the above-mentioned problem.
- Integration of the mathematical model and the algorithms using an appropriate interface tool.
- Implementation of ABC and LA on a PCB manufacturing facility to study the cost estimation and the accounting aspects.

- Development of a framework that integrates the optimisation of production processes, cost estimation and accounting aspects of this research.
- Test the performance of the proposed framework using a case study.
- Verification and validation of the proposed framework.

The reminder of this thesis is organised as follows: Chapter 2 includes the literature review on PCB manufacturing focusing on the three problems under consideration in this research and on the use of TS and GA in solving these problems where appropriate. In addition, the literature review will consider the other two aspects of this research: cost estimation (ABC as an example) and accounting (LA as an example). Chapter 3 explains the research methodology adopted in this research in addition to some background information about PCB production procedures, PCB production problems, TS and GA metaheuristics. In Chapter 4, the mathematical model used to solve PCB production problems mentioned earlier is formulated and the two metaheuristics (TS and GA) used to solve the mathematical model are also detailed in this chapter. ABC and LA are explained in detail, implemented on a PCB manufacturing facility and the results are analysed in Chapter 5. The validation of the results obtained and a critical evaluation for the research work as a whole are presented in Chapter 6. Finally, the thesis is concluded in Chapter 7 where the conclusion and the future work are presented.

CHAPTER TWO

2. LITERATURE REVIEW

2.1. Introduction

In this chapter, essential background information is presented, the previously published research on PCB, ABC and LA is summarised and the main results and conclusions are stated in order to distinguish this research from what have been considered before. Regarding PCB manufacturing, some problems associated with it are listed and the three PCB production problems of board type sequencing, feeder assignment and pick-and-place sequencing considered in this work are explained in detail. Since TS and GA are also the only metaheuristics considered in this research, the review focuses on the publications that considered these two metaheuristics, even though some other methods are also mentioned. As for ABC and LA, a detailed review is considered in this chapter and a special attention is paid to the PCB related cases whenever possible as PCB manufacturing is the area of industry considered in this research.

2.2. Potential problems in PCB production

The area of PCB assembly has been the focus of intensive research during the last two and a half decades. Although a detailed search has been carried out, the focus in this section is on the problems that are under discussion in this research work. In addition, a special attention is paid to the use of TS and GA as search techniques.

Crama *et al* (2002) listed the problems associated with the production planning process of PCBs as follows:

1. The assignment of PCB types to product families and to machine groups. This means, to decide which board type should be processed by which machine (machine group). In addition, to decide what board types (similar in terms of component commonality), which would be processed by one machine, should be grouped in one family so that this family can be processed using one feeder assignment.
2. The allocation of machine feeders to machines. This means, to assign the feeders to the machines taking into consideration problem number 1 above.
3. For each board type, a partition of the set of component locations on this board type, indicating which components will be placed by each machine. For the component

locations of each board type, it should be decided which set of locations should be processed by which machine. This arises when the same component type is assigned to feeders on different machines.

4. The sequence of board types, indicating the order in which the board types will be produced (to be explained further in subsection 2.2.1).
5. The location of feeders on the carrier or feeder assignment (to be explained more in subsection 2.2.2).
6. The component placement sequence (to be explained more in subsection 2.2.3).
7. The component retrieval plan indicating from which feeder a component should be retrieved. This problem arises when the same component type is assigned to more than one feeder.
8. The motion control specification indicating a specification of where the pick-and-place device should be located when it picks or places the component. This is a machine-dependable problem. For machines with a mobile board and mobile feeder carrier it should be determined where the machine head should meet the feeder carrier to pick the component and where it should meet the board to place it.

Amongst the above potential problems associated with PCB assembly, problems 4, 5 and 6 are detailed thereafter.

2.2.1. Sequencing the PCB types on the assembly line

This problem is to find the sequence of board types, amongst a set of possible sequences, which minimises the total assembly (or processing) time. Since each different board type requires a different machine set-up, the sequence that the board types follow when entering the machine affects the total set-up time. The issue here is to find the sequence of board types that minimises the set-up time. The problem of sequencing the PCB types on the assembly line is considered an instance of the job sequencing or job scheduling problem. For more information about job-shop scheduling problem the reader can refer to Käschel *et al* (1999) and Applegate and Cook (1991).

A considerable amount of research has been devoted to solving the job scheduling problems such as Nawaz *et al* (1983), Proust *et al* (1991) and Ji *et al* (2001), to mention just a few. However, as this research is interested in the sequencing problem rather than the general job scheduling problem, the only relevant research works are considered here.

Logendran and Nudtasomboon (1991) proposed a new heuristic to minimise the total completion time or makespan of the job sequencing problem. The proposed heuristic showed, the authors claimed, a high performance relative to the other methods proposed in the literature. Hashiba and Chang (1991) followed a three-step approach to reduce the number of setups for PCB assembly machines by improving the assembly sequence. The first step included grouping PCBs by applying a new heuristic grouping method. In the second step, they sequenced the groups by treating the problem as the Travelling Salesman Problem. A new algorithmic method was presented in the third step to solve the component assignment problem. The presented approach showed to be efficient for large-size industrial problems.

Sadiq *et al* (1993) developed the intelligent slot-assignment algorithm (a knowledge-based approach) to sequence a group of printed wired boards assembly jobs on a placement machine to minimise the production time. The developed algorithm consisted of two stages. In the first stage, new parts were assigned on the machine with the objective of minimising the set-up time, whereas in the second stage the parts were reassigned to minimise the runtime. Their performance evaluation studies showed that the developed algorithm tended to obtain near-optimal solutions. Bhaskar and Narendran (1996) introduced a new measure of similarity for PCB grouping, called the cosine similarity coefficient, in order to reduce the total set-up time for a single machine. They performed that by developing a heuristic, which performed very well for a number of trial problems, based on the maximum spanning tree. The problem of sequencing the groups was approximated to the Travelling Salesman Problem.

Rossetti and Stanford (2003) presented a heuristic for estimating the expected number of setups from the sequence dependent setups which may occur given a board-feeder setup configuration. The estimates were used to measure the similarity between boards in clustering algorithms and in nearest neighbour heuristics for group sequencing. The results indicated that grouped sequences generated by using the heuristic had better makespan performance compared to sequences based on the more traditional Hamming distance. Narayanaswami and Iyengar (2005) used a heuristic that resembled greedy tree traversal to efficiently sequence PCB groups. They proposed a new grouping strategy that combines the feeder contents into the similarity measure for efficient grouping, which they claimed outperformed existing methods of grouping.

As can be seen from the review, there is lack of research on the use of metaheuristics to solve the board type sequencing (or groups sequencing) problem in

the PCB industry. Metaheuristics have proven to be successful for obtaining near-optimal solutions for medium- to large-scale combinatorial problems (Ong & Khoo 1999; Wan & Ji 2001; Loh *et al* 2001; Ho & Ji 2003; Ho & Ji 2005). This is why this research is considering the use of TS and GA as search techniques to solve the PCB types sequencing problem. The use of these two metaheuristics is an opportunity to test whether or not they are successful in solving this type of problems.

2.2.2. Assigning component types to feeders

The basis of this problem is to find the component assignment that minimises the assembly time. The assignment of component types to feeders (or the location of feeders on the carrier) can be explained as follows. Since the distance between a component and its location on the board is variable depending on which feeder this component is assigned to, the time needed for the machine head to travel this distance is also variable. A solution to this problem is to find the specific feeder assignment that minimises the total distance between the components and their corresponding locations on the board. This problem is an instance of a type of combinatorial problems called Quadratic Assignment Problems (QAP). QAP is simply to assign a set of n facilities to a set of n locations. The objective is to find the assignment that corresponds to the minimum cost. The number of possible assignments in this case is $n!$. The cost of each assignment is the sum of the costs of all pairs, which can be calculated by multiplying the flow cost for each successive pair of facilities by the distance between the two locations which that pair of facilities is assigned to. For example, for $n=4$, there are four locations 1, 2, 3 and 4 and four facilities *a*, *b*, *c* and *d*. A possible assignment might be *c*, *b*, *d* and *a*. This means, facility *c* is assigned to location 1, facility *b* is assigned to location 2, facility *d* is assigned to location 3 and facility *a* is assigned to location 4 as shown in Figure 2.1.

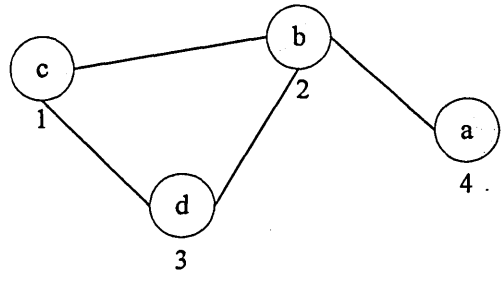


Figure 2.1. One possible assignment for the QAP

Let's assume that the distances between the locations are as follows:

$$dis(1,2)=7,\ dis(1,3)=5,\ dis(2,3)=6\ and\ dis(2,4)=8$$

and the costs of flows between facilities are:

$$fl(c,b)=12,\ fl(c,d)=15,\ fl(d,b)=22\ and\ fl(b,a)=10$$

This means, the cost of this assignment is:

$$dis(1,2)\times fl(c,b)+dis(1,3)\times fl(c,d)+dis(2,3)\times fl(d,b)+dis(2,4)\times fl(b,a)=$$

$$7\times12+5\times15+6\times22+8\times10=371$$

The solution for the problem is to find the assignment that corresponds to the lowest cost. The reader can refer to Burkard *et al* (1997) for more information about the Quadratic Assignment Problem.

Most of the literature related to this problem has focused on single-machine cases, and mainly on single-machine-single-board cases. This problem was first identified by Drezner and Nof (1984). Ahmadi *et al* (1995) discussed the same problem but under a different name (the reel positioning problem). They applied their research to a dual delivery pick-and-place machine (Dynapert MPS 500). The proposed heuristic took into account the engineering concerns for minimising the carrier movement. The component placement sequence was assumed to be given in this case and a 7 to 8% reduction in cycle time was achieved. Simulated Annealing (SA) was used to solve this problem by van Laarhoven and Zijm (1993). Other approaches were discussed by Francis *et al* (1994) and Younis and Cavalier (1990).

Gronalt *et al* (1997) discussed the switching component problem (the component set-up and the feeder assignment problem) on an SMT (Surface Mount Technology) placement machine. The approach used was based on decomposing the switching problem into the set-up problem and the assignment problem and then solving the two problems iteratively with the solutions to the set-up problem providing the basis for solving the assignment problem. The proposed heuristic performed well when applied on data reflecting actual industrial application requirements.

Regarding the multi-board type problem, there have been fewer publications. Dikos *et al* (1997) used Genetic Algorithm to generate a solution for the feeder assignment problem. Another approach was used by Crama *et al* (1990), who decomposed the planning problem into a number of sub-problems, where a new

heuristic based on the individual board characteristics was proposed. Using this heuristic, a solution to the planning problem was achieved and, according to the computational results, this approach worked well. Klomp *et al* (2000) developed a heuristic algorithm to solve the problem (they called it "the feeder rack assignment problem") for a line of placement machines and a family of boards. The algorithm was applied to real-life examples and the authors claimed that the results showed the superiority of the algorithm when compared to the approaches that were commonly in use at that time. Finally, Yuan *et al* (2006) analyzed optimization algorithms of assembly time for a multi-head machine. They developed a four-step algorithm. In the first step, the algorithm assigns the components to feeders. In the second and third steps, it assigns nozzles to the heads and organizes the feeder groups so that the heads can pick and place components on a group-by-group basis. In the last step, the algorithm assigns feeder groups to slots. The results showed that the performance of the algorithm proved to be good in practice.

2.2.3. Pick-and-place sequencing problem and the combined problem with feeder assignment

The pick-and-place sequencing problem is about finding the placement sequence of components on the board so that the assembly time is minimised. This combinatorial problem is a typical example of the Travelling Salesman Problem (TSP). TSP is simply to find the cheapest way of visiting a number of cities once and returning to the start city given the cost of travel between each pair of cities. For a specific feeder assignment, the sequence that the machine head follows to place the components on the board (which component should be placed first and which second etc.) also affects the total distance travelled. Minimising this distance requires a unique placement sequence, which is considered to be a solution to this problem. For more information about the Travelling Salesman Problem the reader can refer to Bellmore and Nemhauser (1968), Reinelt (1991) or Moscato (2003).

There have been a number of publications devoted to component sequencing problem. Ball and Magazine (1988) described some of the problems related to the PCB production. The specific problem of determining the best sequence of insertion was formulated as an instance of the Travelling Salesman Problem. An optimal solution was reached under certain conditions (when the travelled path of the head was rectilinear) using an algorithm developed for this particular problem. Sanchez and Priest (1990)

developed a component-insertion sequencing methodology and a "proof-of-concept" expert system for PCBs. They applied Artificial Intelligence and Expert Systems techniques to represent the human reasoning involved in semi-automated PCB assembly planning. Based on established assembly criteria, sequencing decision rules and data available from a CAD system, they claimed that the methodology optimally solved the component-insertion sequencing problem for semi-automated work cells.

Su and Srihari (1996) designed, implemented and validated a decision-support system that combined the use of artificial neural networks and artificial-intelligence-based technologies (expert system techniques) to identify a near-optimal solution for the placement sequence problem. Khoo and Ng (1998) developed a prototype GA-based planning system to provide near-optimal PCB component placement sequence. PCB placement priority and sequencing decisions rules were incorporated as constraints. A PCB model already used in the literature was used to validate the prototype system. The results showed that an improvement of 19.80% in the total distance was possible. Jeevan *et al* (2002) modelled the component placement sequence problem for a multi-headed machine as a Travelling Salesman Problem with the tool change factor included and the problem was optimised by Genetic Algorithms. The paper suggested that GA was a better alternative to other heuristic solution approaches such as Variable Neighbourhood Search (VNS) and local optimum search since it was simple and more promising as a global and robust method.

As for the combined problem of feeder assignment and placement sequence, Moyer and Gupta (1996) proposed a methodology for efficient process planning of a high speed chip shooter for surface mount assembly. They proposed an asynchronous model which they claimed that it would strengthen the benefits of the chip shooter features. A heuristic algorithm was developed, referred to as the Acyclic Assembly Time (AAT) algorithm, based on the asynchronous model. The authors claimed that the algorithm produced excellent results, compared to previously published problems, when tested on real-life examples. Khoo and Ong (1998) and Ong and Khoo (1999) applied GA in order to optimise the sequence of component placements onto a PCB and the arrangement of component types to feeders simultaneously. They reported a 7.4% improved results compared to Leu *et al* (1993). Su *et al* (1998) used a TS approach to solve the combined problem of sequencing placement points and magazine assignment. They applied this approach on a robotic assembly system with a moving magazine, a moving board and a moving robot. With an average reduction of 14.26% in cycle time,

they concluded that TS was more effective in solving the problem of occasionally changing coordinates than other conventional approaches.

Burke (1999) presented a new model for multi-headed placement machine. Two heuristics were modified to be suitable for the problem presented: the nearest neighbour tour construction heuristic and K-opt (a local search algorithm). Altinkemer *et al* (2000) provided an integrated approach which tackled the two sub-problems (component sequence and feeder assignment) simultaneously as a single problem with the aim of minimising the machine head movement. Khoo and Loh (2000) presented a prototype system that used GA to generate the component placement sequence and the feeder assignment for a concurrent pick and place machine equipped with a time-delay function. The sequencing process was formulated as a multi-objective optimisation problem (a multi-objective function that depends on other three functions was formulated). The prototype system was validated using examples from the literature and the authors reported an improvement of 13.0% compared to the work of Sanchez and Priest (1990). The two problems of assigning component types to feeder slots and the determination of locally optimal pick and place sequences for a multi-headed component placement machine were solved by Burke *et al* (2000). Up to 35% improvement was achieved using a local search approach to improve initial solutions determined by nearest neighbour construction heuristic and up to 40% using the variable neighbourhood search approach. The authors suggested the use of metaheuristics such as TS and Simulated Annealing (SA) to overcome the problem of local optima which they encountered. Loh *et al* (2001) developed an algorithm based on GA to solve the feeder assignment and the placement sequencing problems in PCB assembly. By comparing the performance of that algorithm to previous algorithms, they proved its superiority especially for large problems (> 50 components). Furthermore, they found that the possibility of assigning one component type to more than one feeder provided additional flexibility and reduced the assembly time.

Van Hop and Tabucanon (2001b) proposed a new approach to solve the combined problem of feeder assignment and placement sequence in a Dynamic Pick-and-Place (DPP) model where a robot arm, a board and a magazine move together with different speeds. Their approach was based on the trade-off between two strategies, assembly by area and assembly by component types, to give better results. They applied the new approach on numerical examples, which proved to be efficient. An improvement to the DPP, called the Extended Dynamic Pick-and-Place (EDPP) was introduced later (Van

Hop & Tabucanon 2001a). Deo *et al* (2002) developed a GA-based program for simultaneously optimising component placement sequence and feeder assignment in the assembly of PCBs. The program proved to be a valuable tool for providing good solutions to the problem in a multiple set-up and multiple sourcing PCB assembly machine arrangement. Ong and Tan (2002) demonstrated the application of GA to solving the placement sequencing and feeder assignment problems. They focused on solving the moving board with time delay problem associated with high-speed turret-head chip-shooters. The results obtained showed improvements compared to the results of previous work carried out by Leu *et al* (1993) and Nelson and Wille (1995).

Ayob and Kendal (2002) used a new approach with the objectives of minimising the robot assembly time, feeder movements and PCB table movements. Their approach was a revised dynamic pick and place point (DPP) approach which they called Chebychev DPP (CDPP). The difference between DDP and CDPP was that some unnecessary movements were eliminated in the latter by taking into consideration the next PCB coordinate when determining the current pick up and placement locations. A 3.29% improvement in the cycle time was achieved compared to the approach of Wang *et al* (1998). Ho and Ji (2003) presented a hybrid Genetic Algorithm to optimise the sequence of component placement and feeder assignment simultaneously for a chip shooter machine with the objective to minimise the assembly time. The search heuristics used in the GA included nearest-neighbour, 2-opt and an iterated swap procedure. The hybrid GA, the authors claimed, performed better in terms of assembly time when compared to the results obtained by other researchers. Later on, Ho and Ji (2005) considered the case where the components of the same type were assigned to more than one feeder. This added the retrieval problem to the feeder assignment and the placement sequence. The three problems were solved simultaneously using a hybrid Genetic Algorithm for a sequential pick-and-place machine. The performance of the algorithm was compared to Leu *et al* (1993) and to Ong and Khoo (1999) where the improvements were 7.64% and 0.23% respectively. Finally, Yilmaz and Gunther (2005) presented a novel approach for group setup strategies for the case of a single assembly machine with the aim of minimising the makespan. They proposed a methodology based on applying machine-specific algorithms for optimising feeder setups for each PCB family and providing placement sequences for each PCB type.

As can be seen from the above-mentioned review, there has been a great deal of research on the three PCB production problems under discussion (i.e. board type

sequence, feeder assignment and placement sequence). The review shows how the previous research has considered these three problems either individually or in pairs. However, an attempt to solve the three problems simultaneously has not yet been considered. That is what Chapter 4 of this thesis is focusing on. The three problems are combined and solved using two search techniques: Taboo Search and Genetic Algorithms. These two techniques, as mentioned in subsection 2.2.1, have not yet been considered for solving the board type sequencing problem.

2.3. Activity-Based Costing

Ever since Johnson and Kaplan (1987) wrote their book "Relevance lost: The Rise and Fall of Management Accounting" and the term ABC came to existence, a great deal of research has been attributed to this term. Since it is not viable to mention every published research work about ABC, the focus in this section is on the application of ABC in the manufacturing domain in general and on PCB manufacturing in particular. ABC has been developed as a cost estimating technique that can be used in different types of organisations such as manufacturing, service, retail, etc. Furthermore, the use of ABC has not been limited to product costing but to a variety of applications; even the term Activity-based Cost Management (ABCM) has become widespread in the academic domain (Bellis-Jones & Develin 1999). The following are some of the applications of ABC adopted from Innes and Mitchell (1995).

Product pricing:

In order to price a product or service their actual cost must be known. ABC provides the means that can be used to calculate the actual cost of a cost object. Hence, the "cost plus" approach can be used flexibly and successfully to provide a competitive price.

Decision making:

The make-or-buy decision is very important to the success of any organisation. By knowing the true cost of a product or service it can be assured that the right make-or-buy decision has been taken.

Cost reduction:

ABC helps identify the value added and non-value added activities. This allows the management to pay more attention to the non-value added activities and try to eliminate them or at least to reduce their consumption for resources. As for the value

added activities, by knowing the way these activities consume resources, the possibility for cost reduction becomes available.

Budgeting:

ABC provides the required details for good budgeting practice. The statistics of cost drivers provided by ABC allow for reliable assessment for future needs of resources.

Product design:

Considerable amount of research has been devoted to the relationship between ABC and product design. For example, Ong (1995) developed an ABC estimating system to help designers estimate the manufacturing cost of a PCB assembly at the early stage of design. Ong stated that his system would enable designers to identify the activities that incur high cost and, hence, they would be able to make efforts to reduce their costs. Tornberg *et al* (2002) investigated the possibilities of ABC and the modelling of design, purchasing and manufacturing processes in providing the designers with useful cost information. They concluded that ABC and process modelling would provide a good starting point in heading towards cost-conscious design. Ben-Arieh and Qian (2003) presented a methodology for using ABC to evaluate the cost of the design and development activity for machined parts. The methodology was tested using sample rotational parts developed in a controlled environment. The methodology proved to be more accurate than the traditional cost estimation provided by the shop accountant. In addition, it would provide the ability to expand the most costly activities and investigate the causes of the cost. Giachetti and Arango (2003) developed an ABC model for PCB cost estimation based on the design parameters. A design example was used to verify the model, which revealed important relationship between cost and design parameters. The model enabled the designers to assess the impact of their decisions on the cost and this helped them generate lower cost alternatives.

It is widely known that 80% (Duverlie & Castelain 1999) or up to 85%, as Whitney (1988) claimed, of product cost is determined at the design stage. This shows how important the relationship between the design decisions and the final cost of the product. Since ABC provides cost drivers rates, this help designers design less costly products.

Cost modelling:

According to Cooper (1990), ABC helps structure costs into different levels which gives a realistic representation of them (more on that in Chapter 5):

- Unit-level costs: costs are affected by each additional unit
- Batch-level costs: costs are affected by each additional batch
- Product-level costs: costs are affected by the existence of a product
- Facility-level costs: costs are not related to products but to the facility as a whole.

Ong and Lim (1993) developed cost models for PCB assembly taking into consideration the activities that incurred cost due to complexity, volume and batch size of the produced PCB. The activities were allocated into three levels: unit-level, batch-level and product-level, where the cost of assembling one PCB was the total of all costs in these three levels. The application of the models developed was illustrated by presenting an example. Later on, Ong (1995) presented an ABC estimating cost system to help estimating the cost or PCB production at the design stage. He used activity charts, worksheets and a build-up table to calculate the activity costs. By presenting an example, Ong illustrated how to apply his estimation system. A case study of successfully implementing ABC against all odds (the top management were not convinced that the benefits would outweigh the cost of implementing ABC) was presented by Dedera (1996). In spite of the top management's initial objection and against the experts' advice regarding the timing and the scope of the project, an integrated ABC system was successfully implemented within nine months. Dedera attributed the success of the project to two factors: people, where the best available were selected, and communication, where different departments in the company were all consulted and kept up to date regarding the implementation of the ABC system. The successful implementation was reflected in many areas: accurate cost data to support the decision-making process, better communication between manufacturing and administrative departments, better financial outcome, etc. Sohal and Chung (1998) presented two case studies on the implementation of ABC. They discussed the introduction of ABC and the benefits and problems experienced during implementation process and identified the factors critical to successful implementation of ABC (e.g. educating and training of people, commitment, acquiring external experience, keeping it simple, adequate allocation of resources and continuous feed back to top management). These factors were extensively explored by Marri and Grieve (1999) who introduced a framework for the justification and implementation of ABC in the small- to medium-sized enterprises.

The issue of combining ABC with simulation models was considered by Spedding and Sun (1999). They argued that using a simulation model made it easier to implement ABC on their case study. The importance of their research lies in the fact that using simulation and ABC together provides a more powerful tool than either of them. This can be used for providing more useful management information. Tornberg *et al* (2002) produced a study on the use of ABC and process modelling in providing cost information for designers. The result of their study suggested that using ABC alongside process modelling was a step forward towards creating better cost-conscious design. Gupta and Galloway (2003) showed how an ABC management system could be used as an information system to support the decision-making processes of different operations (product planning and design, inventory management, capacity management, etc.). They demonstrated how their system enabled the operations manager to increase the quality of the decision-making process. Özbayrak *et al* (2004) applied ABC alongside a mathematical and simulation model on a manufacturing system that used either MRP or JIT (push or pull system) in order to estimate the product costs. Their work showed how valuable ABC was as a tool for providing not only more accurate cost information compared to the traditional cost allocation system but also important management information. They concluded that the manufacturing planning and control strategies greatly affected the manufacturing costs and that the pull system provided lower manufacturing costs. Homburg (2004) argued that since ABC allocated overhead costs proportionally it was a heuristic, therefore he used simulations and mixed-integer programming to analyse the extent of the sub-optimality incurred by ABC-heuristics. Homburg argued that previous research used a simple set of cost drivers, which restricted the potential of ABC as a heuristic. Therefore, he analysed the effects of establishing a cost driver corresponding to a higher cost level (e.g. portfolio-based cost driver). This driver was used to allocate the costs to inflexible overhead resources proportionally, which improved the quality of ABC-heuristics significantly. Although Homburg had reservations about generalising the results since the simulations he used were based on simplified scenarios, he argued that the results provided some insight in to the quality of using ABC for decision making.

The success rate of implementing ABC has been acceptable in general, however, the fact that ABC assigns costs to products on the basis of cost drivers, which may or may not be proportional to the volume of the output, has attracted some criticism. Noreen and Soderstrom (1994) presented a case study in which they proved that the

assumption that the overhead costs are proportional to activities is not always correct. Therefore, there are conditions (e.g. all costs must be strictly proportional to their cost drivers) under which ABC can provide relevant costs (Noreen 1991; Christensen & Demski 1995). When the conditions are not met alternative solutions can be used. For example, Homburg (2005) suggested an alternative to using the cost driver rates by calculating relative profits by using "concepts of multi-criteria decision-making". Another example is the research work by Kim and Han (2003) in which they proposed the use of hybrid artificial intelligence techniques. They used GA to identify optimal or near-optimal cost drivers and used artificial neural networks to allocate indirect costs with nonlinear behaviour to products. The results of their work revealed that their proposed model performed better than the conventional ABC model. Furthermore, Ben-Arieh and Qian (2003) argued that the cost estimation of the design activity has been rather difficult. This claim is disputable since an ABC model that focused on the design stage of PCB fabrication was developed by Giachetti and Arango (2003). In this cost model the cost centres and cost drivers were all based on design parameters so that model could be used as an evaluation tool by PCB designers when making design parameter trade off decisions.

In general and as this review shows, there has been a positive attitude towards ABC amongst researchers in spite of the existence of some scepticism. Innes and Mitchell (1995) argued that there was a lack of ABC implementation in the industry. This might be true at the time they conducted their survey but may not be correct nowadays. The outcome that could be drawn from this review is that ABC can be successfully applied with the condition that the costs must be proportional to their cost drivers. Therefore, ABC will be implemented in this research and the pros and cons of its implementation in PCB manufacturing industry will be highlighted.

2.4. Lean Accounting

Lean Accounting can be defined as "the general term used for the change required to a company's accounting, control measurement, and management processes to support lean manufacturing and lean thinking" (Maskell 2004). In order for lean manufacturing to succeed it has to be accompanied with a supporting LA system. Lean manufacturing has been developed and applied for a few decades now; however, LA is a more recent subject. Åhlström and Karlsson (1996) argued that the need for a change in the accounting system was realized during the 1980s; however, there were no clear-cut

suggestions at that time. They explored the role of management accounting in the changes occurring to the production system by introducing lean production. They concluded that in order for the management accounting system to create an "impetus for change" towards lean it had to shift the focus from a machine/operator level to the whole production flow and from the operation level to the whole production system. This "impetus of change" could not be created until "traditional performance measures have reached a certain threshold" (Åhlström & Karlsson 1996).

A major step forward occurred with the introduction of the principles of lean thinking by Womack and Jones (1996). These principles (value, value stream, flow, pull and perfection) have been the drivers for introducing new methods for management accounting. A series of articles were then published presenting more details about LA. For example, Baggaley (2003a) explained how standard costing was not only unsuitable for lean manufacturing but also harmful to lean principles. Therefore, standard costing measurements of labour efficiency, machine utilization, departmental budget focus, etc. had to be replaced with new lean measurements such as cycle time, first time through, value stream focus, etc. Again, Baggaley (2003b) emphasized the importance of organizing and costing by value stream arguing that some people within the organization might not fit into a particular value stream (e.g. plant manager, IT, human resources, etc.). Kroll (2004) agreed that standard cost accounting was not suitable for lean operations and using alternative accounting concepts was necessary to solve some problems. However, she argued that there was maybe a problem with accurately pricing products when considering the value stream rather than the individual product. This argument is true to some extent since 'value stream costing' calculates the average cost of a family of products produced by a value stream not the cost of a particular product.

Finally, Maskell and Baggaley (2004) presented what they called a "proven path" for lean accounting in their book "Practical lean accounting" supported by case studies from the manufacturing industry. They explained in detail how lean manufacturing alongside LA could transform the business. They explained step by step the path to a lean enterprise focusing on the management accounting part.

As LA is a relatively recent subject, it has not been given the required amount of research that makes it widely spread. Considering LA in this research is another step towards a better understanding of this accounting system and to give the reader a better idea of how LA could be implemented in practice. As mentioned earlier, a PCB manufacturing facility is the example to which the three aspects of this research

(optimisation of production processes, cost estimation and accounting) will be applied. This potentially helps the author make a comparison between ABC and LA if it is deemed necessary.

2.5. Summary

A review of past research on PCB, ABC and LA has been considered in this chapter. The review shows how the PCB production problems considered in this research have been researched in detail and how the past researchers have tried solving them individually or in pairs. Too many approaches and techniques have been tried each of which was claimed to perform better than its predecessors. This has limited the number of choices still available for research in this area; however, there is still room for manoeuvre. An attempt to solve these production problems simultaneously has not yet been considered before. In addition and as mentioned in subsection 2.2.1, metaheuristics have not yet been considered for solving the board type sequencing problem. In order to cover this area, the PCB production problems considered in this research will be solved simultaneously using two metaheuristics: Taboo Search and Genetic Algorithms.

The main outcome of this review is that the implementation of ABC may lead to better results when compared to the traditional costing system. However, since the results of implementing ABC would vary depending on whether or not the costs are proportional to the cost drivers, this means they would depend on the type of industry ABC is applied to. Taking this into account and considering the fact that there is lack of ABC implementation in the PCB industry, ABC will be considered in this research to cover these two issues.

As for LA, the review has shown that there is a consensus between the researchers considered in the review that the standard accounting system is no longer suitable for companies implementing lean manufacturing; however, the debate about an alternative is still ongoing. A good alternative would be an accounting system that is based on the same principles as lean manufacturing. Therefore, LA will be studied in this research to enrich the debate about the suitability of its implementation in manufacturing in general and in PCB industry in particular.

CHAPTER THREE

3. RESEARCH METHODOLOGY

3.1. Introduction

The research methodology adopted in this research is presented in this chapter in order to make the research easier to comprehend. The methodology is explained and justified and, then, its implementation on the three aspects considered in this research (optimisation of production processes, cost estimation and accounting) is explained in detail. In addition, some essential background information with some illustrating examples is discussed. This is mainly performed for the Taboo Search and Genetic Algorithms search techniques and for the production process of PCBs.

3.2. The methodology

The word "methodology" is used by different researchers to have different meanings. Lehaney & Vinten (1994) outline the different uses of the word "methodology" as follows:

- "the ways in which hypotheses become theories – scientific methodology;
- the ways in which techniques are chosen to address a particular problem;
- the ways in which problems are chosen, which addresses the question of sponsorship;
- methods or techniques;
- the modelling process, which include hard and soft systems approaches, and the ways in which the relevant variables are chosen for a model, and how reality is concomitantly simplified;
- the chronological planning of events – the research programme."

This means, there are many types of methodologies (e.g. surveys, experimental, questionnaires, case studies, mathematical, comparative, etc.) a researcher can use. Which methodology to choose depends on the nature of the research undertaken and what outcome is expected to be achieved. It is possible to use a combination of more than one methodology if the nature of research undertaken necessitates that.

The research undertaken here deals with three different aspects, therefore, the use of more than one methodology is required. Since in the first aspect considered in this

research (the optimisation of production processes), some of the production problems associated with PCB manufacturing will be solved, the appropriate methodology to use is the mathematical programming. Mathematical programming is used to formulate the manufacturing problems considered into mathematical equations and since these problems could be combinatorial optimisation problems, they can be solved using metaheuristics. Once the problems have been solved, numerical example has to be used to test the solutions provided by the metaheuristics; therefore, the solutions provided are applied to a case study. This means, the methodology that will be followed, as far as the optimisation of production processes is concerned, is a combination of mathematical programming to formulate the production problems, metaheuristics to solve them and a case study to test the solutions. It has to be noted that the mathematical part of this methodology will be of greater importance and will play a bigger role than the case study part.

The same discussion regarding the optimisation of production processes can be applied to the other two aspects considered in this research. The cost estimation and accounting aspects are of different nature compared to the optimisation of production processes; still, a similar methodology can be used. Although there are no problems to solve regarding the cost estimation and the accounting aspects, the use of mathematical programming is still required to solve the equations that will be used in both aspects. ABC and LA will be implemented in this research as examples of the cost estimation and accounting aspects respectively and both use mathematical equations in their implementation process. To test the performance of both ABC and LA, the implementation procedures are applied to a case study. This means, for the cost estimation and accounting aspects considered in this research, the methodology that will be followed is a combination of the mathematical methodology and the case study methodology. However, in contrast to the methodology mentioned above, the case study part of this methodology will play a greater role compared to the mathematical part.

The case study methodology is a non-experimental method and it involves collecting detailed data about the case under consideration. The data are mostly descriptive and this provides the researchers with the ability to fully understand the case but not necessarily explain it. Therefore, the conclusions drawn may not have a satisfactory support and they may or may not be generalised since what applies to the case study considered does not necessarily apply to other cases (Anonymous 1989). In spite of this limitation, choosing the case study as part of the methodology considered in

this research, especially for the cost estimation and accounting aspects, can be justified due to the fact that case studies are used when a detailed investigation is required (Feagin *et al* 1991 cited in Tellis 1997), which is the case in this research. The nature of the subjects considered in this research needs in-depth investigation in order to understand the relationship between them, especially the problems considered in the production process optimisation. In addition, the generalisation issue has been refuted by researchers such as Yin (1994) and Stake (1995).

There are many applications for the case study, some of which has been identified by Yin (1994):

1. Explaining complex causal links in real-life investigation.
2. Describing the real-life context in which the intervention has occurred.
3. Describing the intervention itself.
4. Exploring those situations in which the intervention being evaluated has no clear set of outcomes.

In this research the application of the case study methodology belongs to the 3rd and 4th cases since the optimisation of production processes, cost estimation and accounting aspects of this research and how they will be applied to the case study will be explained in this research. In addition, some of the subjects considered here (e.g. TS, GA, ABC and LA) are still being developed and more modifications are still being applied to them. The author believes that implementing a case study to these subjects will add to the base of knowledge that already exists.

3.3. Implementation of the methodology on PCB production problems

In order to be able to implement the methodology chosen on the PCB production problems they have to be explained first, which is already presented alongside the literature review in Chapter 2. The mathematical formulation for these problems will then be performed by deriving the equations required to calculate the total processing time. Since the mathematical equations for the production problems cannot be solved mathematically because this will take a long CPU time, an algorithm based on two search techniques will be developed to solve the equations. The two search techniques used in the algorithm are TS and GA metaheuristics. The reason of choosing these two techniques is that TS has been used successfully to solve combinatorial problems in general but not widely used to solve PCB related problems. In fact, just two cases, Su *et al* (1998) and Wan & Ji (2001), have used TS for PCB related problems. In addition, TS

has never been used to simultaneously solve the three PCB related problems considered in this research, which give the researcher the opportunity to test how TS performs in this case. As for GA, it has been widely used in the literature to solve PCB related problems (e.g. Dikos *et al* (1997), Khoo & Ng (1998), Khoo & Ong (1998), Deo *et al* (2002), Jeevan *et al* (2002), Ho & Ji (2005), etc.) and it is used in this research for comparison reasons only. Since metaheuristics do not guarantee to give the optimum solution, therefore, the outcome of the proposed algorithms could provide optimum or near optimum solution for the three problems (board type sequencing, feeder assignment and component placement sequence).

Having solved the production problems using the proposed metaheuristics-based algorithm, the implementation of the other part of the methodology (i.e. the case study) can be initiated now. The first step in the case study methodology is the data collections. It would have been better to adopt a clinical methodology by collecting the data through direct contact with the company since it would have given the researcher a chance to collect more data than it is usually available from other sources (Schein 1987). However, this was not possible and hypothetical data will be used instead. The proposed algorithm will then be applied to the PCB case study using the hypothetical data, and the results will be analysed and comparisons will be made when different parameters are used. Finally, the conclusions, recommendations and implications will be developed based on the evidences found.

3.3.1. Taboo Search

In order to be able to explain the TS algorithm proposed, some background information about this search technique should be presented first. Taboo Search, which is a search technique first proposed by Glover (1989; 1990), is a metaheuristic used mainly to solve combinatorial problems. What differentiates TS from other search methods is its unique memory structure that guides the search to avoid entrapments in local optima. In its basic version, TS procedure maintains a record or a *taboo list* of the characteristics of the most recently found solutions. It starts with an initial feasible solution generated randomly (or by using other techniques) and, then, the *neighbourhood* of the initial solution is then determined. The neighbourhood of a solution is the set of solutions that can be generated from this solution using what is called a *move* as presented in Figure 3.1, which shows a graphical representation of the neighbourhood solutions and moves. The move that leads to a better (or the best)

solution in the neighbourhood is performed and is then added to the taboo list. The neighbourhood of the new solution is then generated and a new move is performed and added to the taboo list and so on. A search move will not be performed as long as it is in the taboo list unless it leads to a specific 'good' solution (*aspiration criterion*). An example for a good solution may be a solution that is better than the best solution found so far. This process is repeated until the terminating condition is satisfied.

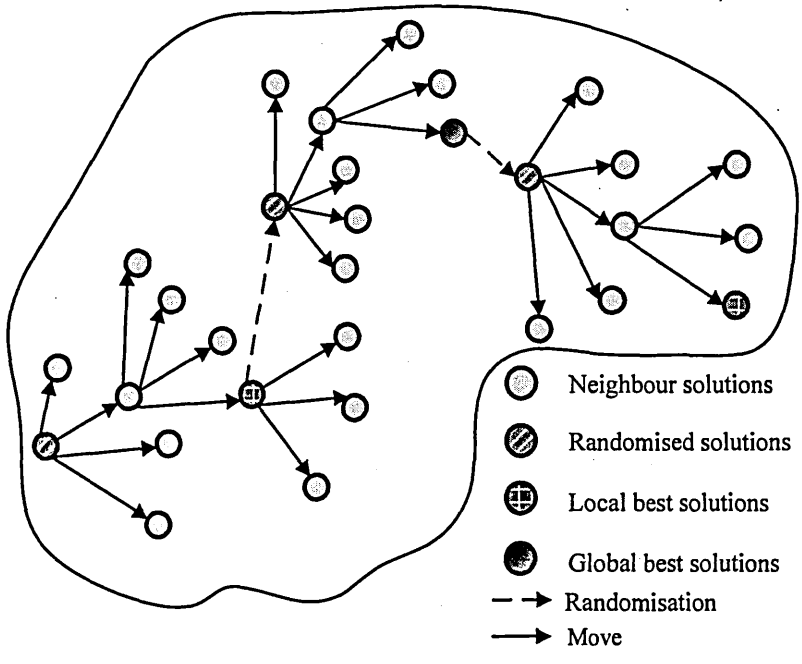


Figure 3.1. A representation of the neighbourhood solutions

TS procedure is mainly used to prevent entrapment in local optima. However, a short-term memory structure (*taboo list*) does not guarantee that the principles of local intensification, in a promising region, and global diversification, into new regions, are considered. The local intensification is not guaranteed because the short-term memory may prevent choosing a potentially good solution because it is in the taboo list. Preventing potentially good solutions to be chosen means that the search is not intensified in a particular area (i.e. good solutions in this area are not considered). This usually happens when the taboo list is long. As for the global diversification, it is not guaranteed because although the short-term memory may prevent small loops, it does not guarantee preventing bigger loops (when the taboo list is short) because the size of taboo list is limited. To overcome this weakness, intermediate and long-term memory

structures have been introduced. The intermediate memory is used to maintain a record of some good solutions. These solutions, when encountered later in the search, are forced to be the next ones to be chosen so that the search is focused on good region (intensification). In contrast, the long-term memory is used to maintain a record of solutions that have not been encountered for a quite long time. Using such solutions guides the search to regions with fewer frequent visits with the aim to scout the areas of the solution scope where potentially hidden good quality solutions might be found (diversification).

The TS algorithm used in this research, which will be detailed in Chapter 4, adopts three different methods to provide the initial solution. The first is a random solution and the other two are problem-specific solutions and have better quality compared to the random one. The rationale behind this is to start the search from a good quality solution rather than a random one. This may reduce the search time and result in a better final solution. The best solution, rather than a better solution than the current one, amongst the neighbourhood solutions is chosen to be the next current solution. This is performed to reduce the search time although this may or may not affect the quality of the final solution. The algorithm also uses the three types of memory mentioned in the previous paragraph, short term, intermediate and long term, to ensure that the intensification and diversification attributes of TS are met. In addition, different taboo list sizes are used in the algorithm in order to check the effect on the quality of the final solution.

3.3.2. Genetic Algorithms

GA, which was first proposed by John Holland during the 1960s, is a search technique based on the emulating of the evolutionary principles of natural selection and survival of the fittest. The procedures of GA start with a *population* of chromosomes (parents) that generates another population of chromosomes (children) using the process of *crossover* and variation operators such as *mutation* and *inversion*. The *fitness function* is used to evaluate the population and offspring chromosomes so that they can be ranked accordingly. The fittest offspring chromosomes are chosen to become members of the population chromosomes. The process is repeated until the terminating criterion is satisfied. The crossover process ensures that some characters (genes) of the parents move to the children. By choosing the best children to become parents again it is ensured that good genes are passed to the next generations. The mutation/inversion

variation operators are used to introduce new, potentially good, genes to the process. The following example, derived from Khoo and Ng (1998), clarifies how crossover and mutation work.

Let us have the two following parents:

Parent 1: *a b c d e f g h i j*

Parent 2: *d i j a h b g c e f*

To generate two children from these parents using crossover, the parents are cut randomly into three parts as follows:

Parent 1: *a b c* | *d e f* | *g h i j*

Parent 2: *d i j* | *a h b* | *g c e f*

The first child generated by the process of crossover should maintain one of the parts of Parent 1. Let's assume it is the middle part, this means Child 1 should look like:

Child 1: ? ? ? *d e f* ? ? ? ?

One possibility for completing Child 1 is to start filling the empty spaces in by using genes from Parent 2 as they appear from left to right excluding already used genes (*d*, *e* and *f* in this case). Child 1 should then look like:

Child 1: *i j a d e f h b g c*

Following the same procedures, Child 2 can be derived and it should look like:

Child 2: *c d e a h b f g i j*

One possible mutation could be to randomly choose two genes in the preserved part of a child and remove the first gene from its position, then shift all the genes between the two chosen genes (including the second chosen gene) one position to the left, and finally put the first chosen gene in the empty position created by the shift (the original position of the second chosen gene). By applying this procedure to Child 2 a mutated Child 2 is created as follows:
The two randomly chosen genes are *a* and *b*, the mutated Child 2 should then look like:

Child 2: *c d e h b a f g i j*

Another example for mutation could be to exchange the two randomly chosen genes. The mutated Child 2 in such case should look like this:

Child 2: *c d e b h a f g i j*

In order to apply GA to a problem, the solutions must be coded into strings. These strings represent the population in GA. The fitness function is a function that evaluates the solutions of the problem so that the best can be determined. The termination criterion is satisfied when a particular good solution has been found or until a specified number of iterations have been performed. For example, GA is used to find the optimum/near optimum solution to the component placement sequencing problem in PCB manufacturing. The fitness function in this case will be the function that calculates the time required to place the components on the board (or the distance travelled by the machine head). The termination condition could be to stop the search when the number of iterations reaches 200.

The GA algorithm used in this research is similar to what has been described in this subsection. The initial population is created randomly and the children are created using the crossover and mutation and/or inversion described in the example above. The best solution is updated every generation, if applicable, and the algorithm stops when the predefined number of generation is reached. The detailed description of the algorithm will be presented in Chapter 4.

3.4. Implementation of the methodology on ABC and LA

Implementing the mathematical part of the methodology adopted in this research on ABC involves the following steps:

- Determining the cost of indirect resources and their drivers:

Once the indirect resources and their drivers have been determined and the total cost of each resource is calculated, the *resource rate* is calculated according to the following equation:

Resource rate = Total cost (per year or per product life) / indirect resource driver spent

- Identifying the cost centres and assigning the resources to them:

After the cost centres and their resources have been identified and the total cost of each cost centre is calculated using the *resource rate* calculated in the previous step, the *cost centre rate* is calculated according to the following equation:

Cost centre rate = total cost of cost centre / cost centre driver spent

- Identifying activities, calculating their costs and the rates of their cost drivers:

The activities are identified in this step and the total cost of each activity is identified using the *cost centre rate* calculated in the previous step, then, the *activity cost driver rate* is calculated according to this equation:

Activity cost driver rate = cost of activity / activity cost driver spent

- Calculating the costs of products:

The cost of each product can now be calculated, using the *activity cost driver rate* calculated in the previous step, for each activity involved in the manufacturing of that product.

Similar process can be applied to implement the mathematical part of the methodology on LA. In this case, some of the steps involved are the following:

- Performance measurements,
- Calculating the financial benefits of applying lean manufacturing,
- Eliminating wasteful financial transactions,
- Value stream costing,
- Features and characteristics costing,
- Target costing, etc.

Each of these steps involves a great deal of calculations using many mathematical equations, which will be detailed in Chapter 5.

Having implemented the mathematical part of the methodology on ABC and LA, the case study part of the methodology can now be implemented. The implementation steps for both ABC and LA can be implemented again using the same PCB case study details used for the optimisation of production processes. Understanding the manufacturing process of PCB is a perquisite to implement the case study part of the methodology adopted; therefore, the production procedures and the manufacturing process will be explained in the next two subsections. The detailed implementation of the case study on ABC and LA will be detailed in Chapter 5.

3.4.1. Production procedures

In general, the assembly process of PCBs consists of the following steps:

- Once the order is accepted, it is input into a Master Production Scheduling (MPS), which provides weekly product requirements over 6 to 12 months.
- Depending on the MPS results, the Material Requirements Planning (MRP) is calculated. The volume and timing of each order (order release and due-date) are determined as a result of the MRP.

- Components are prepared (grouped into kits, marked for future tracking, etc.).
- The components are inserted onto the board according to specific sequencing, assignment to the available machines and assignment of corresponding feeders.
- The final step includes inspection, soldering the components and performing the final board testing.

Amongst the previous steps, insertion is considered the most costly and time-consuming step. Hence, the efficiency of the assembly line is largely determined by the efficiency of this step. There are two basic factors that affect the insertion step: set-up time and placement time. Set-up time is the time needed to prepare the machine to be ready for operation, which basically involves assigning the component types to feeders and programming the SMT machine. The set-up time is affected by the board type sequence, which is the subject of one of the PCB production problems considered in this research, as explained in subsection 2.2.1. The placement time is the time needed by the machine to place the components on the board. There are two optimisation problems that affect the placement time. These two problems are already explained in subsections 2.2.2 and 2.2.3 and they are the assignment of component types to feeders and the sequence of placing the components on the board by the machine head. As mentioned earlier, these three problems represent the optimisation of production processes part of the problem considered in this research.

3.4.2. General PCB manufacturing process

The general manufacturing process of PCB is outlined in Figure 3.2. In the first step the components and boards are prepared for assembly by cleaning and kitting them. In the second step, the assembly is performed using pick-and-place machine for standard components and robotic machine or manual assembly for odd-shaped components. One of two different soldering methods is then performed in the third step, either wave soldering (laminar or turbulent) or reflow soldering, which can be applied using one of these methods: thermal conduction, infra red, vapour, soldering iron, laser and hot gas. In the forth step another assembly process is performed using either robotic machine or manual assembly for heat-sensitive components. The board is then cleaned and tested using in-circuit testing. Any required rework or repair is performed in the final step.

There are two types of components, surface-mount and through-hole. In general, a PCB may contain both types. In this case the soldering and assembling processes can be

achieved through one of two main technologies: Adhesive attach-wave soldering and reflow soldering, which are illustrated in Figure 3.3 "a" and "b" (Coombs 1988).

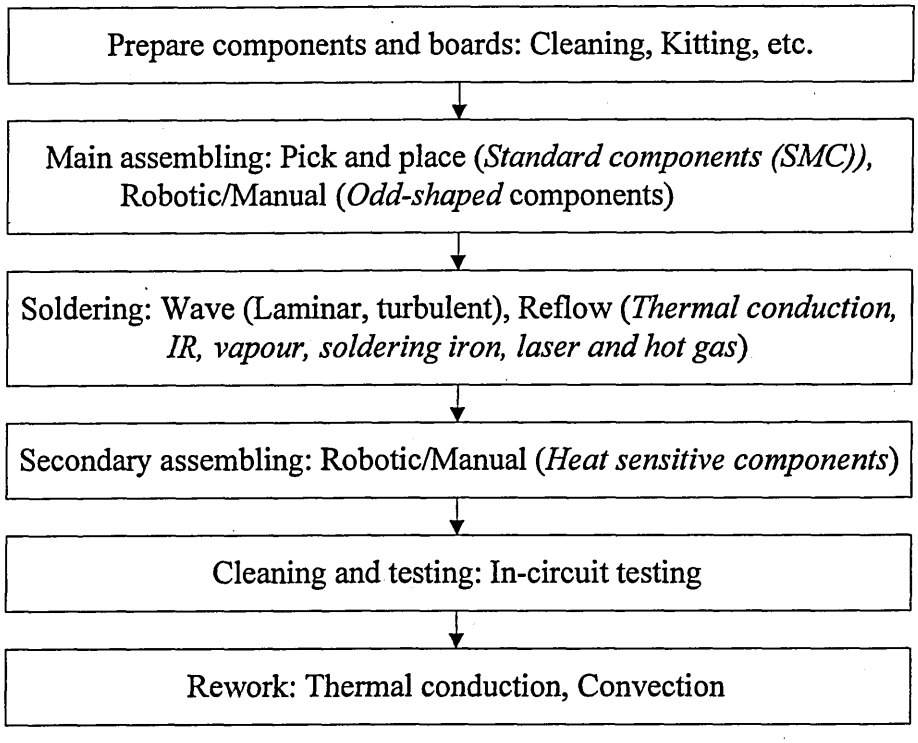
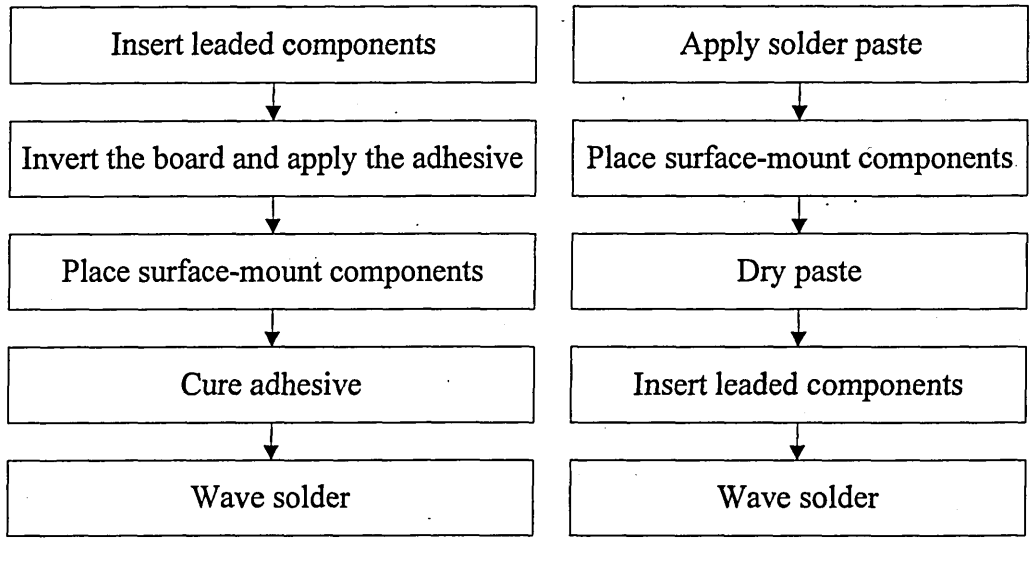


Figure 3.2. General PCB manufacturing process



(a) (b)

Figure 3.3. Flowcharts of PCB production process; a: adhesive attach-wave, b: reflow

- Adhesive attach-wave: According to this technique the leaded components (through-hole) are inserted on top of the board, then the board is inverted and the adhesive is applied (using syringe dispensing, pin transfer or screen printing). The surface-mount components (SMCs) are then placed and the adhesive is cured (using heat, UV or IR). The board is then wave-soldered.
- Reflow: The solder paste is applied onto the board and used to hold the SMCs. The paste is then dried, to remove the solvent, and reflowed (using thermal conduction, IR or vapour phase). In the next step, the board is cleaned and the through-hole components are inserted. In the final step, the board is wave-soldered.

3.5. Summary

The research methodology adopted in this research has been discussed in this chapter. It has been explained that since the three aspects considered in this research relies on mathematical calculations and equations to solve some of the production problems and implement ABC and LA, the use of the mathematical programming is more appropriate. The need to use numerical example to test the solutions provided by the metaheuristics and to test the performance of ABC and LA a case study has to be used. As a result, a two-part methodology based on the mathematical and case study methodologies has been chosen to be adopted in this research. It has been discussed how the mathematical part of the methodology has a greater relevance when the optimisation of the production processes is considered and how the case study part is more relevant when the cost estimation and the accounting aspects are considered.

The case study methodology has been outlined and the justification for choosing it has been stated. Some required background information about TS, GA and PCB production process has also been presented in this chapter due to the fact that this background information is required to understand the implementation of the methodology on the subjects under consideration. The implementation of the methodology chosen in this research on the three aspects considered has been outlined in this chapter and the detailed implementation will be presented in Chapter 4 and Chapter 5.

CHAPTER FOUR

4. PROPOSED FRAMEWORK AND PCB ALGORITHMS

4.1. Introduction

In Chapter 2, some detailed literature review and background information about the three PCB production problems considered in this research are presented. This chapter explains how these three problems are mathematically formulated and how the proposed algorithm is used to solve them. In addition, the two proposed metaheuristic algorithms (TS and GA) are developed and explained in detail. As mentioned in Chapter 3, the proposed algorithm is tested using a case study. The results from the case study are discussed and analysed, and some recommendations are made.

4.2. Mathematical formulation

In general, the processing time required to assemble the printed circuit boards is the sum of two main parts: set-up time and placement time. Set-up time is machine related and it depends on many factors. The only factor that is of interest in this research is the number of feeders that are going to be replenished. Assuming that the set-up time is proportional to the number of feeders replenished (Sadiq *et al* 1993), the set-up time equation can be written as:

$$ST = f(f_n) = af_n + b \tag{4.1}$$

where

ST is the set-up time,

f_n is the number of replenished feeders,

a is a constant and it is experimentally calculated, and

b is the time required to program the machine for a particular board type k and is considered constant for all board types in this research.

The placement time is more complicated and in addition to being dependent on the machine type, it is also dependent on the feeder assignment and the placement sequence. Therefore, to clarify this issue, an example of component placement is presented here. Table 4.1 contains the definition of some of the variables used for the mathematical formulation.

Table 4.1. Definition of variables for the mathematical formulation

Symbol	Value	Description
k	$1, 2, \ldots, K$	Number of board types
b_k	$1, 2, \ldots, B_k$	Number of boards of type k
f	$1, 2, \ldots, F$	Number of feeders
i	$1, 2, \ldots, I$	Number of component types
p_i	$1, 2, \ldots, P_i$	Number of components of type i
c_k	$1, 2, \ldots, C_k$	Number of components on board k

There are four types of components: 1, 2, 3 and 4 placed in 7 feeders. Assuming that the feeder assignment and the placement sequence are given as presented in Figure 4.1:

Feeder assignment: 0,2,1,0,4,0,3 (0 represents an empty feeder)

Placement sequence: 2,4,3,1

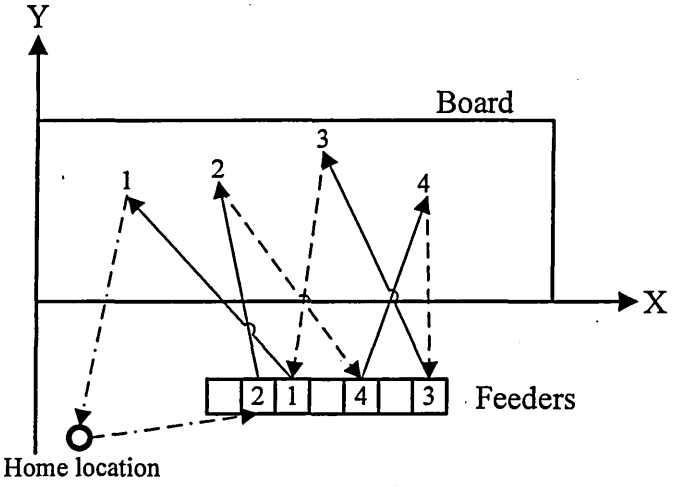


Figure 4.1. Layout of the board and feeders

The placement time (or pick and place time) is the sum of the following times, which can be expressed according to their successive occurrences as:

- *Travel Time* of the machine head from the home location to the feeder that contain the first component to be placed (feeder 2 in the example),
- *Pick Time*,

- *Travel Time* of the machine head from the first feeder to the location on which the first component is to be placed (location 2 in the example),
- *Insertion Time*,
- *Travel Time* from the first location to the feeder which contains the second component to be placed (feeder 5 in the example),
- *Pick Time*,
- ...
- *Travel Time* from the feeder which contains the last component to be placed (feeder 3 in the example) to the location on which the last component is to be placed (location 1 in the example),
- *Insertion Time*, and
- *Travel Time* from the last location to the home location of the machine head.

It should be noted that this description is only applicable to the case where the machine head has sequential movements (both the board and the feeder are stationary). When other cases are considered, new considerations have to be taken into account. The reader can refer to Egbelu *et al* (1996) for details about these cases as they are not considered in this research.

The objective function is to minimise the placement time. This means, the optimum placement sequence π, amongst a set of possible permutations Π, has to be found for a particular feeder assignment. The number of permutations in the set Π is:

$$\Pi = C_k!$$

where

C_k is the number of components of board type k.

This optimum placement sequence provides the shortest placement time. In the example provided, $\pi = 2,4,3,1$ is one possible placement sequence of 24 $(4! = 24)$ sequences comprising the set Π. The placement time can now be represented by an equation as follows:

$$\begin{aligned} \textit{Pick and Place Time} = {} & \textit{Travel Time}(L_h, F_1) + \textit{Pick Time} + \textit{Travel Time}(F_1, L_1) + \\ & \textit{Insertion Time} + \textit{Travel Time}(L_1, F_2) + \textit{Pick Time} + \textit{Travel Time}(F_2, L_2) + \textit{Insertion} \\ & \textit{Time} + \ldots + \textit{Travel Time}(F_C, L_C) + \textit{Insertion Time} + \textit{Travel Time}(L_C, L_h) \end{aligned}$$

where

L_h is home location of the machine head,

L_1 is insertion location 1 (location of component 2 in the example),

L_2 is insertion location 2 (location of component 4 in the example),

L_C is insertion location C (location of component 1 in the example),

F_1 is feeder contains the 1st component to be placed (feeder No 2 in the example),

F_2 is feeder contains the 2nd component to be placed (feeder No 5 in the example), and

F_C is feeder contains the C^{th} component (last components) to be placed (feeder No 3 in the example).

Let:

PPT be Pick and Place Time,

TT be Travel Time,

PT be Pick Time, and

IT be Insertion Time,

then, the previous equation can be rewritten as:

$$PPT = TT_{(L_h,F_1)} + PT + TT_{(F_1,L_1)} + IT + TT_{(L_1,F_2)} + PT + TT_{(F_2,L_2)} + IT... + TT_{(F_C,L_C)} + IT + TT_{(L_C,L_h)}$$

or:

$$PPT = TT_{(L_h,F_1)} + \sum_{c=0}^{C-1} TT_{(F_{c+1},L_{c+1})} + \sum_{c=1}^{C-1} TT_{(L_c,F_{c+1})} + TT_{(L_C,L_h)} + C(PT + IT) \quad (4.2)$$

The pick and place time required for any sequential pair of components c, $c+1$ from the insertion of c to the insertion of $c+1$ can then be calculated as:

$$PPT_{(c,c+1)} = TT_{(L_c,F_{c+1})} + PT + TT_{(F_{c+1},L_{c+1})} + IT, \qquad \text{for } c = 0,1,2,...,C-1 \quad (4.3)$$

where

$$L_0 = L_h$$

Now, the objective function can be written as:

$$Minimise \sum_{c=0}^{C-1} PPT_{\pi(c,c+1)} \quad (4.4)$$

where

$PPT_{\pi(c,c+1)}$ is subject to equation (4.3)

subject to

$\pi \in \Pi$

$PT > 0$

$IT > 0$

$TT_{(L_c,F_{c+1})} > 0$ for $c = 0,1,2,...,C-1$

$TT_{(F_{c+1},L_{c+1})} > 0$ for $c = 0,1,2,...,C-1$

$TT_{(F_{c+1},L_{c+1})} = constant \ \forall \pi \in \Pi$ for $c = 0,1,2,...,C-1$

Pick Time (PT) and Insertion Time (IT) depend on the type of the pick-and-place machine and it is assumed that they are not affected by any other constraints of interest in this research (e.g. the environment around the machine). However, travel times from feeders to locations ($TT_{(F_{c+1},L_{c+1})}$), which are represented by plain arrows (→) in Figure 4.1, depend on the feeder assignment but not on the placement sequence π, whereas travel times from locations to feeders ($TT_{(L_c,F_{c+1})}$), which are represented by dashed arrows (-->) in Figure 4.1, are affected by both feeder assignment and the placement sequence π. Taking this into consideration and since equation (4.4) is for a particular feeder assignment, the previous formulation has to be amended in order to take into account the effects of different feeder assignment. To solve the objective function with both the placement sequence and the feeder assignment taken into consideration, an optimum feeder assignment σ (in addition to the optimum placement sequence π), amongst a set of possible assignments Σ has to be found so that the placement time is minimised. The number of possible assignments in the set Σ is:

$\Sigma = F!/E!$

where

F is the total number of feeders, and

E is the number of empty feeders.

In the example above the feeder assignment $\sigma = 0,2,1,0,4,0,3$ is one possible feeder assignment of an 840 ($7!/3! = 840$) assignments comprising the set Σ.

Equation (4.4) can now be rewritten as:

$$Minimise \sum_{c=0}^{C-1} PPT_{\pi\sigma(c,c+1)} \tag{4.5}$$

where

$PPT_{\pi\sigma(c,c+1)}$ is subject to equation (4.3)

subject to

$\pi \in \Pi$

$\sigma \in \Sigma$

$PT > 0$

$IT > 0$

$TT_{(L_c,F_{c+1})} > 0$ for $c = 0,1,2,...,C-1$

$TT_{(F_{c+1},L_{c+1})} > 0$ for $c = 0,1,2,...,C-1$

$TT_{(F_{c+1},L_{c+1})} = constant \ \forall \pi \in \Pi \ and \ \sigma = constant$ for $c = 0,1,2,...,C-1$

All the above discussion is for one particular board type k. However, in this research a mixed-model case is considered and there are more than one board type. As mentioned in subsection 2.2.1, different sequences of board types require different set-up times as represented in equation (4.1). Therefore, to have a minimised total processing time (*TPT*) and in addition to what is considered in equation (4.5), an optimum board type sequence φ (in addition to the optimum placement sequence π and the optimal feeder assignment σ) amongst a set of a possible sequences Φ has to be found so that the total processing time is minimised. The number of possible sequences in the set Φ is:

$$\Phi = K!$$

where

K is the number of board types.

Taking this into consideration, the final objective function can be written as:

$$Minimise \sum_{k=1}^{K} TPT_{\varphi(k)} \tag{4.6}$$

where

$TPT = PPT + ST,$

PPT is the Pick and Place Time and is subject to equation (4.5), and

ST is the Set-up Time and is subject to equation (4.1).

subject to

$\varphi \in \Phi$

$PPT > 0$

$ST > 0$

$PPT = constant \ \forall \varphi \in \Phi, \sigma = constant \ and \ \pi = constant$

Travel times can be calculated by dividing the distances between the feeders and the locations by the speed of the machine head. The distances can be calculated using one of the following equations:

$$D_{lf} = |X_f - X_l| + |Y_f - Y_l| \quad (4.7.a)$$

$$D_{lf} = \max\left(|X_f - X_l|, |Y_f - Y_l|\right) \quad (4.7.b)$$

$$D_{lf} = \sqrt{|X_f - X_l|^2 + |Y_f - Y_l|^2} \quad (4.7.c)$$

where

D_{lf} the distance between location l and feeder f,

X_f, X_l the X co-ordinates of feeder f and location l respectively, and

Y_f, Y_l the Y co-ordinates of feeder f and location l respectively.

Equation (4.7.a) is used when the movement of the machine head (gripper) is the sum of two different movements performed successively: the movement of the arm of the machine head on one hand and the movement of the gripper of the machine head on the other hand (Manhattan metric). This is represented in Figure 4.2 by the movement of the machine head from point A to point B first and then from point B to point C. This situation happens when the machine head does not move diagonally. Equation (4.7.b) is used when both movements are performed concurrently (Chebychev metric). This is represented in Figure 4.2 by the movement of the machine head from point B to point C and at the same time the gripper moves from point A to point B. This situation happens when the machine head and the gripper move independently. Equation (4.7.c) is used

when the machine head (gripper) moves in a direct straight line in one movement (Euclidean metric).

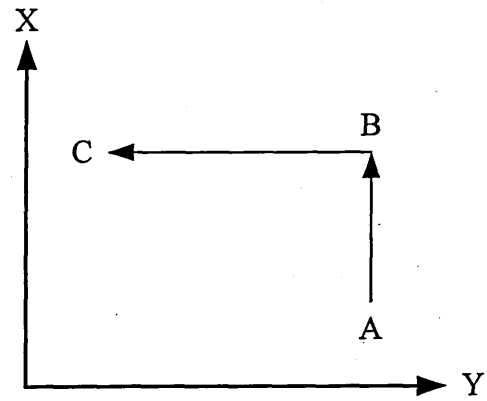


Figure 4.2. Calculating the travel times

4.3. The proposed framework:

The proposed framework for this research is presented in Figure 4.3. As mentioned in subsection 3.4.1, the multi-model manufacturing assembly process of PCBs starts with the order acceptance. Then, it is input into a Master Production Scheduling, which provides the weekly product requirements over 6 to 12 months. The next step includes calculating the Material Requirements Planning (MRP), where the volume and timing of the order (order release and due-date) are determined as a result of the MRP. The process planning system provides the required processes plans and, then, the balancing and sequencing are achieved using information from the optimisation module. The optimisation module is responsible for providing the optimum board type sequence, feeder assignment and component sequencing. Finally, ABC and/or LA are used to evaluate and analyse the cost related issues. The expected results from this step are operational and financial as shown in Figure 4.3. The first part of the proposed framework (Order acceptance, Master Production Scheduling and Material Requirements Planning) is not discussed in this research. The focus will be on the rest of the framework.

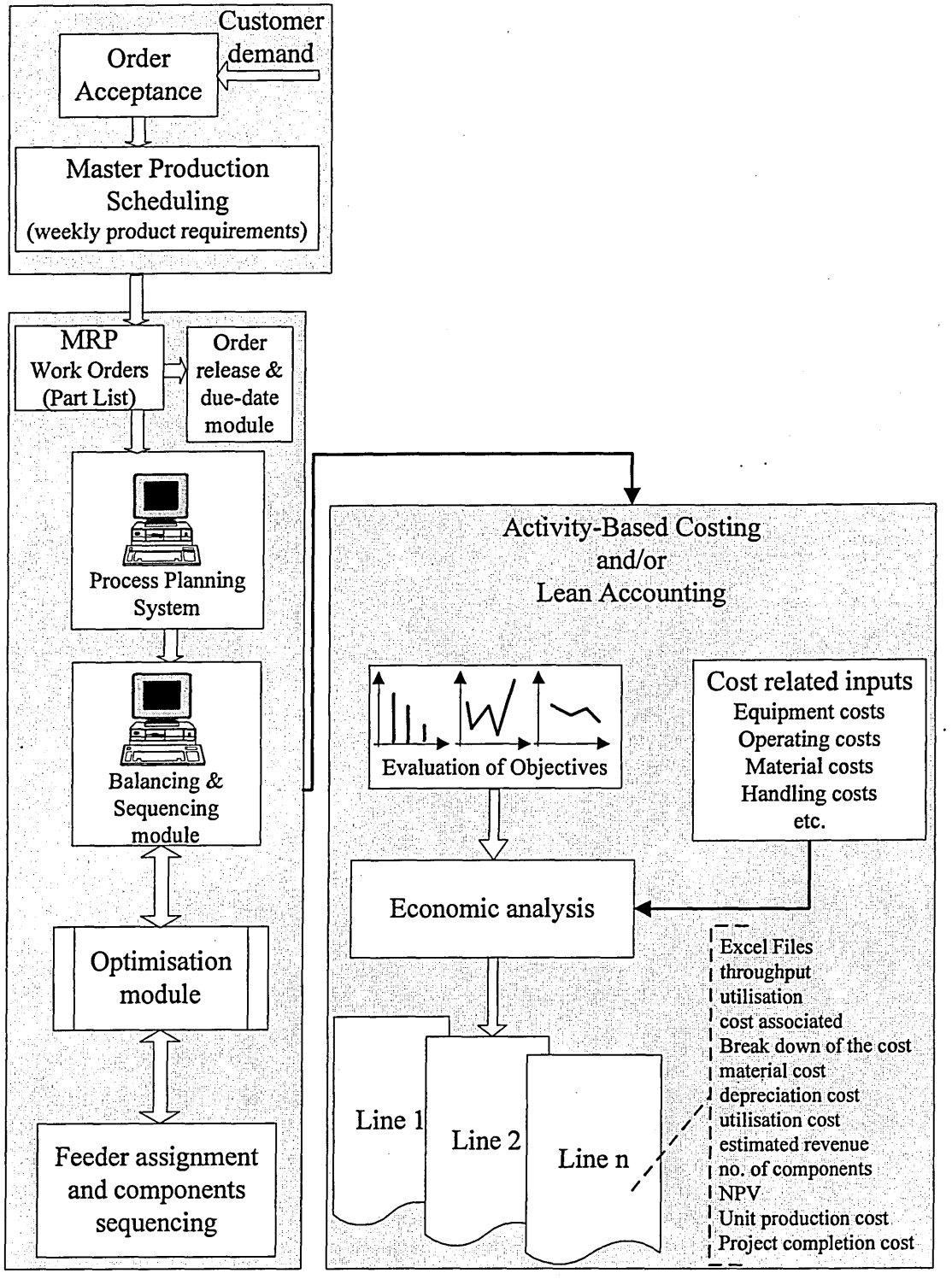


Figure 4.3. Proposed framework of the research

4.4. The proposed algorithm

The proposed algorithm that is used to solve the three problems under consideration is illustrated in Figure 4.4 and outlined as follows:

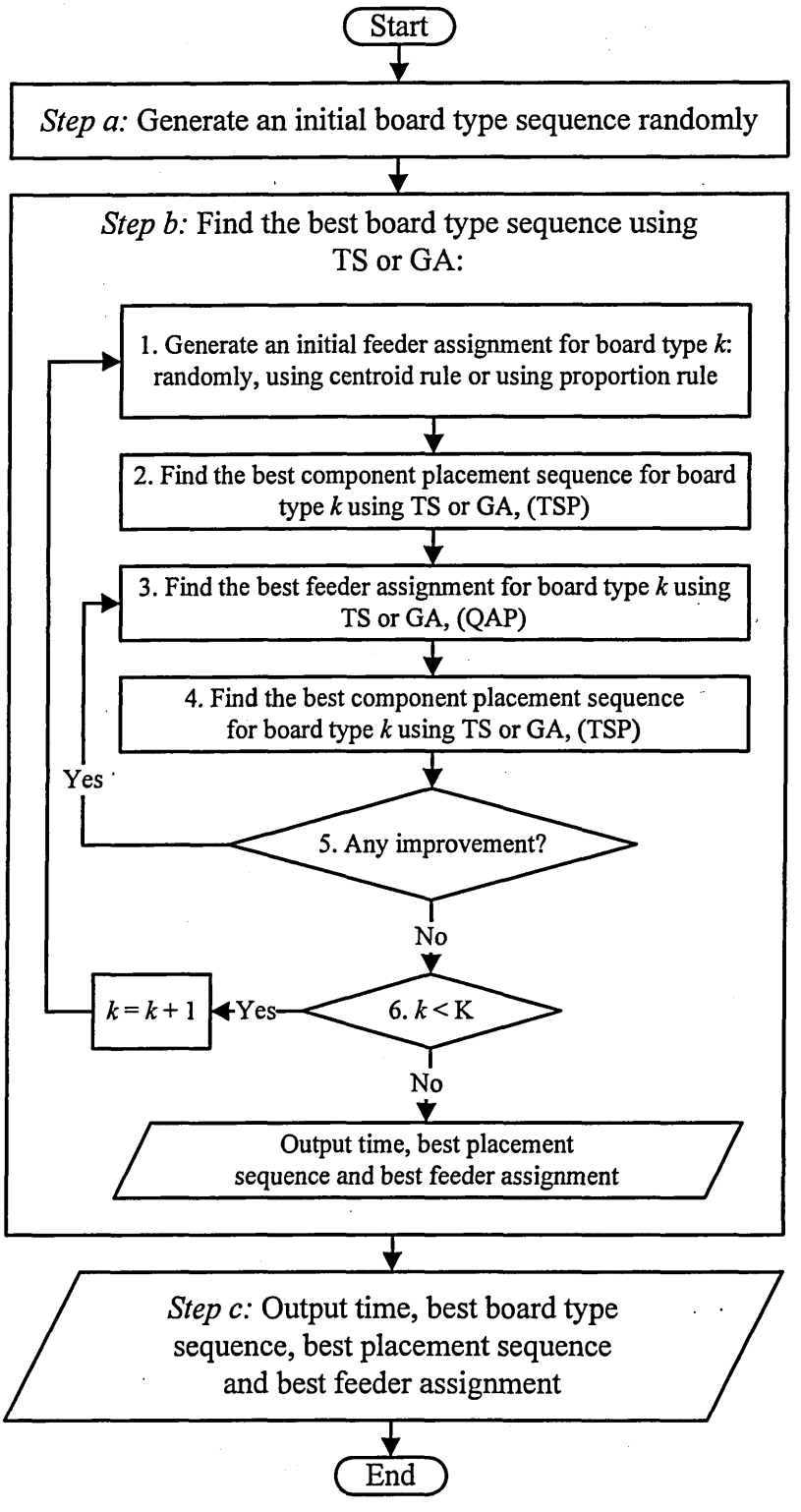


Figure 4.4. Flowchart for the solution algorithm

Step a: start with a randomly-generated board type sequence.

Step b: use the Taboo Search or Genetic Algorithms to find the optimum or near optimum board type sequence as follows:

1. Generate an initial feeder assignment for the first board type in the sequence either randomly, using the centroid rule or using the proportion rule:
    a. The random assignment assigns the component types to feeders by randomly choosing a component type and assigns it to the first feeder and then another component type is randomly chosen and assigned it to the second feeder and so on.
    b. As for the centroid rule method, it requires the centroid location of the components that are of the same type to be calculated as shown in Figure 4.5. For example, in the figure, there are two component types. The first type has four locations and is represented by a circle (o) and the second has three locations and is represented by a diamond (◊). The centroid locations for type o and type ◊ are calculated. They are represented in the figure by a bold circle (o) and a bold diamond (◊) respectively. Now, starting with the type of the most frequency (type o in the example), the component types are assigned to the empty feeders nearest to the centroid locations of the component types. In the example, type o is assigned to feeder number 3 first and then type ◊ is assigned to feeder number 8. In the case where the nearest feeder is occupied, the next nearest is chosen. In the case where there are two empty feeders located at the same distance from the centroid location (one to the left and the other to the right), which one to choose depends on the centroid location of the next type. If the centroid location of the next type is located to the left of the centroid location of the current type then the current type is assigned to the right feeder and vice versa. By doing that it is ensured that the component types are assigned to the feeders so that the total distance between the components of the same type and the feeder that hold this type is minimised.
    c. The proportion rule, as described by Egbelu *et al* (1996), is similar to the centroid rule in calculating the centroid locations for the components of the same types. However, the assignment of component types to the feeders is

different. Here, starting with the component type of highest frequency, the distance D_i is calculated as follows:

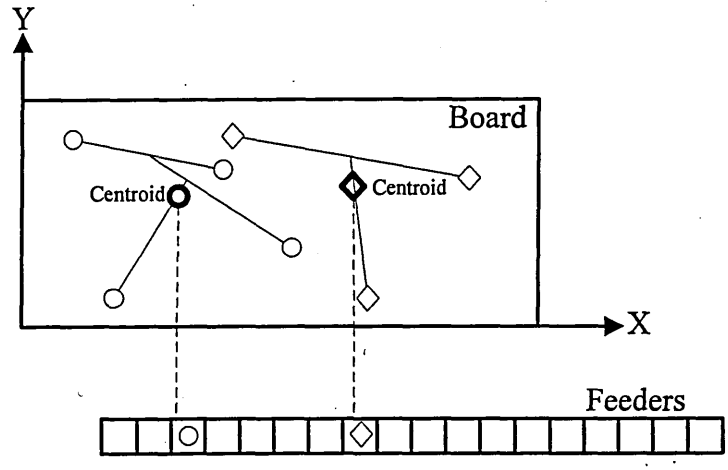


Figure 4.5. Representation of centroid rule

$$D_i = B_i \times L / B$$

where

D_i is the distance between the beginning of the feeders to the feeder to which the component type i is assigned,

B_i is the X-co-ordinate of the centroid location of the component type i,

L is the total length of the feeders, and

B is the length of the board.

After D_i is calculated, the component type i is assigned to the empty feeder nearest to the end of D_i as shown in Figure 4.6. In the example above, type o is assigned to feeder number 6 first and then type ◊ is assigned to feeder number 13. As discussed regarding the centroid rule, in the case where the nearest feeder is occupied, the next nearest feeder is chosen. In the case where there are two empty feeders located at the same distance from the end of D_i, the same method applied to the centroid rule is applied here.

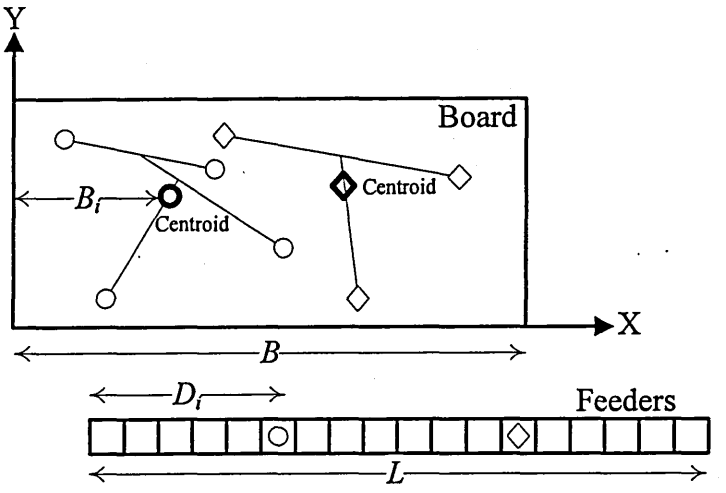


Figure 4.6. Representation of proportion rule

2. The optimum or near optimum components placement sequence for the first board type is determined using TS or GA. As mentioned earlier, this problem is an instance of the travelling salesman problem (TSP).
3. The optimum or near optimum placement sequence from step 2 is used as an input to the feeder assignment problem, which is also solved for the first board type using TS or GA. This problem is an instance of quadratic assignment problem (QAP).
4. A new component placement sequence is generated in this step using TS or GA to take into consideration the new feeder assignment found in step 3.
5. Steps 3 and 4 are repeated successively as long as the terminating condition (when the placement time in step 4 is equal to the placement time in step 3) has not been satisfied.
6. If the number of processed board types k is still smaller than the total number of board types K, steps 1 to 5 are repeated, otherwise the assembly time, optimum or near optimum placement sequence and optimum or near optimum feeder assignment are output.

Step c: the final results are generated including information about assembly time, board type sequence, feeder assignment and placement sequence.

4.5. Taboo Search and Genetic Algorithms

Some background information has been presented in Chapter 3 about TS and GA. However, in this chapter, more detailed and more problem-related information about TS and GA is presented.

4.5.1. The size of TS neighbourhood

As mentioned earlier in section 4.2, the number of possible permutations for the placement sequence problem is $\Pi = C!$, for the feeder assignment problem is $\Sigma = F!/E!$ and for the board type sequence is $\Phi = K!$, which means that the total number of possible permutations is $\Pi \times \Sigma \times \Phi$. However, a much smaller number of permutations will be explored by TS metaheuristic (which is the basic idea of all metaheuristics). This number is equal to the total number of the neighbours of the permutations generated by TS moves. A neighbour of a permutation is another permutation generated by performing a move on the original permutation. The two types of moves considered in this research for the board type sequence, feeder assignment and placement sequence problems are:

- *Swap move*: the two swapped items occupy the positions of each other. For example, the series 1, 2, 3, 4, 5 becomes 1, 4, 3, 2, 5 by swapping the pair (2, 4).
- *Insertion move*: the inserted item occupies the position where it is inserted and the items situated between the old and the new positions of the inserted item are shifted one position to the left if the insertion position is on the right of the old position, and to the right otherwise. For example, the series 1, 2, 3, 4, 5 becomes 1, 3, 4, 2, 5 by inserting 2 in the 4th position.

The size of the neighbourhood for the swap move (Saad & Lassila 2002) and for the insertion move (derived experimentally) can be calculated using the equations in Table 4.2.

Table 4.2. The size of neighbourhood for the swap and the insertion moves

		Move type	
		Swap	Insertion
Size of neighbourhood	Board type & placement sequence	$\frac{C(C-1)}{2}$	$(C-1)^2$
	Feeder assignment	$\frac{F(F-1)}{2} - \sum_{n=1}^{N} \frac{X_n(X_n-1)}{2}$	$(F-1)^2 - \sum_{n=1}^{N}(X_n-1)(F-1)$

C is the number of components or board types
F is the number of feeders

Regarding the equations for the feeder assignment, $N(N \leq F)$ is the number of component types that are going to be assigned to more than one feeder (X_n feeders) each, although this case is not considered in this research but the equation is used to take into consideration the empty feeders which are considered as feeders with dummy components of type 0 assigned to them. The equation which calculates the size of neighbourhood for the feeder assignment when using the insertion move can be applied only under certain conditions. It is applicable when the component types which will be assigned to more than one feeder each are assigned to adjacent feeders only, e.g. 2, 3, 4, 4, 4, 1, 1, 1, 5, 0, 0, 0, 6, where 0 represents an empty feeder.

Figure 4.7 illustrates the size of the neighbourhood when using both the swap and the insertion methods for the board type sequence and placement sequence problems. The X axis represents the number of the board types K or the number of component locations C on the board, whereas the Y axis represents the number of possible neighbours associated with K or C. Figure 4.7 below can also represent both the swap and the insertion methods for the feeder assignment problem but under the condition that the number of feeders is equal to the number of component types. In this case the X axis represents the number of feeders F (or the number of component types I).

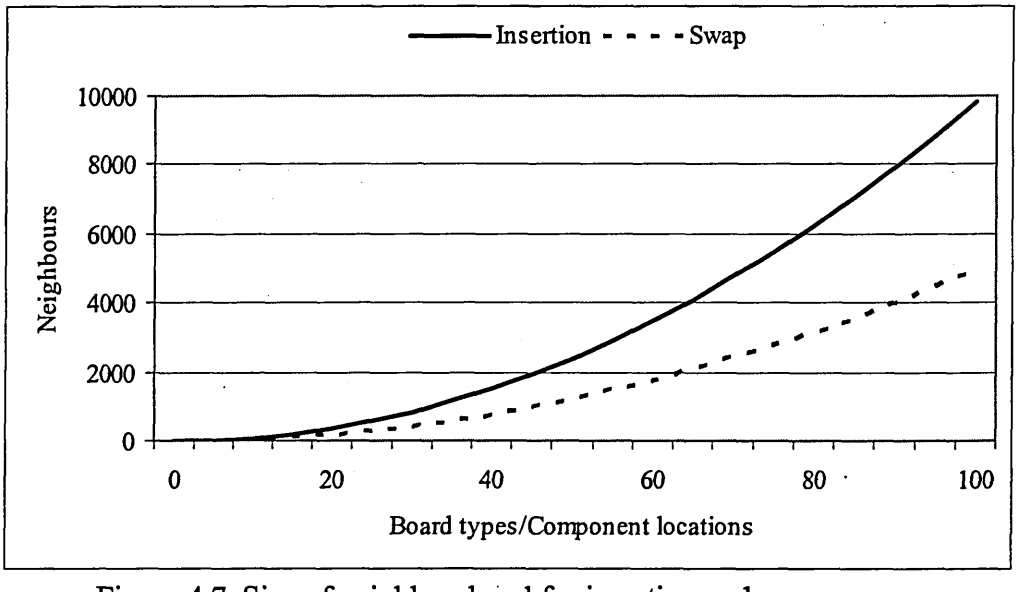


Figure 4.7. Size of neighbourhood for insertion and swap moves

4.5.2. Development of TS algorithm

The TS algorithm used in this research was used previously by Saad and Lassila (2002) where it proved to be successful. This algorithm is modified where necessary to reflect the addition of the insertion move. The algorithm is explained below and its flowchart is shown in Figure 4.8.

1. The algorithm starts with an initial solution (provided randomly or by a previous step in the solution algorithm (section 4.3)).
2. The neighbourhood for that solution is created using the swap or insertion method.
3. Determine the best neighbour in the neighbourhood (the neighbour that provides the shortest processing time) which is selected as the next solution. The move that led to this solution is compared with the taboo list. If it is found and does not lead to a new *best* solution (aspiration criterion), the second best is selected, which in turn will undergo the same treatment until a successful move is chosen.
4. When a move is chosen, the solution associated with it is checked if it is after a *local best* (that is if the previous move was a *local* best). If it is, the move is remembered to be prevented from being chosen later when a restart from this particular *local best* is performed.
5. The solution is checked if it is a new *local best*. If yes, the *local best* is updated, the *non-improvement-moves* counter is set to zero and the restarting is allowed from this point. If not, the *non-improvement-moves* counter is incremented.
6. The solution is compared to the *best* solution found so far. The *best* solution is updated accordingly.
7. Check if the *non-improvement-moves* counter reaches the maximum limit.
   - If yes, the restarting counter is checked to see whether restarting is still allowed.
     a. If yes, the *local best* is used as a restart point and the move that left this point last time is added to the taboo list in order not to follow the same path.
     b. Otherwise, a random solution is used as a restart.
   - Otherwise, make the move and insert it into the taboo list
8. If the predefined maximum number of moves is reached the search stops, otherwise, it cycles back to the second step.

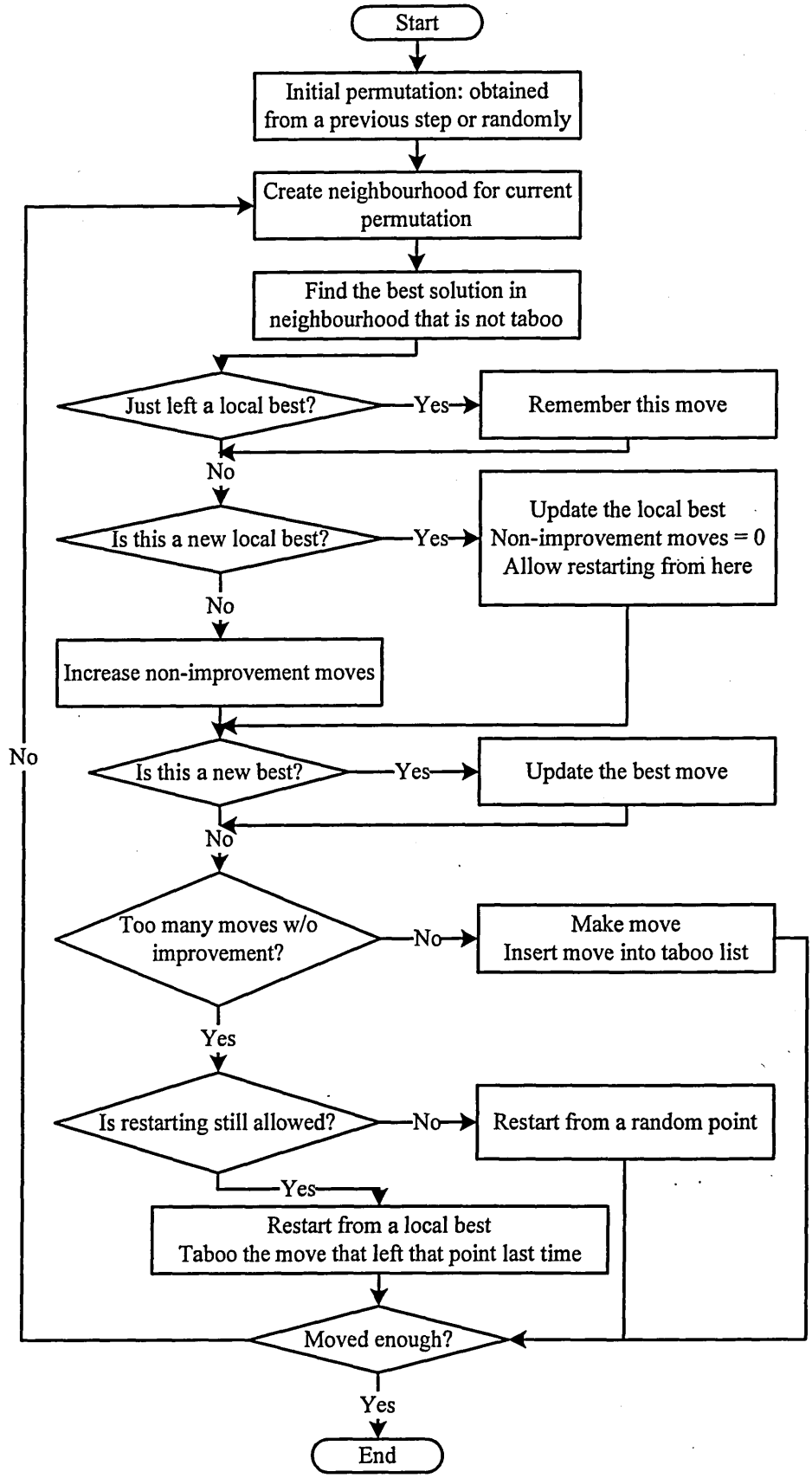


Figure 4.8. Flowchart of Taboo Search algorithm

Since the taboo list has a limited length, it will not be able to prevent loops that have a number of moves larger than the limited length of the taboo list. Therefore, the number of non-improvement moves is stored and used as a criterion to restart the search from the local best solution (intensification) or from a random solution (diversification). The criterion that determines whether to restart from the local best or from a random point is the number of restarts performed from that local best. When this number has exceeded a previously specified number of restarts, the search is restarted from a random point; otherwise, it is restarted from the local best. When the search is restarted from a local best, the move that was performed previously just after this local best has to be prevented from being performed again by adding it to the taboo list. This is performed in order not to follow the same path followed before and, hence, avoiding cycling. Therefore, the first performed move after each local best is stored in order to be added to the taboo list whenever a restart is performed from this local best.

4.5.3. Genetic Algorithms

The GA algorithm used in this research is explained below and its flowchart is represented in Figure 4.9.

1. Initialisation:
   - Create initial population of chromosomes (solutions).
   - Calculate time (the fitness function) for the chromosomes using the objective function.
   - Best solution = the best chromosome in the population.
2. Create children using crossover and mutation and/or inversion.
3. Calculate time (the fitness function) for the children using the objective function.
4. Compare to global best and update as appropriate.
5. Check termination criterion (number of generations) and go to step 2 if not fulfilled.
6. Stop when all generations have been created and output the best solution.

The creation of initial population of chromosomes is achieved randomly or obtained from a previous step in the solution algorithm (section 4.3). The crossover and mutation used here for creating children are the same as explained in subsection 3.3.2, whereas the inversion is performed by rearranging the preserved genes in the child as explained in the following example.

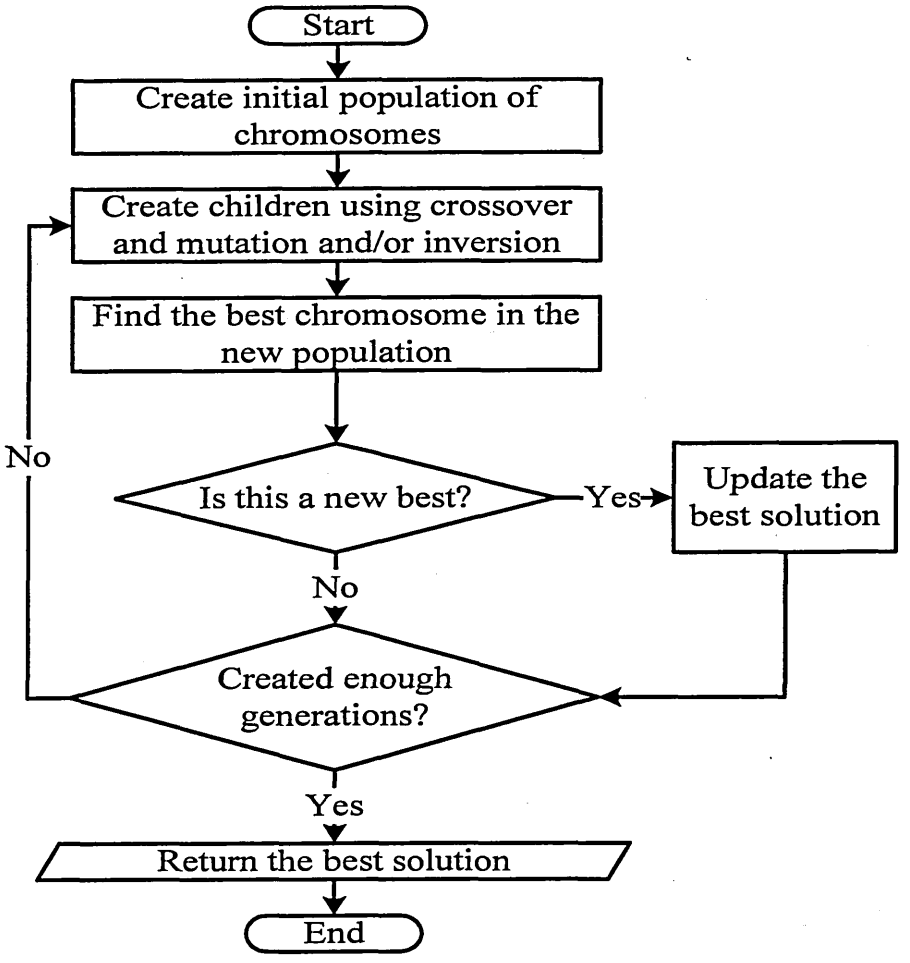


Figure 4.9. Flowchart of Genetic Algorithms

Let's have the following two parents:

Parent 1: *a b c | d e f | g h i j*

Parent 2: *d i j | a h b | g c e f*

The two children created from them (see subsection 3.3.2) are:

Child 1: *i j a | d e f | h b g c*

Child 2: *c d e | a h b | f g i j*

Performing an inversion on Child 1 results in the following:

Child 1: *i j a | f e d | h b g c*

The reason behind performing mutation or inversion is to diversify the population and create more alternative, and potentially better, solutions.

4.6. Case study

In the previous sections of this chapter, the proposed solution algorithm, TS algorithm and GA algorithm are outlined. The next step is to validate these algorithms by applying them to a case study. The case study allows for the algorithms to be tested using different values for the parameters of TS and GA algorithms. This eventually allows for choosing the best values, which can be used when the proposed algorithm is used in a real-life situation. The best values for parameters are chosen depending on the results (total processing time, board type sequence, feeder assignment and component placing sequence) generated by the solution algorithm. The total processing times generated by the algorithm using different values of parameters are compared to each other and the values of parameters that correspond to the shortest assembly time are considered to be the best values. Since it has not been possible to acquire real-life data during the period of this research, hypothetical data are used. The rest of this chapter explains in detail the hypothetical case study used, including data, parameters, tests performed and the results generated.

4.6.1. Case details

A pick-and-place machine is used to assemble eight types of printed circuit boards: A, B, C, D, E, F, G and H. Each board type has a specific number of components (some components are of the same type) to be placed on it. The components of each board type are assigned to feeders located on the machine. Each feeder can hold one component type only and each component type is assigned to one feeder only. Once the first board to be assembled is placed on the machine table, the machine head moves from its home location to the feeder which contains the first component to be placed and picks it up. Then the head moves to the location on the board where the component is to be placed and places it. This process is repeated for each component following a predefined sequence until the board is fully assembled. This process is performed for each board and for each board type according to a predefined sequence. Before each board type is processed on the machine the machine has to be set up for this particular board type. The minimum set-up strategy (only necessary replenishment of components is performed on the feeders) is used here. The head of the pick and place machine moves sequentially and the Euclidean metric is applied. Both the board table and feeders are stationary. Figure 4.10 illustrates a simple representation of the positions of the board and feeders on the machine. In this figure

the board has four components: 1, 2, 3 and 4, the feeder assignment is: 4, 0, 2, 0, 0, 1, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0 and the component placement sequence is: 2, 4, 3 and 1. The movement of the machine head during the assembly process starts from the home location and follows the arrows from 1 to 9 back to the home location.

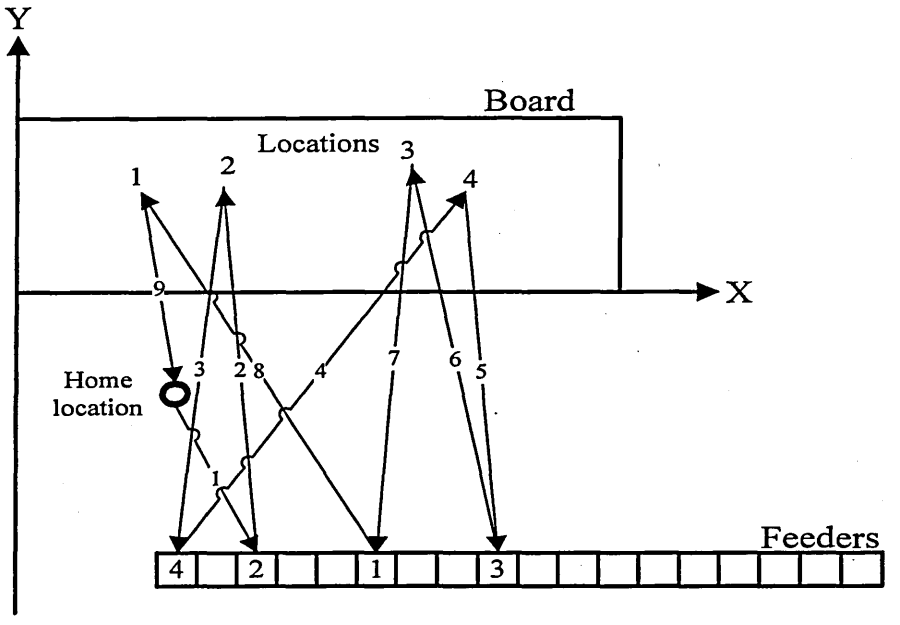


Figure 4.10. A representation of the board and feeders

The machine in this case study has the specifications presented in Table 4.3. The specifications of the boards in terms of the number of component types and the number of component locations on the board are presented in Table 4.4, whereas Table 4.5 contains the specifications of TS and GA algorithms. The number of component types and the X- and Y-coordinates of the locations of each board type can be found in Appendix I. In Table 4.5, the 'maximum number of moves without improvements' is used as a condition to restart the search from a local best (if the maximum 'number of restarts' has not been reached yet) or from a random point (if the maximum 'number of restarts' has been reached).

Table 4.3. Specifications of the machine

Discretion	Value
Pick time	2 seconds
Place time	2 seconds
Average head speed	500 mm/sec
Number of feeders	24
Length of feeder	20 mm
Set-up time (per feeder)	10 sec
X-coordinate of first feeder	X = +200 mm
Y-coordinate of first feeder	Y = −500 mm
X-coordinate of machine head	X_H= +200 mm
Y-coordinate of machine head	Y_H= −200 mm

Table 4.4. Specifications of board types

Board Type	A	B	C	D	E	F	G	H
No. of Component Types	22	18	14	13	10	9	7	5
No. of Locations	45	37	23	22	17	15	20	12

Table 4.5. Specifications of TS and GA algorithms

TS			GA		
	Board sequence	Feeder assign. & place. seq.		Board sequence	Feeder assign. & place. seq.
No. of moves	Up to 40	Up to 100	No. of generations	Up to 20	Up to 160
Taboo-list size	4 - 6	4 - 6	Population size	Up to 96	Up to 160
No. of restarts	3	3	Mutation	Up to 50%	Up to 100%
Max. no. of moves w/o improvements	3	3	Inversion	Up to 40%	Up to 40%

4.6.2. Program code

The solution algorithm is developed into a Windows-based program written in C++ programming language. The full code of the program is presented in Appendix II and a snapshot of the interface is presented in Figure 4.11.

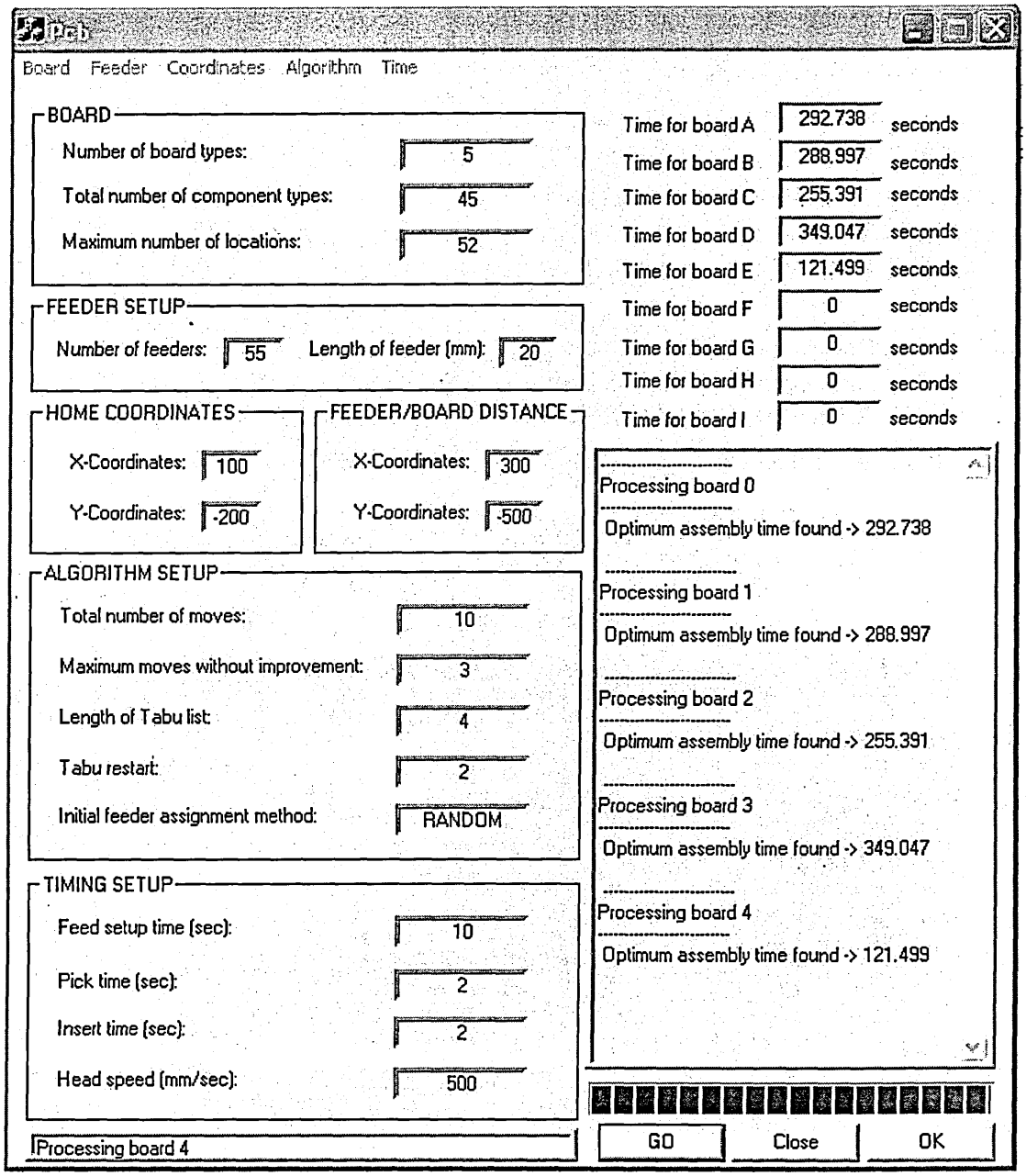


Figure 4.11. Snapshot of the program interface

The program is written in such a way that most of the input and output data can easily be accessed and edited since they are provided in separate text files. These data include:

- Component types: name, number and frequency.
- Component locations: number, X- and Y-coordinates.
- Machine specifications: head speed, home location coordinates, pick time and insertion time.
- Results: processing time, board type sequence, feeder assignment and placement

sequence.

However, the program also has some restrictions:

- Each component type can be assigned to one feeder only.
- Each feeder can hold one component type.
- Each feeder has sufficient capacity to process the whole board.

As mentioned earlier, the initial feeder assignment is generated using three methods: randomly, by applying the centroid rule or by applying the proportion rule. As for the move types, swap and insertion are used.

4.6.3. Experimentation, results and discussion

The algorithms parameters, presented in Table 4.5, are chosen by performing several pilot test runs using different number of moves (10 to 1000) and different taboo list sizes (3 to 12) regarding TS algorithm, and different number of generations (20 to 500) and different population sizes (20 to 300) regarding GA algorithm. Depending on the results of these test runs the parameters of the TS and GA algorithms presented in Table 4.5 are chosen to be used for the actual program runs. Pilot runs helped in identifying the parameters levels that should be included in the experiments. For example, the number of moves considered in the TS algorithm was 10 to 1000; however, when the number of moves was increased between 100 and 1000, the results did not show any changes to the total processing time. The same discussion applies to all the values of parameters considered in the pilot test runs but not in the actual program runs. The program runs are performed on a 3.2 GHz PC. Each run is performed at least four times and the average results are considered. Different combinations of parameters from Table 4.5 are used to study the effect of each parameter.

As mentioned in section 4.2, the set-up time depends on the number of replenished feeders (f_n), which in turn depends on the board type sequence and the feeder assignment. However, the placement time depends on the placement sequence and the feeder assignment (but not on the board type sequence). This means that in order to study the effect of some parameters, it is possible to study their effects on the placement time only (which means one board type only can be used) rather than the total processing time (set-up and placement). For example, when studying the effect of the type of initial feeder assignment (random, using the centroid rule or using the proportion rule) on the processing time, the same results have been achieved when

considering one board type and when considering all board types. Board type A is used when one board type only is considered.

The reminder of this subsection will concentrate on studying the effects of different parameter and their levels on the processing time. This includes the following:

- For TS algorithm: number of moves, methods of initial feeder assignment, taboo list size, type of move, maximum number of moves without improvements and number of restarts.
- For GA algorithm: number of generations, methods of initial feeder assignment, population size and mutation/inversion.

Above all, the effects of the type of algorithm used (TS or GA) will also be studied. The results will be analysed and possible reasons about the behaviour of the algorithms will be given when possible. In addition, a comparison between different parameters will be made and recommendations will be presented.

4.6.3.1. The effect of the number of moves/generations

The increase in the number of moves performed by TS or the number of generations performed by GA should logically increase the improvement in the processing time since it allows for more space to be searched and, potentially, for more good solutions to be found. This is confirmed by the results of the program runs as can be seen in Figure 4.12 and Figure 4.13. Figure 4.12 illustrates the results of running the program when TS algorithm (taboo list = 6), centroid rule and swap move are chosen. The number of moves in this figure refers to the number of moves of the TS algorithm responsible for processing the board type sequence not the TS algorithm responsible for processing feeder assignment and placement sequence (the number of moves for this TS algorithm is fixed to 100). As can be seen from the figure, increasing the number of moves from 10 to 20 led to an increase from 5.82% to 6.06% (a 0.24% increase) in the improvement of the processing time. However, an increase in the number of moves from 20 to 40 resulted in a mere 0.01% increase in the improvement. This means, there is no point of increasing the number of moves more than 20 since it hardly improves the processing time.

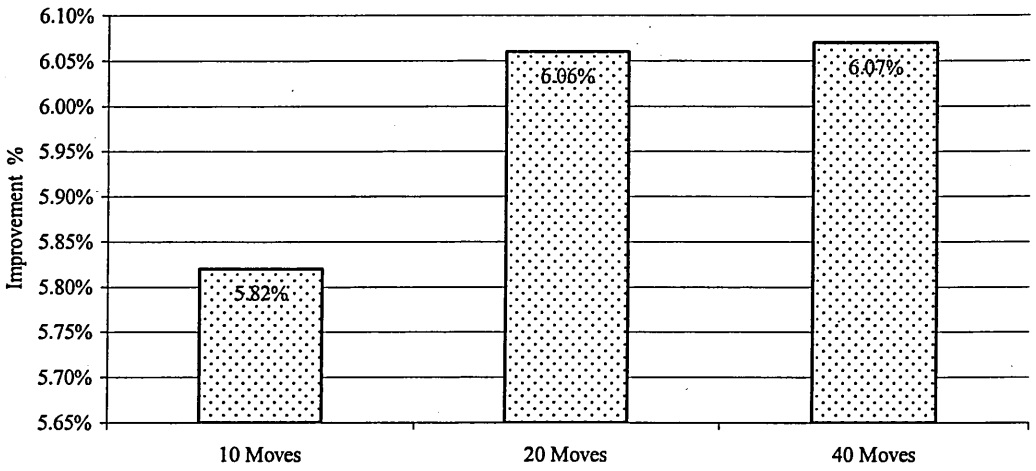


Figure 4.12. The effect of number of moves on processing time

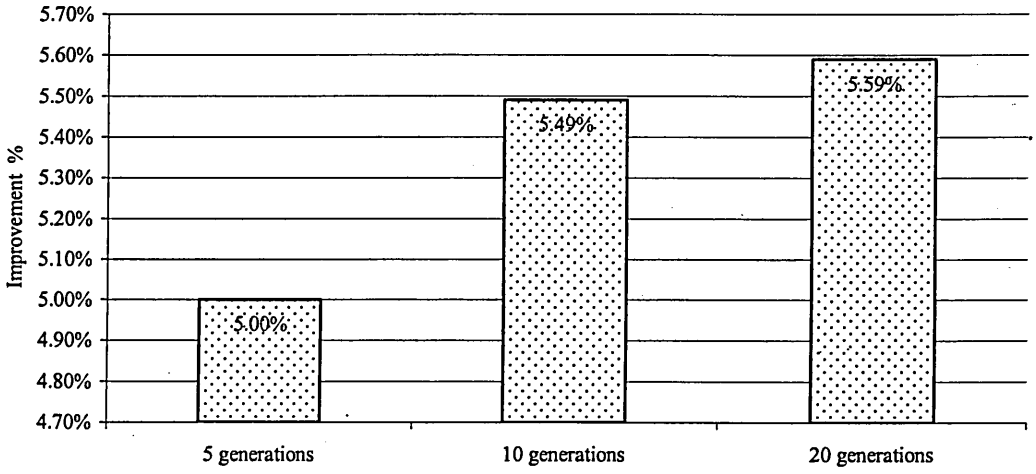


Figure 4.13. The effect of number of generations on processing time

Figure 4.13 illustrates the results of running the program when GA algorithm (population size = 100 and mutation and inversion = 5% each) and proportion rule are chosen. As explained for the TS algorithm, the number of moves here refers to the GA algorithm responsible for processing the board type sequence not the GA algorithm responsible for processing feeder assignment and placement sequence (the number of generation for this GA algorithm is fixed to 20). Figure 4.13 shows the same trend explained for Figure 4.12. When the number of generations is doubled (from 5 to 10), this led to a 0.49% increase in the improvement of the processing time, whereas this increase is limited to 0.1% when the number of generations doubled again (from 10 to

20). This means that increasing the number of moves/generations leads to an improvement in the processing time up to a certain point beyond which the improvement is negligible. This observation is not only applicable to the total processing time of all board types but also to the placement times of individual board types as can be seen in Figure 4.14. It shows how TS and GA algorithms improve (reduce) the processing time of board type A after each move/generation is performed. As the figure clearly shows, the level of improvement is high at first then it slows down towards the end until it reaches a point beyond which there is no improvement. The same observation about board type A is repeated for the rest of the board types considered in the case study. This proves that increasing the number of moves/generations is useful up to a certain limit.

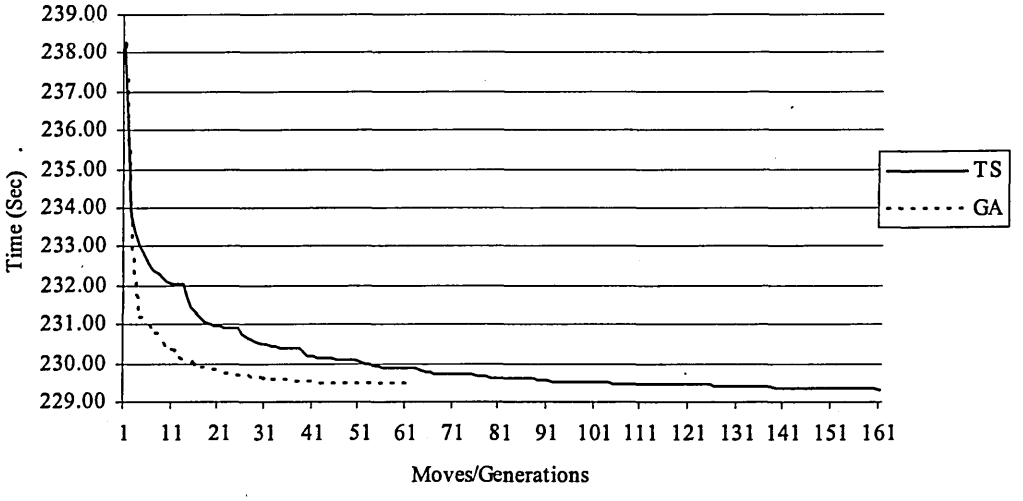


Figure 4.14. The relationship between processing time and number of moves/generations for board type A

4.6.3.2. The effect of the methods used for initial feeder assignment

In the solution algorithm, three different methods for initial feeder assignment are used as explained earlier. The aim of the two non-random methods (centroid and proportion rules) is to start the search from a good initial solution so that the solution algorithm would take less time to find the optimum or near optimum solution. The program is run with random initial feeder assignment first and then using the centroid rule and finally using the proportion rule. The rest of the parameters are left constant

apart from the algorithm type since both TS and GA are used. The results obtained from running the program are presented in Table 4.6.

Table 4.6. The effect of initial feeder assignment on the total processing time

	Improvement in processing time (%)		
	Random	Centroid	Proportion
TS	5.76	5.96	5.90
GA	5.32	5.74	5.36

As can be seen from the table, the improvement in the processing time when the centroid rule is used is the highest (5.96% when TS algorithm is used). The next highest improvement is achieved when the proportion rule is used (5.90% when TS algorithm is used) followed by the random rule (5.76% when TS algorithm is used). The results also show the same trend when GA algorithm is used. The differences in the improvements when different initialisation rules are used are quite small. For example, the percentage of improvements recorded in the case of TS and GA were 0.2% and 0.42% respectively.

The same discussion applies to each individual board type as well as the board types together. Figure 4.15 shows the relationship between the processing time and the number of moves for the three different initial feeder assignments for board type A. It shows how the program reaches the optimum or near optimum solution faster when the centroid rule is used followed by the proportion rule then by the random assignment. Based on the outcome of this case study, starting the search from an improved feeder assignment not only reduces the time required by the program to find the optimum/near optimum solution but it also produces a marginally better solution and the centroid rule has proven to be superior compared to the other two methods.

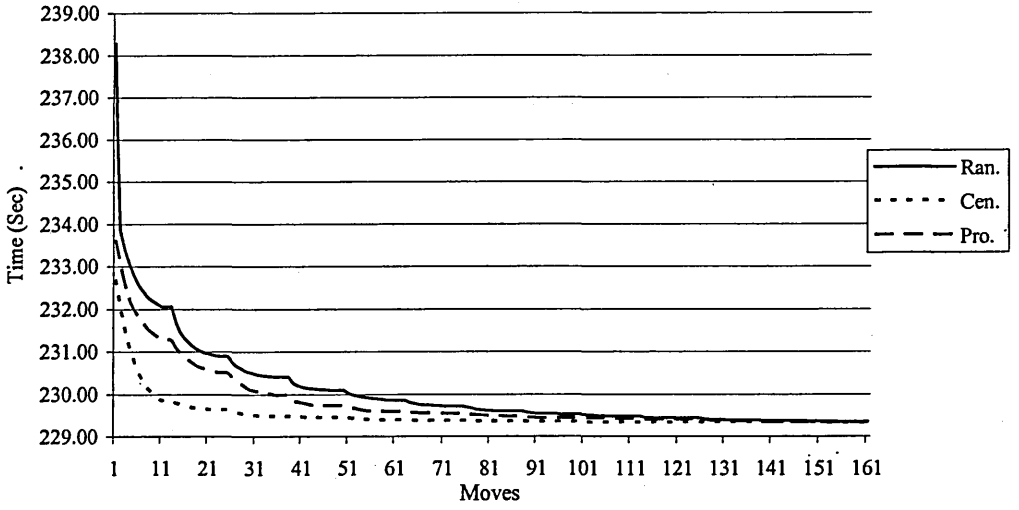


Figure 4.15. The effect of initial feeder assignment on processing time of board type A using TS algorithm

4.6.3.3. The effect of other parameters of TS algorithm

The other parameters of TS algorithm are the type of move (swap and insertion), the taboo list size (4, 5 and 6), the maximum number of moves without improvement (3) and the number of restarts (3). As Table 4.7 shows, the results obtained from running the program using different combinations of these parameters have not shown any notable differences on the total processing time for all board types neither on the placement times of the individual board types.

Table 4.7. The effect of move type and taboo list size on the total processing time

	Improvement in processing time (%)				
	Move type		Taboo list size		
	Swap	Insertion	4	5	6
Total processing time	5.96	5.95	5.94	5.94	5.96
Placement time (A)	3.76	3.75	3.73	3.74	3.76

Although the use of insertion move takes slightly less CPU time to find the optimum or near optimum solution when compared to the swap move as can be seen in Figure 4.16; however, both moves lead to the same placement time.

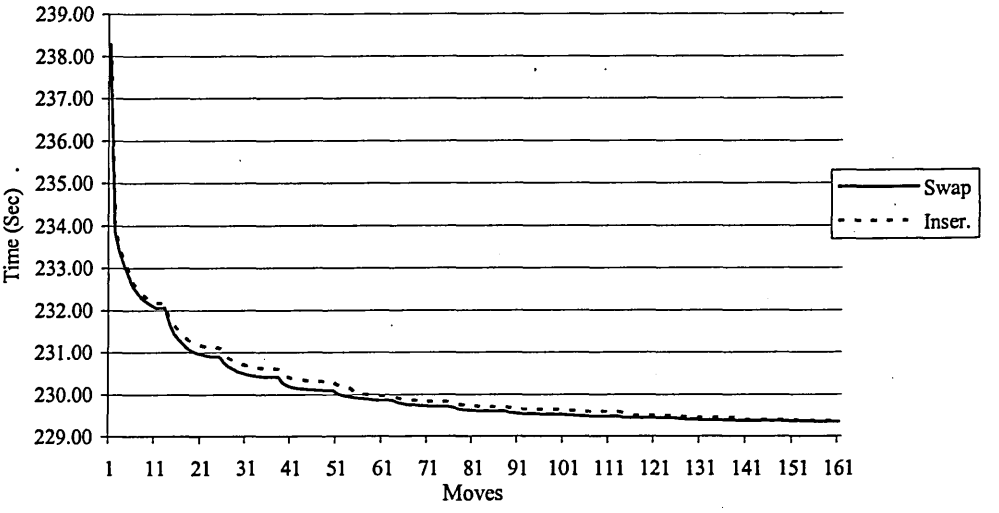


Figure 4.16. The effect of move type on processing time of board type A using TS algorithm (random feeder assignment)

4.6.3.4. The effect of other parameters of GA algorithm

The population size (up to 96) is one of the parameters of GA that affects the processing time. Increasing the population size increases the number of solutions processed by the algorithm, which in turn increases the chance of finding a better solution. This is reflected by the results shown in Figure 4.17. However, as discussed regarding the number of generations, this improvement has a certain limit after which no improvement is achieved.

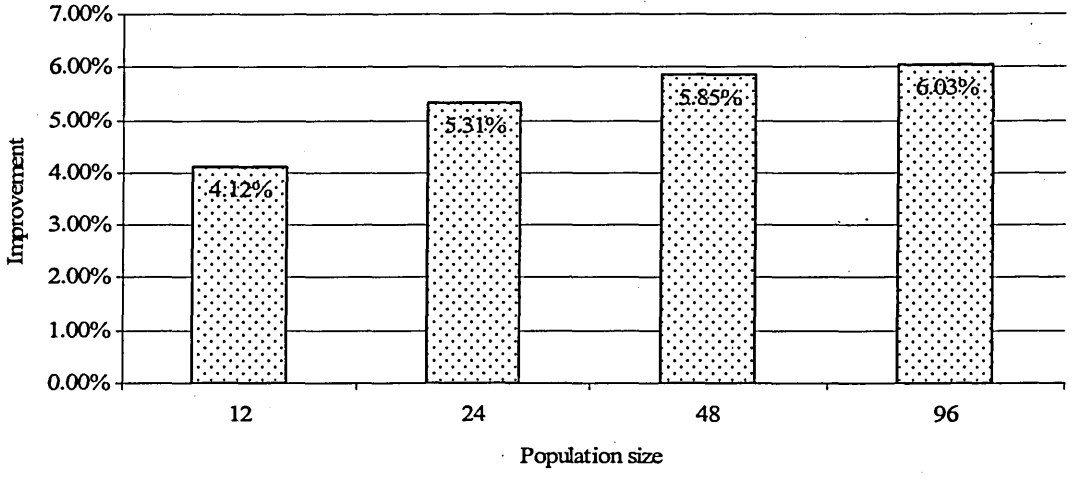


Figure 4.17. The effect of the population size on processing time

As for mutation and insertion, they would slightly affect the processing time. Mutations and insertions are introduced to GA to diversify the solution space by distorting the chromosomes. They are applied to a percentage of chromosomes in the population. The results of the case study show that when the mutation percentage is increased from 0% to 4% the improvement of the processing time increases from 5.02% to 5.83% (i.e. by 0.81%) as shown in Figure 4.18. Any increase of the mutation percentage above that 4% value, results in a fluctuated improvement. Regarding inversion, the results also show the same trend as shown in Figure 4.19, but the effect is less apparent, since an increase in the inversion percentage from 0% to 4% leads to a 0.42% (from 5.03% to 5.45%) increase in the improvement of the processing time.

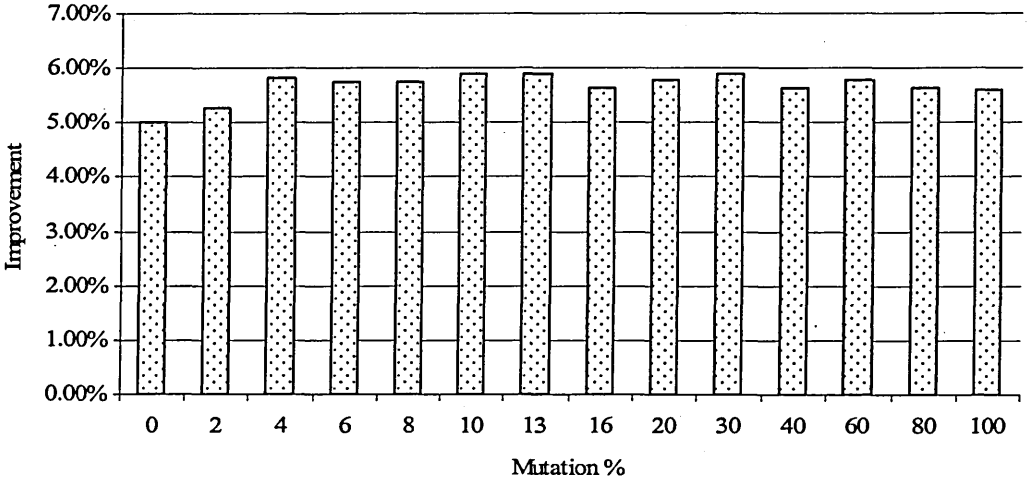


Figure 4.18. The effect of using mutation on the processing time

A possible reason why mutation has a better effect, albeit small, on the processing time compared to inversion could be because of the way each approach distorts the generation of child chromosomes. The change that mutation makes to the preserved part of the child is small compared to the change made by inversion. Note this on the following placement sequence for board type A:

30 2 32 29 9 38 22 21 26 14 23 11| 42 20 39 18 43 6 12 27 25 33 37 24| 41 3 8 4 40 36 16 5 45 7 10 |13 15 31 28 17 19 34 1 35 44

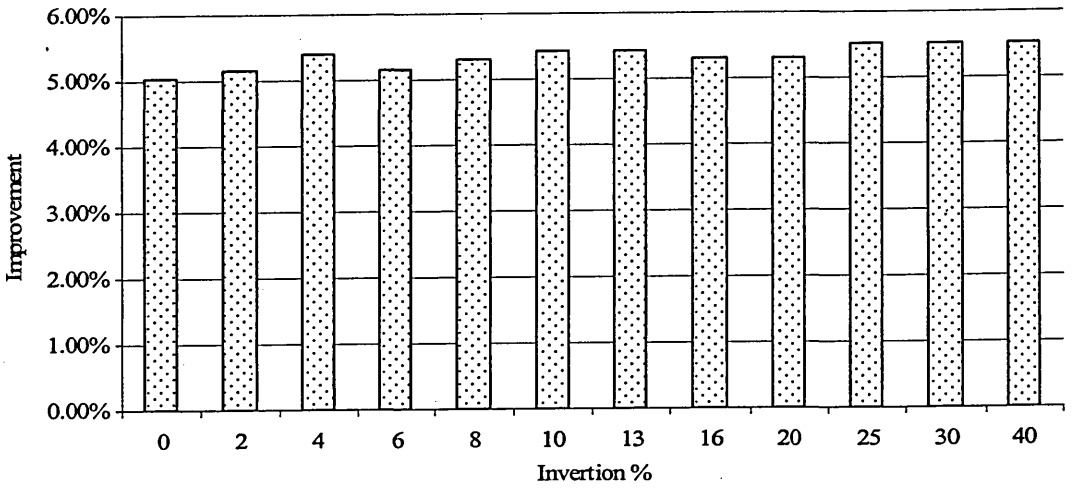


Figure 4.19. The effect of using inversion on the processing time

Let's assume that the second part of this child is the preserved part from the parent. Applying the mutation to this part (18 and 27) leads to one of the following two parts:

Insertion: |42 20 39 43 6 12 27 18 25 33 37 24|

Swap: |42 20 39 27 43 6 12 18 25 33 37 24|

However, applying an inversion to the same part leads to the following result:

|24 37 33 25 27 12 6 43 18 39 20 42|

As can be seen, the inversion has completely distorted the preserved part of the child, whereas the mutation has kept some resemblance. This complete distortion works adversely to the basic idea of GA, which is keeping the good features of chromosomes transferred to next generations. This could be the reason why changing the percentage of inversion leads to a less effect on the processing time compared to when the mutation percentage is changed. Although the aim of using mutation and/or inversion is to distort the child in order to search otherwise unsearched regions in the solution space, this could lead to negative results when the distortion is exaggerated (like when inversion is used).

4.6.3.5. The effect of the algorithm type (TS or GA) used

The results from this case study presented in Figure 4.20 show that the performance of TS algorithm is slightly better than GA. These results represent the average results obtained by running the program several times considering all the

combinations in Table 4.5. TS algorithm checks all the neighbours of a particular solution in every move performed, whereas GA checks a number of solutions equals to the population size. Since the number of neighbours is much higher than the population size in this case, this allows TS to take longer time scouting more solutions in the search space, which gives it a better chance of finding a better solution compared to GA. This is confirmed by the fact that, in general, TS takes around 10% to 20% more CPU time than GA. It should be noted that the two algorithms performed better on the set-up time compared to the placement time as presented in Table 4.8. The reason could be that the set-up time has better margin for improvement compared to the placement time.

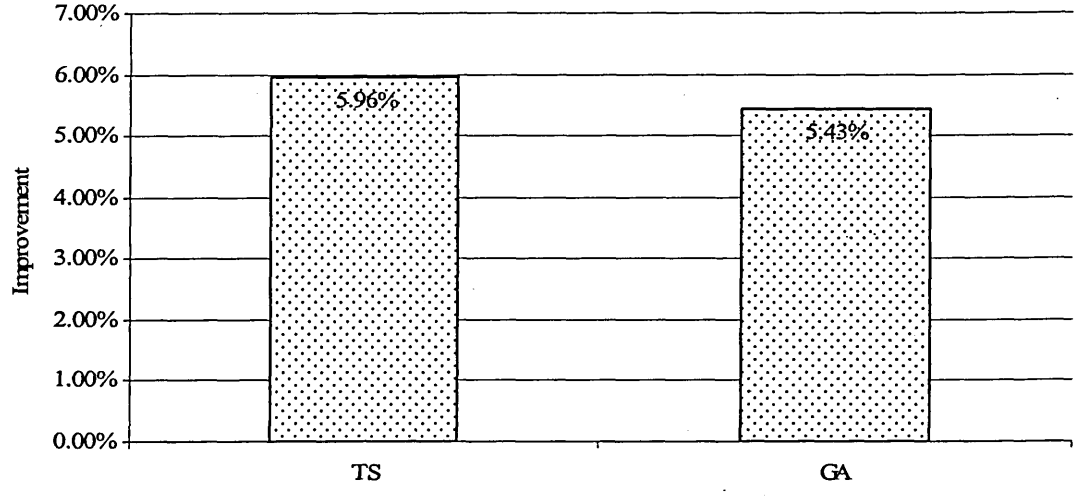


Figure 4.20. The effect of the algorithm type on the processing time

Table 4.8. The effect of algorithm type on the placement/setup times

	Improvement %		
	Placement time	Set-up time	Total processing time
TS	3.76	8.27	5.96
GA	3.74	7.22	5.43

4.7. Summary

In this chapter, the mathematical formulation for the PCB production problems under discussion and the algorithm used to solve these problems were developed. The solution algorithm was based on two metaheuristics, Taboo Search and Genetic

Algorithms, which were developed and explained. A case study was then presented using hypothetical data since real-life data could not be obtained. The results showed an average of 5.96% reduction in total processing time when TS was used and 5.43% when GA was used. The reduction in processing time is calculated by comparing two processing times: the first processing time is calculated using board type sequence, feeder assignment and placement sequence generated by the solution algorithm, and the second processing time is calculated by using random board type sequence, random feeder assignment and random placement sequence. The use of TS proved to be marginally preferable to GA when the goal is to obtain the best total processing time. However, if the case is to obtain a good placement time within short CPU time, the use of GA is preferable. The effects of the number of moves, number of generations, population size, mutation and inversion all follow the same pattern. The processing time is reduced when these parameters are increased up to a certain point after which no reduction is noted. Interestingly, little or no noticeable difference was found in the processing time between runs based on randomly initialised solutions and runs based on the use of centroid rule and proportion rule to generate the initial feeder assignment. However, the use of the centroid rule gave a better initial solution and, hence, it required shorter CPU time. Regarding the effects of other parameters (e.g. move type, taboo list size) on processing time, the results showed no noticeable effects.

CHAPTER FIVE

5. COST ESTIMATION AND ACCOUNTING ASPECTS

5.1. Introduction

Chapter 4 focused on the optimisation of production processes where three PCB production problems were studied and an algorithm was proposed to solve them. In this chapter, which contains the cost estimation and accounting parts of this research, a cost estimation method and an accounting system are considered. The implementation procedures of ABC, as an example of a cost estimation method, and LA, as an example of an accounting system, are explained in detail and applied to the same case study presented in Chapter 4. A comparison between ABC and LA is also presented.

5.2. Basics of ABC

ABC is based on the concept that products consume activities and activities consume resources. Taking this into consideration, the implementation of ABC involves the following steps:

- Identification of activities,
- Identification of activity cost drivers,
- Calculation of cost rates for the cost drivers, and
- Calculation of costs of products by multiplying the cost drivers rates by the volumes of the cost drivers consumed by the product.

In order to understand these steps some of the terms associated with ABC are explained below. In addition, each step will be explained in detail when ABC is implemented on the case study in section 5.4.

- *Activity*:

An activity is a task performed to produce a product. In the ABC model, the true relationship between activities and products is identified as a causal relationship (Bellis-Jones & Develin 1999). According to this cause-and-effect relationship, there are four types of activities:

1. Front-line activities: the activities that have strong relationships with the products through the cost drivers (e.g. purchasing and order processing).

2. Support activities: the activities that have indirect relationships with the products (e.g. training and payroll processing).
3. Sustaining activities: the activities that have little or no relationships with the products (e.g. research and development and market research).
4. Infrastructure activities: these activities have no cost drivers, hence, no relationships with the products. These activities are necessary for the company to stay in the business (e.g. annual audit and the chairperson's lunch).

– *Cost driver*:

A cost driver is a cause that drives the cost. For example, the cost driver of the activity "machine set-up" is the number of set-ups since there is a direct relationship between the number of set-ups performed and the cost of the set-up activity. There are cases where more than one activity is driven by one cost driver. In this case, the activities are called a cost pool.

– *Cost object*:

The cost objects are the products, services, customers, etc. that the cost is assigned to.

5.3. Using ABC for cost estimation in the PCB industry

The use of ABC for estimating the cost of PCB production has been quite rare although other methodologies have been used. For example, Keys *et al* (1986 cited in Giachetti and Arango (2003)), modelled the cost of PCB assembly by taking into account the costs of materials, assembly, test, repair, overhead and maintenance. Boothroyd and Dewhurst (1989 cited in Ong (1995)) developed a methodology to estimate the component assembly cost in PCB manufacturing. Russell (1986 cited in Ong (1995)) developed a cost-estimation methodology for PCBs taking into consideration the cost of delivery, assembly and test.

As for the use of ABC in PCB manufacturing, Ong (1995) developed an ABC-based estimating system in which costs were allocated based on the amounts and types of activities used. In order to determine the costs of activities, Ong used activity charts, worksheets and a cost build-up table. The estimating procedures in his work, Ong argued, would not help the designers to redesign the product but would help them improve the design configurations, choosing the appropriate process and selecting the least-cost design. Spedding and Sun (1999) combined ABC with simulation and applied them on a PCB assembly line. They concluded that the use of a simulation model would

make it easier to implement ABC since without it "the number of combinations and testing variations required by ABC would be extremely time consuming and costly". The authors argued that using simulation and ABC together provided a more powerful tool for providing more useful information for the management. For example, the graphical representation of information provided by the simulation software provided an automatic and powerful tool for the management to analyse the results. Furthermore, the time-based animation could also help identifying some potential problems.

Locascio (2000) developed an ABC method for a PCB assembly line to help designers calculate manufacturing costs from limited design information. This allowed them make trade-offs between materials and manufacturing costs at the design stage and, hence, achieve significant savings in product cots. Giachetti and Arango (2003) argued that the cost models presented so far could not be used for PCB fabrication because it utilised chemical processing steps. Therefore, they developed an ABC model which linked the cost to the design decisions made prior to the layout and routing, in contrast to the model presented by Agrawal and Graves (1999) in which the cost estimation was performed after the layout and routing. The model was used by designers to compare different design alternatives in order to assess the effect of their decisions on the final manufacturing costs of the products.

5.4. Implementation of ABC on the case study

Implementing ABC requires an understanding of the processes on which ABC is to be implemented. In addition to the case study details mentioned in Chapter 4, other necessary details are presented here. The specifications of the board types are shown in Table 5.1. The number of boards (2850 boards) produced within the production period considered (one month) will be calculated later in subsection 5.8.5. Any further data not mentioned in Table 5.1 are presented later when required. The implementation procedures are followed thereafter to calculate the cost of producing each board type. The procedures followed here are similar to that of Cooper and Kaplan (1999) with some modifications to suite the PCB production process.

Table 5.1. Specifications of board types

Board Type	A	B	C	D	E	F	G	H	Total
Number of comp. types	22	18	14	13	10	9	7	5	98
Number of locations	45	37	23	22	17	15	20	12	191
Amount produced	291	296	324	360	354	385	416	424	2850
Number of batches	3	4	4	5	6	7	8	8	45
Number of joints	29	26	22	15	12	12	15	8	139

5.4.1. Determining the cost of indirect resources and their drivers

The indirect resources that can be identified in a facility for manufacturing PCBs are shown in Table 5.2. The resource cost driver rate (RR), which is used to calculate the cost of the cost centres for the indirect resources, can be calculated according to the following equation:

Table 5.2. Indirect resources at the PCB production facility

	Resource	Resource cost driver
Manpower	Administrator	Labour hours
	Secretary/Receptionist	Labour hours
	Human resource	Labour hours
	Security	Labour hours
Build-ings	Rent/Construction cost	Area (m^2)
	Cleaning	Area (m^2)
	Maintenance	Area (m^2)
Utilities	Gas	Area (m^2)
	Electricity	Number of people
	Water	Number of people
	Phone	Number of people
Computing and network	Computer	Number of people
	General Software	Using hours
	Special software package	Using hours
	Network	Using hours
	Printing	Number of people
	Stationery	Number of people
	Copying	Number of projects
	Fax	Number of projects

Resource rate = Total cost (per year or product life) / indirect resource driver spent (5.1)

The 'indirect recourse driver spent' is the amount of driver spent during the production period considered. Since the annual salaries, utility bills, other costs and the amounts of cost drivers spent throughout the production period are presumed to be known, as shown in the third column of Table 5.3, the cost of each resource throughout the production period can easily be calculated. The numbers between parentheses following the names of some of the resources represent the numbers of individuals involved with these resources. For example, considering the Human Resources (HR) indirect resource, there is one employee who works 150 hours per month and receives an annual salary of £20,160. Therefore, within the production period considered the total cost of the HR resource is:

$$£20,160/12 = £1,680$$

Table 5.3. The costs of indirect resources and their drivers

Resource	Cost driver	Total cost (£)	RD*/month	RR**
Administrator (6)	Labour hours	11,320.00	900.00	12.58
Secretary (1)	Labour hours	1,680.00	150.00	11.20
HR (1)	Labour hours	1,680.00	150.00	11.20
Security (1)	Labour hours	1,520.00	150.00	10.13
Rent/Construction	Area (m^2)	20,356.00	580.00	35.10
Cleaning (1)	Area (m^2)	3,540.00	365.00	9.70
Maintenance	Area (m^2)	2,980.00	580.00	5.14
Gas	Area (m^2)	1,120.00	365.00	3.07
Electricity	No. of people	879.00	45	19.53
Water	No. of people	450.00	45	10.00
Phone	No. of people	975.00	45	21.67
Computer	No. of people	980.00	45	21.78
General Software	Using hours	347.00	1190.00	0.29
Special software	Using hours	438.00	460.00	0.95
Network	Using hours	256.00	450.00	0.57
Printing	No. of people	246.00	45	5.47
Stationery	No. of people	356.00	45	7.91
Copying	No. of projects	249.00	1	249.00
Fax	No. of projects	137.00	1	137.00
Total		49,509.00		

* RD: Resource Driver spent.
** RR: Resource Rate (£/unit).

Since the resource driver spent (RD) within this period is 150 hours and using equation (5.1), the resources rates (RR) can be calculated:

$$HR\ resource\ rate = 1,680/150 = £11.20\ per\ hour$$

The resources rates (RR) for the other indirect resources are calculated in the same way and the results are presented in Table 5.3.

5.4.2. Identifying the cost centres and assigning the resources to them

The cost centres include any resources that involve directly in the production process, such as human power, equipments, etc. The cost centres that can be identified for the PCB manufacturing process are presented in Table 5.4. The cost of each cost centre is the sum of the costs of all resources (direct and indirect) consumed in this cost centre throughout the production period.

Table 5.4. Cost centres at the PCB production facility

	Cost centres
Manpower based	Project manager
	Design engineer
	Manufacturing engineer
	Quality assurance engineer
	Technician
	Operator (skilled worker)
	Worker
Machine based	Pick and place machine
	Robotic machine
	Screen printing machine
	Adhesive-application machine
	Soldering machine
	Cleaning machine
	Curing machine
Mixed based	Set-up centre
	Manual placement centre
	Testing centre
	Burning-in centre
	Rework (repair) centre
	Material handling centre
	Inventory centre

As for the costs related to the direct resources, they can be directly added to each cost centre according to how much this cost centre consumes of these direct resources. The direct resources can be divided into two types:

- Type I: the direct resources that are consumed by one cost centre, in which case their costs are added up to form part of the total cost of that cost centre. An example would be the direct resource "electricity" (different from the indirect resource "electricity" mentioned in Table 5.2 and Table 5.3) consumed by "pick-and-place machine" cost centre.
- Type II: the direct resources that are consumed by more than one cost centre. In this case the resources rates (RR) for these direct resources are calculated as explained in subsection 5.4.1. Then, the resource rates are multiplied by the corresponding cost drivers amounts consumed in the cost centres throughout the production period. An example for type II would be the direct resource "engineer", which is consumed by the following cost centres: "project manager", "manufacturing engineer", "quality assurance engineer", etc. The cost of the direct resource "engineer" consumed by the "project manager" cost centre is £244.80 as will be calculated in the example below.

In order to clarify the previous discussion the "project manager" cost centre is used as an example. Regarding the direct resources of type I, there are none. However, there are direct resources of type II, they are: manager, engineer, technician, operator and worker. The resource rate (RR) for each of these direct resources is calculated and then multiplied by the corresponding cost driver amount of each resource consumed by the "project manager" cost centre as shown in Table 5.5. The figures in the second, third and fifth columns of the table are assumed to be known and the data in the fifth column (RD) represent the amounts of resource drivers consumed by the "project manager" cost centre.

Table 5.5. The costs of direct resources (type II) for the "project manager"

Resources	Total cost	RD/month	RR(£/hr)	RD (hrs)	RR×RD (£)
Manager (1)	2,480.00	150.00	16.53	90	1,848.00
Engineer (3)	6,480.00	450.00	14.40	17	244.80
Technician (2)	4,320.00	300.00	14.40	8	115.20
Operator (20)	30,400.00	3000.00	10.13	10	101.33
Worker (10)	15,200.00	1500.00	10.13	5	50.67
Total	£58,880.00				2,000.00

As for the costs related to the indirect resources, they can be calculated by allocating the indirect resources identified in the previous subsection 5.4.1 to the cost centres using the cost drivers and the resource rates of these indirect resources. For example, for the project manager cost centre, the costs related to the manpower indirect resources (administrator, secretary, HR, etc.) can be allocated based on the number of hours the project manager consumes dealing with these indirect resources. The number of hours for each indirect resource is then multiplied by the resource rate (RR) of that particular indirect resource to give the total cost related to that indirect resource. The total costs of the indirect resources involved are, then, added up to give the total cost of indirect resources involved in the project manager cost centre. The indirect resources can also be divided into two types:

- Type A: the indirect resources that are consumed by more than one cost centre. For example, the indirect resource "administrator" consumed by the following cost centres: "project manager", "manufacturing engineer", "quality assurance engineer", etc. In this case, the costs of their consumption by the cost centres can be calculated the same way as type II of the direct resources (i.e. through calculating RR).
- Type B: the indirect resources that are consumed by more than one cost centre via other indirect resources. For example, the indirect resource "rent/construction" consumed by the indirect resource "administrator", which in turn consumed by other cost centres as mentioned in type A above. In this case, the indirect resources that are consumed by the cost centres, "administrator" in the example, are considered as pseudo-cost centres, in which case, the costs of their consumption of the indirect (or even direct) resources, "rent/construction" in the example, are calculated the same way as type A of indirect resources (i.e. through calculating RR). Once the costs of these pseudo-cost centres are calculated, they are allocated to the "real" cost centres proportionately (i.e. according to the same proportion the "real" cost centres consume the pseudo-cost centres).

Applying this to the "project manager" cost centre example, the cost of the indirect resources can be calculated as follows. The indirect resources of type A that are consumed by the "project manager" cost centre are:

- Manpower: administrator, secretary, human resources and security.
- Buildings: rent/construction cost, cleaning and maintenance.
- Utilities: gas, electricity, water and phone.

- Computing and network: computer, general software, network, printing, stationary, copying and fax.

Here, the resource rate (RR) is already calculated for each of these indirect resources as shown in Table 5.3. These resource rates are multiplied by the corresponding amounts of the cost drivers of the indirect resources consumed by the "project manager" cost centre. The results are presented in Table 5.6. The figures in the fourth column of the table represent the amounts of resources drivers consumed by the "project manager" cost centre and assumed to be known data.

Table 5.6. The costs of indirect resources (type A) for the "project manager"

	Resource	RR(£/unit)	RD (unit)	RR×RD (£)
Manpower	Administrator (6)	12.58	80.00	1,006.22
	Secretary (1)	11.20	23.00	257.60
	HR (1)	11.20	5.00	56.00
	Security (1)	10.13	4.00	40.53
Buildings	Rent/Construction	35.10	30.00	1,052.90
	Cleaning (1)	9.70	30.00	290.96
	Maintenance	5.14	30.00	154.14
Utilities	Gas	3.07	30.00	92.05
	Electricity	19.53	1.00	19.53
	Water	10.00	1.00	10.00
	Phone	21.67	1.00	21.67
Computer/ network	Computer	21.78	1.00	21.78
	General Software	0.29	50.00	14.58
	Special software	0.95	0.00	0.00
	Network	0.57	30.00	17.07
	Printing	5.47	1.00	5.47
	Stationery	7.91	1.00	7.91
	Copying	249.00	0.10	24.90
	Fax	137.00	0.10	13.70
Total				3,107.01

Regarding type B of the indirect resources, their costs that are consumed by the "project manager" cost centre are calculated as follows. The pseudo-cost centres in this case are: administrator, secretary, human resources and security. The total cost of the "administrator" pseudo-cost centres is calculated as any real cost centre as shown in Table 5.7. Again, the figures in the fourth column of the table are the amounts of the resources drivers consumed by the "administrator" pseudo-cost centre and are assumed

to be known data. The same calculations are performed on the other pseudo-cost centres and the results are presented in Table 5.8.

Table 5.7. The total cost of the "administrator" pseudo-cost centre

	Resource	RR(£/unit)	RD (unit)	RR×RD (£)
Direct resources	Manager (1)	16.53	7.00	115.73
	Engineer (3)	14.40	5.00	72.00
	Technician (2)	14.40	14.00	201.60
	Operator (20)	10.13	60.00	608.00
	Worker (10)	10.13	20.00	202.67
Indirect resources	Administrator (6)	12.58	283.00	3,559.51
	Secretary (1)	11.20	8.00	89.60
	HR (1)	11.20	10.00	112.00
	Security (1)	10.13	20.00	202.67
	Rent/Construction	35.10	75.00	2,632.24
	Cleaning (1)	9.70	75.00	727.40
	Maintenance	5.14	75.00	385.34
	Gas	3.07	75.00	230.14
	Electricity	19.53	6.00	117.20
	Water	10.00	6.00	60.00
	Phone	21.67	6.00	130.00
	Computer	21.78	6.00	130.67
	General Software	0.29	700.00	204.12
	Special software	0.95	200.00	190.43
	Network	0.57	150.00	85.33
	Printing	5.47	6.00	32.80
	Stationery	7.91	6.00	47.47
	Copying	249.00	0.50	124.50
	Fax	137.00	0.50	68.50
Total				10,329.92

Table 5.8. The total costs of the pseudo-cost centres

Pseudo-cost centre	Total cost (£)
Administrator	10,329.92
Secretary	2,321.01
Human Resources	2,162.73
Security	2,230.58

The total costs of the pseudo-cost centres are allocated to the "real" cost centres that consume them. In this case they are: "project manager", "design engineer",

"manufacturing engineer", "quality assurance engineer", "technician", "operator" and "worker". The allocation process depends on the proportion at which each cost centre consumes of the indirect resources (pseudo-cost centres). Table 5.9 shows the amounts of the resource drivers (in hours) of the pseudo-cost centres consumed by the cost centres. These figures are assumed to be known data.

Table 5.9. The amounts of cost drivers of pseudo-cost centres spent

	Administrator	Secretary	H. Resources	Security
Project manager	80	23	5	4
Design eng.	50	8	7	4
Manufacturing eng.	50	8	7	4
Quality assurance eng.	50	4	7	4
Technician	80	10	15	8
Operator	130	18	21	62
Worker	80	12	15	32
Total	520	83	77	118

Taking into account the figures in Table 5.9, the costs of type B of the indirect resources that are consumed by the "project manager" cost centre can be calculated as:

$$10,329.92\times(80/520)+2,321.01\times(23/83)+2,162.73\times(5/77)+2,230.58\times(4/118)=£2,448.44$$

Now, the total cost of the "project manager" cost centre is the sum of the costs of the direct resources (types I and II) and indirect resources (types A and B) consumed by the "project manager" cost centre:

$$0.00+2,000.00+3,107.01+2,448.44=£7,555.45$$

When the total cost for each cost centre is calculated, a cost driver is identified for each cost centre and the cost centre rate (CCR) is calculated according to the following equation:

$$\textit{Cost centre rate} = \textit{total cost of cost centre} / \textit{cost centre driver spent} \qquad (5.2)$$

The 'cost centre driver spent' is the amount of driver spent during the production period considered. The final results for all cost centres are presented in Table 5.10. The fourth column of the table represents the cost centres drivers (CCD) amounts spent during the production period and are assumed to be known data.

Table 5.10. The costs of cost centres, their cost drivers and their rates

Cost centre	Total cost	Cost driver	CCD*/month	CCR**
Project manager	£7,555.45	Working hours	150	50.370
Design engineer	£6,418.04	Working hours	150	42.787
Manufacturing eng.	£6,279.65	Working hours	150	41.864
Quality ass. eng.	£4,987.50	Working hours	150	33.250
Technician	£6,540.34	Working hours	300	21.801
Operator	£15,295.05	Working hours	3000	5.098
Worker	£9,000.16	Working hours	1500	6.000
Set-up centre	£2,280.85	No. of set-ups	45	50.686
Pick-&-place machine	£4,668.01	Mach. hours	155	30.116
Robotic machine	£3,133.01	Mach. hours	155	20.213
Screen print. machine	£4,631.01	Mach. hours	155	29.877
Adhesive machine	£4,631.01	Mach. hours	155	29.877
Soldering machine	£4,638.01	Mach. hours	155	29.923
Manual place. Centre	£1,688.68	No. of units	2850	0.593
Cleaning machine	£2,988.01	Mach. hours	155	19.277
Curing machine	£2,977.01	Mach. hours	155	19.207
Testing centre	£3,991.52	No. of units	3420	1.167
Burning-in centre	£5,194.02	No. of units	2850	1.822
Rework centre	£3,072.18	Faulty units	570	5.390
Material handl. centre	£4,982.56	Distance	7985	0.624
Inventory centre	£5,094.92	No of part types	98	51.989
Total	£110,047.00			

* CCD: Cost Centre Driver spent

** CCR: Cost Centre Rate (£/unit)

5.4.3. Identifying activities, calculating their costs and the rates of their cost drivers

In this step, the activities that are used in the PCB manufacturing process are identified. The total cost of each activity is calculated depending on the cost centres involved in this activity. This is achieved by multiplying the cost centre rate (CCR), calculated in the previous subsection (5.4.2), by the corresponding cost centre amount consumed in the activity. Each activity is then assigned a cost driver, which is the factor that explains how the activity consumes cost. For example, the cost driver for the machine set-up activity is the number of setups performed. The final part of this step is to calculate the activity cost driver rate (ACDR), which can be calculated according to this equation:

$$\textit{Activity cost driver rate} = \textit{cost of activity} / \textit{activity cost driver spent} \qquad (5.3)$$

The activity cost driver rates are used to calculate the cost of the PCB as will be explained in the next step.

The activities that can be identified in the PCB manufacturing process are obtained from Ong (1995) and presented in Table 5.11. As for facility level activities such as sustaining activities (e.g. research and development, market research, etc.), support activities (e.g. recruitment, training, management, etc.) and infrastructure activities (e.g. annual audit, producing year-end statutory accounts, etc.), the cost of these activities are arbitrarily allocated to the products (Ong 1995) or dealt with depending on different basis (Bellis-Jones & Develin 1999).

Table 5.11. The activities that can be identified in the production of PCBs

	Activities	Cost driver	Cost centres
Unit level	Sequencing parts	No. of parts sequenced	Project manager, Design engineer, Manufacturing engineer, Quality assurance engineer, Operator
	Loading & unloading	No. of times handled	Project manager, Manufacturing engineer, Technician, Operator, Worker
	Screen printing	No. of prints	Project manager, Manufacturing engineer, Quality assurance engineer, Technician, Operator, Worker, Screen printing machine
	Applying adhesive	No. of applications	Project manager, Manufacturing engineer, Quality assurance engineer, Technician, Operator, Worker, Adhesive application machine
	Placing components	No. of parts	Project manager, Manufacturing engineer, Quality assurance engineer, Technician, Operator, Worker, Pick-&-place machine, Robotic machine, Manual placement centre
	Soldering	No. of panels	Project manager, Manufacturing engineer, Quality assurance engineer, Technician, Operator, Worker, Soldering machine
	Cleaning	No. of panels	Project manager, Manufacturing engineer, Quality assurance engineer, Technician, Operator, Worker, Cleaning machine
	Curing and baking	No. of panels	Project manager, Manufacturing engineer, Quality assurance engineer, Technician, Operator, Worker, Curing machine
	Testing	No. of parts	Project manager, Manufacturing engineer, Quality assurance engineer, Technician, Operator, Worker, Testing centre
	Burning-in	No. of panels	Project manager, Manufacturing engineer, Quality assurance engineer, Technician, Operator, Worker, Burning-in centre
	Rework (repair)	No. of parts	Project manager, Manufacturing engineer, Quality assurance engineer, Technician, Operator, Worker, Rework centre
	Visual & touch-up	No. of joints	Project manager, Manufacturing engineer, Technician, Operator, Worker
	Kitting/other operations	No. of operations	Project manager, Manufacturing engineer, Quality assurance engineer, Operator, Worker
Batch level	Purchase order	No. of orders	Project manager, Manufacturing engineer
	Acceptance sampling	Sampling size	Project manager, Manufacturing engineer, Quality assurance engineer, Operator
	Inventory retrieval	No. of part types	Project manager, Manufacturing engineer, Worker
	Material handling	Distance	Project manager, Manufacturing engineer, Technician, Worker, Material handling centre
	Setting-up	No. of machines/set-ups	Project manager, Manufacturing engineer, Quality assurance engineer, Operator, Worker, Setting-up Centre
Product level	Inventory holding	No. of part types	Project manager, Manufacturing engineer, Worker, Inventory centre
	Designing	Time of design	Project manager, Design engineer, Manufacturing engineer, Quality assurance engineer
	Place-&-route design	Time of design	Project manager, Design engineer, Manufacturing engineer, Quality assurance engineer
	Programs & fixtures	No. of programs/fixtures	Project manager, Design engineer, Manufacturing engineer, Quality assurance engineer, Technician, Operator, Worker

The cost of each activity is the total cost of all cost centres, each according to the rate of its contribution, used in this activity. For example, the cost centres that are used in the "sequencing parts" activity (as presented in Table 5.11) are: "project manager", "design engineer", "manufacturing engineer", "quality assurance engineer" and "operator". The cost of each activity can be calculated by multiplying the cost centre rate (CCR), obtained from Table 5.10, for each cost centre involved in the activity by the amount of that cost centre driver (CCD) consumed in the activity throughout the production period. As an example, the cost of "sequencing parts" activity is calculated as shown in Table 5.12. The data in the third column of the table (CCD) are assumed to be known.

The costs of all other activities are calculated using the same way and the activity cost driver rate (ACDR) is calculated according to equation (5.3); the results are presented in Table 5.13. The amounts of the activity cost drivers spent during the production period (fourth column of the table) are assumed to be known data. For the sake of better representation the results in Table 5.13 are depicted graphically as shown in Figure 5.1. As can be seen, the most costly activities are "placing components", "programs and fixtures", "testing" and "rework". By reducing the cost of the activities, especially the most costly ones, the cost of PCB production is reduced. For example, it is possible to reduce the cost of the "placing components" activity by optimising the pick-and-place machine, which was the main subject in Chapter 4. The cost reduction is achieved by reducing the production time through optimising the component placement sequence and the feeder assignment. Another example would be to reduce the cost of "rework" activity by improving the quality, which reduces the amount of reworked units and, hence, the cost. The actual cost reduction attributed to the optimisation work performed on the placement machine will be calculated and discussed later in section 5.5.

Table 5.12. Calculating the cost of "sequencing parts" activity

Cost centre involved	CCR	CCD (Hour)	CCR×CCD (£)
Project manager	50.370	2.00	100.74
Design engineer	42.787	5.00	213.93
Manufacturing engineer	41.864	4.00	167.46
Quality assurance eng.	33.250	4.00	133.00
Operator (skilled worker)	5.098	100.00	509.84
Total			1,124.97

Table 5.13. The costs of activities, their cost drivers and their rates

Activities	Activities costs	Cost driver	ACD*	ACDR**
Sequencing parts	£1,124.97	No. of parts sequenced	64620	0.0174
Loading & unloading	£2,267.95	No. of times handled	19950	0.1137
Screen printing	£5,315.54	No. of prints	2850	1.8651
Applying adhesive	£5,315.54	No. of applications	2850	1.8651
Placing components	£15,912.06	No. of parts	64620	0.2462
Soldering	£5,702.86	No. of panels	2850	2.0010
Cleaning	£3,708.41	No. of panels	2850	1.3012
Curing and baking	£3,712.52	No. of panels	2850	1.3026
Testing	£10,307.76	No. of parts	3420	3.0140
Burning-in	£5,878.55	No. of panels	2850	2.0626
Rework (repair)	£6,363.11	No. of parts	570	11.163
Visual & touch-up	£4,600.99	No. of joints	47163	0.0976
Kitting/other operations	£3,425.90	No. of operations	2850	1.2021
Purchase order	£1,073.45	No. of orders	49	21.907
Acceptance sampling	£1,754.38	Sampling size	68	25.800
Inventory retrieval	£682.61	No. of part types	98	6.9654
Material handling	£5,783.81	Distance	7985	0.7243
Setting-up	£2,795.75	No. of machines/set-ups	45	62.128
Inventory holding	£5,829.03	No. of part types	98	59.480
Designing	£3,225.91	Time of design	60	53.765
Place-&-route design	£2,188.78	Time of design	40	54.720
Programs & fixtures	£13,077.15	No. of programs/ fixtures	45	290.60
Total	£110,047.00			

* ACD: Activity Cost Driver spent

** ACDR: Activity Cost Driver rate

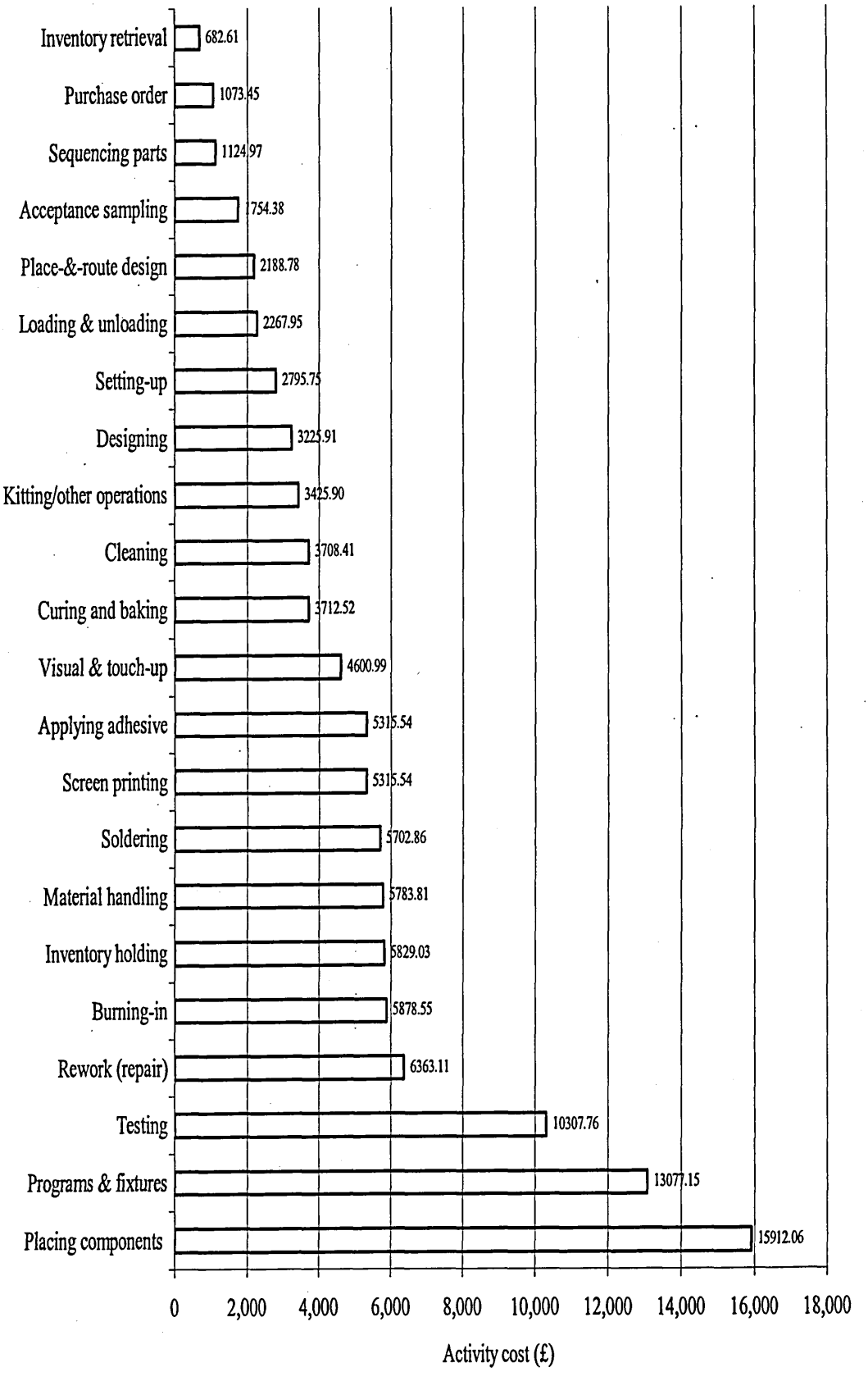


Figure 5.1. The costs of activities

5.4.4. Calculating the costs of PCBs

The production cost of PCBs is calculated in this step depending on the activities involved in the production process. This is achieved by multiplying the activity cost driver rates (ACDR) obtained from the previous step by the corresponding activity cost driver amounts spent during the production process. Usually, different types of PCBs require different activities; however, when they require same activities, they, at least, require different amounts of the activity cost drivers. This means, different types of PCBs should incur different costs depending on what and how many activities are involved in the production process. Since ABC is based on the idea of identifying the production activities, the results of applying ABC on PCB production should accurately reflect the real cost of the production of PCBs.

Using activity cost driver rates from Table 5.13 and data from Table 5.1, the production cost of any particular PCB type can be calculated. Since board type A have been used as an example throughout this chapter, it is used here again to calculate its production cost as presented in table 5.14. The first 13 rows in the table are quite clear and they do not require any explanation. However, the last nine rows require some explanation as presented hereafter.

- Purchase order: assuming that the number of purchase orders during the production period is 49 and since the total number of batches produced during the same period is 45, this means one purchase order is equivalent to the production of:

  $$45/49 = 0.918 \ batches.$$

  Since there are 3 batches of board type A, this means, one purchase order is equivalent to:

  $$0.918 \times (291/3) = 89.081 \ boards$$

  of type A. This means, one board of type A is equivalent to:

  $$1/89.081 = 0.0112 \text{ purchase order,}$$

  which is the amount specified in Table 5.14.
- Acceptance sampling: assuming that the sampling size for the production period is 68 boards, this means, for all boards of type A the sampling size is equivalent to:

  $$68 \times 3/45 = 4.533 \ boards,$$

  which means, for one board it is:

$4.533/291 = 0.0156$ *board*,

which is the amount specified in Table 5.14.

Table 5.14. Calculating the production cost of one PCB of type A

Activities	ACDR	Amount	Description of amount	Cost (£)
Sequencing parts	0.0174	45	Components sequenced	0.78
Loading & unloading	0.1137	7	No. of times handled	0.80
Screen printing	1.8651	1	No. of boards	1.87
Applying adhesive	1.8651	1	No. of boards	1.87
Placing components	0.2462	45	Components placed	11.08
Soldering	2.0010	1	No. of boards	2.00
Cleaning	1.3012	1	No. of boards	1.30
Curing and baking	1.3026	1	No. of boards	1.30
Testing	3.0140	1.2	20% tested twice	3.62
Burning-in	2.0626	1	No. of boards	2.06
Rework (repair)	11.1633	0.2	20% reworked	2.23
Visual & touch-up	0.0976	29	No. of joints	2.83
Kitting/other operations	1.2021	1	No. of boards	1.20
Purchase order	21.9071	0.0112	1/((45/49)×(291/3))	0.25
Acceptance sampling	25.7996	0.0156	1/((45/68)×(291/3))	0.40
Inventory retrieval	6.9654	0.0756	(22/291)	0.53
Material handling	0.7243	1.8293	(7985/45)/(291/3)	1.33
Setting-up	62.1279	0.0103	(3/291)	0.64
Inventory holding	59.4799	0.0756	(22/291)	4.50
Designing	53.7651	0.0486	(60×45/191)/291	2.61
Place-&-route design	54.7195	0.0324	(40×45/191)/291	1.77
Programs & fixtures	290.6033	0.0103	(45/45)/(291/3)	3.00
Total				£47.95

- Inventory retrieval: The cost driver of the inventory retrieval is the number of component types. According to Table 5.1, board type A has 22 component types and since 291 boards of type A are produced during the production period, this means one board of type A is equivalent to:

  $22/291 = 0.0756$ component type,

  which is the amount specified in Table 5.14.
- Material handling: assuming that the distance travelled by the material handling system is 7985m during the production period and since 45 batches are produced during the same period, this means one batch is equivalent to:

$$7985/45 = 177.44\ m$$

Since three batches (equivalent to 291 boards) of board type A are produced, one board of type A is equivalent to:

$$177.44/(291/3) = 1.8293\ m,$$

which is the amount specified in Table 5.14.

- Setting-up: the setting-up is performed for every batch, which means, one board for type A is equivalent to:

$$3/291 = 0.0103 \text{ setting-up},$$

which is the amount specified in Table 5.14.

- Inventory holding: similar to inventory retrieval.
- Designing: assuming that the total time for designing all board types is 60 hours and since the total number of locations (components) is 191, as shown in Table 5.1, this means the designing time for each component is:

$$60/191 = 0.314\ hours.$$

Since the number of locations for board type A is 45, this means the designing time for board type A is:

$$0.314 \times 45 = 14.14\ hours,$$

which is equivalent to the production of 291 boards. This means, one board type A is equivalent to:

$$14.14/291 = 0.0486\ hour,$$

which is the amount specified in Table 5.14.

- Place-and-route design: assuming that the time for design is 40 hours and following the same calculations performed for "designing", one board of type A is equivalent to:

$$40 \times (45/191)/291 = 0.0324\ hour,$$

which is the amount specified in Table 5.14.

- Programs and fixtures: assuming that the number of programs and fixtures during the production period is 45 and since 45 batches are produced during the same period, this means one batch is equivalent to:

$$45/45=1$$

Since three batches (equivalent to 291 boards) of board type A are produced, one board of type A is equivalent to:

$$1/(291/3)=0.0103,$$

which is the amount specified in Table 5.14.

The same calculations are performed to calculate the production costs for the rest of the board types. The results are presented in Table 5.15.

Table 5.15. The production costs of all PCB types

Board type	Production cost (£)	Amount produced	Total cost (£)
A	47.95	291	13,954.85
B	45.47	296	13,460.12
C	38.06	324	12,330.12
D	37.20	360	13,391.06
E	36.36	354	12,870.76
F	35.92	385	13,829.54
G	37.93	416	15,777.11
H	34.04	424	14,433.44
Total		2850	110,047.00

The cost of materials is usually obtained from the bill of materials. As for this case study it is assumed that the cost of materials during the production period of the 2850 boards is £340,510. This cost is allocated to the PCB types according to the number of locations of each board type. This means that the total cost of producing one PCB of each board type can be calculated. For example, for one PCB of board type A, the cost of materials can be calculated as follows.

From Table 5.1, the number of locations of board type A is 45, which means, the total number of locations for the 291 boards is:

$$45\times 291=13095 \textit{ locations.}$$

The total number of locations for each board type can be calculated in the same way, then, the total number of locations for all board types can be calculated:

$$(45\times 291)+(37\times 296)+(23\times 324)+(22\times 360)+(17\times 354)+(15\times 385)+(20\times 416)$$
$$+(12\times 424)=64620 \textit{ locations.}$$

This means, for one board of type A the cost of materials is:

$$(340,510 \times 13095 / 64620) / 291 = £237.12$$

Now, the total cost (production and materials) for each board of type A produced is:

$$47.95 + 237.12 = £285.08$$

The same calculations are performed for the rest of the board types and the results are presented in Table 5.16. It should be emphasised that this cost does not include the cost of facility level activities mentioned earlier in subsection 5.4.3.

Table 5.16. The total production costs of all PCB types

Board type	Total cost/board (£)	Amount produced	Total (£)
A	285.08	291	82,957.92
B	240.44	296	71,170.82
C	159.25	324	51,597.85
D	153.12	360	55,124.87
E	125.94	354	44,582.14
F	114.96	385	44,260.45
G	143.31	416	59,618.70
H	97.27	424	41,244.25
Total		2850	450,557.00

5.5. The effects of applying the algorithm

As mentioned in Chapter 4, the application of the algorithm reduced the time required by the component placement process by 3.76% and the set-up time process by 8.27%. This reduction is considered in this section to see how it affects the production costs of PCBs. The reduction in placement time means that the operating times of some resources on the "placement machine" and the "robotic machine" cost centres are expected to be reduced proportionally. The resources that are involved with these two cost centres and are expected to be affected by this time reduction are: "engineer", "technician", "operator" and "worker". Less operating time for these resources means lower total cost for the cost centres involved ("placement machine" and the "robotic machine"), which leads eventually to lower costs for the activities these two cost centres are linked to. In our case, there is only one activity that is affected; it is the "placing components activity". The same discussion applies to the reduction in the set-up time.

The affected resources in this case are: "engineer", "technician", "operator" and "worker". As for the cost centres and activities affected, they are "Set-up centre" and "setting-up" respectively.

In the case study, let's assume that the times spent by the resources on the cost centres are known as presented in Table 5.17. Since the time reduction is expected to be 3.76% and 8.27% for the placement time and the set-up time respectively, the time saved by applying the algorithm can be calculated according to these two percentages. For example, the resource "engineer" spends 15 hours in each of the following cost centres: "set-up centre", "placement machine" and "robotic machine". This means, the time saved for the resource "engineer" in the set-up centre" cost centre is:

$$15-(15\times 8.27/100)=13.76\ \textit{hours},$$

in the "placement machine" cost centre is:

$$15-(15\times 3.76/100)=14.44\ \textit{hours}$$

and in the "robotic machine" cost centre is:

$$15-(15\times 3.76/100)=14.44\ \textit{hours}$$

Table 5.17. Operating times before time reduction (hours)

	Engineer	Technician	Operator	Worker
Set-up centre	15	10	110	20
Placement machine	15	20	300	50
Robotic machine	15	20	150	50

The same calculations can be repeated for the rest of the resources as shown in Table 5.18. The table shows the operating times of the resources in the cost centres for the production period as would they be expected after the algorithm is applied. By subtracting the amounts in Table 5.18 from the corresponding amounts in Table 5.17 the times saved by the application of the algorithm can be calculated. The results are presented in Table 5.19.

Table 5.18. Operating times after reduction (hours)

Cost centre	Engineer	Technician	Operator	Worker
Set-up centre	13.76	9.17	100.9	18.35
Placement machine	14.44	19.25	288.7	48.12
Robotic machine	14.44	19.25	144.4	48.12

Table 5.19. The saved times of resources (hours)

Cost centre	Engineer	Technician	Operator	Worker
Set-up centre	1.24	0.83	9.1	1.65
Placement machine	0.56	0.75	11.3	1.88
Robotic machine	0.56	0.75	5.6	1.88
Total	2.36	2.33	26.0	5.41

From Table 5.19, the total time saved can be calculated:

$2.36+2.33+26.0+5.41=36.10$ *hours.*

In addition to the reduction in the operating times, there are also reductions in the costs of direct utilities and the maintenance & depreciation of the cost centres involved. The costs of the direct utilities of the cost centres "placement machine" and "robotic machine" before the reduction are assumed to be known data. The costs after applying the algorithm can be calculated in the same way the times were calculated for the resources in the previous paragraph. The results are presented in Table 5.20.

Table 5.20. Costs of direct utilities and the maintenance & depreciation (£)

	Direct utilities (£)			Maintenance & depreciation (£)		
	before	after	difference	before	after	difference
placement machine	65.00	62.56	2.44	150.00	144.36	5.64
Robotic machine	60.00	57.74	2.26	140.00	134.74	5.26
Total			5.70			11.90

The changes to operating times, the direct utilities and maintenance & depreciation will affect the costs of the involved cost centres. The total costs of the cost centres involved after applying the algorithm can be calculated using the same way the

costs were calculated before applying the algorithm, as described in subsection 5.4.2, taking into consideration the operating times in Table 5.18 and the costs in Table 5.20. The results are presented in Table 5.21. The cost centre driver spent (CCD) for the "placement machine" and the "robotic machine" cost centres can be calculated as follows:

$$155-(155\times 3.76/100)=149.17 \textit{ hours.}$$

Table 5.21. The new costs of cost centres, their cost drivers and their rates

Cost centre	Total cost before reduction	Total cost after reduction	Change	CCD*/month before reduction	CCD*/month after reduction	CCR**
Set-up centre	£2,280.85	£2,142.14	£138.72	45 set-ups	45 set-ups	47.603
Pick-&-place machine	£4,668.01	£4,507.62	£160.39	155 hours	149.17 hours	30.218
Robotic machine	£3,133.01	£3,030.34	£102.67	155 hours	149.17 hours	20.314
Total			£401.78			

* CCD: Cost Centre Driver spent
** CCR: Cost Centre Rate (£/unit)

The costs of activities that are affected by the changes to the costs of cost centres presented in Table 5.21 can be calculated as described in subsection 5.4.3 taking into consideration the cost centre rates (CCR) in Table 5.21. The results are presented in Table 5.22. As shown in Table 5.21 and Table 5.22 the total savings expected from applying the algorithms during the production period of one month is £401.78.

Table 5.22. The new costs of activities, their cost drivers and their rates

Activities	Activities costs before reduction	Activities costs after reduction	change	ACD*	ACDR**
Placing components	£15,912.06	£15,648.99	£263.06	64620	0.2422
Setting-up	£2,795.75	£2,657.04	£138.72	45	59.0453
Total			£401.78		

* ACD: Activity Cost Driver spent
** ACDR: Activity Cost Driver rate

5.6. Lean Accounting basics and principles

LA is a supportive system to lean manufacturing; therefore, it cannot be implemented alone, rather, it has to be implemented as a supplement to lean manufacturing. As lean manufacturing is a system that implements lean principles on the operational and production aspects, LA is a system that does the same to the financial and accounting aspects. Therefore, in order to be able to understand LA, some background information about lean principles in general and about lean manufacturing in particular has to be presented first.

Lean manufacturing has been introduced to increase customer value by eliminating, or at least reducing, waste from the production system. Waste has many forms and can be found in many areas in the company. For example, time, materials, inventory, rework, idle machines can all be considered as forms of waste. Since there are different forms of waste, lean manufacturing depends on different tools and principles to deal with waste reduction/elimination. The following are some of these tools and principles:

- Cellular manufacturing.
- Pull, rather than push, system (JIT and kanban).
- Value stream mapping.
- Low inventory.
- Less rework (high quality).
- Small orders of materials.
- Continuous improvement.

The implementation of lean manufacturing depends on the state of the company and should be gradual and continuous. Womack and Jones (1996) present the principles of lean thinking as:

- Value: the value provided to the customer.
- Value stream: the processes that contribute to manufacturing the product.
- Flow: the flow of products and services through the value stream.
- Pull: make on demand (just-in-time system).
- Perfection: total quality management through continuous improvement.

The principles of lean thinking and lean manufacturing are being applied in most of today's manufacturing companies. Therefore, it is necessary that they are applied to the accounting system as well and this is why LA has been developed.

The main idea of lean (whether it is lean manufacturing, lean accounting, lean thinking, etc.) is to reduce waste, which leads to creating free capacity. If this capacity is not used, the improvement to the financial outcome will be equivalent to the amount of waste reduced. However, the financial outcome will improve much more when the capacity is benefited from (e.g. laying people off, increasing sales, introducing new products, etc.). In LA, reducing waste means reducing financial transactions and control processes. This can be performed when the need for such transactions and processes cease to materialize. This means that the thinking of how to use the would-be freed capacity should be parallel to the thinking of introducing lean principles.

LA is necessary for lean manufacturing, not only because standard accounting system is not suitable but also because it is harmful to lean manufacturing. Standard accounting measurements show an increase in cost and reduction in profit, as will be seen in subsection 5.8.2, as a result of applying lean manufacturing especially at the early stage when trying to reduce the work-in-progress inventory (Maskell & Baggaley 2004). Furthermore, overhead absorption (a standard accounting principle) encourages workers/employees to do things that contradict lean manufacturing (large batch size, high inventory, large quantities of raw materials, etc.). For this reason, any company starting to apply lean manufacturing and keeping the standard accounting system in place could be forced to choose to cancel the implementation of lean manufacturing when facing negative results. Therefore, it is important to implement an accounting system that appreciates the improvements introduced by lean manufacturing to the company, an accounting system that shows clearly the impact of applying lean manufacturing on the bottom line. Such an accounting system should be based on the same lean thinking principles and culture that lean manufacturing is based on, in other words, a lean accounting system.

5.7. How Lean Accounting system works

As mentioned earlier, eliminating transactions, which will be explained more in section 5.8, is one of the main features of LA. Transactions in the standard accounting system are used to control business operations. Therefore, in order for LA to be able to eliminate these transactions it has to find a way to maintain the control over the business operations and processes. The idea is to replace the eliminated transactions with fewer, less detailed and simpler transactions. This process is a continuous one since the number of new transactions is further reduced over time to even fewer transactions. In

other words, the journey to a lean enterprise is a never-ending one. For example, in the standard accounting system as a product is being made, the labour hours and the movement of raw materials are tracked and reported. This is replaced in LA with back-flushing, which is done, when the job is completed, by the information system. This is achieved by reading the bill of materials and the production routings and standards. Over time, when the inventory levels are low and consistent and when the operations and processes are controlled, even back-flushing is not necessary and can be eliminated.

It is important to note that the process of transaction elimination should be gradual. Most transactions inherited from standard accounting should be maintained at the early stage of implementing LA in order to maintain the control of the business operations. A step-by-step elimination process should, by time, be able to reduce the number of transactions and keep the business under control.

5.8. Implementation of LA on the case study

Lean accounting has been introduced as an alternative to the standard accounting system in companies implementing lean manufacturing. In this section, the implementation steps of lean accounting are introduced, briefly explained and applied to a case study. Throughout the case study, the importance of implementing lean accounting is being emphasised by clarifying the financial benefits of such implementation, since lean accounting is designed to help lean manufacturing cut the production costs. In addition, it is explained how lean accounting provides better information for the management in the company. It allows them to see the financial benefits of lean improvements through achieving better decision-making and saving money by reducing costs, eliminating waste and providing more control over production processes. Lean accounting should be implemented alongside lean manufacturing since they complement each other. However, the implementation of lean accounting should always fall some steps behind the implementation of lean manufacturing because most of lean accounting tools and methods do not work unless some of lean manufacturing tools have been established.

In order to implement lean accounting one has to know the current state of the organization, company, etc. In their book "Practical Lean Accounting" Maskell and Baggaley (2004) presented a "diagnostic tool" to help assess the current status of the organization that is preparing to implement lean accounting. They divided the "maturity path" to lean accounting into four stages:

1. Traditional: just started with lean manufacturing but not lean accounting.
2. Piloting lean cells: have successfully implemented pilot lean cells and therefore can start implementing some of lean accounting principles.
3. Managing by value stream: value streams are created to link the cells implemented in the previous stage. This allows for more lean accounting principles and methods to be implemented.
4. Lean enterprise (lean business management): the value streams are extended outside the company walls to include suppliers and customers. At this stage some other tools of lean accounting such as 'target costing', can be applied.

The diagnostic tool that Maskell and Baggaley provide is a questionnaire about the state of the organization in different categories: "Financial accounting, Operational accounting, Management accounting, Support for lean transformation and Business management". The outcome of the questionnaire is used by the team responsible for the transition towards lean accounting to decide how to plan this transition. The implementation steps of lean accounting differ according to the state of the company implementing lean; however, there are general steps that could be followed as shown in Figure 5.2. When the current state of lean manufacturing at the company under consideration has been assessed, there are three possible options to consider:

1. If the company is at an early stage of lean manufacturing and the pilot cells are in place, lean accounting procedures should start with the top block of steps.
2. If lean manufacturing is widespread at the company and the company has started managing by value stream, lean accounting procedures can start with the top and middle blocks of steps at the same time.
3. If lean manufacturing is widespread at the company and its suppliers and customers (where applicable), lean accounting procedures can start with the top, the middle and the bottom blocks of steps at the same time.

The general implementation steps of lean accounting are explained throughout this section and most of these steps are applied to a case study in order to clarify how they can be applied in a real-life situation. However, the order at which these steps are explained in this chapter does not necessarily follow the order shown in Figure 5.2.

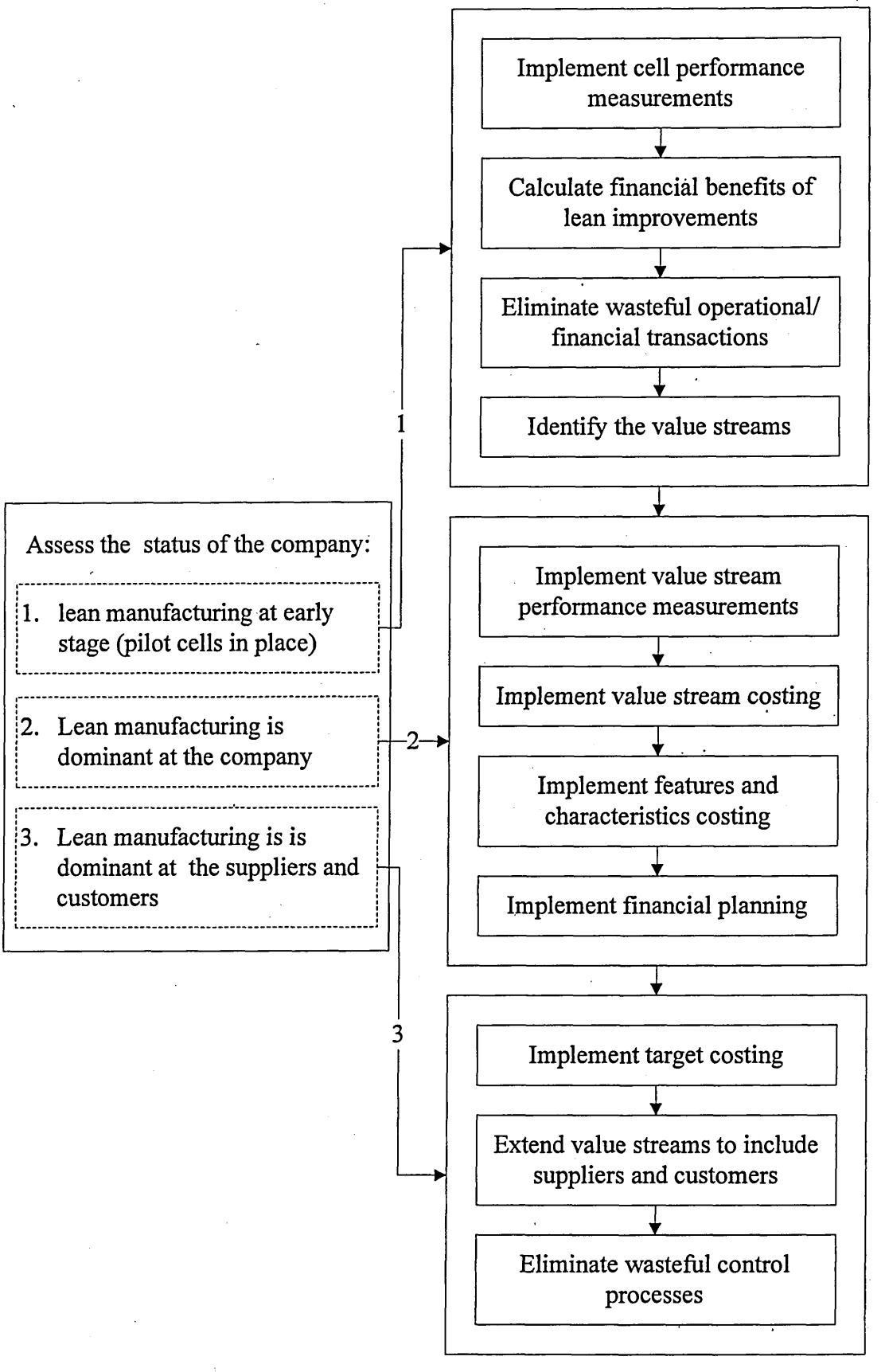


Figure 5.2. The general implementation steps of lean accounting

The company in the case study follows a traditional production system: push rather than pull system, high inventory, maximum machine utilization, high rework percentage, etc. The features of this production system are presented in Table 5.23. The lead time is 25 days, which is quite long compared to the cycle time (37.5min). The company employs 46 people for the PCB value stream, produces an average of 2850 boards per month (45 batches), have daily shipments and weekly delivery from the suppliers.

Table 5.23. Features of the current production system

	SMT machine	Manual/ robotic load	Test/ rework	Burn-in	Package/ shipment
Cycle time (min.)	1.5	15	8	7	6
Set-up time (min.)	38	8	10	12	n/a
Downtime/rework	10%	7%	20%	n/a	n/a
No. of people	5	10	9	4	2
No. of machines	1	1	1	0	0

The company management decided to undergo a lean approach. The lean consultant team responsible for the job started implementing lean manufacturing principles:

- Lean pilot cells have been introduced by reorganizing and redesigning the current cells, which led to forming the value stream.
- A pull system was introduced (kanban) to the value stream that reflects the customers' real needs.
- A training program in lean principles was introduced.
- Key suppliers were identified and agreements were signed to deliver daily according to the requirements of the SMT cell controlled by the pull system.
- Introduction of standardized work and quality from the source.

The operational changes resulted from implementing lean manufacturing were:

- The lead time was reduced to 6.5 days.
- The required floor area reduced from 3200 m^2 to 2200 m^2.
- Shorter cycle times and reduced rework percentage (quality increased).
- Some resources were freed up (machine hours, labour hours).

– Percentage of on-time shipments increased.

As lean manufacturing and lean accounting complement each other, the detailed implementation of lean accounting is discussed and the operational and financial aspects of the process are explained thereafter.

5.8.1. Performance measurements

There are performance measurements for both the production cells and the value streams. The number of the measurements should not be too high since this would be against the principles of lean. The measurements should be visual, simple, manual and should also be focused around the principles of lean thinking (see Womack and Jones (1996)). The data required for these measurements should be collected following the principles of lean (simple, effective and does not incur waste). Some examples for these measurements, obtained from Maskell and Baggaley (2004), are explained and applied to the case study hereafter. It should be noted that any other measurements are also possible.

5.8.1.1. Cell measurements

– Day-by-the-hour report:

This report ensures that the cell cycle time is consistent with the customers demand. The produced PCBs are counted hourly and the report is presented visually so that everyone can see it. This allows for problems to be discovered and resolved as early and as quickly as possible. Descriptions of the problems encountered are also written down and, over time, the gathered information can be used to avoid potential problems in future. This will eventually lead to a better process quality. The planned production rate may differ according to the type of products manufactured (mixed-model systems), which is the case in this case study. Since the company produces different types of PCBs, the planned hourly rate is different from one hour to another. Table 5.24 presents an example of the day-by-the-hour report for the case study.

– Work-in-progress to standard work-in-progress ratio:

This is used to measure the inventory in the cell. Each cell is designed to hold a specific amount of inventory (standard work in progress) to account for any problems that may happen in the cell. In a lean cell, when the amount of inventory is equal to the amount of standard inventory, this means the pull system is working smoothly; otherwise, the system is failing. This measure is calculated by dividing the amount of

inventory (kanbans, items, family of products) in the cell by the standard amount of inventory that the cell is designed to hold. When the ratio is less than 1 the inventory level is low and when it is bigger than 1 the inventory level is high. Since there are too many components (parts) to count in this case study, this measurement is calculated by counting the number of boards rather than the number of components. The amount of standard inventory for the SMT cell is 8 boards and the cell has 9 boards, which means the work-in-progress to standard work-in-progress ratio in this case is:

$$9/8 = 1.13$$

Table 5.24. Day-by-the-hour report

Time	No. of boards planed	No. of boards manufactured	Total No. of boards planned	Total No. of boards manufactured
09:00 – 10:00	17	17	17	17
10:00 – 11:00	17	16	34	33
11:00 – 12:00	16	16	50	49
12:00 – 13:00	16	15	66	64
13:30 – 14:30	15	15	81	79
14:30 – 15:30	15	14	96	93
15:30 – 16:30	13	12	109	105
16:30 – 17:30	14	14	123	119

– First time through:

This measurement (also called "yield") reports the number of reworked or rejected items in the cell. It is a measure for the effectiveness of the cell producing the product correctly and to the right cycle time. The data required for this measurement are collected by the cell operators and the results are presented visually. The first time through measurement is presented as a percentage and calculated as follows:

$$\textit{First time through} = (\textit{correct items manufactured} / \textit{total items manufactured}) \times 100\%$$

In the case study, assuming that the SMT cell produced 13 boards per hour and one is rejected and one is reworked, the first time through in this case is:

$$((13-1-1)/13) \times 100\% = 84.62\%$$

Since rejected and reworked items are forms of waste, whenever the first time through measurement is low, the cell has to be investigated and any problem found should be resolved. This measurement allows the cell operators to quickly detect any quality problems in the cell and, hence, reduces the amount of waste.

– Operational equipment effectiveness:

This measurement (can also be called Overall Equipment Effectiveness (OEE)) is used to measure the ability of machines to do the required job according to the specified quality at the specified time. It is a combination of three measures: availability, first time through (yield) and production efficiency (utilisation). Applying this measurement to all machines could be time consuming and wasteful, therefore, this measurement is usually applied to the machine that determines the flow rate of the cell or of the production line, in other words, the bottleneck machine. The first element of this measurement can be calculated as follows:

$$Availability = ((total\ time - down\ time) / total\ time) \times 100\%$$

The total time is the time scheduled for production and the downtime is the time when the machine is not working due to maintenance, setting-up, tooling, etc. The second element of this measurement, first time through, is calculated as described in the previous measurement, whereas the third element is calculated as follows:

$$Production\ efficiency = (actual\ flow\ rate / ideal\ flow\ rate) \times 100\%$$

The ideal flow rate is the rate at which the machine should run to achieve the required cycle time, which is determined by the customer requirements. Therefore, it is not necessarily the maximum flow rate which the machine has been designed to work at. The actual flow rate is usually smaller than the ideal flow rate due to the unexpected stoppages that could result from a lack of feeding materials.

In the case study, the operational equipment effectiveness of the SMT machine (the bottleneck machine) can be calculated as follows:

Availability:

Assuming that the total scheduled time is 8 hours a day (one shift) and the total down time during this period is 25 minutes, the availability is calculated as:

$$((8 - (25/60))/8) \times 100\% = 94.79\%$$

First time through:

It is already calculated: 84.62%

Production efficiency:

Assuming that the actual flow rate is 13 boards per hour and the ideal flow rate is 14 boards per hour, the production efficiency is calculated as:

$$(13/14)\times 100\% = 92.86\%$$

Operational effectiveness efficiency:

The operational effectiveness efficiency of the SMT machine can now be calculated as:

$$94.79\%\times 84.62\%\times 92.86\% = 74.48\%$$

5.8.1.2. Value stream measurements

Value streams are identified to include the cells (including non-production processes such as, customer services, purchasing, production planning, etc.) that work together to produce similar products or a family of products. In fact, the process of value stream mapping is very important to lean manufacturing and lean accounting because many of lean accounting elements (e.g. performance measures, value stream costing, etc.) are based on value streams. Value stream mapping illustrates how materials and information flow through the value stream and, hence, gives the management a wider and clearer view of the value stream. The value stream performance measurements for the case study are calculated for the PCB value stream as detailed below.

- Sales per person:

This measurement is used to measure the productivity of the value stream. Sales-per-person measurement is important to track the productivity and it should be increased over time to increase the value stream profit. To calculate the sales per person for a particular value stream, the value of the sales associated with that value stream within a specific period should be known in addition to the number of full-time people working in that value stream. In the case where there are part-time or temporary people, the equivalent number of full-time people should be used.

In this case study, the monthly sales for the value stream are assumed to be £901,470 and the number of full-time people is 46, therefore, the sales-per-person measurement is calculated as:

$$901,470/46 = £19,597.17$$

- On time delivery:

This measurement is used to control the value stream. It measures the percentage of products delivered to the customers on the day the customers requested them to be delivered. When the control over the value stream is high, the on time delivery is high and vice versa. This measurement gives an indication when the value stream goes beyond control and, hence, provokes an action to be taken. One way of calculating on time delivery is by tracking the number of units delivered to the customer on a particular date compared to the number of units requested to be delivered on that date.

In this case study, the number of units requested by the customers to be delivered in a particular month is assumed to be 2750 boards. The actual number of PCBs delivered during that month is assumed to be 2430 boards. Hence, the on time delivery can be calculated as:

$$(2430/2750)\times 100\% = 88.36\%$$

- Dock-to-dock time:

Dock-to-dock time measures the time required for raw materials to be converted into finished products (starting at the time they are delivered to the receiving dock and ending at the time they are ready for shipment on the shipping dock). This is a measurement of the flow of materials in the value stream. The aim is to increase the flow in the value stream in order to reduce the inventory and this requires short dock-to-dock time. This measurement is calculated by dividing the inventory within the value stream by the shipment rate. The shipment rate is the shipped amount per week/month divided by the number of hours/days in the week/month.

In this case study, the total inventory in the value stream is calculated by counting the number of PCBs that can be produced from the available raw materials, the PCBs available in the work in progress and the finished PCBs. These amounts are assumed to be 930, 1720 and 470 boards respectively. This means, the total inventory in the PCB value stream is 3120 boards. The shipment rate is the amount of shipped units per month divided by the number of working days in this month. Assuming that the number of boards shipped is 2750 boards and the number of working days in the month under consideration is 22 days, this means, the dock-to-dock days can be calculated as:

$$3120/(2750/22) = 25\ days$$

- First time through:

This measurement is similar to that of the cell performance measurements. However, it is applied here to the whole value stream. It is calculated by multiplying the first time through for all the cells (again, production and non-production cells) in the value stream.

In this case study, the first time through for the PCB value stream is the product of multiplying the first time through of these cells: purchase order, material purchasing, SMT, manual/robotic assembly, test and rework, burning-in, shipping and invoicing. The first time through for the SMT machine is already calculated before and it is 84.62%. Following the same calculations, the first time through is calculated for the other cells. Assuming that the first time through for the purchase order, material purchasing, manual/robotic assembly, test and rework, burning-in, shipping and invoicing cells are 96.23%, 97.50%, 90.04%, 96.38%, 98.10%, 92.54% and 88.75% respectively, the first time through for the PCB value stream can be calculated as:

$$96.23\% \times 97.50\% \times 84.62\% \times 90.04\% \times 96.38\% \times 98.10\% \times 92.54\% \times 88.75\% = 55.51\%$$

- Average cost per unit:

Average cost per unit is calculated by dividing the total cost of the value stream (may or may not include the cost of raw materials) by the number of units shipped to customers within a specific period of time. Including or not including the cost of raw materials depends on whether or not the materials used for different products in the value stream have similar costs. Calculating the total cost of the PCB value stream will be detailed in subsection 5.8.4. The importance of this measurement lies in the fact that it is used in both 'features and characteristics costing' and 'target costing' methods. Additionally, it gives an indication to the state of the value stream. When the average cost is increasing abnormally, this means the value stream is producing more than selling and the inventory is building up, and vice versa. In both cases, the situation should be investigated.

The total cost of the PCB value stream in the case study is the sum of the cost of materials and the conversion costs. The cost of materials is obtained from the bill of materials and it is £340,510 as mentioned in subsection 5.4.4. The conversion cost is calculated as described in subsection 5.4.4 and it is £110,047. This means, the average cost per unit can be calculated as:

$$(340,510+110,047)/2850=£158.09$$

- Accounts receivable days outstanding:

This measurement gives an indication as to at what extent the process of receiving cash from customers is properly used. It basically measures the cash flow in the value stream. The number of outstanding days for the accounts receivable is calculated by dividing the accounts receivable amount by the average daily sales. When the number of days is high this means there is shortage of cash flow in the value stream and the process of receiving cash from customers must be revised.

For this case study, the average daily sales amount is calculated by dividing the amount of monthly sales by the number of working days per month:

$$901,470/22=£40,976$$

Assuming that the amount of accounts receivable is £740,500, the number of outstanding days for accounts receivable can be calculated as:

$$740,500/40,976=18\ days$$

5.8.2. Calculating the financial benefits of applying lean manufacturing

It is important to check the effects of applying lean manufacturing on the bottom line. Improvements to the bottom line are the only way to convince the management that lean manufacturing is not only good operationally but also financially. The standard accounting system is unable to detect any financial improvements resulting from implementing lean manufacturing as they are usually long term. In addition, the standard accounting system may show that the immediate financial effects of implementing lean manufacturing are the same or even worse than before applying it. This may raise the issue of whether implementing lean manufacturing was a good idea in the first place. This idea is further explained while the financial benefits of applying lean manufacturing are being identified in the case study.

The standard accounting system usually calculates the cost of sales as follows:

$$Cost\ of\ sales = cost\ of\ purchased\ materials + conversion\ cost + starting\ inventory - ending\ inventory$$

Now, assuming that the data for the case study before and after applying lean manufacturing are as presented in Table 5.25, the cost of sales before lean according to the above-mentioned equation can be calculated as:

$$340,510+110,047+(40,020-40,020)=£450,557$$

and the cost of sales after lean can be calculated as:

$$317,240+102,977+(40,020-9,100)=£451,137$$

This means, the cost after lean is higher than before lean by:

$$451.137-450,557=£580$$

Table 5.25. The data of the case study before and after applying lean manufacturing

	Before lean	After lean
Starting inventory (£)	40,020	40,020
Cost of materials (£)	340,510	317,240
Conversion cost (£)	110,047	102,977
Ending inventory (£)	40,020	9,100

This negative result is actually not due to an increase in either the conversion cost or the cost of materials. On the contrary, the conversion cost and the cost of materials were reduced after applying lean manufacturing, which means, the negative result would have been higher by:

$$(340,510-317,240)+(110,047-102,977)=£30,340$$

In fact, the increase in the cost of sales after applying lean is the result of a reduction in inventory by $(40,020-9,100)=£30,920$, which is a normal consequence of applying lean manufacturing.

In order to show the real financial improvements to the value stream after implementing lean manufacturing, the negative impact of inventory reduction should be eliminated from the financial statements. Lean accounting presents the financial results in a way that reflects the real changes to the value stream by eliminating the inventory reduction effect. Lean accounting makes the value stream statement even clearer by including not only financial information but also operational and resources information. This new improved representation of the financial aspects is called the "Box Score" (Maskell & Baggaley 2004). An example of the box score for the case study is presented in Table 5.26.

Table 5.26. The box score for the PCB value stream in the case study

		Current state	Future state	Change	Long term future	Change
Operational	Dock-to-dock days	25				
	First time through	55.51%				
	On time delivery	88.36%				
	Floor space	3200 m^2				
	Sales per person	19,597				
	Average cost/unit	158.09				
Resource capacity	Productive					
	Non productive					
	Available					
Financial	Inventory value	40,020				
	Revenue	901,470				
	Material costs	340,510				
	Conversion costs	110,047				
	Value stream profit	450,913				

The operational and the financial information for the value stream have already been discussed in subsections 5.8.1.2 and 5.8.2 respectively. Hence, the information is filled in the table accordingly. However, one entry of the financial information needs to be calculated; it is the value stream profit. This can be calculated by subtracting the material and conversion costs from the value stream revenue:

$$901,470 - 340,510 - 110,047 = £450,913$$

As for the information related to resource capacity, it has been explained before that the implementation of lean manufacturing frees some resource capacities. The successful exploitation of these freed capacities is what will increase the value stream profit. This can be achieved by, firstly, calculating how much capacity has been freed and, secondly, deciding what to do with this capacity. This is discussed in the following two subsections.

5.8.2.1. Calculating freed capacity

As shown in Table 5.26, resource capacity can be divided into three categories: productive, non-productive and available; this categorisation applies to the two types of resources (people and machines). Productive capacity is the ability to do the work that is directly involved in manufacturing the required products (e.g. fabrication, assembly,

etc.). Non-productive capacity is the ability to do the work that is not directly involved in manufacturing the required products but is necessary for the production process (e.g. set-up, maintenance, rework, planning, administration, etc.). Available capacity is the ability to do work during the spare time after productive and non-productive capacities have been fulfilled (e.g. the time when the required jobs has been performed and, still, there is time to do something else).

The freed capacity can be calculated by calculating the capacities before and after lean for all the cells in the value stream and for all the resources. Calculating the capacity for a particular cell requires analysing the cell and determining the types of activities (productive or non-productive) that are performed in the cell and the time required for each activity. The total time for the productive and non-productive activities can then be calculated. To calculate the percentage of the productive, non-productive and available capacities, the total available time during the period under consideration should be calculated. The total available time is basically the total number of hours within which the people/machines should, according to the company policy, be working during the period under consideration. The resource capacities, therefore, can be calculated as follows:

$$Productive = (total\ productive\ time / total\ available\ time) \times 100\%$$

$$Non\text{-}productive = (total\ non\text{-}productive\ time / total\ available\ time) \times 100\%$$

$$Available = 100 - Productive - Non\text{-}productive$$

These three equations are applied to the cells in the PCB value stream to calculate the resource capacities for the box score presented in Table 5.26. The SMT cell is used as an example to show how the resource capacity can be calculated.

The data required to calculate the time of activities for the SMT cell are assumed to be known as presented in Table 5.27.

Table 5.27. The required data for some of the SMT activities

Activity	No. of times/month	Average time (min)
Material moves	148	8
Meetings/training	6	50
Other activities (cleaning etc.)	11	45
Waiting (per product)	2850	1.391
Downtime (machine)	10	45

The available time for machines can be calculated as follows:

$$8\ (hours\ per\ day) \times 22\ (days\ per\ month) = 176\ hours\ per\ month\ per\ machine$$

As for people, the available time (with two 10-minute breaks) can be calculated as:

$$(8 - 2 \times (10/60)) \times 22 = 168.67\ hours\ per\ month\ per\ person$$

The machines continue to run during the breaks since not all machine operators take their breaks at the same time. The activities performed in the SMT cell and the time of each activity for both people and machines can now be calculated. The results are presented in Table 5.28. Using the data from Table 5.23 and Table 5.27, the time of the activities can be calculated as follows.

- *Machine set-up*: This is a non-productive activity and the time of the activity can be calculated as follows.
  - For people: number of batches × set-up time × number of people

    $$45 \times (38/60) \times 5 = 142.50\ hours$$

  - For machine: number of batches × set-up time

    $$45 \times (38/60) = 28.50\ hours$$

- *Moving materials*: This is a non-productive activity and the time of the activity can be calculated as follows.
  - For people: number of times moved × average time per move

    $$148 \times (8/60) = 19.37\ hours$$

- *Attending machine/product*: This is a non-productive activity for people but productive for machine and the time of the activity can be calculated as follows.
  - For people & machine: number of boards produced × cycle time

    $$2850 \times (1.5/60) = 71.25\ hours$$

- *Training/meetings*: This is a non-productive activity and the time of the activity can be calculated as follows.
  - For people: number of times × average time of meeting/training × number of people

    $$6 \times (50/60) \times 5 = 25.00\ hours$$

Table 5.28. Activities in the SMT cell and the time of each activity

	People		Machine	
Activity	Productive	Non-productive	Productive	Non-productive
Set-up		142.50		28.50
Moving materials		19.73		
Attend machine/prod.		71.25	71.25	
Training/meetings		25.00		
Wait/downtime		66.07		73.57
Other (cleaning, etc.)		41.25		
Total		365.81	71.25	102.07

- *Wait/downtime*: This is a non-productive activity and the time of the activity can be calculated as follows.
  - For people: number of boards produced × average waiting time

    $$2850 \times (1.391/60) = 66.07 \textit{ hours}$$

  - For machine: number of boards produced × average waiting time + number of downtimes × average time of downtime

    $$2850 \times (1.391/60) + 10 \times (45/60) = 73.57 \textit{ hours}$$

- *Other (cleaning, etc.)*: this is a non-productive activity and the time of the activity can be calculated as follows.
  - For people: number of times × average time × number of people

    $$11 \times (45/60) \times 5 = 41.25 \textit{ hours}$$

The resource capacities (people and machine) for the SMT cell can now be calculated.

For people:

- Productive: 0.00%
- Non-productive:

  $$(365.81/(168.67 \times 5)) \times 100\% = 43.38\%$$

- Available:

  $$100 - 0.00 - 43.38 = 56.62\%$$

For machine:

- Productive:

$$(71.25/176)\times 100\% = 40.48\%$$

- Non-productive:

$$(102.07/176)\times 100\% = 58.00\%$$

- Available:

$$100 - 40.48 - 58.00 = 1.52\%$$

The resource capacities for the cells in the PCB value stream are calculated again after the implementation of lean manufacturing in order to check how much free capacity has been achieved. Again, the SMT cell is used as an example. Assuming that the introduction of lean manufacturing has led to the following improvements:

- Set-up time is reduced to 25 minutes and only 3 people are required to perform the set-up activity.
- Material moves are eliminated since the suppliers provide the materials daily and directly to the SMT cell.
- Waiting is eliminated and down time is reduced to 4 times.

Taking into consideration these new improvements, the data in Table 5.27 (before lean) should be amended as presented in Table 5.29 (after lean). Now, using data from Table 5.29 and following the same calculations used to calculate the times of activities before lean, the times of activities after lean can be calculated. The results are presented in Table 5.30.

Table 5.29. The data for the SMT activities after lean

Activity	No. of times/month	Average time (min)
Material moves	0	0
Meetings/training	6	50
Other activities	11	45
Waiting (per product)	2850	0
Downtime (machine)	4	45

Table 5.30. Activities in the SMT cell and the time of each activity after lean

	People		Machines	
Activity	Productive	Non-productive	Productive	Non-productive
Set-up		85.50		18.75
Moving materials		0.00		
Attend machine/prod.		71.25	71.25	
Training/meetings		25.00		
Wait/downtime		0.00		3.00
Other (cleaning, etc.)		41.25		
Total		223.00	71.25	21.75

The resource capacities for the SMT cell after implementing lean manufacturing can now be calculated following the same calculations performed above. The final results regarding the resource capacities for the SMT cell before and after the implementation of lean manufacturing and the changes achieved are shown in Table 5.31. As can be seen from the table, the implementation of lean manufacturing has converted part of the non-productive capacity into an available capacity in the SMT cell. The percentage of capacity freed by lean manufacturing is 16.93% for people and 45.64% for machines.

Table 5.31. Resource capacities for the SMT cell before and after implementing lean

	People			Machines		
	Before lean	After lean	Change	Before lean	After lean	Change
Productive (%)	0.00	0.00	0.00	40.48	40.48	0.00
Non productive (%)	43.38	26.44	-16.93	58.00	12.36	-45.64
Available (%)	56.62	73.56	16.93	1.52	47.16	45.64

The same procedures are applied to the other cells in the PCB value stream and the capacities of the resources are calculated before and after implementing lean manufacturing. The complete picture can now be seen after filling the information obtained above in the box score for the case study. Table 5.32 represent the new box score, which includes, in addition to the information presented in the old box score, the

capacities of resources and the future state (after lean) for the three categories: operational, resource capacity and financial.

Table 5.32. The new box score for the case study

		Current state	Future state	Change		
Operational	Dock-to-dock days	25	6.5	-18.5		
	First time through	55.51%	91.50%	35.99%		
	On time delivery	88.36%	97.20%	8.84%		
	Floor space	3200m^2	2200 m^2	-1000 m^2		
	Sales per person	£19,597	£19,597	£0		
	Average cost per unit	£158.09	£147.44	-£10.65		
Resource capacity	Productive	23.18%	21.50%	-1.68%		
	Non productive	58.46%	38.05%	-20.41%		
	Available	18.36%	40.45%	22.09%		
Financial	Inventory value	£40,020	£9,100	-£30,920		
	Revenue	£901,470	£901,470	£0		
	Material costs	£340,510	£317,240	-£23,270		
	Conversion costs	£110,047	£102,977	-£7,070		
	Value stream profit	£450,913	£481,253	£30,340		

In addition to the £30,340 improvement to the value stream profit, there are also operational improvements (e.g. 1000m^2 of freed-up floor space and 40.45% available capacity) that can be exploited to further increase the financial profit of the value stream. A lean option would be to use the freed resources to increase production. The other option would be to sell/rent the extra resources. For example, extra people can be made redundant, extra machines can be sold or rented to another company. The information for long-term future state can be calculated once the decision of what to do with the extra resources is taken and, then, the box score can be completed.

5.8.3. Eliminating wasteful financial transactions

Financial transactions and processes are used to maintain control over the business. Accounts payable, accounts receivable and general lodger are examples for such financial processes. As these transactions and processes incur costs, lean accounting tries to eliminate some of them while maintaining the required control over the business. In addition, eliminating transactions frees up more resources (mainly

accountants), which can then be used for lean improvements. The best way for eliminating transactions is to eliminate the reasons behind the existence of these transactions in the first place, and that is what lean accounting does. In the case study, some financial transactions will be used as examples to show how lean accounting can eliminate these transactions.

5.8.3.1. Accounts payable and accounts receivable processes

The company accountants spend a great deal of time auditing invoices from suppliers and producing invoices to customers. The reason for that is to match these invoices with the purchase orders produced and receipts received. This process is time and cost consuming and the company would do better without it. Reducing the number of invoices, and ultimately eliminating them, is the way to eliminate this process. The lean accounting way of dealing with this issue is explained hereafter. The accounts payable is used as an example.

The accounts payable elimination process starts with the company examining its suppliers to identify the key ones. Then, the key suppliers are certified and blanket purchase orders are established with them. The blanket purchase order contains the terms for supplying, the prices, the amounts, etc. which are agreed with the suppliers. The outcome of this action is that the need for invoices is reduced dramatically. The company now has a firm relationship with a small number of certified suppliers who will deliver the required materials daily to the production cells. Furthermore, the matching of purchase orders with the invoices has been eliminated since the trust between the company and its certified suppliers is now higher. Reducing the number of invoices means less accounts payable and, hence, less wasteful transactions. Furthermore, the number of invoices could be completely eliminated when the lean practices mature enough in the company and the payments are made to the suppliers when the materials are used.

The same approach discussed above can be applied to the accounts receivable with one difference. The company takes on the role of the supplier and its customers take on the role of the company. The company becomes a certified supplier to its main customers, who are encouraged to create blanket purchase orders with the company. The results achieved here are the same as discussed for the accounts payable.

5.8.3.2. The general lodger and end-of-month close process

In the case study, the company follows the traditional way of processing the general ledger and end-of-month close. Since the company is organised by departments there are resources allocated to each department. Hence, there are different accounts for the resources of each department, as shown in Table 5.33. This leads to a complex general ledger and end-of-month close process compared to the case when the company is organised by value stream.

Table 5.33. The resources and their accounts for each department for the case study

Resources	Accounts
People	Salary & benefits, taxes, training, travel, entertainment
Machines	Outside maintenance, depreciation
Tools	Tools used, depreciation
Supplies	Supplies costs
IT equipments	Supplies, depreciation
Office equipments	Supplies, depreciation
Utilities	Electricity, gas, water, telephone
Warehouses	Warehouses costs
other	Property tax, land rent
Total No. of accounts	45

Having introduced lean accounting, the company is now organised and managed by value streams. The number of departments is now reduced to three: PCB value stream, research & development and administration & overhead. Furthermore, the company reduced the number of accounts per department to five (materials, people, machines, external costs and other costs). Due to the reduction of the number of departments and the reduction of accounts per departments the number of accounts in the general ledger is reduced from 45 (as shown in Table 5.33) to $5 \times 3 = 15$. The reduction of number of accounts in the general ledger means the way of producing financial statements has become easier and faster and this leads to time and, hence, cost reduction.

Table 5.34 represents the financial statement for the case study. The profit rate of the PCB value stream is calculated by dividing the value stream profit by the revenue. As for the other two columns (new products and admin/overhead), the information mentioned is assumed to be known data.

Table 5.34. A financial statement for the case study

	Value streams			
	PCB	New products	Admin/overhead	Total
Revenue	£901,470	£0		£901,470
Material Costs	£340,510	£8,240		£348,750
Conversion costs	£110,047	£260,255		£370,302
Value stream profit	£450,913	-£268,495		£182,418
Employees costs			£16,720	£16,720
Expenses			£9,201	£9,201
Previous inventory				£40,020
Current inventory				£40,020
Change				£0
Gross profit				£156,497
Profit rate	50.02%			17.36%

5.8.4. Value stream costing

The value stream includes all the cells that contribute to fulfil a customer order starting with order entry and ending with after-sale support. Since standard costing (a process used in standard accounting) is not suitable for lean companies due to its way of dealing with overhead and to its non-lean characteristics (e.g. detailed data collection), a more suitable costing method that relies on lean principles and that is easy to understand is required. The lean accounting alternative to standard costing is value stream costing. At the early stage of implementing lean manufacturing, back-flushing is used to calculate the product cost without the need to track the costs while the product is being produced. This process can gradually be eliminated and replaced with value stream costing since value stream costing provides the appropriate information for decision-making and it is easy to understand and simple to implement as will be seen when implemented on the case study.

The basic idea of value stream costing is to calculate the costs (direct and indirect) incurred by the value stream. Then, the product average cost can be calculated by dividing the total cost of the value stream by the number of items produced in that value stream. This idea is actually not new and it is already being used in the process industries (e.g. oil refining, petrochemical industries, etc.). In fact, because lean companies are starting to have the characteristics of process-based companies (short lead time, small batch size, the level of inventory is low and stable, etc.), it would be

suitable for them to use the same costing method. The implementation of value stream costing can be performed when some conditions are met:

- Lean practices have progressed to a certain stage (when the level of inventory is low and consistent, short lead time, etc.).
- The business should be organized by value streams not by departments because one product or one product family can be produced by one value stream but not by one department.
- There should be as little as possible of overlap between value streams since overlapping makes it difficult to allocate the costs to each individual value stream.
- The inventory and the production processes should be under control.

There are some costs that cannot be attributed to any particular value stream in the business (e.g. research and development, market research, recruitment, training, annual audit, etc.); these costs are usually, but not always, small when compared to the other costs and are not allocated to the value streams. Instead, they are considered as sustaining costs and treated separately.

The cost of a product is required to be calculated by the standard accounting system because some decisions (e.g. pricing, make/buy, etc.) are made based on it. However, in a lean organization, the focus is on the value that the product provides to the customer and the pricing decision is made according to this value. The make/buy decisions are made according to how the decisions will affect the profitability of the value stream (the existence or non-existence of required capacity). The same discussion can be applied when a new product is introduced. This means, in a lean organization, the need to calculate the cost of a particular product is not always required. However, when it is required other methods should be used since value stream costing does not provide it. The solution provided by lean accounting to this problem is 'features and characteristics costing', which will be studied in subsection 5.8.5.

The value stream costing method is used to calculate the average cost of the PCBs produced by the PCB value stream in the case study. The calculated costs are for a period of one month and calculated as follows. The costs of employees working (directly or indirectly) in the value stream are calculated as shown in Table 5.35 in which the number of employees and the cost per employee are assumed to be known data. The number of employees represents the equivalent number of employees working for the PCB value stream. For example, the number of employees working in

"Purchasing" is 3 but since they work for two different value streams (PCB and new products development), the equivalent number of employees working in the PCB value stream is less than 3. Depending on how much time these 3 employees spend working for each value stream, the equivalent number of employees working for the PCB value stream can be calculated. Assuming that the amount of time they spend working for the PCB value stream is double the amount they spent working for the new products development value stream, this means, the equivalent number of employees working in "Purchasing" for the PCB value stream is equivalent to $3 \times (2/3) = 2$ people as shown in Table 5.35.

Table 5.35. The cost of employees working in PCB value stream

	No. of employees	Cost per employee	Total cost
Manager	1	£2,480	£2,480
SMT machine	5	£1,520	£7,600
Manual load	10	£1,520	£15,200
Test/rework	9	£1,520	£13,680
Burn-in	4	£1,520	£6,080
Package/shipment	2	£1,520	£3,040
IT	1	£1,960	£1,960
Purchasing	2	£1,680	£3,360
Customer services	1	£1,680	£1,680
Human resources	1	£1,680	£1,680
Secretary	1	£1,680	£1,680
Accounting	2	£2,160	£4,320
Quality assurance	1	£2,160	£2,160
Design/manufacturing eng.	2	£2,160	£4,320
Maintenance/technical supp.	2	£2,160	£4,320
Security/cleaning	2	£1,520	£3,040
Total	46		£76,600

The costs of materials, machines and other costs are also assumed to be known as presented in Table 5.36. The cost of materials include the costs of raw materials used in the manufacturing processes of the PCBs; the costs of machines include maintenance, utilities and depreciation; other costs include any outside processing costs, consumable materials, tools, etc. The total of the costs in Table 5.35 and Table 5.36, excluding the costs of materials, represents the conversion cost in the PCB value stream.

Table 5.36. The costs of material, machines and other costs for PCB value stream

	Costs of materials	Costs of machines	Other costs
SMT machine	£320,450	£999	£12,912
Manual load	£7,998	£257	£5,430
Test/ rework	£1,230	£142	£1,530
Burn-in	£10,832	£120	£1,056
Package/shipment		£140	£3,150
Design/manufacturing eng.			£5,691
Security/Cleaning			£2,020
Total	£340,510	£1,658	£31,789

Therefore, the conversion cost, which is used as known data earlier in this chapter, can now be calculated:

$$76,600+1,658+31,789=£110,047$$

Now, the total cost of the PCB value stream can be calculated:

$$110,047+340,510=£450.557$$

The company produces an average of 2850 PCBs per month, this means the average cost of one PCB is:

$$450,557/2850=£158.09$$

5.8.5. Features and characteristics costing

Features and characteristics costing method is based on the idea that there are some features and characteristics in the product that determine the cost of manufacturing it. Since the value stream is designed to produces similar but not identical products (product family), the average cost calculated by the value stream costing method does not accurately represent the cost of any of the products. Rather, it represents the cost of an 'average product' that has the average features and characteristics found in the other products. Therefore, the actual cost of any of the products is probably close, but not identical, to the average cost. The difference between the two costs depends on how much the two products are different in their features and characteristics. Therefore, to calculate the cost of a product more accurately, the features and characteristics that differentiate it from the 'average product' should be identified and the cost generated from having them should be calculated.

In general, the products that require more time to be manufactured consume more cost; this means, the cost of a product is related to its production rate of flow, which in turn is determined by the flow at the bottleneck cell in the value stream. Therefore, all these issues should be considered when using the features and characteristics costing method for calculating the cost of a product. Not only can features and characteristics costing method be used to calculate the cost of a product already being manufactured but it can also be used to calculate the cost of products that are still in the design stage.

The features and characteristics costing method is applied to the case study to calculate the costs of each PCB produced by The PCB value stream. The PCBs that have high number of components require longer cycle times on the SMT machine than the average cycle time (1.5 minutes). The same principle applies to the PCBs that have small number of components.

- Determining the bottleneck cell:

The bottleneck cell in the case study is the SMT cell. The average time required by the PCB to pass through the cell is the sum of the cycle time, waiting time and set-up time:

$$1.5+1.391+(38\times 45/2850)=3.491 \; minutes$$

- Determining the features and characteristics that affect the production of the bottleneck cell:

In PCB manufacturing, the product features that affect the SMT production time is the number of components that have to be placed on the board and the types of these components. In the case study, the number of components is categorised into low, average and high, which corresponds to 1.2 min, 1.5 min and 1.9 min cycle times respectively. Regarding the types of components, it can affect the set-up time of the SMT machine in the case study. When the components include a high percentage of components that are not dedicated to the SMT machine, the set-up time will be higher than the average set-up time and vice versa. To take this into account, the percentage of the non-dedicated components is, again, categorised into low, average and high, which corresponds to 30 min, 38 min and 53 min set-up times respectively. Table 5.37 shows the categories of the features and characteristics that affect SMT production time. It has to be noted that for the average number of components and the average number of non-dedicated components, the cycle time is 1.5 min and the setup time is 38 min. These two amounts have been used previously in the case study as presented in Table 5.23.

Table 5.37. Categories of the features and characteristics that affect SMT production

Category	Number of components	Cycle time (min)	Percentage of non-dedicated components	Set-up time (min)
Low	< 17	1.2	< 2%	30
Average	17-25	1.5	(2-4)%	38
High	> 25	1.9	> 4%	53

– Calculating the cost of PCBs:

The conversion costs of the PCBs are calculated taking into account the nine combinations (3 categories for the number of components × 3 categories for the non-dedicated components) listed in Table 5.37 as follows. The conversion cost of an average PCB (a PCB that has a number of components of 17-25 and a percentage of non-dedicated components of 2%-4%) is the conversion cost of the PCB value stream divided by the average number of PCBs produced per month. The average number of PCBs produced per month is calculated by dividing the production time of the SMT machine by the cycle time. The production time is the number of working minutes per month minus the downtime minutes. Assuming that the average number of working days per month is:

$$5\ (\textit{days per week}) \times 52\ (\textit{weeks per year}) / 12\ (\textit{months per year}) = 21.67\ \textit{days}$$

Then, the working minutes per month can be calculated:

$$8\ (\textit{hours per day}) \times 60\ (\textit{minutes per hour}) \times 21.67 = 10,400\ \textit{minutes}$$

Taking into account the data in Table 5.27, the downtime minutes per month can be calculated:

$$10 \times 45 = 450\ \textit{minutes}$$

This means, the production time is:

$$10,400 - 450 = 9950\ \textit{minutes}$$

Since the cycle time for the SMT, as calculated above, is 3.491 min, the average number of PCBs produced per month is:

$$9950 / 3.491 = 2850\ \textit{boards}$$

This amount has been used in previous calculations throughout this chapter.
Now, the conversion cost of an average PCB is calculated:

$110,047 / 2850 = £38.61$

The same calculations can be performed for the other eight combinations taking into consideration the cycle times and set-up times for each combination in Table 5.37. For example, when considering a high number of components and a low percentage of non-dedicated components, the cycle time and the set-up time for this combination are 1.9min and 30min respectively. The conversion costs for all types of PCBs are presented in Table 5.38.

Table 5.38. The conversion costs for all PCB types in the PCB value stream

		Percentage of non-dedicated components		
		< 2%	(2-4)%	> 4%
Number of components	< 17	£33.15	£34.60	£37.69
	17-25	£36.99	£38.61	£42.05
	> 25	£42.11	£43.95	£47.87

As can be seen from Table 5.38, the conversion cost per PCB starts with £33.15 for the easiest to manufacture and ends with £47.87 for the most difficult to manufacture. As for the cost of materials, it is obtained for the intended PCBs from the bill of materials. Usually, PCBs with a higher number of components tend to have a higher cost of materials.

It has been explained above how features and characteristic costing method could be used to calculate the cost for different PCB types. Following the same analogy, this method could potentially be used to calculate the costs of products in other industries. However, there is a limitation to this method that should be noted. This method provides one cost for a range of similar products (i.e. sub-family of products). In reality, non-identical products consume similar but not identical costs. For example, in the case study, a board type with 18 components and 5% non-dedicated components has the same conversion cost as a board type with 24 components and 7% non-dedicated components because these two board types fall into the same range according to the classifications shown in Table 5.38 (average number of components and high percentage of non-dedicated components). In some cases, the downside of this costing method can be ignored due to the fact that the exact cost of a product is not always required.

5.8.6. Target costing

Target costing is used in lean accounting to focus on customer value, which is one of the principles of lean thinking. Target costing is used to calculate the allowable cost, which is the cost that represents the difference between the selling price and the required profit of a product. The allowable cost of a product should satisfy the customer and the value stream profitability required by the company management. Target costing is implemented by establishing the customer value by considering the product and any other services associated with it. The allowable cost is then calculated as the customer value (selling price) minus the required profit, and then compared to the average product cost created by value stream costing as explained in subsection 5.8.4. If the allowable cost is less than the average product cost, some improvements should be made to reduce the average product cost. These improvements may include changes to any process in the value stream from order entry to after-sale services. Target costing can be used for products currently being manufactured and for new products alike. It should be noted that target costing is not just a method for calculating the cost of a product to satisfy the customer and the company management, it is also a method for continuous improvement across the value stream in order to increase customer value and, at the same time, to increase value stream profitability.

The process of implementing target costing requires many steps. It usually starts by understanding the customer needs through conducting surveys, then, identifying the features and characteristics that will meet these needs. This step is followed by identifying the target costs for the products and services, and finally, balancing the value stream costs with the target costs through continuous improvement. These steps will be applied to the case study as explained below.

The company in the case study receives a customer order to manufacture a new type of PCB. The rate of production will need to be 350 boards per month to fulfil the customer demand. Having negotiated the price with the customer, it has been agreed that the price of the new board type is to be £325.00 per board. The company manufactures PCBs according to the requirements needed by its customers, which means the first steps of implementing target costing (understanding customer needs and value) are already fulfilled.

The new PCB type consists of 32 components and 3.8% of them are non-dedicated to the SMT machine. This means that the new PCB falls into the category

high number of components and average percentage of non-dedicated components. According to Table 5.38, the conversion cost for this PCB is £43.95. The cost of materials according to the bill of materials is assumed to be £168.65, hence, the total cost of the new PCB type can be calculated as:

$$43.95 + 168.65 = £212.60$$

The company has decided to keep the profit rate for the PCB value stream constant (i.e. 50.02%, as shown in Table 5.34) after the new PCB type is introduced, which means the new PCB should be sold with a profit rate of 50.02%. With such profit rate, the allowable cost can be calculated as the selling price minus required profit:

$$325.00 - (325.00 \times 50.02/100) = £162.43$$

Since the total cost is higher than the allowable cost, improvements to the value stream have to be made to get rid of the cost gap of:

$$212.60 - 162.43 = £50.17 \textit{ per board}$$

or:

$$£50.17 \textit{ (per board)} \times 350 \textit{ (boards)} = £17,558 \textit{ per month}$$

for the value stream. Without bridging this gap the profit rate of the new product would be:

$$\textit{profit / revenue} \times 100\% = (325.00 - 212.60)/325.00 \times 100\% = 34.58\%$$

and for the value stream as a whole it would be:

- The value stream revenue is the revenue of the value stream (from Table 5.26) plus the revenue obtained from the new PCB type per month:

  $$901,470 + (325.00 \times 350) = £1,015,220$$

- The value stream cost is the cost of the value stream (calculated in subsection 5.8.4) plus the cost of the new PCB type per month:

  $$450,557 + (212.60 \times 350) = £524,968$$

This means, the profit rate of the value stream without any improvements made would be:

$$\textit{profit / revenue} \times 100\% = (1,015,220 - 524,968)/1,015,220 \times 100\% = 48.29\%$$

as shown in Table 5.39.

Table 5.39. Calculations of the target costs for the case study

	New product	Current value stream	Future state
Allowable cost	£162.43	£158.09	£158.57
Conversion cost	£43.95	£38.61	£39.19
Material costs	£168.65	£119.48	£124.86
Total costs	£212.60	£158.09	£164.05
Cost gap	£50.17	£0.00	£5.49
No. of products	350	2850	3200
Current value stream cost	£74,411	£450,557	£524,968
Target value stream cost	£56,853	£450,557	£507,410
Cost gap	£17,558	£0	£17,558
Profit rate	34.58%	50.02%	48.29%

The figures in the second column of the table are already calculated above apart from the target value stream cost, which can be calculated as follows:

$$162.43 \times 350 = £56,853$$

As for the third column, the same calculations performed for the second column can be performed here after calculating the average selling price, which is calculated as follows:

$$901,470 / 2850 = £316.31$$

and calculating the average cost of materials, which is calculated as follows:

$$340,510 / 2850 = £119.48$$

The revenue and the total cost of materials are assumed to be known as presented in Table 5.26. Regarding the future state column, the number of products, the current value stream cost, the target value stream cost and the cost gap are all calculated by adding up the corresponding amounts of the new product and the current value stream columns. As for the rest of the amounts, they are calculated as follows:

- The allowable cost is calculated by dividing the target value stream cost by the number of products:

$$507,410 / 3200 = £158.57$$

- The conversion cost can be calculated as follows:

  (conversion cost of the new product × amount of new product + conversion cost of the current value stream × amount of current value stream) / amount of the future state

  $$(43.95 \times 350 + 38.61 \times 2850) / 2300 = £39.19$$

- The total costs is calculated by dividing the current value stream cost by the number of products:

  $$524,968 / 2300 = £164.05$$

- The material costs is calculated by deducting the conversion cost from the total cost:

  $$164.05 - 39.19 = £124.86$$

The following lean improvements have been introduced to the PCB value stream in order to eliminate the cost gap in the value stream:

- Reduction in the cost of materials due to lean processes (pull systems, short lead time, etc.), which have been agreed with the suppliers.
- Reduction in the conversion costs due to a reduction in the waste rate of the value stream cells (as explained in subsection 5.8.2) and the elimination of some transactions (as explained in subsection 5.8.3).

Assuming that the improvements have reduced the conversion and the material costs by 3.44% and 3.10% respectively for both the new PCB type and the current PCBs, this means the new conversion and the material costs can be calculated:

- For new PCB type:

  $$43.95 \times (100 - 3.44\%) = £43.44$$

  $$138.65 \times (100 - 3.10\%) = £163.42$$

- For current PCBs:

  $$38.61 \times (100 - 3.44\%) = £37.28$$

  $$119.48 \times (100 - 3.10\%) = £115.77$$

Following the same calculations performed to obtain the figures presented in Table 5.39, the new figures, after the improvements have been applied to the PCB value stream, can be obtained as shown in Table 5.40. The table shows how the cost gap in

the value stream is reduced to £846 and the profit rate is increased from what it would have been without the lean improvements to 49.94%.

Table 5.40. Financial impact of the introduction of the lean improvements

	New product	Current value stream	Future state
Allowable cost	£162.43	£158.09	£158.57
Conversion cost	£42.44	£37.28	£37.85
Material costs	£163.42	£115.77	£120.98
Total costs	£205.86	£153.05	£158.83
Cost gap	£43.42	-£5.04	£0.26
No. of products	350	2850	3200
Current value stream cost	£72,051	£436,205	£508,256
Target value stream cost	£56,853	£450,557	£507,410
Cost gap	£15,198	-£14,352	£846
Profit rate	36.66%	51.61%	49.94%

In general, the lean improvements introduced may not necessarily improve the situation to the extent required by the company. In this case study, since the lean improvements have not been sufficient to reach the company target, the company could decide not to introduce the new product. However, implementing lean improvements is required even when no new products are introduced. In fact, the process of implementing lean improvements is an ever continuous process and is necessary for the survival of the company in an ever-competitive manufacturing world.

5.8.7. Financial planning

Lean accounting, in common with other accounting systems, requires planning so that the future customer needs can be assessed in order to prepare the required resources and capacities to meet these needs. Furthermore, the business should be flexible enough, in terms of capacity, in order to cope with unexpected situations. In contrast to traditional annual budgeting, lean planning is more flexible and can be updated when needed (often regularly). Lean planning process gives the managers the required information to successfully manage the business in a proactive way. Lean financial planning is a team-work process. It involves the cooperation of cross-functional people in the value stream: sales and marketing, finance, engineering and operations. The end result of this cooperation is a plan that includes sales, new products introduction,

operation capacity and finance for each value stream. The plan is put into action, which includes short- and long-term actions. This plan is updated periodically (usually monthly) in order to reflect changing customer needs.

5.9. Activity-based costing versus lean accounting

Activity-based costing and lean accounting have been the main subjects of Chapter 5 in which they have been applied to the case study. However, here are some problems associated with them that should be outlined. The main problem with ABC is that it requires a great deal of work to collect the relevant data for its implementation. This problem is more noticeable when the organisation is large and produces high number of product types. In fact, a large organisation requires a team of full-time people to collect the necessary data for ABC implementation. This is a wasteful and time-consuming process and it may force organisations experimenting with ABC to abandon it. Another problem associated with ABC is that ABC allocates costs to products on the basis of cost drivers that may not be proportional to the volume of the output. Furthermore, it can be argued that ABC, compared to LA, is a method for cost estimation not for cost reduction. It helps provide better understanding of how costs are incurred and allocated to products but it does not provide a plan for minimising these costs. It is up to the management to find a way of doing that having understood how the costs are incurred.

Kaplan, who is the cofounder of ABC, acknowledges the complexity of implementing ABC and how it is difficult to sustain it over time (Kaplan & Anderson 2003). The solution to this problem is to find other methods that are less data demanding and more lean based. Bearing this in mind, Kaplan suggested a new approach to ABC. This new approach is time and capacity based and has some similarities to lean accounting. Maskell (2006) presented those similarities as shown in Table 5.41. This new approach shows that ABC is evolving so that it can solve the problems associated with it on one hand and adopts the principles of lean on the other hand. In other words, the gap between ABC and lean accounting is continuously narrowing and soon a new hybrid system which has the positive features of both could be developed.

As for lean accounting, it has also its own problems. The lean accounting way of calculating the cost of a particular product is represented by 'features and characteristics costing'. As explained in section 5.8.5, this method relies on dividing the product types

into groups of similar products depending on their features and characteristics. The cost of the products in each group is then calculated assuming that all the products in this group have the same cost. In reality, this assumption may or may not be accurate and when it is not, this could eventually lead to wrong decisions being made.

Table 5.41. The similarities between new ABC and lean accounting

	New ABC	Lean Accounting
Cost allocation	Collected by departments within the processes	Step by step allocation as in value stream costing
Product cost	Calculated according to the capacity required to perform the job within the process	Considering the rate of flow through the bottleneck operation
Capacity	Used capacity and unused capacity	Productive, non-productive and available capacity
Cost	Process cost	Value stream cost
Activity features	Incorporating activity characteristics that cause processing time to vary	Features and characteristics costing

5.10. Summary

ABC and LA have been presented in this chapter as two examples for the cost estimation and accounting aspects of this research. The implementation procedures of both methods have been studied and applied to a case study. The implementation process of ABC on the case showed how ABC could be used to analyse the production process and perform the right steps to understand and potentially reduce the production costs. In the case study, ABC was used to identify the activities used for producing PCBs and the cost of each activity. This allowed for more attention to be paid to the most costly ones in order to reduce their costs. The most costly activities were found to be "placing components", "programs and fixtures", "testing" and "rework". As shown in Chapter 4, it was possible to reduce the cost of the "placing components" activity by optimising the pick-and-place machine. This optimisation was achieved through optimising the component placement sequence and feeder assignment. The use of ABC in this case study has proven how it could be successfully used for estimating the cost of the PCB production.

The importance of implementing lean accounting alongside lean manufacturing and how it could be achieved were explained and the explanation was supported by examples for illustration purposes. It was illustrated how lean accounting could help companies implementing lean manufacturing see the financial benefits of such implementation in addition to the operational benefits. The case study clarified the implementation steps of lean accounting and made it easier for the reader to comprehend what lean accounting was about. It explained the financial benefits of implementing lean principles. Finally, the problems associated with activity-based costing and lean accounting were outlined and how a new activity-based costing approach was being developed.

CHAPTER SIX

6. RESEARCH VALIDATION & EVALUATION

6.1. Introduction

In this chapter, the validation of the results obtained in this research is presented. This is achieved by comparing the results and the approaches used in this research to similar research works in the literature. In addition, the importance of this research and its contributions to the advancement of knowledge are outlined.

6.2. Research validation

In this section, the proposed algorithm to solve the PCB related three problems is validated against other algorithms used in the literature to solve PCB related problems. Since the three PCB problems considered in this research has not been solved simultaneously before, the results obtained in this research will be compared, where available, to the results obtained by researchers who simultaneously solved two (feeder assignment and component sequencing) of the three PCB problems considered in this research.

6.2.1. Validation of the work on the optimisation of production processes

The most similar study in the literature to the case considered in this research is that of Su *et al* (1998). In their study, Su *et al* presented a TS-based approach to obtain the shortest (or near shortest) cycle time, feeder assignment and component sequencing based on a dynamic pick and place (DPP) robot motion model. In a DPP model, the robot moves in two directions (X and Y axes) and the board and the feeders move in one direction (X-axis). This means that Su's study is different from the research presented in this thesis in the number of the problems solved as Su focused on the feeder assignment and the component sequencing problems, however, in this research work an additional problem (board type sequencing) was considered. Another difference between the two studies is the movement of the board and feeders (mobile in Su's and fixed in this research). The comparison between the two cases, because of these two differences, will be of limited value and importance. Therefore, the case considered in this research is adjusted as follows so that a comparison can be made.

The board type sequencing is affected by the set-up time and the feeder assignment. By setting the set-up time in the program to zero, the problem of board type sequence is eliminated and it is no longer affecting the results. Regarding the issue of fixed board and feeders, equation (4.6) in Chapter 4 is still applicable, however, the set-up time is now equal to zero and the way the time calculated in the program is adjusted to take into account the movement of the board and the feeders. Now, the board and the feeders move horizontally (x-axis) at a speed equal to the speed of the machine head (500 mm/sec).

Su *et al* compares their results to the work of Wang *et al* (1997 cited in Su *et al* (1998)) and they claim that their results are better than Wang's (who considered the feeder assignment problem only). In the case where the number of insertion points is 30 and the number of component types is 15, the cycle time achieved by Su's approach is 14.26% less compared to the cycle time achieved by Wang's approach. However, the average improvement for all the combinations considered is 9.08%. The average improvement to the assembly time achieved in this research is 5.96% and after adjusting the case to be similar to Su's study, the improvement has increased to 7.87%. Although this percentage is less than the results achieved by Su *et al* (1998), but this does not necessarily mean that Su's approach is superior to the approach adopted in this research. Su's TS-based approach does not consider two of the basic attributes of TS: intermediate and long-term memories (diversifications and intensification). Taking these two attributes into consideration in this research means that the TS algorithm adopted in this research should be superior to Su's. Unfortunately, the results obtained do not support this claim because the data used in this research are different from that of Su's since the author did not have access to Su's data in order to use them.

It should be noted that the increase of the improvement in assembly time from 5.96% to 7.87% after eliminating the board type sequence problem and considering a moving table and moving feeders, leads to the belief that the movement of the board and feeders have positively affected the TS algorithm. A possible reason could be that the movement of the board and feeders have provided the TS algorithm more room for manoeuvre and, hence, more chance to improve the assembly time.

As for the GA algorithm, it has been used intensively in the literature to solve PCB problems and using it here for the same goal (solving PCB problems) would be of little usefulness. Therefore, it is used in this research mainly for comparison reasons.

The results obtained from using GA are compared to that obtained from using TS and the effects of changing the parameters of the algorithm are also considered as presented in Chapter 4. Therefore, a validation process is not necessary for GA algorithm as performed for TS algorithm.

6.2.2. Validation of the work on the cost estimation aspect

As mentioned in Chapter 5, there has not been much work on the implementation of ABC in PCB manufacturing facilities. However, the work of Ong (1995) is considered to be the most suitable to compare to this work due to the fact that Ong applied his work on a case study which is quite similar to the research presented in this thesis. Ong uses a different way of implementing ABC compared to what is used in this research. He uses worksheets, activity charts and a cost build-up table to calculate the cost of PCBs with the aim of allowing designers to estimate the cost of PCBs at the design stage. However, in this research the implementation is a four-step process as presented in section 3.4.

Since the data used by Ong are different from the data used in this research, the comparison between the results obtained in both research works can be carried out only when the costs of activities are represented as percentage values as shown in Table 6.1. For reasons beyond the author's knowledge, some of the activities in the example presented in Ong's research have no cost (e.g. Burning-in, Applying adhesive, designing, etc.). In spite of this and in addition to the 10-year gap in time between the two research works, there are similarities between the costs of activities in Ong's example and the costs of activities in the case study presented in this research. As can be seen from Table 6.1, the three activities of the highest costs are the same (Placing components, Programs & fixtures and Testing) in both cases. However, the notable difference is that the cost percentage is higher in Ong's example which could be due to the technological advantage available these days. As for the rest of activities, there are some similarities in the costs (e.g. Rework, Soldering, loading & unloading, etc.) and some differences (e.g. Inventory holding, Screen printing, Curing & baking, etc.), which is bound to exist due to the differences in the data used (e.g. different number of components, different layout, etc.) between Ong's example and the case study presented in this research.

The problem with Ong's approach is its dependence on worksheets and activity charts that have to be updated continuously and the fact that they are suitable for PCB

cases only. The worksheets, which describe the components, their placement, set-up costs, etc., have to be updated since new types of components are introduced continuously. In addition, the continuous technological advancement means that the costs of components are continuously changing. The same discussion applies to the activity charts since they require updating when new components or new placement machines are introduced. Updating the worksheets and activity charts is not an easy process; it requires intensive experimentation to obtain accurate information.

Table 6.1. Costs of activities (in percentage) in this research and in Ong's research

Activity	This research	Ong's research
Placing components	14.46%	26.83%
Programs & fixtures	11.88%	25.82%
Testing	9.37%	12.80%
Rework (repair)	5.78%	6.04%
Burning-in	5.34%	0.00%
Inventory holding	5.30%	1.91%
Material handling	5.26%	n/a
Soldering	5.18%	4.94%
Screen printing	4.83%	0.43%
Applying adhesive	4.83%	0.00%
Visual & touch-up	4.18%	7.61%
Curing and baking	3.37%	0.14%
Cleaning	3.37%	1.13%
Kitting/other operations	3.11%	6.37%
Designing	2.93%	0.00%
Setting-up	2.54%	0.88%
Loading & unloading	2.06%	2.56%
Place-&-route design	1.99%	0.00%
Acceptance sampling	1.59%	1.41%
Sequencing parts	1.02%	0.00%
Purchase order	0.98%	0.96%
Inventory retrieval	0.62%	0.17%
Total	100.00%	100.00%

The deficiency of Ong's approach explained in the previous paragraph is not an issue in the approach used in this research. Any changes to the component types or to the machines used, or even any changes introduced to the facility which affect the production cost, are reflected during the implementation of the four-step process explained in section 5.4. In addition, the implementation process can be tailored to suit

different manufacturing facilities other than the manufacturing of PCBs. These advantages give the implementation process considered in this research the upper hand when ABC implementation is considered.

6.2.3. Validation of the work on the accounting aspect

Lean Accounting has been considered in this research as an example to show how it is possible to improve the accounting system in order to reduce the cost on one hand and show the financial benefits of implementing lean manufacturing on the other hand. Since LA is a relatively recent subject, not much research has been designated to it. In fact, the only substantial work about LA with a full detailed case study is the work of Maskell and Baggaley (2004). In their work, Maskell and Baggaley present what they call "a proven system for measuring and managing the lean enterprise". The LA implementation process considered in this research is adopted from Maskell and Baggaley's work. Table 6.2 summarises the improvements achieved after implementing LA on the case studies presented in this research and in Maskell and Baggaley's. The improvement percentage is calculated by dividing the amount of change between the future state and the current state in the box score presented in Chapter 5 (Table 5.32) by the current state. For example, the improvement percentage for the Dock-to-dock days can be calculated as follows:

Table 6.2. Improvements achieved by implementing LA on this research and on Maskell and Baggaley's research

		This research	Maskell and Baggaley's research
Operational	Dock-to-dock days	-74.00%	-78.05%
	First time through	64.84%	100.00%
	On time delivery	10.00%	10.00%
	Floor space	-31.25%	-50.00%
	Sales per person	0.00%	4.56%
	Average cost per unit	-6.73%	-5.99%
Resource capacity	Productive	-7.25%	-15.00%
	Non productive	-34.91%	-41.94%
	Available	120.32%	161.11%
Financial	Inventory value	-77.26%	-76.07%
	Revenue	0.00%	0.00%
	Material costs	-6.83%	-6.83%
	Conversion costs	-6.42%	-4.45%
	Value stream profit	6.73%	7.36%

$(-18.5/25)\times 100\% = -74.00\%$,

which means the Dock-to-dock days were reduced by 74%.

As can be seen from the table, the improvements achieved in both research works are similar to some extent. In both case studies, the implementation of LA improved the status of the company considered in the case study financially and operationally.

6.3. Research evaluation

The evaluation of this research is presented in this section, where alternatives to how it is introduced are considered and its importance and the contribution it has made to the advancement of knowledge are also discussed. Each of the three aspects of this research is considered separately at first and the three aspects are then considered as a whole.

6.3.1. Evaluation of the work on the optimisation of production processes

The optimisation of production processes part of this research focuses on process optimisation and, hence, on cost reduction in the manufacturing industry. PCB manufacturing has been chosen as an example and three production problems have been solved using the proposed algorithm in Chapter 4. The problems associated with PCB manufacturing have been considered by many researchers, as presented in the literature review. Some of these problems are interrelated and should be solved simultaneously rather than individually. Therefore, most researchers have solved PCB assembly problems in pairs such as the component placement and feeder assignment problems. A further step has been considered in this research by considering a three-problem situation. Since the problem of board type sequence is affected by the set-up time, this means it is related to the component placement and feeder assignment problems and any solution to this problem should take the other two problems into consideration. This research considers solving the three problems simultaneously, which is a development to what other researchers have considered before.

The algorithm used to solve the three production problem is based on two metaheuristics: TS and GA. Other metaheuristics (simulated annealing, mimetic algorithms, random optimisation, local search, greedy algorithm, etc.) could have also been used instead. TS is used because it has been successfully used to solve combinatorial problems (Saad & Lassila 2002; Wan & Ji 2001; Su *et al* 1998) on one hand and it has been rarely used to solve PCB related problems so it is used here to

assess its potential on the other hand. GA is used mainly for comparison reasons. Since GA has been used frequently to solve PCB manufacturing problems (Ho & Ji 2005; Ho & Ji 2003; Jeevan *et al* 2002; Deo *et al* 2002; Ji *et al* 2001; Loh *et al* 2001), it is used in this research in order to compare the results obtained from using it to the results obtained from using TS algorithm. This comparison between TS and GA, and even between different parameters within each metaheuristic, is a new subject that provides recommendations to which metaheuristic to use and which values to be used for the parameters of the metaheuristic used.

6.3.2. Evaluation of the work on the cost estimation aspect

Cost estimation can be calcified into four types (Rabuñal & Dorado 2005):

- Analogical: the cost of a product is estimated by comparing it to the cost of a similar product whose cost is known.
- Analytical: the product cost is estimated by analysing the product manufacturing process and dividing the process into small tasks whose costs are either known or can be calculated easily compared to the whole process. The information gathered is then integrated to provide the cost of the product.
- Intuitive: this type of cost estimation depends on experience. A person experienced in cost estimation estimates the cost of a product depending on his past knowledge.
- Parametric: this method relies on the parameters used by the designers of the product. The relationships between the parameters and the cost of the product are represented by equations, which are then used to calculate the cost of the product depending on the values of the parameters used to design the product.

Each of the four types has its own pros and cons. The analogical method provides good estimation but it is limited to the existence of a similar product to the product that its cost being estimated. The problem with the intuitive method is its reliance on personal judgments; whereas the problem with the parametric method is the lack of accuracy since the equations used are approximate. As for the analytical method, it has attracted more attention than the other types of cost estimation methods due to the fact that it lacks the deficiencies associated with them. For this reason, ABC, which is an analytical method, is used in this research as an example for cost estimation. In addition, since ABC is more suitable for small and medium-sized enterprises

(Gunasekaran *et al* 1999), it is therefore more suitable for the PCB manufacturing example considered in this research.

The ABC implementation procedures adopted to in this research can also be used to further examine the activities identified and decompose them into sub-activities (Ben-Arieh & Qian 2003). This is important because it helps study costly activities in more detail and, hence, identify the exact cause of the high cost, which ultimately leads to a reduction in the cost of these activities.

6.3.3. Evaluation of the work on the accounting aspect

The standard accounting system has been used successfully for mass productions for few decades now. However, the introduction of lean manufacturing as an alternative to mass production has led to changes in the accounting system to reflect the new features of lean manufacturing. A new accounting system, called Lean Accounting, that takes into consideration the features of lean manufacturing has been developed. This research has considered LA because of the following reasons:

- It is a relatively recent subject and quite limited research has been devoted to it.
- It is the only alternative for standard accounting.
- It is designed to suit lean manufacturing, which is rapidly replacing mass production.
- It has the potential of reducing the cost of production by implementing lean techniques and eliminating financial transactions (or replacing them with less costly ones).

The implementation of LA in this research has shown how it can reduce the production cost as follows:

- Cell and value stream performance measures provide visual indications about the operational state of the cells and value streams. This allows for solving any emerging problems at an early stage before they become more costly to solve. In addition, performance measures promote continuous lean improvements throughout the company.
- Calculating the free capacity of cells generated by the implementation of lean manufacturing. The freed capacity can be used to increase the value stream profit as explained in subsection 5.8.2.
- Eliminating wasteful financial transactions. This leads to eliminating the costs required to perform these transactions.

- The use of value stream costing provides more accurate and easily-understood information, when compared to standard costing, for decision-making. Butter decisions lead, amongst other things, to cost reductions.

In addition to the cost reduction, LA provides better understanding to the financial information due to the simplification of the accounting reports and statements. For example, the box score provides a simple but comprehensive view of the state of value stream: operational, financial and capacity usage. Therefore, LA can be defined as an accounting system that implements the principle of lean thinking leading to more accurate and understandable information for decision making and leading also to the elimination of some of the transactions associated with standard accounting.

6.3.4. Evaluation of the work on the research as a whole

The intense competition between companies has led cost reduction to be considered as a necessity rather than a luxury. There are two different types for cost reduction, direct and indirect, and each type can be applied to different areas in the company. Therefore, this research has considered cost reduction in the manufacturing industry by studying three aspects: optimisation of production processes, cost estimation and accounting. These three aspects include most of the areas in a company that cost reduction can be applied to. The cost reduction in the production area has been considered by optimising three production problems in the PCB industry. However, the TS- and GA-based algorithm used to provide the near optimum solution to the problems can also be applied to other combinatorial problems in other types of industry. The scheduling problem, for example, is a combinatorial problem that can be found in many types of industries such as the chemical industry, the automotive industry, the manufacturing industry, the food industry, etc. Therefore, the algorithm used in this research can be modified to suit optimisation problems other than the problems mentioned in this research.

As for the cost reduction in the cost estimation and accounting areas, this research has considered ABC as a cost estimation method and LA as an accounting system to reduce the cost in these two areas. ABC can reduce the cost by providing the right costing information for the management. This is achieved by analysing the production processes and studying the activities required to manufacture the product. By calculating the costs of these activities, the product cost can be calculated depending on how much the activities contribute to the manufacturing of the product. ABC shows

exactly how the costs are incurred, therefore allowing the management to pay more attention to costly activities and try to reduce their costs. This means, in addition to being a cost estimation method, ABC reduces the production cost indirectly. Regarding LA, it reduces the cost directly and indirectly as mentioned in subsection 6.3.3.

The importance of this research lies also in the fact that it has presented a framework for PCB manufacturing process. As described in Chapter 4 and Chapter 5, the framework provides an integrated and comprehensive description of the processes that can be used to help solve the production problems, optimise the work and reduce the production cost. The framework can also be used for other products when the production problems associated with PCB are replaced with the production problems associated with the other products.

6.4. Summary

In order to validate this research, the results obtained in this research were compared, where possible, to the results obtained by other researchers. Regarding the optimisation of production processes, the case considered in this research had to be modified so that it could be compared to that of Su *et al* (1998). Since the data used in Su's research were different from the data used in this research, the comparison between the two sets of results was found to be inconclusive. However, the use of intensification and diversification (two important attributes of TS) in this research and ignoring them by Su *et al* should indicate that the TS-based algorithm used in this research is better than that of Su's. As for the cost estimation aspect, when the results obtained in this research were compared to the work of Ong (1995), the similarities that were found surpassed the dissimilarities. It was also noted that the approach used here to implement ABC was better than the approach used by Ong because it could be used for products other than PCBs and it did not require continuous updating as was required for Ong's approach. Finally, regarding the accounting aspect, the results obtained in this research were similar to the results obtained by Maskell and Baggaley (2004) since the same implementation process was used. The results in both research works revealed how LA could help reduce the production costs directly and indirectly.

The evaluation of this research was also considered in this chapter. Some alternatives to the approaches and methods used were considered and the reasons of considering some particular techniques were explained. For example, it has been explained that TS was used instead of other metaheuristics because it was used

successfully to solve combinatorial optimisation problems but rarely used to solve PCB related problems. In addition, the importance of this research has been identified and explained. It has been explained how the algorithm presented in this research could be used to solve PCB related problems in particular and other combinatorial optimisation problems for other types of industry in general. Furthermore, the framework presented in this research for PCB manufacturing can also be applied to other products by changing the part that deals with the production problems and generalise it to suit other manufacturing processes.

7. CONCLUSION

The issue of cost reduction is a necessity for today's industries due to the high competition created by the advancement of technology and the emergence of new economic powers. Cost reduction can be achieved either directly (waste elimination) or indirectly (process optimisation). The majority of research on cost reduction has been devoted to specific and individual aspects rather than considering multiple aspects. The research presented in this thesis has attempted to fill this gap by considering the issues associated with the optimisation of production processes, cost estimation and accounting aspects of manufacturing in general and multi-assembly systems in particular with the aim of achieving process optimisation and cost reduction.

PCB manufacturing has been used in this research as a platform for experimentation for the three aspects considered in this research. The reason behind using PCB manufacturing is mainly related to the optimisation of production processes part of this research. The vast collection of PCB related production problems provides an ideal test-bed for the optimisation of production processes part of this research. All the researchers who considered PCB production problems have focused on these production problems individually or in pairs. Therefore, a further step has been taken here by studying three interrelated problems. Additionally, two examples for the cost estimation and accounting aspects considered in this research have been applied to PCB manufacturing. This is achieved by implementing ABC, as an example for the cost estimation aspect, and LA, as an example for the accounting aspect, on a PCB manufacturing facility. Ultimately, the goal has been to develop a framework that integrates the optimisation of production processes, cost estimation and accounting aspects for PCB manufacturing in particular and for manufacturing in general in order to achieve cost reduction.

For the rest for this chapter, the main findings of this research are stated for each of the aspects considered and the contributions to the literature are summarised. In addition, the suggestions for future work including the reasons behind these suggestions, the methods involved and the expected outcomes are presented.

7.1. Optimisation of production processes

A proposed algorithm for concurrently solving the components sequencing, feeder assignment and board sequencing problems has been developed based on two search techniques: TS and GA. The results obtained from the case study show that the use of TS is preferable to GA when the goal is to obtain a better assembly time, whereas GA is preferable if the goal is to obtain acceptable results within short time. Regarding the parameters of the search techniques such as the number of moves in TS and number of generations and the population size in GA, the results obtained showed a reduction in the assembly time when these parameters were increased. The reduction in the assembly time was limited to a certain level after which no improvement was achieved in spite of the increase of the number of moves in TS and number of generations and the population size in GA. It could be concluded that the ideal values for the parameters are dependent on the type of the case study and the data used and should be obtained experimentally.

The use of different data in the case study has limited the comparison to the other research work used to validate this research. However, the use of intensification and diversification in the TS-based algorithm in this research has given it an advantage over the TS-based approach adopted in the research work used for validation. Unfortunately, this could not be proved by comparing the two research works due to the use of different sets of data in each one.

7.2. Cost estimation aspect

ABC has been used as an example for cost estimation and it has been applied to a PCB production facility. The results obtained from the case study show how ABC can be used to calculate the costs of activities so that the most costly ones can be given more attention and more effort can be taken to reduce their costs. A reduction in the costs of the most costly activities would achieve more cost reduction than the reduction in the less costly activities because the margin for reduction is higher. ABC is not a method for cost reduction as such but it leads to cost reduction indirectly by identifying the costs of activities.

There are some downsides for ABC; for example, in order for the application of ABC to be accurate, there must be a linear relationship between the cost drivers and the volume of the product. In addition, ABC is not suitable for large companies due to the

resources it requires to collect required data. In spite of these downsides, ABC is still a better alternative to standard costing.

7.3. Accounting aspect

Lean accounting has recently been introduced as an accounting system for those companies which have chosen to implement lean manufacturing. The research explained the basic ideas behind lean accounting and illustrated how lean accounting could help companies identify the financial benefits of implementing lean manufacturing. The results from implementing LA on a case study show how implementing lean accounting can help identify the financial benefits of implementing lean manufacturing and how lean accounting itself could lead to cost reduction through eliminating some financial transactions.

Standard accounting relies heavily on a huge number of transactions whereas LA tries to eliminate some of these transactions or replace them with less costly ones. This, hence, does not only lead to creating capacity but also leads to saving money by eliminating the direct and indirect costs that are associated with the transactions eliminated. LA, in contrast to standard accounting, motivates people towards actions that are in accordance with lean manufacturing principles. This allows lean manufacturing improvements to continue smoothly and gives a better chance for lean manufacturing to succeed and for the company to prosper. LA provides also more accurate and more useful information for decision-making because better tools, such as value stream costing, target costing and sales, operations and financial planning, are used in LA.

It can also be concluded from this research that there is a deficiency in calculating the exact costs of products using "features and characteristics costing", which is an LA tool. Some might argue that the exact product cost is not required in lean companies since they focus on the customer value and the value stream profitability rather than the profitability of a product when making pricing decisions. However, there are cases when the exact cost of a product is required and the inaccuracy in calculating this cost by "features and characteristics costing" might lead to the wrong decision being made.

7.4. Final thoughts

Cost minimisation has been forced on manufacturers due to the development of new technologies and the high competition created by the emergence of new

competitors. The research presented here represents a good first step towards cost minimisation and process optimisation in the production, cost estimation and accounting aspects of the manufacturing industry. The framework developed presents an integrated approach that can help solve production problems, with the aim of minimising the production cost, and provide helpful information for the management in order to make the right decisions. Some of the solutions provided are problem specific; therefore, the generalisation issue needs to be addressed. In addition, regarding the accounting aspect, this research has considered the companies that adopt lean manufacturing; therefore, what has been discussed here is not applicable otherwise.

7.5. Future work

Although the research undertaken has provided a step forward towards cost reduction and process optimisation in different areas of manufacturing, there are some issues that can still be addressed in order to add more value and benefit to the issues considered in this research.

This research has considered the area of PCB manufacturing only. Therefore, the generalisation issue can be proposed as the subject of future research. This can be achieved by applying this research project on products of different nature to PCBs, and the use of real-life data will be an added benefit. Naturally, the proposed algorithm that deal with the production problems will have to be modified to suit the production problems associated with the new products under consideration. Additionally, search techniques other than TS and GA can also be used. If the outcome of the future research is similar to the outcome of the research presented in this thesis, the research can be generalised and confidently applied to different industries.

In this research, the proposed algorithm has dealt with the problems associated with the pick-and-place machine only and the improvements achieved are related to this machine only. However, any effects caused by the improvements to the pick-and-place machine on other cells have not been considered in this research. By modelling the whole PCB assembly line using computer simulation, the effects of the improvements to the pick-and-place machine on other cells can be studied. Additionally, introducing simulation can also help implement ABC (Spedding & Sun 1999) and improve the accuracy of cost estimation as well (Homburg 2004; Özbayrak *et al* 2004). Therefore, integrating simulation into this research can be considered as a potentially good subject for any future work.

REFERENCES

Agrawal, A. & Graves, R.J., (1999). A distributed systems model for estimation of printed circuit board fabrication costs. *Production Planning & Control*, 10(7), p.650-658.

Åhlström, P. & Karlsson, C., (1996). Change processes towards lean production – the role of management accounting system. *International Journal of Operations & Production Management*, 16(11), p.42-56.

Ahmadi, J.H., Ahmadi, R., Matsuo, H. & Tirupati, D., (1995). Component fixture positioning for printed circuit board assembly with concurrent operations. *Operations Research*, 43(3), p.444-457.

Altinkemera, K., Kazazb, B., Köksalanc, M. & Moskowitza, H., (2000). Optimization of printed circuit board manufacturing: Integrated modeling and algorithms. *European Journal of Operational Research*, 124(2), p.409-421.

Anonymous: Maricopa Center for Learning and Instruction, (1989). Research methods [Online]. (Updated 26 Oct 2001) Available at: http://www.mcli.dist.maricopa.edu/proj/res_meth/login.html [accessed 08 Oct 2007].

Applegate, D. & Cook, W., (1991). A Computational Study of the Job-Shop Scheduling Problem. *ORSA Journal on Computing*, 3(2), p.149-157.

Ayob, M. & Kendall, G., (2002). A new dynamic point specification approach to optimise surface mount placement machine in printed circuit board assembly. In: *IEEE ICIT'02*, Bangkok.

Baggaley, B., (2003a). Solving the standard costing problem [Online]. Available at: http://www.maskell.com/SolvingStdCost.htm [accessed 15 May 2005].

Baggaley, B., (2003b). Value Stream Management for Lean Companies. *The Journal of Cost Management*, 17(2), p.23-27.

Ball, M.O. & Magazine, M.J., (1988). Sequencing of Insertions in Printed Circuit Board Assembly. *Operations Research*, 36(2), p.192-201.

Bellis-Jones, R. & Develin, N., (1999). *No Customer – No Business: The true value of activity based cost management.* Chippenham, UK: Antony Rowe Limited.

Bellmore, M. & Nemhauser, G.L., (1968). The traveling salesman problem: a survey. *Operations Research*, 16(3), p.538-558.

Ben-Arieh, D. & Qian, L., (2003). Activity-based cost management for design and development stage. *International Journal of Production Economics*, 83(2), p.169-183.

Berliner, C. & Brimson, J.A., (1988). *Cost Management for Today's Advanced Manufacturing: The Cam-I Conceptual Design.* Harvard: Harvard Business School Press.

Bhaskar, G. & Narendran, T.T., (1996). Grouping PCBs for set-up reduction: a maximum spanning tree approach. *International Journal of Production Research*, 34(3), p.621-632.

Burkard, R.E., Karisch, S.E. & Rendl, F., (1997). QAPLIB – A Quadratic Assignment Problem Library. *Journal of Global Optimization*, 10(4), p.391-403.

Burke, E., Cowling, P.I. & Keuthen, R., (1999). New Models and Heuristics for Component Placement in Printed Circuit Board Assembly. In: *International Conference on Information Intelligence and Systems*

Burke, E., Cowling, P.I. & Keuthen, R., (2000). Effective Heuristic and Metaheuristic Approaches to Optimize Component Placement in Printed Circuit Board Assembly. In: *The 2000 Congress on Evolutionary Computation CEC00.*

Christensen, J. & Demski, J.S., (1995). The classical foundations of 'Modern' costing. *Management Accounting Research*, 6(1), p.13-32.

Coombs, J., C. f., (1988). *Printed Circuits Handbook.* 3rd ed. New York: McGraw-Hill, Inc.

Cooper, R., (1990). Cost classification in unit-based and activity-based manufacturing cost systems. *Journal of Cost Management for the Manufacturing Industry*, 4(3), p.4-14.

Cooper, R. & Kaplan, R.S., (1999). *The design of cost management systems : text and cases.* 2nd Edition ed. Upper Saddle River, NJ: Prentice Hall.

Crama, Y., Klunder, J.V.D. & Spieksma, F.C.R., (2002). Production planning problems in printed circuit board assembly. *Discrete Applied Mathematics*, 123(1-3), p.339-361.

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* 26(1-4), p.455-480.

Dedera, C.R., (1996). Harris Semiconductor ABC: worldwide implementation and total integration. *Journal of Cost Management*, 10(1), p.44-58.

Deo, S., Javadpourb, R. & Knapp, G.M., (2002). Multiple setup PCB assembly planning using genetic algorithms. *Computers and Industrial Engineering*, 42(1), p.1-16.

Deoa, S., Javadpourb, R. & Knapp, G.M., (2002). Multiple setup PCB assembly planning using genetic algorithms. *Computers and Industrial Engineering*, 42(1), p.1-16.

Dikos, A., Nelson, P.C., Tirpak, T.M. & Wang, W., (1997). Optimization of High-mix printed circuit card assembly using genetic algorithms. *Annals of Operations Research*, 75, p.303-324.

Drezner, Z. & Nof, S., (1984). On optimizing bin picking and insertion plans for assembly robots. *IIE Transactions*, 16, p.262-270.

Duverlie, P. & Castelain, J.M., (1999). Cost Estimation During Design Step: Parametric Method versus Case Based Reasoning Method *The International Journal of Advanced Manufacturing Technology*, 15(12), p.895-906.

Egbelu, P.J., Wu, C. & Pilgaonkar, R., (1996). Robotic assembly of printed circuit boards with component feeder location consideration. *Production Planning and Control*, 7(2), p.162-175.

Francis, R.L., Hamacher, H.W., Lee, C.Y. & Yeralan, S., (1994). Finding placement and sequences and bin locations for Cartesian robots. *IIE Transactions*, 26, p.47-59.

Giachetti, R.E. & Arango, J., (2003). A Design-centric Activity-based Cost Estimation Model for PCB Fabrication. *Concurrent Engineering: Research and Applications*, 11(2), p.139-149.

Glover, F., (1989). Tabu Search – Part I. *ORSA Journal on Computing*, 1(3), p.190-206.

Glover, F., (1990). Tabu Search – Part II. *ORSA Journal on Computing*, 2(1), p.4-31.

Gronalt, M., Grunow, M., Günther, H.O. & Zeller, R., (1997). A heuristic for component switching on SMT placement machines. *International Journal of Production Economics*, 53(2), p.181-190.

Gunasekaran, A., Marri, H.B. & Grieve, R.J., (1999). Justification and implementation of activity based costing in small and medium-sized enterprises. *Logistics Information Management*, 12(5), p.386-394.

Gupta, M. & Galloway, K., (2003). Activity-based costing/management and its implications for operations management. *Technovation*, 23(2), p.131-138.

Hashiba, S. & Chang, T., (1991). PCB assembly setup reduction using group technology. *Computers & Industrial Engineering*, 21(1-4), p.453-457.

Ho, W. & Ji, P., (2003). Component scheduling for chip shooter machines: a hybrid genetic algorithm approach. *Computers & Operations Research*, 30(14), p.2175-2189.

Ho, W. & Ji, P., (2005). A genetic algorithm to optimise the component placement process in PCB assembly *The International Journal of Advanced Manufacturing Technology*, 26(11-12), p.1397-1401.

Homburg, C., (2004). Improving activity-based costing heuristics by higher-level cost drivers. *European Journal of Operational Research*, 157(2), p.332-343.

Homburg, C., (2005). Using relative profits as an alternative to activity-based costing. *International Journal of Production Economics*, 95(3), p.387-397.

Innes, J. & Mitchell, F., (1995). A survey of activity-based costing in the U.K.'s largest companies. *Management Accounting Research*, 6(2), p.137-153.

Jeevan, K., Parthiban, A., Seetharamu, K.N., Azid, I.A. & Quadir, G.A., (2002). Optimization of PCB Component Placement using Genetic Algorithms. *Journal of Electronics Manufacturing*, 11(1), p.69-79.

Ji, P., Sze, M.T. & Lee, W.B., (2001). A genetic algorithm of determining cycle time for printed circuit board assembly lines. *European Journal of Operational Research*, 128(1), p.175-184.

Johnson, H.T. & Kaplan, R.S., (1987). *Relevance Lost: The Rise and Fall of Management Accounting.* Boston, MA: Harvard Business School Press.

Kaplan, R.S. & Anderson, S.R., (2003). Time-Driven Activity-Based Costing [Online]. Available at: http://www.hbs.edu/research/facpubs/workingpapers/papers2/0304/04-045.pdf [accessed 18 Oct 2006].

Käschel, J., Teich, T., Köbernik, G. & Meier, B., (1999). Algorithms for the Job Shop Scheduling Problem - a Comparison of Different Methods. In: *European Symposium on Intelligent Techniques*, Crete, Greece

Khoo, L.P. & Loh, K.M., (2000). A Genetic Algorithms Enhanced Planning System for Surface Mount PCB Assembly. *The International Journal of Advanced Manufacturing Technology*, 16(4), p.289-296.

Khoo, L.P. & Ng, T.K., (1998). A genetic algorithm-based planning system for PCB component placement. *International Journal of Production Economics*, 54(3), p.321-332.

Khoo, L.P. & Ong, N.S., (1998). PCB assembly planning using genetic algorithm. *The International Journal of Advanced Manufacturing Technology*, 14, p.363-368.

Kim, K. & Han, I., (2003). Application of a hybrid genetic algorithm and neural network approach in activity-based costing. *Expert Systems with Applications* 24(1), p.73-77.

Klomp, C., van de Klundert, J., Spieksma, F.C.R. & Voogt, S., (2000). The feeder rack assignment problem in PCB assembly. *International Journal of Production Economics*, 64(1-3), p.399-407.

Kroll, K.M., (2004). The Lowdown on Lean Accounting. *Journal of Accountancy*, 198(1), p.69-76.

Lehaney, B.A. & Vinten, G., (1994). "Methodology": An Analysis of Its Meaning and Use. *Work Study*, 43(3), p.5-8.

Leu, M.C., Wong, H. & Ji, Z., (1993). Planning of component placement/insertion sequence and feeder setup in PCB assembly using genetic algorithm. *Journal of Electronic packaging*, 115(4), p.424-432.

Locascio, A., (2000). Manufacturing Cost Modeling for Product Design. *The International Journal of Flexible Manufacturing Systems*, 12, p.207-217.

Logendran, R. & Nudtasomboon, N., (1991). Minimizing the makespan of a group scheduling problem: a new heuristic. *International Journal of Production Economics*, 22(3), p.217-230.

Loh, T.S., Bukkapatnam, S.T.S., Medieros, D. & Kwon, H., (2001). A genetic algorithm for sequential part assignment for PCB assembly. *Computers and Industrial Engineering*, 40(4), p.293-307.

Marri, H.B. & Grieve, R.J., (1999). Justification and implementation of activity based costing in small and medium-sized enterprises. *Logistics Information Management*, 12(5), p.386-394.

Maskell, B.H., (2004). What is lean accounting? [Online]. Available at: http://www.maskell.com/PDF_Files/What%20is%20Lean%20Accounting.pdf [accessed 15 May 2005].

Maskell, B.H., (2006). Lean accounting & Activity-Based Costing [Online]. Available at: http://www.maskell.com/LeanAcctg_ABC.htm [accessed 18 Oct 2006].

Maskell, B.H. & Baggaley, B., (2004). *Practical Lean Accounting: a proven system for measuring and managing the lean enterprise.* New York, NY, USA: Productivity Press.

Moscato, P., (2003). TSPBIB Home Page [Online]. Available at: http://www.ing.unlp.edu.ar/cetad/mos/TSPBIB_home.html [accessed 08 Jun 2003].

Moyer, L.K. & Gupta, S.M., (1996). Simultaneous component sequencing and feeder assignment for high speed chip shooter machines. *Journal of Electronics Manufacturing*, 6(4), p.271-307.

Narayanaswami, R. & Iyengar, V., (2005). Setup reduction in printed circuit board assembly by efficient sequencing. *The international journal of advanced manufacturing technology*, 26(3), p.276-284.

Nawaz, M., Enscore, E.E. & Ham, I., (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1), p.91-95.

Nelson, K.M. & Wille, L.T., (1995). Comparative study of heuristics for optimal printed circuit board assembly. In: *Conference Record for Southcon*, Fort Lauderdale, FL, USA.

Noreen, E., (1991). Conditions under which activity-based cost systems provide relevant costs. *Journal of Management Accounting Research*, 3, p.159-168.

Noreen, E. & Soderstrom, N., (1994). Are overhead costs strictly proportional to activity? *Journal of Accounting and Economics*, 17(1-2), p.255-278.

Ong, N.S., (1995). Manufacturing cost estimation for PCB assembly: An activity-based approach. *International Journal of Production Economics*, 38(2-3), p.159-172.

Ong, N.S. & Khoo, L.P., (1999). Genetic algorithm approach in PCB assembly. *Integrated Manufacturing Systems*, 10(5), p.256-265.

Ong, N.S. & Lim, L.E.N., (1993). Activity-based cost-modelling procedures for PCB assembly. *The International Journal of Advanced Manufacturing Technology*, 8(6), p.396-406.

Ong, N.S. & Tan, W.C., (2002). Sequence placement planning for high-speed PCB assembly machine. *Integrated Manufacturing Systems*, 13(1), p.35-46.

Özbayrak, M., Akgün, M. & Türker, A.K., (2004). Activity-based cost estimation in a push/pull advanced manufacturing system. *International Journal of Production Economics*, 87(1), p.49-65.

Proust, C., Gupta, J.N.D. & Deschamps, V., (1991). Flowshop scheduling with set-up, processing and removal times separated. *International Journal of Production Research*, 29(3), p.479-493.

Rabuñal, J.R. & Dorado, J., (2005). *Artificial Neural Networks in Real-life Applications.* Hershey, PA: IGI Global.

Reinelt, G., (1991). TSPLIB- A Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3(4), p.376-385.

Rossetti, M.D. & Stanford, K.J.A., (2003). Group sequencing a PCB assembly system via an expected sequence dependent setup heuristic. *Computers & Industrial Engineering*, 35(1), p.231-254.

Saad, S.M. & Lassila, A.M., (2002). Sequencing for Mixed-model Assembly Lines with Bottleneck Resources Using Tabu Search. In: *International Conference on Flexible Automation & Intelligent Manufacturing (FAIM)*, Dresden, Germany.

Sadiq, M., Landers, T.L. & Taylor, G., (1993). A heuristic algorithm for minimizing total production time for a sequence of jobs on a surface mount placement machine. *International Journal of Production Research*, 31(6), p.1327-1341.

Sanchez, J.M. & Priest, J.W., (1990). Optimal component-insertion sequence planning methodology for the semiautomatic assembly of printed circuit boards. *Journal of Intelligent Manufacturing*, 2(3), p.177-188.

Schein, E.H., (1987). *The Clinical perspective in fieldwork.* Beverly Hills, CA: Sage.

Sickafus, E., (2004). *Heuristics for Solving Technical Problems - Theory, Derivation, Application.* Grosse Ile, MI: Ntelleck, LLC.

Sohal, A.S. & Chung, W.W.C., (1998). Activity based costing in manufacturing: two case studies on implementation. *Integrated Manufacturing Systems*, 9(3), p.137-147.

Spedding, T.A. & Sun, G.Q., (1999). Application of discrete event simulation to the activity based costing of manufacturing systems. *International Journal of Production Economics*, 58(3), p.289-301.

Stake, R.E., (1995). *The Art of case study research.* Newbury Park, CA: Sage Publications.

Su, C., Ho, L. & Fu, H., (1998). A novel tabu based approach to find the best placement sequence and magazine assignment in dynamic robotics assembly. *Integrated Manufacturing Systems*, 9(6), p.366-376.

Su, Y. & Srihari, K., (1996). Placement sequence identification using artificial neural networks in surface mount PCB assembly. *International journal of advanced manufacturing technology*, 11(4), p.285-299.

Tellis, W., (1997). Application of a case study methodology. The Qualitative Report [Online serial] 3(3), Available at: http://www.nova.edu/ssss/QR/QR3-3/tellis2.html.

Tornberg, K., Jamsen, M. & Paranko, J., (2002). Activity-based costing and process modeling for cost-conscious product design: A case study in a manufacturing company. *International Journal of Production Economics*, 79(1), p.75-82.

Van Hop, N. & Tabucanon, M.T., (2001a). Extended dynamic point specification approach to sequencing robot moves for PCB assembly. *International Journal of Production Research*, 39(8), p.1671-1687.

Van Hop, N. & Tabucanon, M.T., (2001b). Multiple criteria approach for solving feeder assignment and assembly sequence problem in PCB assembly. *Production Planning & Control*, 12(8), p.736-744.

van Laarhoven, P.J.M. & Zijm, W.H.M., (1993). Production preparation and numerical control in PCB assembly. *International Journal of Flexible Manufacturing Systems*, 5(3), p.187-207.

Wan, Y.F. & Ji, P., (2001). A tabu search heuristic for the component assignment problem in PCB assembly. *Assembly Automation*, 21(3), p.236-240.

Wang, C., Ho, L. & Cannon, D.J., (1998). Heuristics for assembly sequencing and relative magazine assignment for robotic assembly. *Computers and Industrial Engineering*, 34(2), p.423-431.

Watson, J.-P., (2003). Empirical Modeling and Analysis of Local Search Algorithms for the Job-Shop Scheduling Problem. Department of Computer Science. Fort Collins, Colorado, Colorado State University. Doctor of Philosophy.

Whitney, D.E., (1988). Manufacturing by design. *Harvard Business Review*, 66(4), p.83-91.

Womack, J.P. & Jones, D.T., (1996). *Lean thinking: banish waste and create wealth in your corporation.* New York, NY: Simon & Schuster.

Yilmaz, I.O. & Günther, H.O., (2005). A Group Setup Strategy for PCB Assembly on a Single Automated Placement Machine. In: *The Annual International Conference of the German Operations Research Society*, Bremen, Germany, Springer Berlin Heidelberg.

Yin, R., (1994). *Case study research: design and methods.* 2nd ed. Thousand Oaks, CA: Sage Publishing.

Younis, T.A. & Cavalier, T.M., (1990). On locating part bins in a constrained layout area of an automated assembly process. *Computers and Industrial Engineering*, 18(2), p.111-118.

Yuan, P., Hu, Y., Liu, H. & Gao, H., (2006). Feeder assignment optimization algorithm for multi-head mounter. *Journal of Control Theory and Applications*, 4(3), p.223-228.

PUBLICATIONS

Two publications have been produced based on this research:

Saad, S., Khalil, E., Fowkes, C., Basarab-Horwath, I. & Perera, T., (2004). Component and board sequencing - A simultaneous taboo search approach. *2nd international Conference on Advances in Manufacturing Technology (ICRM),* Sheffield, UK.

Saad, S., Khalil, E., Fowkes, C., Basarab-Horwath, I. & Perera, T., (2006). Taboo Search vs. Genetic Algorithms in Solving and Optimising PCB Problems. *Journal of Manufacturing Technology Management*, 17(4), p.521-536.

APPENDICES

Appendix I

The component types and the co-ordinates of their locations of the case study (dimensions in mm).

Board type A												
Comp. type	X-Co-ordinate						Y-Co-ordinate					
	1	2	3	4	5	6	1	2	3	4	5	6
1	386	123	23	670	677		75	55	288	341	354	
2	345	35	32	498	598		454	23	190	10	499	
3	65	55	654	568	679		459	98	480	34	443	
4	354	23	245	600	0		751	551	208	301	0	
5	359	98	59	0	0		414	13	100	0	0	
6	456	654	450	0	0		409	118	420	0	0	
7	400	255	55	0	0		156	454	469	0	0	
8	27	54	0	0	0		355	255	0	0	0	
9	49	151	0	0	0		47	24	0	0	0	
10	245	0	0	0	0		286	0	0	0	0	
11	258	0	0	0	0		245	0	0	0	0	
12	228	0	0	0	0		152	0	0	0	0	
13	278	0	0	0	0		255	0	0	0	0	
14	208	0	0	0	0		248	0	0	0	0	
15	134	0	0	0	0		245	0	0	0	0	
16	248	0	0	0	0		258	0	0	0	0	
17	268	0	0	0	0		124	0	0	0	0	
18	134	0	0	0	0		74	0	0	0	0	
19	76	0	0	0	0		35	0	0	0	0	
20	36	0	0	0	0		42	0	0	0	0	
21	45	0	0	0	0		163	0	0	0	0	
22	173	0	0	0	0		103	0	0	0	0	
Board type B												
4	75	55	288	98	45	55	454	23	190	35	123	65
8	454	23	190	35	123	65	459	98	589	167	213	154
9	459	98	589	167	213	0	156	153	475	41	457	0
13	156	454	569	24	0	0	355	55	127	397	0	0
17	355	255	454	0	0	0	65	35	65	0	0	0
18	69	55	542	0	0	0	69	55	542	0	0	0
5	344	23	0	0	0	0	344	23	0	0	0	0
1	399	0	0	0	0	0	256	0	0	0	0	0
11	356	0	0	0	0	0	128	0	0	0	0	0
34	108	0	0	0	0	0	198	0	0	0	0	0
41	398	0	0	0	0	0	35	0	0	0	0	0
15	186	0	0	0	0	0	56	0	0	0	0	0
14	345	0	0	0	0	0	156	0	0	0	0	0
52	75	0	0	0	0	0	265	0	0	0	0	0
55	58	0	0	0	0	0	268	0	0	0	0	0
58	151	0	0	0	0	0	245	0	0	0	0	0
22	52	0	0	0	0	0	268	0	0	0	0	0
21	111	0	0	0	0	0	125	0	0	0	0	0

Board type C												
Comp. type	X-Co-ordinate						Y-Co-ordinate					
	1	2	3	4	5	6	1	2	3	4	5	6
1	295	55	259	98			694	23	248	354		
5	65	55	298	219			259	98	648	104		
6	694	23	248	0			624	63	324	0		
8	259	98	0	0			269	48	0	0		
11	256	0	0	0			156	0	0	0		
14	298	0	0	0			218	0	0	0		
18	125	0	0	0			145	0	0	0		
22	127	0	0	0			167	0	0	0		
52	95	0	0	0			94	0	0	0		
34	65	0	0	0			255	0	0	0		
55	255	0	0	0			255	0	0	0		
58	248	0	0	0			248	0	0	0		
44	654	0	0	0			654	0	0	0		
46	687	0	0	0			687	0	0	0		
Board type D												
3	55	65	444	128			216	143	344	245		
11	624	63	98	245			345	34	95	434		
13	269	548	623	0			55	65	24	0		
2	156	214	0	0			624	63	0	0		
58	218	0	0	0			269	0	0	0		
34	145	0	0	0			156	0	0	0		
40	305	0	0	0			218	0	0	0		
41	65	0	0	0			335	0	0	0		
5	254	0	0	0			35	0	0	0		
52	459	0	0	0			244	0	0	0		
54	256	0	0	0			449	0	0	0		
55	123	0	0	0			246	0	0	0		
15	245	0	0	0			133	0	0	0		
Board type E												
3	345	634	195	434			345	434	195	434		
9	664	63	75	0			464	63	75	0		
15	269	548	0	0			269	548	0	0		
56	156	214	0	0			156	214	0	0		
48	555	0	0	0			455	0	0	0		
44	35	0	0	0			35	0	0	0		
20	544	0	0	0			544	0	0	0		
79	698	0	0	0			498	0	0	0		
31	646	0	0	0			446	0	0	0		
36	133	0	0	0			133	0	0	0		
Board type F												
11	645	434	195				345	434	195			
41	464	63	75				464	63	75			
23	669	548	0				369	548	0			
8	156	264	0				156	264	0			
12	455	0	0				355	0	0			
35	65	0	0				65	0	0			
79	564	0	0				364	0	0			
45	698	0	0				398	0	0			
2	446	0	0				346	0	0			

Board type G												
Comp. type	X-Co-ordinate						Y-Co-ordinate					
	1	2	3	4	5	6	1	2	3	4	5	6
55	454	23	190	35			354	23	190	35		
4	459	98	589	0			359	98	389	0		
21	156	545	569	0			156	354	369	0		
35	69	55	542	0			69	55	342	0		
48	399	98	654	0			399	98	354	0		
79	356	154	0	0			356	154	0	0		
56	108	219	0	0			108	219	0	0		
Board type H												
20	295	55	259				235	55	259			
73	694	23	248				394	23	348			
46	256	254	0				256	254	0			
12	298	219	0				298	319	0			
1	127	24	0				127	34	0			

Appendix II

The program code written in MS Visual C++.

Header Files:

AlgorithmDlg.h

```
#if
!defined(AFX_ALGORITHMDLG_H__93B376F0_C755_4FC5_9213_DB5809544088__INC
LUDED_)
#define
AFX_ALGORITHMDLG_H__93B376F0_C755_4FC5_9213_DB5809544088__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// AlgorithmDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
///////
// CAlgorithmDlg dialog

class CAlgorithmDlg : public CDialog
{
private:
      int choice;
      CButton& centroid();
      CButton& random();
      int MaxMoves;
      int TabuRestart;
      int TabuSize;
      int MaxNoImp;
// Construction
public:
      void set_para(const int _MaxMoves, const int _MaxNoImp, const
int _TabuSize, const int _TabuRestart, const int _choice);
      BOOL OnInitDialog();
      void UpdateBox(const int _ID, const int _Data);
      int Get_choice();
      int Get_TabuRestart();
      int Get_TabuSize();
      int Get_MaxNoImp();
      int Get_MaxMoves();
      int GetItem(const int _ID);
      CAlgorithmDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
      //{{AFX_DATA(CAlgorithmDlg)
      enum { IDD = IDD_ALGORITHM };
            // NOTE: the ClassWizard will add data members here
      //}}AFX_DATA


// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CAlgorithmDlg)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
      //}}AFX_VIRTUAL

// Implementation
protected:

      // Generated message map functions
```

```
        //{{AFX_MSG(CAlgorithmDlg)
        virtual void OnOK();
        afx_msg void OnRadioMethodClick();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_ALGORITHMDLG_H__93B376F0_C755_4FC5_9213_DB5809544088__INC
LUDED_)
```

boarddlg.h

```
#if
!defined(AFX_BOARDDLG_H__581C98AB_B162_4B2E_95A5_4C10BD79781A__INCLUDE
D_)
#define
AFX_BOARDDLG_H__581C98AB_B162_4B2E_95A5_4C10BD79781A__INCLUDED_

#include <fstream>
#include "Board.h"      // Added by ClassView
using namespace std;

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// boarddlg.h : header file
//

/////////////////////////////////////////////////////////////////////////
///////
// CBoardDlg dialog

class CBoardDlg : public CDialog
{
// Construction
public:
      CBoard *get_pBoardArray();
      bool CompCompareMax(const int _NewComp, const int _Count);
      bool CompCompareBoard(const int _NewComp, const int _Board,
const int _Comp);
      bool CompCompare(const int _NewComp, const int _Next);
      int get_number(const char *_file, const int _next);
      void UpdateBox(const int _ID, const _Data);
      void load_files(const char *_file);
      void ClearBox(const int _ID);
      int Get_NumOfBoardTypes();
      int Get_MaxNumOfLocations();
      int Get_MaxNumOfCompTypes();
      CBoardDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
      //{{AFX_DATA(CBoardDlg)
      enum { IDD = IDD_BOARD };
            // NOTE: the ClassWizard will add data members here
      //}}AFX_DATA


// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CBoardDlg)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
      //}}AFX_VIRTUAL

// Implementation
protected:

      // Generated message map functions
      //{{AFX_MSG(CBoardDlg)
      afx_msg void OnSelchangeCombo();
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
private:
      CBoard *pBoardArray;
      int NumOfBoardTypes;
      int MaxCompFreq;
      int MaxNumOfCompTypes;
      int MaxNumOfLocations;
      int *pCompTypes;
```

```
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_BOARDDLG_H__581C98AB_B162_4B2E_95A5_4C10BD79781A__INCLUDE
D_)
```

Comps.h

```
// Comps.h: interface for the CComps class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_COMPS_H__7428CEC8_1140_41FE_9A2F_BC9DDDEB98B8__INCLUDED_)
#define AFX_COMPS_H__7428CEC8_1140_41FE_9A2F_BC9DDDEB98B8__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CComps
{
public:
	float X_Co;
	float Y_Co;
	int CompType;
	int CompFreq;
	CComps();
	virtual ~CComps();

};

#endif //
!defined(AFX_COMPS_H__7428CEC8_1140_41FE_9A2F_BC9DDDEB98B8__INCLUDED_)
```

FeederDlg.h

```
#if
!defined(AFX_FEEDERDLG_H__E8AD6481_341C_430B_B840_8362521D3E30__INCLUD
ED_)
#define
AFX_FEEDERDLG_H__E8AD6481_341C_430B_B840_8362521D3E30__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// FeederDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
//////
// CFeederDlg dialog

class CFeederDlg : public CDialog
{
// Construction
public:
	void set_para(const int _NumOfFeeders, const int _LengOfFeeder);
	BOOL OnInitDialog();
	void UpdateBox(const int _ID, const int _Data);
	int Get_LengOfFeeder();
	int Get_NumOfFeeders();
	int GetItem(const int _ID);
	CFeederDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(CFeederDlg)
	enum { IDD = IDD_FEEDER };
		// NOTE: the ClassWizard will add data members here
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CFeederDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CFeederDlg)
	virtual void OnOK();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
private:
	int NumOfFeeders;
	int LengOfFeeder;
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_FEEDERDLG_H__E8AD6481_341C_430B_B840_8362521D3E30__INCLUD
ED_)
```

PcbDlg.h

```
// PcbDlg.h : header file
//

#if
!defined(AFX_PCBDLG_H__C4F577C4_2DE2_49F9_A842_8453F4330153__INCLUDED_
)
#define AFX_PCBDLG_H__C4F577C4_2DE2_49F9_A842_8453F4330153__INCLUDED_

#include <strstream>
#include "Permutation.h"      // Added by ClassView
#include "FeederDlg.h"  // Added by ClassView
#include "CoordinatesDlg.h"   // Added by ClassView
#include "TimeDlg.h"    // Added by ClassView
#include "AlgorithmDlg.h"     // Added by ClassView
#include "BoardDlg.h"   // Added by ClassView
#include "Positions.h"  // Added by ClassView
#include "TabuList.h"   // Added by ClassView
#include "Board.h"      // Added by ClassView
using namespace std;

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

/////////////////////////////////////////////////////////////////////////////
//////
// CPcbDlg dialog

class CPcbDlg : public CDialog
{
// Construction
public:
      void ClearTime();
      void UpdateTime(const int _cBoard, const float _BestTime);
      //void UpdateProgress();
      void ClearBox(const int _ID);
      void UpdateTextBox(ostrstream _stream);
      void UpdateStatusBox(ostrstream _stream);
      void UpdateBox(const int _ID, char *_data);
      int GetItem(const int _ID);
      void UpdateBox(const int _ID, const int _data);
      CPcbDlg(CWnd* pParent = NULL);      // standard constructor

// Dialog Data
      //{{AFX_DATA(CPcbDlg)
      enum { IDD = IDD_PCB_DIALOG };
      CButton     m_GoButton;
      CProgressCtrl     m_ProgressStatus;
      CString     m_StringData;
      CString     m_TextData;
      float       m_Board_A_Time;
      float       m_Board_B_Time;
      float       m_Board_C_Time;
      float m_Board_D_Time;
      float m_Board_E_Time;
      float m_Board_F_Time;
      float m_Board_G_Time;
      float m_Board_H_Time;
      float m_Board_I_Time;
      //}}AFX_DATA

      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CPcbDlg)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);      // DDX/DDV
support
      //}}AFX_VIRTUAL
```

```
// Implementation
protected:
	HICON m_hIcon;

	// Generated message map functions
	//{{AFX_MSG(CPcbDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnBoard();
	afx_msg void OnFeeder();
	afx_msg void OnCoordinates();
	afx_msg void OnTime();
	afx_msg void OnAlgorithm();
	virtual void OnOK();
	afx_msg void OnGo();
	afx_msg void OnSetfocusTextout();
	virtual void OnCancel();
	//}}AFX_MSG

private:
	CEdit *pDisplay;
	char *Test;
	int Size;
	CString *CompNames;
	bool FinalOpen;
	double Board_Time;
	double Setup_Time;
	bool ResultOpen;
	bool allowed;
	bool flag;
	bool Repeat;
	int choice;

	int Total_Time;
	int solution;
	int NoImpMoves;
	int RestartMoves;

	ofstream FinalFile;
	ofstream ResultFile;
	ofstream FedNeighFile;
	ostrstream str;
	ostrstream strText;
	ostrstream strBuffer;

	CPermutation LocalBest;
	CPermutation Best;
	CPermutation current;

	CBoard *pBoardArray;
	CTabuList *tabu;
	CPositions *pBest_Exit;
	CPositions move;

	CBoardDlg BoardDlg;
	CAlgorithmDlg AlgorithmDlg;
	CTimeDlg TimeDlg;
	CCoordinatesDlg CoordinatesDlg;
	CFeederDlg FeederDlg;

	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.
```

```
#endif //
!defined(AFX_PCBDLG_H__C4F577C4_2DE2_49F9_A842_8453F4330153__INCLUDED_
)
```

Positions.h

```
// Positions.h: interface for the CPositions class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_POSITIONS_H__A3653B2D_41FE_4783_891E_652FE307FA54__INCLUD
ED_)
#define
AFX_POSITIONS_H__A3653B2D_41FE_4783_891E_652FE307FA54__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CPositions
{
public:
	int b;
	int a;
	CPositions();
	virtual ~CPositions();

};

#endif //
!defined(AFX_POSITIONS_H__A3653B2D_41FE_4783_891E_652FE307FA54__INCLUD
ED_)
```

StdAfx.h

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if
!defined(AFX_STDAFX_H__233F957B_59B4_4819_83D4_14C71A77BB18__INCLUDED_
)
#define AFX_STDAFX_H__233F957B_59B4_4819_83D4_14C71A77BB18__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define VC_EXTRALEAN            // Exclude rarely-used stuff from
Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdisp.h>        // MFC Automation classes
#include <afxdtctl.h>         // MFC support for Internet Explorer 4
Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                 // MFC support for Windows Common
Controls
#endif // _AFX_NO_AFXCMN_SUPPORT


//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_STDAFX_H__233F957B_59B4_4819_83D4_14C71A77BB18__INCLUDED_
)
```

TimeDlg.h

```
#if
!defined(AFX_TIMEDLG_H__1CA23F63_5380_4CD6_BE2D_F61517975501__INCLUDED
_)
#define AFX_TIMEDLG_H__1CA23F63_5380_4CD6_BE2D_F61517975501__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// TimeDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
///////
// CTimeDlg dialog

class CTimeDlg : public CDialog
{
// Construction
public:
      void set_para(const int _FeedSetupTime, const int _PickTime,
const int _InsertTime, const int _HeadSpeed);
      BOOL OnInitDialog();
      void UpdateBox(const int _ID, const int _Data);
      int Get_HeadSpeed();
      int Get_InsertTime();
      int Get_PickTime();
      int Get_FeedSetupTime();
      int GetItem(const int _ID);
      CTimeDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
      //{{AFX_DATA(CTimeDlg)
      enum { IDD = IDD_TIME };
            // NOTE: the ClassWizard will add data members here
      //}}AFX_DATA


// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CTimeDlg)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
      //}}AFX_VIRTUAL

// Implementation
protected:

      // Generated message map functions
      //{{AFX_MSG(CTimeDlg)
      afx_msg void OnChangeFeedsetuptime();
      afx_msg void OnChangePicktime();
      afx_msg void OnChangeInserttime();
      afx_msg void OnChangeHeadspeed();
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
private:
      int HeadSpeed;
      int PickTime;
      int InsertTime;
      int FeedSetupTime;
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.
```

```
#endif //
!defined(AFX_TIMEDLG_H__1CA23F63_5380_4CD6_BE2D_F61517975501__INCLUDED
_)
```

Board.h

```
// Board.h: interface for the CBoard class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_BOARD_H__DE3D2410_58B6_472E_9ACD_EE5DFDF2A25E__INCLUDED_)
#define AFX_BOARD_H__DE3D2410_58B6_472E_9ACD_EE5DFDF2A25E__INCLUDED_

#include "Comps.h"      // Added by ClassView
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CBoard
{
public:
      CComps *pComps;
      int NumOfBoards;
      int NumOfComps;
      int NumOfCompTypes;
      int NumOfLocations;
      int MaxCompFreqOnBoard;
      int *TypesOfComps;
      CBoard();
      virtual ~CBoard();

};

#endif //
!defined(AFX_BOARD_H__DE3D2410_58B6_472E_9ACD_EE5DFDF2A25E__INCLUDED_)
```

Buffer.h

```
// Buffer.h: interface for the CBuffer class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_BUFFER_H__04580654_ECA2_458C_8978_192B7F660FBC__INCLUDED_
)
#define AFX_BUFFER_H__04580654_ECA2_458C_8978_192B7F660FBC__INCLUDED_

#include <cassert>

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CBuffer
{
public:
	//void Buffer_Init(const int _TabuSize);
	void clear_buffer();
	void push_buffer(int max, int item);
	int pop_buffer(int max);
	int get_size() const;
	int get_pointer(int _k) const;
	CBuffer(const int _TabuSize);
	virtual ~CBuffer();

private:
	int *pProd;
	int BufferLast;
	int BufferSize;
};

#endif //
!defined(AFX_BUFFER_H__04580654_ECA2_458C_8978_192B7F660FBC__INCLUDED_
)
```

CoordinatesDlg.h

```
#if
!defined(AFX_COORDINATESDLG_H__DC57E5D8_E44B_4C77_AB3E_E263E0C7348F__I
NCLUDED_)
#define
AFX_COORDINATESDLG_H__DC57E5D8_E44B_4C77_AB3E_E263E0C7348F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// CoordinatesDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
//////
// CCoordinatesDlg dialog

class CCoordinatesDlg : public CDialog
{
// Construction
public:
	void set_para(const int _X, const int _Y, const int _X_Home,
const int _Y_Home);
	BOOL OnInitDialog();
	void UpdateBox(const int _ID, const int _Data);
	int Get_Y_Home();
	int Get_X_Home();
	int Get_Y();
	int Get_X();
	int GetItem(const int _ID);
	CCoordinatesDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(CCoordinatesDlg)
	enum { IDD = IDD_COORDINATES };
		// NOTE: the ClassWizard will add data members here
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CCoordinatesDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CCoordinatesDlg)
	afx_msg void OnChangeX();
	afx_msg void OnChangeY();
	afx_msg void OnChangeXHome();
	afx_msg void OnChangeYHome();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
private:
	int Y;
	int X;
	int Y_Home;
	int X_Home;
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.
```

```
#endif //
!defined(AFX_COORDINATESDLG_H__DC57E5D8_E44B_4C77_AB3E_E263E0C7348F__I
NCLUDED_)
```

Pcb.h

```
// Pcb.h : main header file for the PCB application
//

#if
!defined(AFX_PCB_H__76D9CB6F_3FF5_467A_BBFD_48C0CB5CEA71__INCLUDED_)
#define AFX_PCB_H__76D9CB6F_3FF5_467A_BBFD_48C0CB5CEA71__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
      #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"         // main symbols

/////////////////////////////////////////////////////////////////////////////
//////
// CPcbApp:
// See Pcb.cpp for the implementation of this class
//

class CPcbApp : public CWinApp
{
public:
      CPcbApp();

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CPcbApp)
      public:
      virtual BOOL InitInstance();
      //}}AFX_VIRTUAL

// Implementation

      //{{AFX_MSG(CPcbApp)
            // NOTE - the ClassWizard will add and remove member
functions here.
            //    DO NOT EDIT what you see in these blocks of
generated code !
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////
//////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_PCB_H__76D9CB6F_3FF5_467A_BBFD_48C0CB5CEA71__INCLUDED_)
```

Permutation.h

```
// Permutation.h: interface for the CPermutation class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_PERMUTATION_H__930F4ABA_DEDF_4571_9871_B768B8633648__INCL
UDED_)
#define
AFX_PERMUTATION_H__930F4ABA_DEDF_4571_9871_B768B8633648__INCLUDED_

#include <fstream>
#include <math.h>
#include "Positions.h"  // Added by ClassView
#include "Board.h"      // Added by ClassView
using namespace std;

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CPermutation
{
public:
      void write_NumOfFeeders();
      int find_best_f(const int start, const CBoard *_pBoardArray,
const int _Board);
      int find_best_s(const int start, const CBoard *_pBoardArray,
const int _Board);
      void initialise(const CBoard *_pBoardArray, const int _Board);
      void pick_random_s(const CBoard *_pBoardArray, const int
_Board);
      void pick_random_f(const CBoard *_pBoardArray, const int
_Board);
      void generate_assign(const CBoard *_pBoardArray, const int
_Board);
      void calculate_time(const CBoard *_pBoardArray, const int
_Board);
      int create_neigh_s(const CBoard *_pBoardArray, const int _Board,
const CPermutation _current, ofstream &_OutFile);
      int create_neigh_f(const CBoard *_pBoardArray, const int _Board,
const CPermutation _current, ofstream &_OutFile);
      void poison();
      int* get_pFeeder();
      int* get_pSeq();
      double get_time();
      int get_position_a(int _count);
      int get_position_b(int _count);
      int get_S_Neigh_Size(const CBoard *_pBoardArray, const int
_Board);
      int get_F_Neigh_Size();
      friend CPositions compare_s(const CPermutation &p1, const
CPermutation &p2, const CBoard *_pBoardArray, const int _Board);
      friend CPositions compare_f(const CPermutation &p1, const
CPermutation &p2);
      int Get_NumOfFeeders();
      CPermutation();
      virtual ~CPermutation();

      CPermutation operator = (const CPermutation &source);

private:
      bool FedNeighOpen;
      ofstream FedNeighFile;
      ofstream SeqNeighFile;

      CPositions *pCoordinates;

      int NumOfSeq;
```

```
        int *pFeeder;
        int *pSeq;

        double time;
};

#endif //
!defined(AFX_PERMUTATION_H__930F4ABA_DEDF_4571_9871_B768B8633648__INCL
UDED_)
```

resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Pcb.rc
//
#define IDM_ABOUTBOX                    0x0010
#define IDD_ABOUTBOX                    100
#define IDS_ABOUTBOX                    101
#define IDD_PCB_DIALOG                  102
#define IDR_MAINFRAME                   128
#define IDR_MENU                        129
#define IDD_BOARD                       132
#define IDD_FEEDER                      134
#define IDD_COORDINATES                 135
#define IDD_TIME                        136
#define IDD_ALGORITHM                   137
#define IDD_PROGRESS                    138
#define IDC_TEXTOUT                     1001
#define IDC_STATUSBOX                   1002
#define IDC_NUMOFBOARDTYPESOUT          1004
#define IDC_COMBO                       1005
#define IDC_X_HOME                      1006
#define IDC_Y_HOME                      1007
#define IDC_X                           1008
#define IDC_Y                           1009
#define IDC_NUMOFFEEDERS                1010
#define IDC_LENGOFFEEDER                1011
#define IDC_FEEDSETUPTIME               1012
#define IDC_PICKTIME                    1013
#define IDC_INSERTTIME                  1014
#define IDC_HEADSPEED                   1015
#define IDC_MAXMOVES                    1016
#define IDC_MAXNOIMP                    1017
#define IDC_TABUSIZE                    1018
#define IDC_TABURESTART                 1019
#define IDC_NUMOFBOARDS_A1              1020
#define IDC_NUMOFBOARDS_A2              1021
#define IDC_NUMOFBOARDS_A3              1022
#define IDC_NUMOFBOARDS_A4              1023
#define IDC_NUMOFBOARDS_A5              1024
#define IDC_NUMOFBOARDS_A6              1025
#define IDC_NUMOFBOARDS_A7              1026
#define IDC_NUMOFBOARDS_A8              1027
#define IDC_NUMOFBOARDS_A9              1028
#define IDC_NUMOFCOMPS_A1               1029
#define IDC_NUMOFCOMPS_A2               1030
#define IDC_NUMOFCOMPS_A3               1031
#define IDC_MAXMOVESOUT                 1031
#define IDC_NUMOFCOMPS_A4               1032
#define IDC_MAXNOIMPOUT                 1032
#define IDC_NUMOFCOMPS_A5               1033
#define IDC_TABUSIZEOUT                 1033
#define IDC_NUMOFCOMPS_A6               1034
#define IDC_TABURESTARTOUT              1034
#define IDC_NUMOFCOMPS_A7               1035
#define IDC_X_HOMEOUT                   1035
#define IDC_NUMOFCOMPS_A8               1036
#define IDC_Y_HOMEOUT                   1036
#define IDC_NUMOFCOMPS_A9               1037
#define IDC_XOUT                        1037
#define IDC_MAXCOMPTYPES                1038
#define IDC_YOUT                        1038
#define IDC_NUMOFFEEDERSOUT             1039
#define IDC_LENGOFFEEDEROUT             1040
#define IDC_FEEDSETUPTIMEOUT            1041
#define IDC_PICKTIMEOUT                 1042
#define IDC_INSERTTIMEOUT               1043
#define IDC_HEADSPEEDOUT                1044
#define IDC_MAXCOMPTYPESOUT             1045
```

```
#define IDC_MAXNUMOFLOCATIONS           1046
#define IDC_INITIALMETHODOUT            1046
#define IDC_MAXNUMOFLOCATIONSOUT        1047
#define IDC_GO                          1055
#define IDC_PROGRESS                    1059
#define IDC_BOARD_A_TIME                1062
#define IDC_BOARD_B_TIME                1063
#define IDC_BOARD_C_TIME                1064
#define IDC_BOARD_D_TIME                1065
#define IDC_BOARD_E_TIME                1066
#define IDC_BOARD_F_TIME                1067
#define IDC_BOARD_G_TIME                1068
#define IDC_BOARD_H_TIME                1069
#define IDC_BOARD_I_TIME                1070
#define RANDOM                          8888
#define CENTROID                        9999
#define IDC_BOARD                       32771
#define IDC_FEEDER                      32772
#define IDC_COORDINATES                 32773
#define IDC_TIME                        32775
#define IDC_ALGORITHM                   32777
#define IDC_CENTROID                    32779
#define IDC_RANDOM                      32780

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        140
#define _APS_NEXT_COMMAND_VALUE         32781
#define _APS_NEXT_CONTROL_VALUE         1072
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

TabuList.h

```
// TabuList.h: interface for the CTabuList class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_TABULIST_H__8BE4B658_9AB8_4571_BF87_C76A4187C1F4__INCLUDE
D_)
#define
AFX_TABULIST_H__8BE4B658_9AB8_4571_BF87_C76A4187C1F4__INCLUDED_

#include "Positions.h"  // Added by ClassView
#include "Buffer.h"     // Added by ClassView
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CTabuList
{
public:
      void clear_list();
      void push_list(CPositions &move);
      void pop_list();
      int list_size();
      bool list_find(CPositions &move) const;
      CTabuList(const int _TabuSize);
      virtual ~CTabuList();

private:
      int TabuSize;
      CBuffer *b;
      CBuffer *a;
};

#endif //
!defined(AFX_TABULIST_H__8BE4B658_9AB8_4571_BF87_C76A4187C1F4__INCLUDE
D_)
```

Source Files:

AlgorithmDlg.cpp

```
// AlgorithmDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Pcb.h"
#include "AlgorithmDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
//////
// CAlgorithmDlg dialog


CAlgorithmDlg::CAlgorithmDlg(CWnd* pParent /*=NULL*/)
      : CDialog(CAlgorithmDlg::IDD, pParent)
{
      MaxMoves = 10;
      TabuRestart = 2;
      TabuSize = 4;
      MaxNoImp = 3;
      //{{AFX_DATA_INIT(CAlgorithmDlg)
            // NOTE: the ClassWizard will add member initialization
here
      //}}AFX_DATA_INIT
}


void CAlgorithmDlg::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
      //{{AFX_DATA_MAP(CAlgorithmDlg)
            // NOTE: the ClassWizard will add DDX and DDV calls here
      //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CAlgorithmDlg, CDialog)
      //{{AFX_MSG_MAP(CAlgorithmDlg)
      ON_BN_CLICKED(IDC_CENTROID, OnRadioMethodClick)
      ON_BN_CLICKED(IDC_RANDOM, OnRadioMethodClick)
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

int CAlgorithmDlg::GetItem(const int _ID)
{
      const TEXT_SIZE = 16;
      char szText[TEXT_SIZE + 1];         // buffer for conversions
      CEdit *pGet = (CEdit *)(GetDlgItem(_ID));
      pGet->GetWindowText(szText, TEXT_SIZE);
      return atoi(szText);
}

/////////////////////////////////////////////////////////////////////////////
//////
// CAlgorithmDlg message handlers

int CAlgorithmDlg::Get_MaxMoves()
{
      return MaxMoves;
}
```

```
int CAlgorithmDlg::Get_MaxNoImp()
{
	return MaxNoImp;
}

int CAlgorithmDlg::Get_TabuSize()
{
	return TabuSize;
}

int CAlgorithmDlg::Get_TabuRestart()
{
	return TabuRestart;
}

void CAlgorithmDlg::OnOK()
{
	MaxMoves = GetItem(IDC_MAXMOVES);
	MaxNoImp = GetItem(IDC_MAXNOIMP);
	TabuSize = GetItem(IDC_TABUSIZE);
	TabuRestart = GetItem(IDC_TABURESTART);

	CDialog::OnOK();
}

void CAlgorithmDlg::OnRadioMethodClick()
{
	if (random().GetCheck() == 1)
	{
		choice = RANDOM;
	}

	if (centroid().GetCheck() == 1)
	{
		choice = CENTROID;
	}

}

CButton& CAlgorithmDlg::random()
{
	return *(CButton*) GetDlgItem(IDC_RANDOM);
}

CButton& CAlgorithmDlg::centroid()
{
	return *(CButton*) GetDlgItem(IDC_CENTROID);
}

int CAlgorithmDlg::Get_choice()
{
	return choice;
}

void CAlgorithmDlg::UpdateBox(const int _ID, const int _Data)
{
	const TEXT_SIZE = 16;
	char szText[TEXT_SIZE + 1];   // buffer for conversions
	CEdit *pDisplay = (CEdit *)(GetDlgItem(_ID));
	itoa(_Data, szText, 10);
	pDisplay->SetWindowText(szText);
}

BOOL CAlgorithmDlg::OnInitDialog()
{
	UpdateBox(IDC_MAXMOVES, MaxMoves);
	UpdateBox(IDC_MAXNOIMP, MaxNoImp);
	UpdateBox(IDC_TABUSIZE, TabuSize);
	UpdateBox(IDC_TABURESTART, TabuRestart);
```

```
        if (choice == RANDOM)
        {
                random().SetCheck(RANDOM);
        }
        if (choice == CENTROID)
        {
                centroid().SetCheck(CENTROID);
        }
        return true;
}

void CAlgorithmDlg::set_para(const int _MaxMoves, const int _MaxNoImp,
const int _TabuSize, const int _TabuRestart, const int _choice)
{
        MaxMoves = _MaxMoves;
        MaxNoImp = _MaxNoImp;
        TabuSize = _TabuSize;
        TabuRestart = _TabuRestart;
        choice = _choice;
}
```

BoardDlg.cpp

```
// BoardDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Pcb.h"
#include "boarddlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
//////
// CBoardDlg dialog


CBoardDlg::CBoardDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CBoardDlg::IDD, pParent)
{
	NumOfBoardTypes = 0;
	MaxCompFreq = 0;
	MaxNumOfCompTypes = 0;
	MaxNumOfLocations = 0;
	pCompTypes = 0;
	//{{AFX_DATA_INIT(CBoardDlg)
		// NOTE: the ClassWizard will add member initialization
here
	//}}AFX_DATA_INIT
}


void CBoardDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CBoardDlg)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CBoardDlg, CDialog)
	//{{AFX_MSG_MAP(CBoardDlg)
	ON_CBN_SELCHANGE(IDC_COMBO, OnSelchangeCombo)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
//////
// CBoardDlg message handlers

int CBoardDlg::Get_MaxNumOfCompTypes()
{
	return MaxNumOfCompTypes;
}

int CBoardDlg::Get_MaxNumOfLocations()
{
	return MaxNumOfLocations;
}

int CBoardDlg::Get_NumOfBoardTypes()
{
	return NumOfBoardTypes;
}

void CBoardDlg::ClearBox(const int _ID)
```

```
{
	CEdit *pClear = (CEdit *)(GetDlgItem(_ID));
	pClear->SetWindowText("          ");
}

void CBoardDlg::load_files(const char *_file)
{
	int j = 0;
	int next = 0;
	int CompFreq;
	int CompType;

	ifstream in(_file, ios::in);  // file input variable

	if(!in)
	{
		MessageBox("Unable to open file!", "Fail",
MB_ICONWARNING);
		return;
	}

	char item[20];
	char command[20];

	in >> item >> item >> item >> item >> item >> item >>
item;

	for (int board = 0; board < NumOfBoardTypes; board++)
	{
		pBoardArray[board].NumOfCompTypes = 0;
		pBoardArray[board].NumOfComps = get_number(_file, board);
		UpdateBox(IDC_NUMOFCOMPS_A1 +
next,pBoardArray[board].NumOfComps);
		pBoardArray[board].pComps = new CComps
[pBoardArray[board].NumOfComps + 1];
		for (int k=0; k<=pBoardArray[board].NumOfComps; k++)
		{
			pBoardArray[board].pComps[k].X_Co = 0;
			pBoardArray[board].pComps[k].Y_Co = 0;
			pBoardArray[board].pComps[k].CompType = 0;
			pBoardArray[board].pComps[k].CompFreq = 0;
		}
		in >> pBoardArray[board].NumOfBoards;
		UpdateBox(IDC_NUMOFBOARDS_A1 +
next,pBoardArray[board].NumOfBoards);
		while(1)
		{
			in >> pBoardArray[board].pComps[j].CompType;
			CompType = pBoardArray[board].pComps[j].CompType;
			if (pBoardArray[board].pComps[j].CompType == 0)
			{
				in >> command;
				if (command[0] == 'N')
				{
					next += 1;  // increment next
				}
				if (command[0] == 'E')
				{
					MessageBox("End of file","End of file
reached",MB_ICONINFORMATION);
					return;
					break;
				}
				break;
			}
			if ((CompType!=0) && (CompCompare(CompType,board)))
			{
				MaxNumOfCompTypes += 1;
			}
```

```
                    if ((CompType!=0) &&
(CompCompareBoard(CompType,board,j)))
                    {
                          pBoardArray[board].NumOfCompTypes += 1;
                    }

                    in >> pBoardArray[board].pComps[j].CompFreq;
                    CompFreq = pBoardArray[board].pComps[j].CompFreq;

                    for (int i=0; i<CompFreq; i++)
                    {
                          pBoardArray[board].pComps[j].CompType =
CompType;
                          pBoardArray[board].pComps[j].CompFreq =
CompFreq;

                          in >> pBoardArray[board].pComps[j].X_Co >>
pBoardArray[board].pComps[j].Y_Co;
                          j++;
                    }
              }     // end while loop
              pBoardArray[board].TypesOfComps = new int
[pBoardArray[board].NumOfCompTypes];
              for (k = 0; k < pBoardArray[board].NumOfCompTypes; k++)
              {
                    pBoardArray[board].TypesOfComps[k] = 0;
              }
              int cComps = 0;
              int TempFreq = 0;
              for (int cTypes = 0; cTypes <
pBoardArray[board].NumOfCompTypes; cTypes++)
              {
                    if
(CompCompareBoard(pBoardArray[board].pComps[cComps].CompType,board,cTy
pes))
                    {
                          pBoardArray[board].TypesOfComps[cTypes] =
pBoardArray[board].pComps[cComps].CompType;
                          TempFreq =
pBoardArray[board].pComps[cComps].CompFreq;
                          cComps = cComps + TempFreq;
                    }
              }
              pBoardArray[board].NumOfLocations = j;
              if (MaxNumOfLocations < pBoardArray[board].NumOfLocations)
              {
                    MaxNumOfLocations =
pBoardArray[board].NumOfLocations;
                    UpdateBox(IDC_MAXNUMOFLOCATIONS,MaxNumOfLocations);
              }
              for (int board = 0; board < NumOfBoardTypes; board++)
              {
                    for (int comp = 0; comp <
pBoardArray[board].NumOfComps; comp++)
                    {
                          if (MaxCompFreq <
pBoardArray[board].pComps[comp].CompFreq)
                          {
                                MaxCompFreq =
pBoardArray[board].pComps[comp].CompFreq;
                          }
                          if (pBoardArray[board].MaxCompFreqOnBoard <
pBoardArray[board].pComps[comp].CompFreq)
                          {
                                pBoardArray[board].MaxCompFreqOnBoard =
pBoardArray[board].pComps[comp].CompFreq;
                          }
                    }
              }
              UpdateBox(IDC_MAXCOMPTYPES,MaxNumOfCompTypes);
```

```
            pCompTypes = new int [MaxNumOfCompTypes];
            for (int i = 0; i < MaxNumOfCompTypes; i++)
            {
                  pCompTypes[i] = 0;
            }
            j = 0;
      }
      int comp = 0;
      int TempFreq = 0;
      for (int cBoard = 0; cBoard < NumOfBoardTypes; cBoard++)
      {
            for (int cComp = 0; cComp <
pBoardArray[cBoard].NumOfComps; cComp++)
            {
                  if
(CompCompareMax(pBoardArray[cBoard].pComps[cComp].CompType,comp))
                  {
                        pCompTypes[comp] =
pBoardArray[cBoard].pComps[cComp].CompType;
                        TempFreq =
pBoardArray[cBoard].pComps[cComp].CompFreq;
                        comp++;
                        cComp = cComp + TempFreq - 1;
                  }
            }
      }
      in.close();
      return;
}

void CBoardDlg::UpdateBox(const int _ID, const int _Data)
{
      const TEXT_SIZE = 16;
      char szText[TEXT_SIZE + 1];   // buffer for conversions
      CEdit *pDisplay = (CEdit *)(GetDlgItem(_ID));
      itoa(_Data, szText, 10);
      pDisplay->SetWindowText(szText);
}

int CBoardDlg::get_number(const char *_file, const int _next)
{
      char item[20];
      char command[20];
      int number;
      int types;
      int x_co;
      int y_co;
      int total_number[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
      int next = 0;

      ifstream in(_file);     // file input variable

      if (!in)
      {
            MessageBox("Cannot open file!", "Fail", MB_ICONWARNING);
            return 1;
      }

      in >> item >> item >> item >> item >> item >> item >>
item;

      for (int board = 0; board < NumOfBoardTypes; board++)
      {
            in >> pBoardArray[board].NumOfBoards;

            while(1)
            {
                  in >> types;
                  if (types == 0)
                  {
```

```
                        in >> command;
                        if (command[0] == 'N')
                        {
                              next += 1;
                        }
                        break;
                  }
                  in >> number;
                  total_number[board] += number;

                  for (int i=0; i<number; i++)
                  {
                        in >> x_co >> y_co;
                  }
            }
      }

      in.close();
      return total_number[_next];
}

bool CBoardDlg::CompCompare(const int _NewComp, const int _Next)
{
      for (int board = 0; board < _Next+1; board++)
      {
            for (int j = 0; j < pBoardArray[board].NumOfComps; j++)
            {
                  if (pBoardArray[board].pComps[j].CompType != 0)
                  {
                        if (_NewComp == pBoardArray[board].pComps[j-
1].CompType)
                        {
                              //MessageBox("Same
CompType","Same",MB_OK);
                              return false;
                        }
                  }
            }
      }

      return true;
}

bool CBoardDlg::CompCompareBoard(const int _NewComp, const int _Board,
const int _Comp)
{
      for (int cComp = 0; cComp < _Comp; cComp++)
      {
            if (pBoardArray[_Board].pComps[cComp].CompType != 0)
            {
                  if (_NewComp == pBoardArray[_Board].pComps[cComp-
1].CompType)
                  {
                        //MessageBox("Same CompType on
board","Same",MB_OK);
                        return false;
                  }
            }
      }
      return true;
}

bool CBoardDlg::CompCompareMax(const int _NewComp, const int _Count)
{
      for (int i = 0; i < _Count+1; i++)
      {
            if (i > 0 && _NewComp == pCompTypes[i-1])
            {
                  return false;
            }
```

```
	}
	return true;
}

void CBoardDlg::OnSelchangeCombo()
{
	const TEXT_SIZE = 16;
	char szText[TEXT_SIZE + 1];

	MaxNumOfCompTypes = 0;
	MaxNumOfLocations = 0;
	pCompTypes = 0;

	CComboBox *pBoard = (CComboBox *)(GetDlgItem(IDC_COMBO));

	for (int i = 0; i < 9; i++)
	{
		ClearBox(IDC_NUMOFBOARDS_A1+i);
		ClearBox(IDC_NUMOFCOMPS_A1+i);
	}

	int iCurSel = pBoard->GetCurSel();

	pBoard->GetLBText(iCurSel, szText);
	NumOfBoardTypes = atoi(szText);

	pBoardArray = new CBoard [NumOfBoardTypes];

	for (i = 0; i < NumOfBoardTypes; i++)
	{
		pBoardArray[i].NumOfBoards = 0;
		pBoardArray[i].NumOfComps = 0;
		pBoardArray[i].NumOfCompTypes = 0;
		pBoardArray[i].NumOfLocations = 0;
	}
	load_files("Coordinates.txt");
}

CBoard* CBoardDlg::get_pBoardArray()
{
	return pBoardArray;
}
```

Comps.cpp

```
// Comps.cpp: implementation of the CComps class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Pcb.h"
#include "Comps.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CComps::CComps()
{

}

CComps::~CComps()
{

}
```

FeederDlg.cpp

```
// FeederDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Pcb.h"
#include "FeederDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
//////
// CFeederDlg dialog


CFeederDlg::CFeederDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CFeederDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CFeederDlg)
		// NOTE: the ClassWizard will add member initialization
here
	//}}AFX_DATA_INIT
}


void CFeederDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CFeederDlg)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CFeederDlg, CDialog)
	//{{AFX_MSG_MAP(CFeederDlg)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
//////
// CFeederDlg message handlers

int CFeederDlg::GetItem(const int _ID)
{
	const TEXT_SIZE = 16;
	char szText[TEXT_SIZE + 1];   // buffer for conversions
	CEdit *pGet = (CEdit *)(GetDlgItem(_ID));
	pGet->GetWindowText(szText, TEXT_SIZE);
	return atoi(szText);
}

int CFeederDlg::Get_NumOfFeeders()
{
	return NumOfFeeders;
}

int CFeederDlg::Get_LengOfFeeder()
{
	return LengOfFeeder;
}

void CFeederDlg::OnOK()
{
	NumOfFeeders = GetItem(IDC_NUMOFFEEDERS);
```

```
        LengOfFeeder = GetItem(IDC_LENGOFFEEDER);

        CDialog::OnOK();
}

void CFeederDlg::UpdateBox(const int _ID, const int _Data)
{
        const TEXT_SIZE = 16;
        char szText[TEXT_SIZE + 1];   // buffer for conversions
        CEdit *pDisplay = (CEdit *)(GetDlgItem(_ID));
        itoa(_Data, szText, 10);
        pDisplay->SetWindowText(szText);
}

BOOL CFeederDlg::OnInitDialog()
{
        UpdateBox(IDC_NUMOFFEEDERS, NumOfFeeders);
        UpdateBox(IDC_LENGOFFEEDER, LengOfFeeder);
        return true;
}

void CFeederDlg::set_para(const int _NumOfFeeders, const int
_LengOfFeeder)
{
        NumOfFeeders = _NumOfFeeders;
        LengOfFeeder = _LengOfFeeder;
}
```

PcbDlg.cpp

```
// PcbDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Pcb.h"
#include "PcbDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
///////
// Declaration of global variables
int NumOfFeeders = 40;
int LengOfFeeder = 20;
int TabuSize = 4;
int TabuRestart = 2;
int MaxMoves = 10;
int MaxNoImp = 3;
int FeedSetupTime = 0;
int InsertTime = 2;
int PickTime = 2;
int HeadSpeed = 500;
int X = 300;
int Y = -500;
int X_Home = 100;
int Y_Home = -200;
int MaxNumOfLocations;
int MaxNumOfCompTypes;
int NumOfBoardTypes;
int F_Neigh_Size;
int S_Neigh_Size;
CPermutation *neigh_s;
CPermutation *neigh_f;

/////////////////////////////////////////////////////////////////////////////
///////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
      CAboutDlg();

// Dialog Data
      //{{AFX_DATA(CAboutDlg)
      enum { IDD = IDD_ABOUTBOX };
      //}}AFX_DATA

      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CAboutDlg)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
      //}}AFX_VIRTUAL

// Implementation
protected:
      //{{AFX_MSG(CAboutDlg)
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
```

```
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
///////
// CPcbDlg dialog

CPcbDlg::CPcbDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CPcbDlg::IDD, pParent)
{
	MaxNumOfLocations = 0;
	MaxNumOfCompTypes = 0;
	NumOfBoardTypes = 0;
	NumOfFeeders = 40;
	X_Home = 100;
	Y_Home = -200;
	X = 300;
	Y = -500;
	choice = RANDOM;
	ResultOpen = false;
	FinalOpen = false;
	flag = true;
	RestartMoves = 0;
	NoImpMoves = 0;
	Total_Time = 0;
	Setup_Time = 0;
	Board_Time = 0;

	//{{AFX_DATA_INIT(CPcbDlg)
	m_StringData = _T("");
	m_TextData = _T("");
	m_Board_A_Time = 0;
	m_Board_B_Time = 0;
	m_Board_C_Time = 0;
	m_Board_D_Time = 0.0f;
	m_Board_E_Time = 0.0f;
	m_Board_F_Time = 0.0f;
	m_Board_G_Time = 0.0f;
	m_Board_H_Time = 0.0f;
	m_Board_I_Time = 0.0f;
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon
in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CPcbDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CPcbDlg)
	DDX_Control(pDX, IDC_GO, m_GoButton);
	DDX_Control(pDX, IDC_PROGRESS, m_ProgressStatus);
	DDX_Text(pDX, IDC_STATUSBOX, m_StringData);
	DDX_Text(pDX, IDC_TEXTOUT, m_TextData);
	DDX_Text(pDX, IDC_BOARD_A_TIME, m_Board_A_Time);
```

```
	DDX_Text(pDX, IDC_BOARD_B_TIME, m_Board_B_Time);
	DDX_Text(pDX, IDC_BOARD_C_TIME, m_Board_C_Time);
	DDX_Text(pDX, IDC_BOARD_D_TIME, m_Board_D_Time);
	DDX_Text(pDX, IDC_BOARD_E_TIME, m_Board_E_Time);
	DDX_Text(pDX, IDC_BOARD_F_TIME, m_Board_F_Time);
	DDX_Text(pDX, IDC_BOARD_G_TIME, m_Board_G_Time);
	DDX_Text(pDX, IDC_BOARD_H_TIME, m_Board_H_Time);
	DDX_Text(pDX, IDC_BOARD_I_TIME, m_Board_I_Time);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPcbDlg, CDialog)
	//{{AFX_MSG_MAP(CPcbDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_COMMAND(IDC_BOARD, OnBoard)
	ON_COMMAND(IDC_FEEDER, OnFeeder)
	ON_COMMAND(IDC_COORDINATES, OnCoordinates)
	ON_COMMAND(IDC_TIME, OnTime)
	ON_COMMAND(IDC_ALGORITHM, OnAlgorithm)
	ON_BN_CLICKED(IDC_GO, OnGo)
	ON_EN_SETFOCUS(IDC_TEXTOUT, OnSetfocusTextout)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////
//////
// CPcbDlg message handlers

BOOL CPcbDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	UpdateBox(IDC_X_HOMEOUT, X_Home);
	UpdateBox(IDC_Y_HOMEOUT, Y_Home);
	UpdateBox(IDC_XOUT, X);
	UpdateBox(IDC_YOUT, Y);
	UpdateBox(IDC_FEEDSETUPTIMEOUT, FeedSetupTime);
	UpdateBox(IDC_PICKTIMEOUT, PickTime);
	UpdateBox(IDC_INSERTTIMEOUT, InsertTime);
	UpdateBox(IDC_HEADSPEEDOUT, HeadSpeed);
	UpdateBox(IDC_MAXMOVESOUT, MaxMoves);
	UpdateBox(IDC_MAXNOIMPOUT, MaxNoImp);
	UpdateBox(IDC_TABUSIZEOUT, TabuSize);
	UpdateBox(IDC_TABURESTARTOUT, TabuRestart);
	UpdateBox(IDC_INITIALMETHODOUT, "RANDOM");
	UpdateBox(IDC_NUMOFFEEDERSOUT, NumOfFeeders);
	UpdateBox(IDC_LENGOFFEEDEROUT, LengOfFeeder);

	LengOfFeeder = 20;
	TabuSize = 4;
	TabuRestart = 2;
	MaxMoves = 10;
	MaxNoImp = 3;
	FeedSetupTime = 0;
	InsertTime = 2;
	PickTime = 2;
	HeadSpeed = 500;
	X = 300;
	Y = -500;
	X_Home = 100;
	Y_Home = -200;

	tabu = new CTabuList(TabuSize);

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);
```

```
	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this
automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here

	return TRUE;  // return TRUE  unless you set the focus to a
control
}

void CPcbDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code
below
//  to draw the icon.  For MFC applications using the document/view
model,
//  this is automatically done for you by the framework.

void CPcbDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(),
0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}
```

```
// The system calls this to obtain the cursor to display while the
user drags
//  the minimized window.
HCURSOR CPcbDlg::OnQueryDragIcon()
{
      return (HCURSOR) m_hIcon;
}

void CPcbDlg::OnBoard()
{
      str<<"Setting boards parameters."<<ends;
      UpdateStatusBox(str);
      BoardDlg.DoModal();          // run Board dialog box
      m_GoButton.SetFocus();
      str<<"Board details updated."<<ends;
      UpdateStatusBox(str);

      pBoardArray = BoardDlg.get_pBoardArray();

      NumOfBoardTypes = BoardDlg.Get_NumOfBoardTypes();
      MaxNumOfCompTypes = BoardDlg.Get_MaxNumOfCompTypes();
      MaxNumOfLocations = BoardDlg.Get_MaxNumOfLocations();

      UpdateBox(IDC_NUMOFBOARDTYPESOUT, NumOfBoardTypes);
      UpdateBox(IDC_MAXCOMPTYPESOUT, MaxNumOfCompTypes);
      UpdateBox(IDC_MAXNUMOFLOCATIONSOUT, MaxNumOfLocations);
}

void CPcbDlg::OnFeeder()
{
      str<<"Setting feeder parameters."<<ends;
      UpdateStatusBox(str);
      while(1)
      {
            FeederDlg.set_para(NumOfFeeders, LengOfFeeder);
            FeederDlg.DoModal();    // run Feeder dialog box
            m_GoButton.SetFocus();
            NumOfFeeders = FeederDlg.Get_NumOfFeeders();
            LengOfFeeder = FeederDlg.Get_LengOfFeeder();
            if (NumOfFeeders >= MaxNumOfCompTypes)
            {
                  break;
            }
            else
            {
                  MessageBox("Number of feeders must be greater than
the maximum number of component types!",
                        "Warning",MB_ICONWARNING);
            }
      }

      UpdateBox(IDC_NUMOFFEEDERSOUT, NumOfFeeders);
      UpdateBox(IDC_LENGOFFEEDEROUT, LengOfFeeder);
      str<<"Feeder parameters updated."<<ends;
      UpdateStatusBox(str);
}

void CPcbDlg::OnCoordinates()
{
      str<<"Setting coordinates."<<ends;
      UpdateStatusBox(str);
      CoordinatesDlg.set_para(X, Y, X_Home, Y_Home);
      CoordinatesDlg.DoModal();     // run Coordinates dialog box
      m_GoButton.SetFocus();
      X = CoordinatesDlg.Get_X();
      Y = CoordinatesDlg.Get_Y();
      X_Home = CoordinatesDlg.Get_X_Home();
      Y_Home = CoordinatesDlg.Get_Y_Home();
```

```
        UpdateBox(IDC_XOUT, X);
        UpdateBox(IDC_YOUT, Y);
        UpdateBox(IDC_X_HOMEOUT, X_Home);
        UpdateBox(IDC_Y_HOMEOUT, Y_Home);
        str<<"Coordinates updated."<<ends;
        UpdateStatusBox(str);
}

void CPcbDlg::OnTime()
{
        str<<"Setting timing parameters."<<ends;
        UpdateStatusBox(str);
        TimeDlg.set_para(FeedSetupTime, PickTime, InsertTime,
HeadSpeed);
        TimeDlg.DoModal();      // run Time dialog box
        m_GoButton.SetFocus();
        FeedSetupTime = TimeDlg.Get_FeedSetupTime();
        PickTime = TimeDlg.Get_PickTime();
        InsertTime = TimeDlg.Get_InsertTime();
        HeadSpeed = TimeDlg.Get_HeadSpeed();

        UpdateBox(IDC_FEEDSETUPTIMEOUT, FeedSetupTime);
        UpdateBox(IDC_PICKTIMEOUT, PickTime);
        UpdateBox(IDC_INSERTTIMEOUT, InsertTime);
        UpdateBox(IDC_HEADSPEEDOUT, HeadSpeed);
        str<<"Timing parameters updated."<<ends;
        UpdateStatusBox(str);
}

void CPcbDlg::OnAlgorithm()
{
        str<<"Setting Tabu Search algorithm."<<ends;
        UpdateStatusBox(str);
        AlgorithmDlg.set_para(MaxMoves, MaxNoImp, TabuSize, TabuRestart,
choice);
        AlgorithmDlg.DoModal(); // run Algorithm dialog box
        m_GoButton.SetFocus();
        MaxMoves = AlgorithmDlg.Get_MaxMoves();
        MaxNoImp = AlgorithmDlg.Get_MaxNoImp();
        TabuSize = AlgorithmDlg.Get_TabuSize();
        delete [] tabu;
        tabu = new CTabuList(TabuSize);
        TabuRestart = AlgorithmDlg.Get_TabuRestart();
        choice = AlgorithmDlg.Get_choice();

        UpdateBox(IDC_MAXMOVESOUT, MaxMoves);
        UpdateBox(IDC_MAXNOIMPOUT, MaxNoImp);
        UpdateBox(IDC_TABUSIZEOUT, TabuSize);
        UpdateBox(IDC_TABURESTARTOUT, TabuRestart);

        if (choice == CENTROID)
        {
                UpdateBox(IDC_INITIALMETHODOUT, "Centroid");
        }
        else
        {
                UpdateBox(IDC_INITIALMETHODOUT, "Random");
        }
        str<<"Tabu Search algorithm updated."<<ends;
        UpdateStatusBox(str);
}

void CPcbDlg::UpdateBox(const int _ID, const int _data)
{
        const TEXT_SIZE = 16;
        char szText[TEXT_SIZE + 1];   // buffer for conversions
        pDisplay = (CEdit *)(GetDlgItem(_ID));
        itoa(_data, szText, 10);
        pDisplay->SetWindowText(szText);
        UpdateData(TRUE);
```

```
}

void CPcbDlg::UpdateBox(const int _ID, char *_data)
{
	pDisplay = (CEdit *)(GetDlgItem(_ID));
	pDisplay->SetWindowText(_data);
	UpdateData(TRUE);
}

int CPcbDlg::GetItem(const int _ID)
{
	const TEXT_SIZE = 16;
	char szText[TEXT_SIZE + 1];   // buffer for conversions
	CEdit *pGet = (CEdit *)(GetDlgItem(_ID));
	pGet->GetWindowText(szText, TEXT_SIZE);
	return atoi(szText);
}

void CPcbDlg::OnOK()
{
	CDialog::OnOK();
}

void CPcbDlg::UpdateStatusBox(ostrstream _stream)
{
	m_StringData = _stream.str();
	UpdateData(FALSE);
	UpdateWindow();
}

void CPcbDlg::OnGo()
{
	m_ProgressStatus.SetRange(0, NumOfBoardTypes+1);
	ClearBox(IDC_TEXTOUT);
	if (FinalOpen == false)
	{
		FinalFile.open("Final.txt", ios::out);
		FinalOpen = true;
		if (FinalFile.fail())
		{
			MessageBox("Unable to open final result file",
"Fail", MB_ICONWARNING);
			return;
		}
	}
	if (ResultOpen == false)
	{
		ResultFile.open("results.txt", ios::out);
		ResultOpen = true;
		if (ResultFile.fail())
		{
			MessageBox("Unable to open result file", "Fail",
MB_ICONWARNING);
			return;
		}
	}
	if (NumOfFeeders < MaxNumOfCompTypes)
	{
		MessageBox("Number of feeders < Total number of component
types", "Warning", MB_ICONWARNING);
		return;
	}
	ResultFile<<"Total number of
feeders:\t\t\t"<<NumOfFeeders<<"\n";
	ResultFile<<"Length of each feeder:\t\t\t\t"<<LengOfFeeder<<"
mm\n";
	ResultFile<<"Home coordinates:\t\t\t\t\t"<<X_Home<<" x
"<<Y_Home<<"\n";
	ResultFile<<"Feeder-to-board distance:\t\t\t"<<X<<" x
"<<Y<<"\n";
```

```
      ResultFile<<"Feed setup time:\t\t\t\t\t"<<FeedSetupTime<<"
seconds\n";
      ResultFile<<"Pick time:\t\t\t\t\t\t\t"<<PickTime<<" seconds\n";
      ResultFile<<"Insert time:\t\t\t\t\t\t"<<InsertTime<<"
seconds\n";
      ResultFile<<"Head speed:\t\t\t\t\t\t\t"<<HeadSpeed<<"
mm/seconds\n";
      ResultFile<<"Total number of moves:\t\t\t\t"<<MaxMoves<<"\n";
      ResultFile<<"Maximum moves without
improvement:\t"<<MaxNoImp<<"\n";
      ResultFile<<"Length of Tabu list:\t\t\t\t"<<TabuSize<<"\n";
      ResultFile<<"Number of Tabu restart
allowed:\t\t"<<TabuRestart<<"\n";

      //delete [] pBest_Exit;
      pBest_Exit = new CPositions[TabuRestart+1];
      srand((unsigned)time(NULL));  // initiate random number
generator
      /*CompNames = new CString [pBoardArray[0].NumOfCompTypes];
      CompNames[0] = "Res_1";
      CompNames[1] = "Res_2";
      CompNames[2] = "Cap_1";
      CompNames[3] = "Diode_1";
      CompNames[4] = "Cap_2";
      CompNames[5] = "Res_3";
      CompNames[6] = "Diode_2";
      CompNames[7] = "Cap_3";
      CompNames[8] = "Res_4";*/

      //
**********************************************************************
**
      // Start processing one board at a time
      //
**********************************************************************
**
      for (int cBoard = 0; cBoard < NumOfBoardTypes; cBoard++)
      {
            m_ProgressStatus.SetPos(cBoard+1);
            UpdateData(FALSE);
            UpdateWindow();
            current.initialise(pBoardArray, cBoard);
            Best.initialise(pBoardArray, cBoard);
            LocalBest.initialise(pBoardArray, cBoard);

            str<<"Processing board "<<cBoard<<ends;
            UpdateStatusBox(str);

            strText<<"-------------------------\r"<<endl<<"Processing
board "<<cBoard<<"\r"<<endl;
            strText<<"-------------------------\r"<<endl<<ends;
            UpdateTextBox(strText);

            ResultFile<<"\n\n*******************************";
            ResultFile<<"\n***** PROCESSING BOARD "<<cBoard<<" *****";
            ResultFile<<"\n*******************************\n";
            ResultFile<<"\nNumber of
board:\t\t\t"<<pBoardArray[cBoard].NumOfBoards;
            ResultFile<<"\nNumber of component
types:\t"<<pBoardArray[cBoard].NumOfCompTypes;
            ResultFile<<"\nNumber of
locations:\t\t"<<pBoardArray[cBoard].NumOfLocations;
            ResultFile<<"\nNumber of
components:\t\t"<<pBoardArray[cBoard].NumOfComps;
            ResultFile<<"\nInitialising...\n";

            //
**********************************************************************
            // Generate initial placement sequence randomly
```

```
            //
*****************************************************************
            current.pick_random_s(pBoardArray, cBoard);

            //
*****************************************************************
            // Generate initial feeder assignment either randomly or
using centroid
            // rule
            //
*****************************************************************
            if (choice == RANDOM)
            {
                  //
*****************************************************************
                  // Generate initial feeder assignment randomly
                  //
*****************************************************************
                  current.pick_random_f(pBoardArray, cBoard);

                  //
*****************************************************************
                  // Calculate time based on initial feeder assignment
and placement
                  // sequence
                  //
*****************************************************************
                  current.calculate_time(pBoardArray, cBoard);
                  ResultFile<<"\nInitial generated time Randomly = ";
            }
            else if (choice == CENTROID)
            {
                  //
*****************************************************************
                  // Generate initial feeder assignment using Centroid
Rule
                  //
*****************************************************************
                  current.generate_assign(pBoardArray, cBoard);

                  //
*****************************************************************
                  // Calculate time based on initial feeder assignment
and placement
                  // sequence
                  //
*****************************************************************
                  current.calculate_time(pBoardArray, cBoard);
                  ResultFile<<"\nInitial generated time using Centroid
Rule = ";
            }
            ResultFile<<current.get_time()<<" seconds\n";

            F_Neigh_Size = current.get_F_Neigh_Size();
            S_Neigh_Size = current.get_S_Neigh_Size(pBoardArray,
cBoard);

            ResultFile<<"\n\nF_Neigh_Size = "<<F_Neigh_Size;
            ResultFile<<"\nS_Neigh_Size = "<<S_Neigh_Size<<"\n";

            delete [] neigh_s;
            delete [] neigh_f;
            neigh_s = new CPermutation[S_Neigh_Size];
            neigh_f = new CPermutation[F_Neigh_Size];

            for (int i = 0; i < S_Neigh_Size; i++)
            {
                  neigh_s[i].initialise(pBoardArray, cBoard);
            }
            for (i = 0; i < F_Neigh_Size; i++)
```

```
            {
                  neigh_f[i].initialise(pBoardArray, cBoard);
            }
            int *pTest = current.get_pFeeder();
            for (i = 0; i < NumOfFeeders; i++)
            {
                  ResultFile<<"\nFeeder["<<i<<"] = "<<pTest[i];
            }
            ResultFile<<"\n";

            //
*****************************************************************
            // Current feeder assignment and placement sequence are
both the Best
            // LocalBest
            //
*****************************************************************
            Best = current;
            LocalBest = current;

            //
*****************************************************************
            // Clear tabu list before making moves
            //
*****************************************************************
            tabu->clear_list();

            //
*****************************************************************
            // Start making moves to determine the optimised feeder
assignment and
            // placement sequence
            //
*****************************************************************
            for (int cMoves = 1; cMoves <= MaxMoves; cMoves++)
            {
                  UpdateTime(cBoard, Best.get_time());
                  ResultFile<<"\nMove "<<cMoves<<" :";
                  //strText<<"Move_L "<<cMoves<<" : "<<ends;
                  //UpdateTextBox(strText);
                  //
*****************************************************************
                  // Create neighbourhood based on the initial feeder
assignment and
                  // placement sequence generated. Then, report
position of the best
                  // neighbour found.
                  //
*****************************************************************
                  solution = current.create_neigh_s(pBoardArray,
cBoard, current, ResultFile);

                  //
*****************************************************************
                  // Is the solution found the best?
                  //
*****************************************************************
                  do
                  {
                        //
*****************************************************************
                        // Compare the performance of the new solution
with the current
                        // one
                        // Identify the move
                        //
*****************************************************************
                        move = compare_s(neigh_s[solution], current,
pBoardArray, cBoard);
```

```
                        allowed = true;   // assume that the move is
allowed
                        if (tabu->list_find(move))    // check if the
move is taboo
                        {
                              // Aspiration criteria
                              if (Best.get_time() <=
neigh_s[solution].get_time())
                              {
                                    neigh_s[solution].poison();   //
make this move unattractive
                                    //
*************************************************
                                    // Find a new solution from the
neighbourhood
                                    //
*************************************************
                                    solution =
current.find_best_s(solution, pBoardArray, cBoard);
                                    allowed = false;  // repeat to
check the new solution
                              }
                        }
                  } while (allowed == false);

                  //
*****************************************************************
                  // Is the move just after a local best?
                  //
*****************************************************************
                  if (flag == true)
                  {
                        pBest_Exit[RestartMoves] = move;
                        flag = false;
                  }

                  //
*****************************************************************
                  // Is the current solution a new local best?
                  //
*****************************************************************
                  if (neigh_s[solution].get_time() <
LocalBest.get_time())
                  {
                        LocalBest = neigh_s[solution];      // Set
local best
                        flag = true;                              //
The next move is after a local best
                        NoImpMoves = 0;                           //
Reset the number of NoImpMoves
                        RestartMoves = 0;                   // Allow
all restart attempts
                  }
                  else
                  {
                        //
*************************************************************
                        // If the current solution is not better than
the previous,
                        // then, it is considered as the 'Non-
Improvement Move'. Hence,
                        // increate the NoImpMoves counter.
                        //
*************************************************************
                        NoImpMoves++;     // loose some 'patience'
                  }

                  //
*****************************************************************
```

```
                    // Check if the current solution is even the best
                    //  -> If YES, set it as the best
                    //  -> If NO, proceed
                    //
*************************************************************
                    if (neigh_s[solution].get_time() < Best.get_time())
                    {
                         Best = neigh_s[solution];     // Set new Best
                    }

                    //
*************************************************************
                    // Have we exceeded the max non-improvement moves
limit?
                    //  -> If YES, try to restart, else RANDOMISE again
                    //  -> If NO, make the move
                    //
*************************************************************
                    if (NoImpMoves > MaxNoImp)
                    {
                         tabu->clear_list();           // clear tabu
list
                         NoImpMoves = 0;               // 'be patient'
again

                    //
*********************************************************
                    // Restart, if the restarting limit has not
been reached
                    //
*********************************************************
                    if (RestartMoves < TabuRestart)
                    {
                         RestartMoves++;   // Remember that we
have restarted

                    //
*****************************************************
                    // Fill the tabu list with pBest_Exit
(we don't want to
                    // follow the same path)
                    //
*****************************************************
                    ResultFile<<" RESTART "<<RestartMoves<<"
:  Taboo -> ";
                    //strText<<" RESTART "<<RestartMoves<<"
:  Taboo -> ";
                    for (int x = 0; x < RestartMoves; x++)
                    {
                         //
*************************************************
                         // Make move that left local best
last time taboo
                         //
*************************************************
                         tabu->push_list(pBest_Exit[x]);
      ResultFile<<"<"<<pBest_Exit[x].a<<","<<pBest_Exit[x].b<<">  ";
      //strText<<"<"<<pBest_Exit[x].a<<","<<pBest_Exit[x].b<<">  ";
                    }
                    ResultFile<<"\n";
                    //strText<<"\r"<<endl<<ends;
                    //UpdateTextBox(strText);

                    //
*********************************************************
                    // The move we are going to make should
be appended to
                    // pBest_Exit
```

```
                                    //
*****************************************************
                                    flag = true;
                                    current = LocalBest;    // go back to
local best
                              }

                              //
*************************************************************
                              // If can't restart, RANDOMISE
                              //
*************************************************************
                              else
                              {
                                    ResultFile<<" RANDOMISE\n";
                                    //strText<<" RANDOMISE\r"<<endl<<ends;
                                    //UpdateTextBox(strText);
                                    current.pick_random_s(pBoardArray,
cBoard);          // pick random placement sequence
                                    current.calculate_time(pBoardArray,
cBoard);    // calculate time
                                    LocalBest = current;
                              // reset LocalBest
                                    RestartMoves = 0;
                              // allow restart again
                              }
                        }

                        //
*************************************************************
                        // Make the move if the MaxNoImp has not been
reached
                        //
*************************************************************
                        else
                        {
                              current = neigh_s[solution];  // make move
                              ResultFile<<" Swaping "<<move.a<<" with
"<<move.b<<" -> "<<current.get_time()<<" seconds\n";
                              //strText<<" Swaping "<<move.a<<" with
"<<move.b<<"\t->\t"<<current.get_time()<<" seconds\r"<<endl<<ends;
                              //UpdateTextBox(strText);

                              //
*************************************************************
                              // Update the tabu list
                              //
*************************************************************
                              if (tabu->list_size() == TabuSize)
                              {
                                    tabu->pop_list(); // remove oldest move
from the tabu list
                              }
                              tabu->push_list(move);  // insert current move
into the tabu list
                        }
                  }
                  ResultFile<<"\nBest components placement sequence
found\nTime = "<<Best.get_time()<<" seconds\n\n";

      ResultFile<<"*****************************************************
************\n";
                  ResultFile<<"Start finding optimum placement sequence and
feeder assignment.\n";

      ResultFile<<"*****************************************************
************\n";
                  //strText<<"\r"<<endl<<ends;
                  //UpdateTextBox(strText);
```

```
            //MessageBox("Finished making initial
placement","Test",MB_OK);
            int cycle = 1;
            //
*************************************************************
            // Now, find the best feeder assignment repeatedly, based
on the
            // placement sequence found
            //
*************************************************************
            do
            {
                  ResultFile<<"\n Cycle "<<cycle<<"\n --------\n";
                  //strText<<" Cycle "<<cycle<<"\r"<<endl<<"----------
-----\r"<<endl<<ends;
                  //UpdateTextBox(strText);
                  cycle++;
                  //
*************************************************************
                  // First, find the best feeder assignment
                  //  -> Reset some variables
                  //
*************************************************************
                  NoImpMoves = 0;
                  RestartMoves = 0;
                  flag = true;
                  for (int n = 0; n < TabuRestart+1; n++)
                  {
                        pBest_Exit[n].a = 0;
                        pBest_Exit[n].b = 0;
                  }
                  current = Best;   // start from the best placement
seq found
                  LocalBest = current;

                  tabu->clear_list();

                  //
*************************************************************
                  // Start making moves
                  //
*************************************************************
                  for (cMoves = 1; cMoves <= MaxMoves; cMoves++)
                  {
                        UpdateTime(cBoard, Best.get_time());
                        ResultFile<<"\nMove_f "<<cMoves<<" : ";
                        //strText<<"Move_F "<<cMoves<<" : "<<ends;
                        //UpdateTextBox(strText);
                        //
*************************************************************
                        // Create neighbourhood based on the initial
feeder
                        // assignment generated. Then, report position
of the best
                        // neighbour found.
                        //
*************************************************************
                        solution = current.create_neigh_f(pBoardArray,
cBoard, current, FedNeighFile);

                        //
*************************************************************
                        // Is the solution found the best?
                        //
*************************************************************
                        do
                        {
                              //
*************************************************
```

```
                                    // Compare the performance of the new
found solution
                                    // with the current one
                                    //
*****************************************************
                                    move = compare_f(neigh_f[solution],
current);
                                    allowed = true;   // assume that the
move is allowed
                                    //
*****************************************************
                                    // Check if the move is taboo
                                    //  -> If YES, check if the move is
better
                                    //  -> If NO, check if it is just after
a local best
                                    //
*****************************************************
                                    if (tabu->list_find(move))
                                    {
                                          //
*************************************************
                                          // Check if the tabooed move is
worse than the best
                                          //  -> If YES, make the move
unattractive, then
                                          //     find a better solution.
                                          //
*************************************************
                                          if (Best.get_time() <=
neigh_f[solution].get_time())
                                          {
                                                neigh_f[solution].poison();
                                                solution =
current.find_best_f(solution, pBoardArray, cBoard);
                                                allowed = false;
                                          }
                                    }
                              } while (allowed == false);

                              //
*********************************************************
                              // Check if the move is just after a local
best
                              //  -> If YES, remember the move
                              //  -> If NO, proceed
                              //
*********************************************************
                              if (flag == true)
                              {
                                    pBest_Exit[RestartMoves] = move;
                                    flag = false;
                                    /*FedNeighFile<<"\nMove is after a Local
Best.\n\t";
                                    int *pTest4 =
neigh_f[solution].get_pFeeder();
                                    for (i = 0; i < NumOfFeeders; i++)
                                    {
                                          FedNeighFile<<pTest4[i]<<"  ";
                                    }
                                    FedNeighFile<<"  ->
("<<move.a<<","<<move.b<<")   :  Time  ->
"<<neigh_f[solution].get_time()<<" seconds\n";*/
                              }

                              //
*********************************************************
                              // Check if the current solution is better
than local best
```

```
                        //  -> If YES, set it as the new LocalBest
                        //  -> If NO, increment NoImpMoves
                        //
*********************************************************
                        if (neigh_f[solution].get_time() <
LocalBest.get_time())
                        {
                              LocalBest = neigh_f[solution];      //
Set local best
                              /*FedNeighFile<<"=> Local Best solution
found:\n\t";
                              int *pTest3 = LocalBest.get_pFeeder();
                              for (i = 0; i < NumOfFeeders; i++)
                              {
                                    FedNeighFile<<pTest3[i]<<"  ";
                              }
                              FedNeighFile<<"  ->
("<<move.a<<","<<move.b<<")   :  Time  ->  "<<LocalBest.get_time()<<"
seconds\n";*/
                              flag = true;            // The next move
is after a local best
                              NoImpMoves = 0;         // Reset the
number of NoImpMoves
                              RestartMoves = 0; // Allow all restart
attempts
                        }
                        else
                        {
                              //
*****************************************************
                              // If the current solution is not better
than the
                              // previous, then, it is considered as
the 'Non-
                              // Improvement Move'. Hence, increase
the NoImpMoves
                              // counter.
                              //
*****************************************************
                              NoImpMoves++;
                        }

                        //
*********************************************************
                        // Check if the current solution is even the
best
                        //  -> If YES, set it as the new Best
                        //  -> If NO, proceed
                        //
*********************************************************
                        if (neigh_f[solution].get_time() <
Best.get_time())
                        {
                              Best = neigh_f[solution];     // Set new
Best
                              /*FedNeighFile<<"=> Best =\n\t";
                              int *pTest2 = Best.get_pFeeder();
                              for (i = 0; i < NumOfFeeders; i++)
                              {
                                    FedNeighFile<<pTest2[i]<<"  ";
                              }
                              FedNeighFile<<"  ->
("<<move.a<<","<<move.b<<")   :  Time  ->  "<<Best.get_time()<<"
seconds\n";*/
                        }

                        //
*********************************************************
                        // Have we exceeded the max non-improvement
moves limit?
```

```
                        //  -> If YES, try to restart, else RANDOMISE
                        //  -> If NO, make the move
                        //
*****************************************************
                        if (NoImpMoves > MaxNoImp)
                        {
                              tabu->clear_list();     // clear the
tabu list
                              NoImpMoves = 0;
                              if (RestartMoves < TabuRestart)
                              {
                                    RestartMoves++;   // Remember that
we have restarted
                                    ResultFile<<" RESTART
"<<RestartMoves<<"  : Taboo -> ";
                                    //strText<< " RESTART
"<<RestartMoves<<"  : Taboo -> ";
                                    //FedNeighFile<<" RESTART
"<<RestartMoves<<"  : Taboo -> ";
                                    for (int x = 0; x < RestartMoves;
x++)
                                    {
                                          //
*****************************************
                                          // Taboo move that left
local best last time
                                          //
*****************************************
                                          tabu-
>push_list(pBest_Exit[x]);
      ResultFile<<"<"<<pBest_Exit[x].a<<","<<pBest_Exit[x].b<<">  ";
      //strText<<"<"<<pBest_Exit[x].a<<","<<pBest_Exit[x].b<<">  ";
      //FedNeighFile<<"("<<pBest_Exit[x].a<<","<<pBest_Exit[x].b<<")
";
                                    }
                                    ResultFile<<"\n";
                                    //strText<<"\r"<<endl<<ends;
                                    //UpdateTextBox(strText);
                                    //FedNeighFile<<"\n";

                                    flag = true;
                                    current = LocalBest;
                              }
                              else
                              {
                                    //
*********************************************
                                    // Cannot restart, RANDOMISE
                                    //
*********************************************
                                    ResultFile<<" RANDOMISE\n";
                                    //strText<<"
RANDOMISE\r"<<endl<<ends;
                                    //UpdateTextBox(strText);
                                    current.pick_random_f(pBoardArray,
cBoard);
      current.calculate_time(pBoardArray, cBoard);
                                    LocalBest = current;
                                    RestartMoves = 0;
                              }
                        }
                        else
                        {
                              current = neigh_f[solution];
                              ResultFile<<" Swaping "<<move.a<<" and
"<<move.b<<" -> "<<current.get_time()<<" seconds\n";
```

```
                                    //strText<<" Swaping "<<move.a<<" and
"<<move.b<<"\t->\t"<<current.get_time()<<" seconds\r"<<endl<<ends;
                                    //UpdateTextBox(strText);

                                    //
**************************************************
                                    // MaxNoImp has not reached, make move
taboo
                                    //
**************************************************
                                    if (tabu->list_size() == TabuSize)
                                    {
                                          tabu->pop_list();
                                    }
                                    tabu->push_list(move);
                              }
                        }
                        //strText<<"\r"<<endl<<ends;
                        //UpdateTextBox(strText);
                        ResultFile<<"\nBest feeder assignment found\nTime =
"<<Best.get_time()<<" seconds\n";
                        ResultFile<<"Finding optimum placement sequence.\n";

                        double time = Best.get_time();      // variable to
control the repeatition

                        //
**********************************************************
                        // Second, find the best component placement
sequence
                        //  -> Reset some variables
                        //
**********************************************************
                        NoImpMoves = 0;
                        RestartMoves = 0;
                        flag = true;
                        for (int m = 0; m < TabuRestart+1; m++)
                        {
                              pBest_Exit[m].a = 0;    // Initialise
                              pBest_Exit[m].b = 0;    // Initialise
                        }
                        current = Best;
                        LocalBest = current;
                        tabu->clear_list();
                        for (cMoves = 1; cMoves <= MaxMoves; cMoves++)
                        {
                              UpdateTime(cBoard, Best.get_time());
                              ResultFile<<"\nMove_s "<<cMoves<<" : ";
                              //strText<<"Move_S "<<cMoves<<" : "<<ends;
                              //UpdateTextBox(strText);
                              solution = current.create_neigh_s(pBoardArray,
cBoard, current, ResultFile);
                              do
                              {
                                    move = compare_s(neigh_s[solution],
current, pBoardArray, cBoard);
                                    allowed = true;
                                    if (tabu->list_find(move))
                                    {
                                          if (Best.get_time() <=
neigh_s[solution].get_time())
                                          {
                                                neigh_s[solution].poison();
                                                solution =
current.find_best_s(solution, pBoardArray, cBoard);
                                                allowed = false;
                                          }
                                    }
                              } while (allowed == false);
                              if (flag == true)
```

```
                        {
                              pBest_Exit[RestartMoves] = move;
                              flag = false;
                        }
                        if (neigh_s[solution].get_time() <
LocalBest.get_time())
                        {
                              LocalBest = neigh_s[solution];
                              flag = true;
                              NoImpMoves = 0;
                              RestartMoves = 0;
                        }
                        else
                        {
                              NoImpMoves++;
                        }

                        if (neigh_s[solution].get_time() <
Best.get_time())
                        {
                              Best = neigh_s[solution];
                        }

                        if (NoImpMoves > MaxNoImp)
                        {
                              tabu->clear_list();
                              NoImpMoves = 0;
                              if (RestartMoves < TabuRestart)
                              {
                                    RestartMoves++;
                                    ResultFile<<" RESTART
"<<RestartMoves<<" : Taboo -> ";
                                    //strText<<" RESTART
"<<RestartMoves<<" : Taboo -> ";
                                    for (int x = 0; x < RestartMoves;
x++)
                                    {
                                          tabu-
>push_list(pBest_Exit[x]);
      ResultFile<<"<"<<pBest_Exit[x].a<<","<<pBest_Exit[x].b<<"> ";
      //strText<<"<"<<pBest_Exit[x].a<<","<<pBest_Exit[x].b<<"> ";
                                    }
                                    ResultFile<<"\n";
                                    //strText<<"\r"<<endl<<ends;
                                    //UpdateTextBox(strText);
                                    flag = true;
                                    current = LocalBest;
                              }
                              else
                              {
                                    ResultFile<<" RANDOMISE\n";
                                    //strText<<"
RANDOMISE\r"<<endl<<ends;
                                    //UpdateTextBox(strText);
                                    current.pick_random_s(pBoardArray,
cBoard);
      current.calculate_time(pBoardArray, cBoard);
                                    LocalBest = current;
                                    RestartMoves = 0;
                              }
                        }
                        else
                        {
                              current = neigh_s[solution];
                              ResultFile<<" Swaping "<<move.a<<" and
"<<move.b<<" -> "<<current.get_time()<<" seconds\n";
```

```
                                    //strText<<" Swaping "<<move.a<<" and
"<<move.b<<"\t->\t"<<current.get_time()<<" seconds\r"<<endl<<ends;
                                    //UpdateTextBox(strText);
                                    if (tabu->list_size() == TabuSize)
                                    {
                                          tabu->pop_list();
                                    }
                                    tabu->push_list(move);
                              }
                        }
                        //strText<<"\r"<<endl<<ends;
                        //UpdateTextBox(strText);
                        ResultFile<<"\nBest components placement sequence
found\nTime = "<<Best.get_time()<<" seconds\n";
                        ResultFile<<"Finding optimum feeder assignment.\n";

                        if (Best.get_time() == time)  // no more
improvement?
                        {
                              Repeat = false;   // stop repeating
                        }
                        else
                        {
                              Repeat = true;
                        }
                  } while (Repeat == true);     // if repeat is still
allowed

                  Setup_Time = pBoardArray[cBoard].NumOfCompTypes *
FeedSetupTime;
                  Board_Time = (Setup_Time + Best.get_time()) *
pBoardArray[cBoard].NumOfBoards;
                  ResultFile<<"\nTotal time for board "<<cBoard<<" is : "
                        <<Board_Time/pBoardArray[cBoard].NumOfBoards<<"
seconds\n";
                  //strText<<"\r"<<endl<<"Total time for board "<<cBoard<<"
is : "
                  //    <<Board_Time/pBoardArray[cBoard].NumOfBoards<<"
seconds\r"<<endl<<"\r"<<endl<<"\r"<<endl<<ends;
                  //UpdateTextBox(strText);
                  Total_Time += Board_Time;

                  strText<<"Optimum assembly time found ->
"<<Best.get_time()<<"\r"<<endl<<"\r"<<endl<<endl<<ends;
                  UpdateTextBox(strText);
                  FinalFile<<"Optimum feeder assignment for board
"<<cBoard<<":"<<endl;
                  int *pFinalFeeder = Best.get_pFeeder();
                  for (i = 0; i < NumOfFeeders; i++)
                  {
                        FinalFile<<pFinalFeeder[i]<<"\t";
                  }
                  FinalFile<<"\n\nOptimum component placement sequence for
board "<<cBoard<<":"<<endl;
                  int *pFinalSeq = Best.get_pSeq();
                  for (i = 0; i < pBoardArray[cBoard].NumOfLocations; i++)
                  {
                        //FinalFile<<hex<<CompNames[pFinalSeq[i]]<<"\t";

            FinalFile<<pBoardArray[cBoard].pComps[pFinalSeq[i]].CompType<<"\
t";
                  }
                  FinalFile<<"\n\nAssembly time  ->  "<<Best.get_time()<<"
seconds\n\n\n\n";

            }     // end for (int cBoard = 0; cBoard < NumOfBoardTypes;
cBoard++)

            int t1 = Total_Time/3600;
            int t2 = (Total_Time%3600)/60;
```

```
	int t3 = (Total_Time%3600)%60;

	ResultFile<<"*****************************************************
*";
	ResultFile<<"\n\nTotal time for all boards is : "<<t1<<":"<<t2
		<<":"<<t3<<" h:m:s\n";

	ResultFile.close();
	ResultOpen = false;
	FinalFile.close();
	FinalOpen = false;
	m_ProgressStatus.SetPos(NumOfBoardTypes+1);
	MessageBox("Repetition completed. Optimum feeder assignment and
placement sequence found!",
		"Finish", MB_ICONINFORMATION);
	m_ProgressStatus.SetPos(0);
	m_GoButton.SetFocus();
	delete [] pBest_Exit;
	return;
}

void CPcbDlg::UpdateTextBox(ostrstream _stream)
{
	pDisplay = (CEdit *)(GetDlgItem(IDC_TEXTOUT));
	Size = pDisplay->GetWindowTextLength();
	Test = new char [Size+1];
	pDisplay->GetWindowText(Test,Size+1);

	char *buffer = _stream.str();

	if (Size == 0)
	{
		m_TextData.Format("%s", buffer);
	}
	else
	{
		m_TextData.Format("%s %s", Test, buffer);
	}
	UpdateData(FALSE);
	UpdateWindow();
	delete [] Test;
}

void CPcbDlg::OnSetfocusTextout()
{
	pDisplay = (CEdit *)(GetDlgItem(IDC_TEXTOUT));
	pDisplay->SetFocus();
}

void CPcbDlg::ClearBox(const int _ID)
{
	m_TextData.Empty();
	UpdateData(FALSE);
	//UpdateWindow();
}

void CPcbDlg::OnCancel()
{
	delete [] pBoardArray;
	CDialog::OnCancel();
}

void CPcbDlg::UpdateTime(const int _cBoard, const float _BestTime)
{
	if (_cBoard == 0)
	{
		m_Board_A_Time = _BestTime;
	}
	if (_cBoard == 1)
	{
```

```
            m_Board_B_Time = _BestTime;
        }
        if (_cBoard == 2)
        {
            m_Board_C_Time = _BestTime;
        }
        if (_cBoard == 3)
        {
            m_Board_D_Time = _BestTime;
        }
        if (_cBoard == 4)
        {
            m_Board_E_Time = _BestTime;
        }
        if (_cBoard == 5)
        {
            m_Board_F_Time = _BestTime;
        }
        if (_cBoard == 6)
        {
            m_Board_G_Time = _BestTime;
        }
        if (_cBoard == 7)
        {
            m_Board_H_Time = _BestTime;
        }
        if (_cBoard == 8)
        {
            m_Board_I_Time = _BestTime;
        }
        UpdateData(FALSE);
        UpdateWindow();
}

void CPcbDlg::ClearTime()
{
        for (int i = 0; i < NumOfBoardTypes; i++)
        {
            ClearBox(IDC_BOARD_A_TIME + i);
        }
}
```

Positions.cpp

```
// Positions.cpp: implementation of the CPositions class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Pcb.h"
#include "Positions.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CPositions::CPositions()
{

}

CPositions::~CPositions()
{

}
```

TabuList.cpp

```
// TabuList.cpp: implementation of the CTabuList class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Pcb.h"
#include "TabuList.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CTabuList::CTabuList(const int _TabuSize)
{
	TabuSize = _TabuSize;
	a = new CBuffer(TabuSize);
	b = new CBuffer(TabuSize);
}

CTabuList::~CTabuList()
{
	delete a;
	delete b;
}

bool CTabuList::list_find(CPositions &move) const
{
	int end = a->get_size();
	for (int i = 0; i < end; i++)
	{
		if (a->get_pointer(i) == move.a && b->get_pointer(i) ==
move.b)
		{
			return true;
		}
	}
	return false;
}

int CTabuList::list_size()
{
	return a->get_size();
}

void CTabuList::pop_list()
{
	a->pop_buffer(TabuSize);
	b->pop_buffer(TabuSize);
}

void CTabuList::push_list(CPositions &move)
{
	a->push_buffer(TabuSize, move.a);
	b->push_buffer(TabuSize, move.b);
}

void CTabuList::clear_list()
{
	a->clear_buffer();
	b->clear_buffer();
}
```

Board.cpp

```
// Board.cpp: implementation of the CBoard class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Pcb.h"
#include "Board.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CBoard::CBoard()
{

}

CBoard::~CBoard()
{

}
```

Buffer.cpp

```
// Buffer.cpp: implementation of the CBuffer class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Pcb.h"
#include "Buffer.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//int *pProd;
//int BufferLast;
//int BufferSize;

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CBuffer::CBuffer(const int _TabuSize)
{
      pProd = new int [_TabuSize];
}

CBuffer::~CBuffer()
{
      delete pProd;
}

int CBuffer::get_pointer(int _k) const
{
      return pProd[_k];
}

int CBuffer::get_size() const
{
      return BufferSize;
}

int CBuffer::pop_buffer(int max)
{
      int i;
      assert(BufferSize > 0); // verify that buffer is not empty
      i = BufferLast - BufferSize;
      if (i < 0)
      {
            i += max;
      }
      BufferSize--;
      return pProd[i];
}

void CBuffer::push_buffer(int max, int item)
{
      assert(BufferSize < max);     // verify that the buffer is not
full
      BufferSize++;                       // increment the size of the
buffer
      pProd[BufferLast] = item;     // 'insert' the item into the
buffer
      BufferLast++;                       // increment the buffer's
address
      if (BufferLast >= max)
      {
```

```
            BufferLast = 0;                 // start from the first
location (overwrite)
      }
      return;
}

void CBuffer::clear_buffer()
{
      BufferSize = 0;
      BufferLast = 0;
      return;
}

/*void CBuffer::Buffer_Init(const int _TabuSize)
{
      pProd = new int [_TabuSize];
}*/
```

CoordinatesDlg.cpp

```
// CoordinatesDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Pcb.h"
#include "CoordinatesDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
//////
// CCoordinatesDlg dialog


CCoordinatesDlg::CCoordinatesDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CCoordinatesDlg::IDD, pParent)
{
	X = 300;
	Y = -500;
	X_Home = 100;
	Y_Home = -200;
	//{{AFX_DATA_INIT(CCoordinatesDlg)
		// NOTE: the ClassWizard will add member initialization
here
	//}}AFX_DATA_INIT
}


void CCoordinatesDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CCoordinatesDlg)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CCoordinatesDlg, CDialog)
	//{{AFX_MSG_MAP(CCoordinatesDlg)
	ON_EN_CHANGE(IDC_X, OnChangeX)
	ON_EN_CHANGE(IDC_Y, OnChangeY)
	ON_EN_CHANGE(IDC_X_HOME, OnChangeXHome)
	ON_EN_CHANGE(IDC_Y_HOME, OnChangeYHome)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

int CCoordinatesDlg::GetItem(const int _ID)
{
	const TEXT_SIZE = 16;
	char szText[TEXT_SIZE + 1];   // buffer for conversions
	CEdit *pGet = (CEdit *)(GetDlgItem(_ID));
	pGet->GetWindowText(szText, TEXT_SIZE);
	return atoi(szText);
}

/////////////////////////////////////////////////////////////////////////////
//////
// CCoordinatesDlg message handlers

void CCoordinatesDlg::OnChangeX()
{
	X = GetItem(IDC_X);
}
```

```
void CCoordinatesDlg::OnChangeY()
{
	Y = GetItem(IDC_Y);
}

void CCoordinatesDlg::OnChangeXHome()
{
	X_Home = GetItem(IDC_X_HOME);
}

void CCoordinatesDlg::OnChangeYHome()
{
	Y_Home = GetItem(IDC_Y_HOME);
}

int CCoordinatesDlg::Get_X()
{
	return X;
}

int CCoordinatesDlg::Get_Y()
{
	return Y;
}

int CCoordinatesDlg::Get_X_Home()
{
	return X_Home;
}

int CCoordinatesDlg::Get_Y_Home()
{
	return Y_Home;
}

void CCoordinatesDlg::UpdateBox(const int _ID, const int _Data)
{
	const TEXT_SIZE = 16;
	char szText[TEXT_SIZE + 1];   // buffer for conversions
	CEdit *pDisplay = (CEdit *)(GetDlgItem(_ID));
	itoa(_Data, szText, 10);
	pDisplay->SetWindowText(szText);
}

BOOL CCoordinatesDlg::OnInitDialog()
{
	UpdateBox(IDC_X, X);
	UpdateBox(IDC_Y, Y);
	UpdateBox(IDC_X_HOME, X_Home);
	UpdateBox(IDC_Y_HOME, Y_Home);
	return true;
}

void CCoordinatesDlg::set_para(const int _X, const int _Y, const int
_X_Home, const int _Y_Home)
{
	X = _X;
	Y = _Y;
	X_Home = _X_Home;
	Y_Home = _Y_Home;
}
```

Pcb.cpp

```
// Pcb.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Pcb.h"
#include "PcbDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
//
// CPcbApp

BEGIN_MESSAGE_MAP(CPcbApp, CWinApp)
	//{{AFX_MSG_MAP(CPcbApp)
		// NOTE - the ClassWizard will add and remove mapping
macros here.
		//    DO NOT EDIT what you see in these blocks of
generated code!
	//}}AFX_MSG
	ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
//
// CPcbApp construction

CPcbApp::CPcbApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
//
// The one and only CPcbApp object

CPcbApp theApp;

/////////////////////////////////////////////////////////////////////////////
//
// CPcbApp initialization

BOOL CPcbApp::InitInstance()
{
	AfxEnableControlContainer();

	// Standard initialization
	// If you are not using these features and wish to reduce the
size
	//  of your final executable, you should remove from the
following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC
in a shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC
statically
#endif

	CPcbDlg dlg;
	m_pMainWnd = &dlg;
```

```
	int nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with OK
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with Cancel
	}

	// Since the dialog has been closed, return FALSE so that we
exit the
	//  application, rather than start the application's message
pump.
	return FALSE;
}
```

Permutation.cpp

```
// Permutation.cpp: implementation of the CPermutation class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Pcb.h"
#include "Permutation.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Declaration of global variables
extern int NumOfFeeders;
extern int LengOfFeeder;
extern int TabuSize;
extern int TabuRestart;
extern int MaxMoves;
extern int MaxNoImp;
extern int FeedSetupTime;
extern int InsertTime;
extern int PickTime;
extern int HeadSpeed;
extern int X;
extern int Y;
extern int X_Home;
extern int Y_Home;
extern int MaxNumOfLocations;
extern int MaxNumOfCompTypes;
extern int NumOfBoardTypes;
extern int F_Neigh_Size;
extern int S_Neigh_Size;
extern CPermutation *neigh_s;
extern CPermutation *neigh_f;


//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CPermutation::CPermutation():time(0)
{
      FedNeighOpen = false;
}

CPermutation::~CPermutation()
{

}

CPermutation CPermutation::operator =(const CPermutation &source)
{
      time = source.time;
      for (int i = 0; i < NumOfSeq; i++)
      {
            pSeq[i] = source.pSeq[i];
      }
      for (i = 0; i < NumOfFeeders; i++)
      {
            pFeeder[i] = source.pFeeder[i];
      }
      return *this;
}

int CPermutation::Get_NumOfFeeders()
```

```
{
	return NumOfFeeders;
}

CPositions compare_f(const CPermutation &p1, const CPermutation &p2)
{
	int i = 0;
	CPositions move;
	move.a = -1;
	move.b = -1;
	while (i < NumOfFeeders && move.b < 0)
	{
		if (p1.pFeeder[i] != p2.pFeeder[i])
		{
			if (move.a < 0)
			{
				move.a = i;
			}
			else
			{
				move.b = i;
			}
		}
		i++;
	}
	return move;
}

CPositions compare_s(const CPermutation &p1, const CPermutation &p2,
                                const CBoard *_pBoardArray, const int
_Board)
{
	int i = 0;
	CPositions move;
	move.a = -1;
	move.b = -1;
	while (i < _pBoardArray[_Board].NumOfLocations && move.b < 0)
	{
		if (p1.pSeq[i] != p2.pSeq[i])
		{
			if (move.a < 0)
			{
				move.a = i;
			}
			else
			{
				move.b = i;
			}
		}
		i++;
	}
	return move;
}

int CPermutation::get_F_Neigh_Size()
{
	int F_Neigh_Size = 0;
	int i, j = 0;
	for (i = 0; i < NumOfFeeders-1; i++)
	{
		for (j = i+1; j < NumOfFeeders; j++)
		{
			if (pFeeder[i] != pFeeder[j])
			{
				F_Neigh_Size++;
			}
		}
	}
	return F_Neigh_Size;
}
```

```
int CPermutation::get_S_Neigh_Size(const CBoard *_pBoardArray, const
int _Board)
{
	int S_Neigh_Size = 0;
	int i, j = 0;
	for (i = 0; i < _pBoardArray[_Board].NumOfLocations-1; i++)
	{
		for (j = i+1; j < _pBoardArray[_Board].NumOfLocations;
j++)
		{
			S_Neigh_Size++;
		}
	}
	return S_Neigh_Size;
}

int CPermutation::get_position_b(int _count)
{
	return pCoordinates[_count].b;
}

int CPermutation::get_position_a(int _count)
{
	return pCoordinates[_count].a;
}

double CPermutation::get_time()
{
	return time;
}

int* CPermutation::get_pSeq()
{
	return pSeq;	// return pointer
}

int* CPermutation::get_pFeeder()
{
	return pFeeder;	// return pointer
}

void CPermutation::poison()
{
	time *= 10;
}

int CPermutation::create_neigh_f(const CBoard *_pBoardArray, const int
_Board, const CPermutation _current, ofstream &_OutFile)
{
	int x = 0;
	int i, j, BestX = 0;
	double BestTime;

	for (i = 0; i < NumOfFeeders-1; i++)
	{
		for (j = i+1; j < NumOfFeeders; j++)
		{
			if (_current.pFeeder[i] != _current.pFeeder[j])
			{
				for (int m = 0; m <
_pBoardArray[_Board].NumOfLocations; m++)
				{
					neigh_f[x].pSeq[m] = _current.pSeq[m];
				}
				for (int z = 0; z < NumOfFeeders; z++)
				{
					neigh_f[x].pFeeder[z] =
_current.pFeeder[z];
				}
```

```
                        neigh_f[x].pFeeder[i] = _current.pFeeder[j];
                        neigh_f[x].pFeeder[j] = _current.pFeeder[i];
                        neigh_f[x].calculate_time(_pBoardArray,
_Board);
                        if (x == 0)
                        {
                              BestTime = neigh_f[x].time;
                        }
                        else
                        {
                              if (BestTime > neigh_f[x].time)
                              {
                                    BestTime = neigh_f[x].time;
                                    BestX = x;
                              }
                        }
                        x++;
                  }
            }
      }
      return BestX;
}

int CPermutation::create_neigh_s(const CBoard *_pBoardArray, const int
_Board, const CPermutation _current, ofstream &_OutFile)
{
      int x = 0;
      int i, j, BestX = 0;
      double BestTime;
      for (i = 0; i < (_pBoardArray[_Board].NumOfLocations)-1; i++)
      {
            for (j = i+1; j < _pBoardArray[_Board].NumOfLocations;
j++)
            {
                  for (int m = 0; m < NumOfFeeders; m++)
                  {
                        neigh_s[x].pFeeder[m] = _current.pFeeder[m];
                  }
                  for (int z = 0; z <
_pBoardArray[_Board].NumOfLocations; z++)
                  {
                        neigh_s[x].pSeq[z] = _current.pSeq[z];
                  }
                  neigh_s[x].pSeq[i] = _current.pSeq[j];
                  neigh_s[x].pSeq[j] = _current.pSeq[i];
                  neigh_s[x].calculate_time(_pBoardArray, _Board);
                  if (x == 0)
                  {
                        BestTime = neigh_s[x].time;
                  }
                  else
                  {
                        if (BestTime > neigh_s[x].time)
                        {
                              BestTime = neigh_s[x].time;
                              BestX = x;
                        }
                  }
                  x++;
            }
      }
      return BestX;
}

void CPermutation::calculate_time(const CBoard *_pBoardArray, const
int _Board)
{
      int h = 0, i, j;
      double t1 = 0,
            t2 = 0,
```

```
            t3 = 0,
            t4 = 0;
      int *pCompType = new int [_pBoardArray[_Board].NumOfLocations];
      int *pFeederNo = new int [_pBoardArray[_Board].NumOfLocations];
      pCoordinates = new CPositions
[_pBoardArray[_Board].NumOfLocations];
      for (i = 0; i < _pBoardArray[_Board].NumOfLocations; i++)
      {
            pCompType[i] = 0;
            pFeederNo[i] = 0;
      }
      for (i = 0; i < _pBoardArray[_Board].NumOfComps; i++)
      {
            pCoordinates[h].a = _pBoardArray[_Board].pComps[i].X_Co;
            pCoordinates[h].b = _pBoardArray[_Board].pComps[i].Y_Co;
            h++;
      }
      for (int k = 0; k < _pBoardArray[_Board].NumOfLocations; k++)
      {
            for (i = 0; i < _pBoardArray[_Board].NumOfLocations; i++)
            {
                  int Freq = _pBoardArray[_Board].pComps[i].CompFreq;
                  for (j = 0; j < Freq; j++)
                  {
                        if (_pBoardArray[_Board].pComps[i+j].X_Co ==
pCoordinates[pSeq[k]].a &&
                              _pBoardArray[_Board].pComps[i+j].Y_Co ==
pCoordinates[pSeq[k]].b)
                        {
                              pCompType[k] =
_pBoardArray[_Board].TypesOfComps[i];
                        }
                  }
            }
            for (i = 0; i < NumOfFeeders; i++)
            {
                  if (pFeeder[i] == pCompType[k])
                  {
                        pFeederNo[k] = i + 1;
                  }
            }
            t3 += PickTime + InsertTime +
sqrt(pow((pFeederNo[k]*LengOfFeeder - LengOfFeeder/2 -
pCoordinates[pSeq[k]].a), 2.0)
                  + pow((pCoordinates[pSeq[k]].b - Y),
2.0))/HeadSpeed;
      }
      for (k = 0; k < _pBoardArray[_Board].NumOfLocations-1; k++)
      {
            t2 += sqrt(pow((pFeederNo[k+1]*LengOfFeeder -
LengOfFeeder/2 -
                  pCoordinates[pSeq[k]].a),2.0) +
pow((pCoordinates[pSeq[k]].b - Y),2.0))/HeadSpeed;
      }
      t1 = sqrt(pow((X+pFeederNo[0] * LengOfFeeder - LengOfFeeder/2 -
X_Home),2.0) +
            pow((Y - Y_Home),2.0))/HeadSpeed;
      t4 =
sqrt(pow((pCoordinates[pSeq[_pBoardArray[_Board].NumOfLocations-1]].a
- X_Home),2.0) +

      pow((pCoordinates[pSeq[_pBoardArray[_Board].NumOfLocations-1]].b
- Y_Home),2.0))/HeadSpeed;
      time = t1 + t2 + t3 + t4;
      delete pCompType;
      delete pFeederNo;
      delete [] pCoordinates;
      return;
}
```

```
void CPermutation::generate_assign(const CBoard *_pBoardArray, const
int _Board)
{
	int i;
	CString Type;
	int k = 0;
	CPositions *pFeed = new CPositions [NumOfFeeders+1];
	for (i = 0; i < NumOfFeeders; i++)
	{
		pFeeder[i] = 0;
		pFeed[i].a = 0;
		pFeed[i].b = 0;
	}
	for (i = 0; i < _pBoardArray[_Board].NumOfComps; i++)
	{
		int Freq = _pBoardArray[_Board].pComps[i].CompFreq;
		int temp = _pBoardArray[_Board].pComps[i].X_Co;
		for (int j = 0; j < Freq; j++)
		{
			Type = _pBoardArray[_Board].pComps[i+j].CompType;
			if (_pBoardArray[_Board].pComps[i+j].X_Co > 0)
			{
				temp = (temp <
_pBoardArray[_Board].pComps[i+j].X_Co ?
				(abs(temp -
_pBoardArray[_Board].pComps[i+j].X_Co)/(j+1)) + temp :
				(abs(temp -
_pBoardArray[_Board].pComps[i+j].X_Co)*j/(j+1)) +
_pBoardArray[_Board].pComps[i+j].X_Co);
				if (Type !=
_pBoardArray[_Board].pComps[i+j+1].CompType)
				{
					float location = (temp -
X)/LengOfFeeder;
					pFeed[k].a =
_pBoardArray[_Board].pComps[i+j].CompType;
					pFeed[k].b = location;
					k++;
				}
			}
		}
		i = i + Freq - 1;
	}
	for (int j = _pBoardArray[_Board].NumOfCompTypes-1; j > 0; j--)
	{
		for (int cFeed = 0; cFeed < j; cFeed++)
		{
			if (pFeed[cFeed].b > 0)
			{
				if (pFeed[cFeed].b > pFeed[cFeed+1].b)
				{
					float TempB = pFeed[cFeed+1].b;
					pFeed[cFeed+1].b = pFeed[cFeed].b;
					pFeed[cFeed].b = TempB;
					int TempA = pFeed[cFeed+1].a;
					pFeed[cFeed+1].a = pFeed[cFeed].a;
					pFeed[cFeed].a = TempA;
				}
			}
		}
	}
	for (j = _pBoardArray[_Board].NumOfCompTypes-1; j > 0; j--)
	{
		for (int cFeed = 0; cFeed < j; cFeed++)
		{
			if (pFeed[cFeed].b < 0)
			{
				if (pFeed[cFeed].b < pFeed[cFeed+1].b)
				{
					float TempB = pFeed[cFeed+1].b;
```

```
                                    pFeed[cFeed+1].b = pFeed[cFeed].b;
                                    pFeed[cFeed].b = TempB;
                                    int TempA = pFeed[cFeed+1].a;
                                    pFeed[cFeed+1].a = pFeed[cFeed].a;
                                    pFeed[cFeed].a = TempA;
                              }
                        }
                  }
            }
            for (i = 0; i < _pBoardArray[_Board].NumOfCompTypes; i++)
            {
                  if (pFeed[i].b < 0)
                  {
                        if (pFeeder[0] == 0)
                        {
                              pFeeder[0] = pFeed[i].a;
                        }
                        else
                        {
                              for (int k = 1; k < NumOfFeeders; k++)
                              {
                                    while (pFeeder[k] == 0 && pFeeder[0] !=
0)
                                    {
                                          pFeeder[k] = pFeeder[k-1];
                                          pFeeder[k-1] = 0;
                                          k--;
                                    }
                              }
                              pFeeder[0] = pFeed[i].a;
                        }
                  }
                  else
                  {
                        if (pFeed[i].b > NumOfFeeders)
                        {
                              if (pFeeder[NumOfFeeders-1] == 0)
                              {
                                    pFeeder[NumOfFeeders-1] = pFeed[i].a;
                              }
                              else
                              {
                                    for (int k = NumOfFeeders-2; k > 0; k--)
                                    {
                                          while (pFeeder[k] == 0 &&
pFeeder[NumOfFeeders-1] != 0)
                                          {
                                                pFeeder[k] = pFeeder[k+1];
                                                pFeeder[k+1] = 0;
                                                k++;
                                          }
                                    }
                                    pFeeder[NumOfFeeders-1] = pFeed[i].a;
                              }
                        }
                        else
                        {
                              if (pFeeder[pFeed[i].b] == 0)
                              {
                                    pFeeder[pFeed[i].b] = pFeed[i].a;
                              }
                              else
                              {
                                    int *pEmptyFeeder = new int
[NumOfFeeders];
                                    for (int j = 0; j < NumOfFeeders; j++)
                                    {
                                          if (pFeeder[j] == 0)
                                          {
```

```
                                        pEmptyFeeder[j] =
abs(pFeed[i].b - j);
                                  }
                                  else
                                  {
                                        pEmptyFeeder[j] = 0;
                                  }
                            }
                            int temp = NumOfFeeders;
                            for (j = 0; j < NumOfFeeders; j++)
                            {
                                  if (pEmptyFeeder[j] > 0 &&
pEmptyFeeder[j] < temp)
                                  {
                                        temp = pEmptyFeeder[j];
                                        pFeed[i].b = j;
                                  }
                            }
                            pFeeder[pFeed[i].b] = pFeed[i].a;
                            delete pEmptyFeeder;
                            //delete (pEmptyFeeder);
                      }
                }
          }
      }
      delete [] pFeed;
      return;
}

void CPermutation::pick_random_f(const CBoard *_pBoardArray, const int
_Board)
{
      int i, item, feeder;
      int *pLimiter = new int [_pBoardArray[_Board].NumOfCompTypes];
      int *pFLimiter = new int [NumOfFeeders];
      for (i = 0; i < _pBoardArray[_Board].NumOfCompTypes; i++)
      {
            pLimiter[i] = 0;
      }
      for (i = 0; i < NumOfFeeders; i++)
      {
            pFLimiter[i] = 0;
            pFeeder[i] = 0;
      }
      for (i = 0; i < _pBoardArray[_Board].NumOfCompTypes; i++)
      {
            do
            {
                  item = rand() % _pBoardArray[_Board].NumOfCompTypes;
            } while (pLimiter[item] != 0);
            pLimiter[item]++;
            do
            {
                  feeder = rand() % NumOfFeeders;
            } while (pFLimiter[feeder] != 0);
            pFLimiter[feeder]++;
            pFeeder[feeder] = _pBoardArray[_Board].TypesOfComps[item];
      }
      delete pLimiter;
      delete pFLimiter;
      return;
}

void CPermutation::pick_random_s(const CBoard *_pBoardArray, const int
_Board)
{
      int i, item;
      int *pLimiter = new int [_pBoardArray[_Board].NumOfLocations];
      for (i = 0; i < _pBoardArray[_Board].NumOfLocations; i++)
      {
```

```
            pLimiter[i] = 0;
      }
      for (i = 0; i < _pBoardArray[_Board].NumOfLocations; i++)
      {
            pSeq[i] = 0;
      }
      for (i = 0; i < _pBoardArray[_Board].NumOfLocations; i++)
      {
            do
            {
                  item = rand() % _pBoardArray[_Board].NumOfLocations;
            } while (pLimiter[item] != 0);
            pLimiter[item]++;
            pSeq[i] = item;
      }
      delete pLimiter;
      return;
}

void CPermutation::initialise(const CBoard *_pBoardArray, const int
_Board)
{
      NumOfSeq = _pBoardArray[_Board].NumOfLocations;
      pSeq = new int [NumOfSeq];
      for (int i = 0; i < NumOfSeq; i++)
      {
            pSeq[i] = -1;
      }
      pFeeder = new int [NumOfFeeders];
      for (i = 0; i < NumOfFeeders; i++)
      {
            pFeeder[i] = 0;
      }
}

int CPermutation::find_best_s(const int start, const CBoard
*_pBoardArray, const int _Board)
{
      int Best = start;
      for (int i = 0; i < get_S_Neigh_Size(_pBoardArray, _Board); i++)
      {
            if (neigh_s[i].get_time() < neigh_s[Best].get_time())
            {
                  Best = i;
            }
      }
      return Best;
}

int CPermutation::find_best_f(const int start, const CBoard
*_pBoardArray, const int _Board)
{
      int Best = start;
      for (int i = 0; i < get_F_Neigh_Size(); i++)
      {
            if (neigh_f[i].get_time() < neigh_f[Best].get_time())
            {
                  Best = i;
            }
      }
      return Best;
}
```

StdAfx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
//    Pcb.pch will be the pre-compiled header
//    stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

TimeDlg.cpp

```
// TimeDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Pcb.h"
#include "TimeDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
//////
// CTimeDlg dialog


CTimeDlg::CTimeDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CTimeDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CTimeDlg)
		// NOTE: the ClassWizard will add member initialization
here
	//}}AFX_DATA_INIT
}


void CTimeDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CTimeDlg)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CTimeDlg, CDialog)
	//{{AFX_MSG_MAP(CTimeDlg)
	ON_EN_CHANGE(IDC_FEEDSETUPTIME, OnChangeFeedsetuptime)
	ON_EN_CHANGE(IDC_PICKTIME, OnChangePicktime)
	ON_EN_CHANGE(IDC_INSERTTIME, OnChangeInserttime)
	ON_EN_CHANGE(IDC_HEADSPEED, OnChangeHeadspeed)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

int CTimeDlg::GetItem(const int _ID)
{
	const TEXT_SIZE = 16;
	char szText[TEXT_SIZE + 1];   // buffer for conversions
	CEdit *pGet = (CEdit *)(GetDlgItem(_ID));
	pGet->GetWindowText(szText, TEXT_SIZE);
	return atoi(szText);
}

/////////////////////////////////////////////////////////////////////////////
//////
// CTimeDlg message handlers

void CTimeDlg::OnChangeFeedsetuptime()
{
	FeedSetupTime = GetItem(IDC_FEEDSETUPTIME);
}

void CTimeDlg::OnChangePicktime()
{
	PickTime = GetItem(IDC_PICKTIME);
}
```

```
void CTimeDlg::OnChangeInserttime()
{
      InsertTime = GetItem(IDC_INSERTTIME);
}

void CTimeDlg::OnChangeHeadspeed()
{
      HeadSpeed = GetItem(IDC_HEADSPEED);
}

int CTimeDlg::Get_FeedSetupTime()
{
      return FeedSetupTime;
}

int CTimeDlg::Get_PickTime()
{
      return PickTime;
}

int CTimeDlg::Get_InsertTime()
{
      return InsertTime;
}

int CTimeDlg::Get_HeadSpeed()
{
      return HeadSpeed;
}

void CTimeDlg::UpdateBox(const int _ID, const int _Data)
{
      const TEXT_SIZE = 16;
      char szText[TEXT_SIZE + 1];   // buffer for conversions
      CEdit *pDisplay = (CEdit *)(GetDlgItem(_ID));
      itoa(_Data, szText, 10);
      pDisplay->SetWindowText(szText);
}

BOOL CTimeDlg::OnInitDialog()
{
      UpdateBox(IDC_FEEDSETUPTIME, FeedSetupTime);
      UpdateBox(IDC_PICKTIME, PickTime);
      UpdateBox(IDC_INSERTTIME, InsertTime);
      UpdateBox(IDC_HEADSPEED, HeadSpeed);
      return true;
}

void CTimeDlg::set_para(const int _FeedSetupTime, const int _PickTime,
const int _InsertTime, const int _HeadSpeed)
{
      FeedSetupTime = _FeedSetupTime;
      PickTime = _PickTime;
      InsertTime = _InsertTime;
      HeadSpeed = _HeadSpeed;
}
```