2002

HYBRID FLOW STRATEGIES FOR HIGH VARIETY LOW VOLUME MANUFACTURING FACILITIES TO IMPLEMENT FLOW AND PULL

Mukund Narayanan
University of Kentucky, mnara0@uky.edu

ABSTRACT OF THE THESIS

HYBRID FLOW STRATEGIES FOR HIGH VARIETY LOW VOLUME MANUFACTURING FACILITIES TO IMPLEMENT FLOW AND PULL

Lean Manufacturing has proven to be a very successful strategy for achieving production efficiencies. The basic elements of lean manufacturing are flow and pull. The traditional methods for establishing flow and pull do not fit well in the realm of high variety low volume manufacturing systems. This thesis provides a general framework for establishing flow and pull in high variety low volume manufacturing systems, through the concept of hybrid flow layouts. The existing analytical procedure for forming hybrid flow layouts is described and a new heuristic procedure, that overcomes some of the limitations of the existing procedure, is proposed. The performance of the new procedure in comparison to the existing procedure is illustrated using a real world case study. Finally, certain practical implementation issues that affect the formation of hybrid flow layouts are provided.

Keywords: Group Technology, Hybrid Flow Layouts, Layout Modules

MUKUND NARAYANAN

November 21, 2002

HYBRID FLOW STRATEGIES FOR HIGH VARIETY LOW VOLUME MANUFACTURING FACILITIES TO IMPLEMENT FLOW AND PULL

By

Mukund Narayanan

Dr.Jon C.Yingling

Director of Thesis

Dr.Larry Holloway

Director of Graduate studies

November 21 2002

RULES FOR THE USE OF THESES

Unpublished theses submitted for the Master's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of part may be published only with the permission of the author, and with usual scholarly acknowledgements.

Extensive copying or publication of the thesis in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky

THESIS

Mukund Narayanan

The Graduate School

University of Kentucky

2002

HYBRID FLOW STRATEGIES FOR HIGH VARIETY LOW VOLUME MANUFACTURING FACILITIES TO IMPLEMENT FLOW AND PULL

THESIS

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Manufacturing Systems Engineering from the College of Engineering at the University of Kentucky

By

Mukund Narayanan

Lexington, Kentucky

Director: Dr.Jon C.Yingling, Professor of Mining Engineering

Lexington, Kentucky

2002

MASTER'S THESIS RELEASE

I authorize the University of Kentucky

Libraries to reproduce this thesis in

whole or in part for purpose of research.

Signed: Mukund Narayanan

Date: November 21st 2002

ACKNOWLEDGEMENTS

I am extremely grateful to my Thesis Director Dr. Jon. C. Yingling for introducing me to the exciting field of Lean Manufacturing. I am highly obliged to him for his expert guidance throughout the course of this research. I am very thankful to Dr.Janet Lumpp for setting up the project at the Electronic Manufacturing Plant. I thank Dr.Larry Holloway and Dr.Janet Lumpp for serving on my Thesis committee. I would also like to thank Mr.Greg Howard, Mr.Frank Grimard and Mr.Bryan Gillespie for their support during the course of the project at the Electronic Manufacturing Plant. I sincerely thank the Center for Robotics and Manufacturing Systems for providing financial support in the form of Research Assistantship. Thanks are also due to Mohan Swaminathan for his valuable suggestions and support.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS……iii

LIST OF TABLES……. vi

LIST OF FIGURES……... vii

LIST OF FILES……. ix

CHAPTER 1 INTRODUCTION

1.1 IMPORTANCE OF LEAN MANUFACTURING…… 1

1.2 FLOW AND PULL…… 3
1.2.1 FLOW……... 3
1.2.2 PULL…… 6

1.3 THE PROBLEM WITH HIGH VARIETY LOW VOLUME MANUFACTURING ENVIRONMENTS……... 8

1.4 PULL AND HYBRID FLOW…….. 10

1.5 SUMMARY…….. 15

CHAPTER 2 PROCEDURE FOR FORMING HYBRID FLOW LAYOUTS….. 17

2.1 EXISTING PROCEDURE…… 17

2.2 A DISCUSSION ON MERGER COEFFCIENT…….. 21

2.3 FORMATION OF MODULES USING EXISTING PROCEDURE… 24

2.4 SUMMARY……... 28

CHAPTER 3 NEW PROCEDURE FOR FORMING LAYOUT MODULES……. 29

3.1 LIMITATIONS OF EXISTING PROCEDURE……. 29

3.2 NEW PROCEDURE……33

3.3 APPLYING THE NEW PROCEDURE…….. 43

3.4 SUMMARY……..50

CHAPTER 4 COMPARISON OF THE TWO APPROACHES USING A REAL WORLD CASE STUDY……...52

4.1 ELECTRONIC MANUFACTURING CASE STUDY……52

4.2 APPLYING THE EXISTING PROCEDURE TO THE CASE STUDY……54

4.3 APPLYING THE NEW PROCEDURE TO THE CASE STUDY…….60

4.4 SUMMARY……64

CHAPTER 5 LIMITATIONS OF ALL HYBRID FLOWSHOP FORMATION PROCEDURES…….….65

5.1 LIMITATIONS OF THE NEW PROCEDURE……65

5.2 OVERCOMING THE LIMITATIONS OF THE NEW PROCEDURE USING U-SHAPED CELLS……...67

5.3 PRATICAL ISSUES GOVERNING THE FORMATION OF LAYOUT MODULES……..70

5.4 SUMMARY……..73

CHAPTER 6 CONCLUSIONS……..74

6.1 SUMMARY OF RESEARCH……..74

6.2 FUTURE DIRECTION……75

APPENDIX A……76
APPENDIX B……84
APPENDIX C……88
APPENDIX D……89
APPENDIX E……92
REFERENCES……..96
VITA…….98

LIST OF TABLES

TABLE 2.1: OPERATION SEQUENCE OF PRODUCTS [10]……..17

TABLE 2.2: COMMON SUB-STRINGS OBTAINED FROM OPERATION SEQUENCE [10]……..19

TABLE 2.3 MATRIX OF MERGER COEFFICIENTS FOR THE DATA IN TABLE 2.2 ……..25

TABLE 2.4: STRUCTURE OF THE MODULES FOR THE CASE STUDY [10] …….26

TABLE 2.5: OPERATION SEQUENCES EXPRESSED IN TERMS OF MODULES [10] ……...27

TABLE 3.1: STRUCTURE OF THE MODULES FOR THE CASE STUDY [10] …….29

TABLE 3.2: COUNT OF DUPLICATES RESULTING FROM THE EXISTING PROCEDURE ……..29

TABLE 3.3: COMMON SUB-STRINGS OBTAINED FROM OPERATION SEQUENCE [10] ……31

TABLE 3.4: EXHAUSTIVE LIST OF COMMON SUB-STRINGS ……33

TABLE 3.5: OPERATION SEQUENCE OF THE PRODUCTS [10] ……..44

TABLE 3.6: OPERATION SEQUENCES EXPRESSED IN TERMS OF MODULES ……..45

TABLE 3.7: COMPARISON OF THE NUMBER OF DUPLICATES USED BY THE TWO PROCEDURES ……50

TABLE 4.1: COMMON SUB-STRINGS OBTAINED USING THE EXISTING PROCEDURE ……...54

TABLE 4.2: CLUSTERS OF COMMON SUB-STRINGS…….56

TABLE 4.3 COUNT OF DUPLICATES REQUIRED BY THE EXISTING PROCEDURE……...59

TABLE 4.4 COUNT OF DUPLICATES AVAILABLE…….59

LIST OF FIGURES

FIGURE 1.1: GROUP TECHNOLOGY WORK CELLS …….5

FIGURE 1.2: HYBRID FLOW LAYOUT CONFIGURATION……9

FIGURE 1.3: A FRAMEWORK FOR ESTABLISHING PULL IN A HYBRID FLOW LAYOUT …….12

FIGURE 1.4: CONFIGURATION OF A CONWIP [8] ……14

FIGURE 2.1: DENDOGRAM OBTAINED AFTER CLUSTER ANALYSIS [10] ……...26

FIGURE 3.1: FROM / TO GRAPH SHOWING PRODUCT FLOW BETWEEN THE MODULES ……..46

FIGURE 3.2: CONFIGURATION OF MODULES OBTAINED FROM THE NEW PROCEDURE…….49

FIGURE 4.1: DENDOGRAM OBTAINED AFTER CLUSTER ANALYSIS OF THE MATRIX OF MERGER COEFFICIENTS ……55

FIGURE 4.2: CONFIGURATION OF MODULES OBTAINED USING THE NEW PROCEDURE ……63

FIGURE 5.1: STRUCTURE OF A LAYOUT MODULE FORMED USING THE MERGER COEFFICIENT ALGORITHM …….66

FIGURE 5.2: STRUCTURE OF A LAYOUT MODULE FORMED USING THE NEW PROCEDURE ……...67

FIGURE 5.3: STRUCTURE OF A LAYOUT MODULE FORMED USING THE NEW PROCEDURE, SHOWING POSSIBLE FLOWS BETWEEN THE PROCESSES ……..68

FIGURE 5.4: A U-SHAPED CELL DEPICTING THE FLEXIBILITY OF FLOW BETWEEN THE PROCESSES …….69

FIGURE 5.5: CONFIGURATION OF THE MODULES OBTAINED USING THE NEW PROCEDURE FOR THE ELECTRONIC MANUFACTURING CASE STUDY ……71

FIGURE 5.6: MODIFIED STRUCTURE OF THE MODULES FOR THE ELECTRONIC MANUFACTURING CASE STUDY, BASED ON SPAN OF CONTROL ……..72

LIST OF FILES

Name	Type	Size
Mukund_thesisrep	pdf	325 KB

CHAPTER 1 INTRODUCTION

1.1 Importance of Lean Manufacturing

Manufacturing is becoming extremely competitive by the day. Gone are the days of the sellers market when the manufacturers fixed the selling price of their products. In the present scenario, the equation governing the business is:

Profit = Selling price – Cost

The market determines the selling price. This is attributed to the competitive nature of the environment. It is thus clear that any increase in profits could only be achieved by reducing the costs involved in furnishing the products. This brings into question how cost reduction could be accomplished. The manufacturing of a product involves traversing the raw material through a set of well-defined stages, thus converting the raw material gradually into the final product. There are seven types of wastes [1] associated with each stage of manufacturing, namely:

1. Over production
2. Over processing
3. Waiting
4. Transportation
5. Inventory

6. Motion
7. Defects

Over-production relates to producing anything that is more than required in the immediate future. It results in extra inventory and requires more resources, ultimately increasing the costs. Over-production shows the existence of a poor correspondence between production activity and demand. ***Over-processing*** is another wasteful activity that adds more than value to the product, resulting in over quality. It is considered a waste since it requires additional resources and expertise for performing something beyond that which the customer is willing to pay. ***Waiting*** refers to the act of having an improper operator machine interface. When operators wait on machines, we lose time, which relates to money. Waiting also refers to delays experienced by the product as it flows through the system, e.g., batches in front of processes, resulting in increased lead times. The waste of ***Transportation*** is incurred when there are long distances to traverse between the processes, thus depicting poor flow in the system. These require more resources and increase the costs. ***Inventory*** is a major waste associated with manufacturing systems. Inventory not only requires investment, but also requires additional resources for maintaining it. Another waste associated with large inventory is that it is used as a buffer for countering the variability in the system, e.g. long setups, breakdowns, e.t.c). Industrial experience is that when inventory is used as such a buffer, the underlying problems are seldom resolved and the costs they incur are substantial and ongoing. The ***Motions*** that the operators exercise during their work cycles can result in wastes. The customer is just willing to pay for the value

added motions that result in the product. The motions also affect physical changes in the product. The non-value added motions do not contribute to the value of the product and need to be eliminated or reduced to the greatest possible extent. Last but not the least, we have the waste of ***Defects***. Defective products are not only a reflection of bad processes, but also increase the lead times and costs in the form of scrap and rework. Moreover, anything that makes it difficult to get at the root cause of defects (e.g., batch building) is also considered a waste.

Each of the above wastes forces a manufacturer to spend more money than required in the course of traversing a product from the raw material to its final form. Hence it is important to eliminate these wastes to the maximum possible extent to reduce the costs involved in the manufacturing of a product. The Lean philosophy aims at producing the product in the least costly fashion by eliminating the seven types of wastes.

1.2 Flow and Pull

Two major elements of the Lean philosophy are:

1. Flow
2. Pull

1.2.1 Flow

Traditionally the factories have been organized in terms of departments, where the departments consist of similar functions or processes. The products are then routed through the departments to complete their processing steps. It is not necessary for all

the products to flow through the same sequence of departments, as they could have different types of processing requirements. Hence the physical orientation of the departments on the floor would typically not support a unidirectional flow of the products. A product routed from department X to department Y could require some processes in X after the operations in Y are complete. So it has to be brought back to department X again. If the number of products is large, the amount of such backtracking also increases, resulting in increased transportation. Also, because of large distances, the products must move in batches between the departments. The lead-time of the product is thus the sum of the processing times of the product batch across the departments. This holds if the product batch is processed immediately after it is transported to a department, which is seldom true. The products wait in queues before the processes since the processes are kept busy most of the time. Hence there are unsynchronized arrivals, occurring in fits and starts, to the department. Because of these factors, the actual lead-time of the product is large. Lean advocates the implementation of Flow to overcome the above problems. Flow can be viewed as moving a product through processes that are co-located. When the processes are co-located, they do not have to be moved in batches but may be moved piece by piece between the processes. Now the lead-time of a product is closer to just the sum of the processing times of that product across the different processes. Transportation waste is also reduced to a large extent by co-locating all the processes that are required to produce a product. The co-located processes form a work-cell that is capable of completing all the operations that are required by a product. When there are products with similar processing requirements, all such products could be accommodated by a

Group Technology cell. Group Technology is a general strategy that seeks to gain manufacturing efficiencies through grouping products and processes based on commonalties or similarities. Products with similar processing requirements are classified as product families. Hence we would have a number of group technology cells that are dedicated to their respective product families. The structure of the cells is demonstrated in Figure 1.1. There are two Group technology work cells (Cell 1 and Cell 2) represented in Figure 1.1. These cells process the product families 1 and 2 respectively. As we could observe, the products flow in a unidirectional manner in each cell and the processes within each cell (represented by boxes) are also co-located to aid smooth flow of the products.

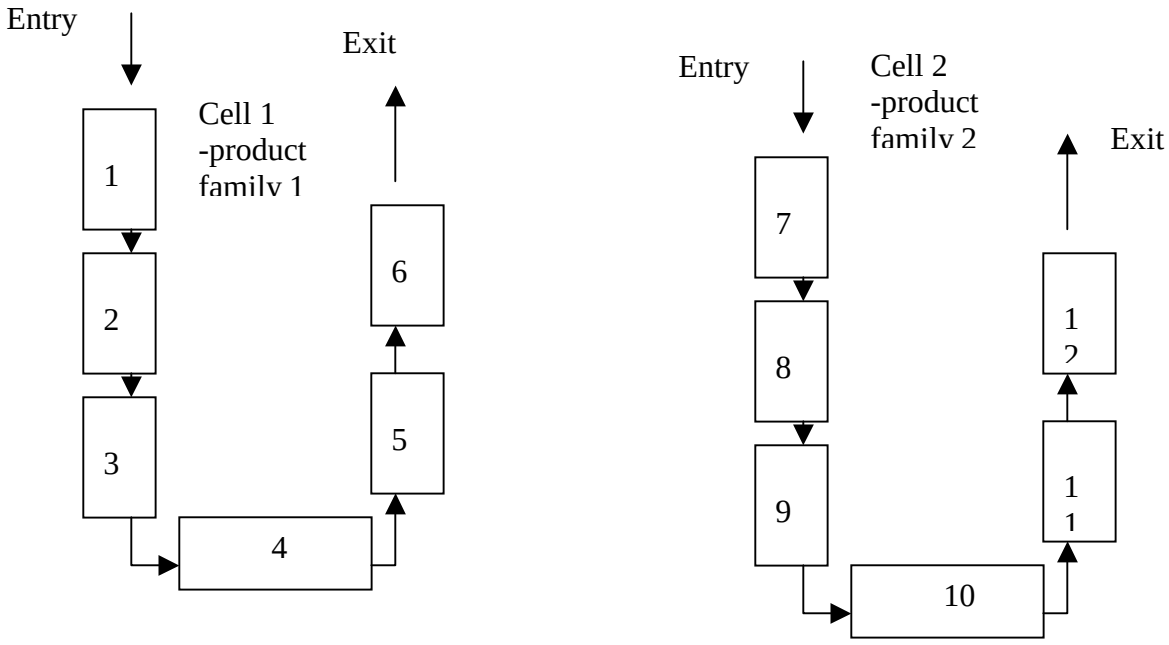


Figure 1.1: Group Technology work cells

All the necessary operations for product family 1 are finished within cell 1 and this holds for product family 2 also. Thus cells 1 and 2 are dedicated to product families 1 and 2 respectively. The above configuration helps is eliminating the wastes incurred in the form of waiting and transportation, thus reducing the lead-times of the products. The cell configuration also helps in improving the functionality of the processes as they are now dedicated to specific product groups and hence could specialize on those product groups alone. This helps in eliminating the defective products. Standardized work procedures [2] could be developed within the cells using the principles of motion economy [3], to eliminate motion wastes and over-processing wastes. The procedures for forming mutually exclusive cells like the one shown above are widely published in literature. ([4] to [7])**.** Thus it is clear that imparting a smooth flow could help in eliminating a number of wastes.

1.2.2 Pull

Once flow is achieved, the next step is to develop an efficient production control system. Lean advocates the use of pull systems. Traditionally push systems have been used to guide the flow of products in a facility. Push systems schedule work releases based on demand, while pull systems authorize work releases based on floor status [8]**.** In a push system, the jobs are released into the floor based on an MRP (Materials Requirement Planning) determined schedule. When a particular process breaks down, the production stops for sometime. But this does not stop the MRP system from releasing the products. Hence inventory builds up within the floor. Excess inventory is highly undesirable since it not only leads to higher costs, but also hides problems

inherent in the floor. Excess inventory is often used to prevent the starvation of the downstream processes when an upstream process fails. But this hinders the motivation to find the cause of the failure, which could lead to more failures. A pull system helps in capping the inventory within the system. Toyota uses kanban cards for implementing the pull system. Under the kanban system [2], each product type is associated with a unique kanban card. When a customer pulls a product from the finished goods inventory, the kanban card associated with that product is returned to the last process. The last process pulls a product from the output buffer of the penultimate process and starts processing it so as to replace the void created in the finished goods inventory. At the same time, the penultimate process pulls a product from the output buffer of its upstream process using a kanban card and starts processing to fill the void created by the last process. This type of signaling is transmitted in the reverse direction of material flow till reaching the first process. Note that is a process is disrupted (e.g., a breakdown), the flow of the pull authorization signal is blocked and the upstream process is not requested to replenish the product. Hence by making production activity responsive to floor status, pull systems can counteract variability with much lower WIP levels than push systems. A kanban system will process a product only if a customer needs that product. The number of kanban cards determine the amount of in-process inventory. A pull system like that of kanban helps in eliminating the wastes of over-production and inventory.

1.3 The problem with High Variety Low Volume Manufacturing Environments

The previous sections showed that flow and pull are essential for eliminating the seven types of wastes associated with manufacturing systems. Group technology work cells promote flow to a great extent especially if they have rapid setup and can manufacture products in small batches, perhaps even perform mixed model production where variants change piece to piece. If we go back and observe Figure 1.1, it is clear that the cells are dedicated to their respective product families. This type of dedication is easiest to justify when the product families have constant demand and are high volume. If a manufacturing facility is characterized by a high variety of products that are produced in low volumes and have fluctuating demands, having dedicated cells would prove to be a costly option. This is because, we would have a large number of product families. If these families have overlapping machine requirements between the cells, those machines will have to be duplicated. Moreover the product families do not have a constant demand and hence dedicating a configuration (group technology cell) to a product may result in expensive wasted capacity. Also, we could have a facility in which it is very difficult to define product families if the majority of the products use the majority of the processes (e.g. electronic manufacturing). These cases increase the difficulty in forming dedicated group technology cells. But still we are definitely interested in promoting flow as it is critical to any manufacturing facility. Hence we would have to come up with a different strategy to impart good flow in the case of high variety low volume manufacturing environments. Shukla J [9] has proposed a methodology for structuring high variety low volume facilities. Figure 1.2 represents a

Hybrid Cellular Layout consisting of three levels, A, B and C. All the products enter level A and they flow in the forward direction through levels B and C before exiting the system. Some products flow directly from level A to level C or from level B to level C, but still in the forward direction. Each level consists of a number of processes that are co-located to promote continuous flow of products between them. Within a level, a product need not use all the processes, but those processes that a product needs are arranged in a way so as to aid unidirectional flow.

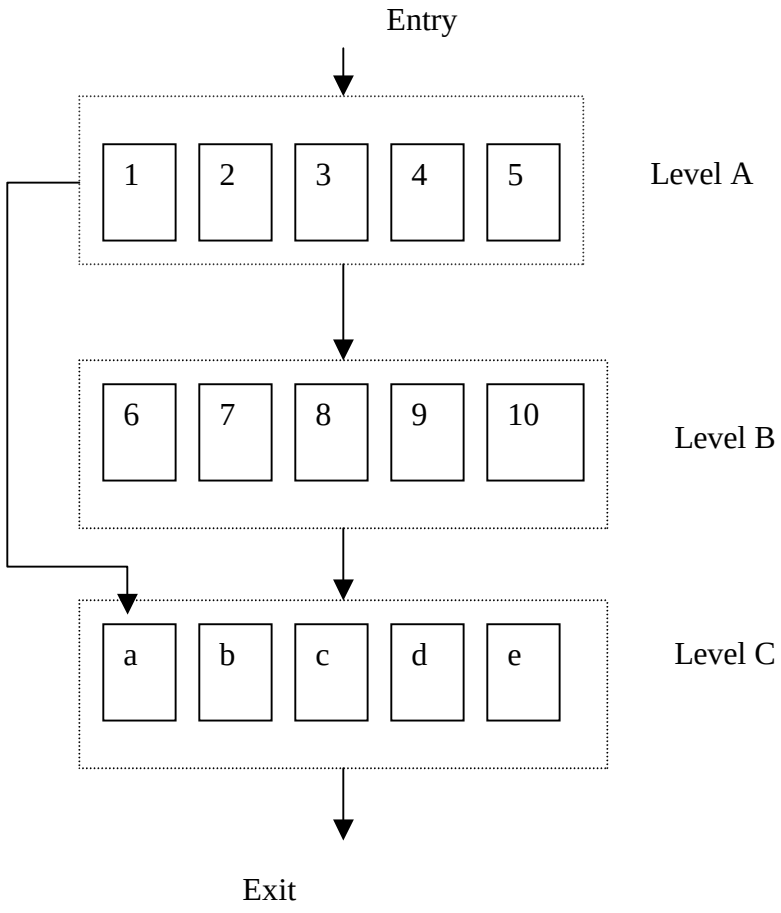


Figure 1.2: Hybrid Flow Layout Configuration

This type of layout is most suitable for high variety low volume manufacturing facilities since we are not dedicating any part of facility to a particular product variety. This reduces the amount of duplication required, but still the product variety is accommodated with a smooth flow within and between levels. A single level can also consist of two or more machine groups that support flow for different types of products. This could be used if machine duplicates are available. For example operation sequence of some products in level A could 1,2,3,4,5 while it could be 1,4,5 for another set of products. If we have duplicate machines for 1, 4 and 5 we could have another group within level A that will consist of machines 1,4 and 5. Groups of machines are referred to as modules [10]. Thus the flow of products will be between modules in the different levels. The procedures for forming such layouts will be discussed in the ensuing chapters. A hybrid cellular thus promotes good flow of products and is highly compatible to a high variety low volume manufacturing system. The next thing that we would be interested is the applicability of pull to the hybrid cellular layouts, which is discussed in the following section.

1.4 Pull and Hybrid flow

We saw previously how kanban cards are used to implement pull systems. Kanbans, as traditionally defined in the Toyota Production System (TPS) [2] are part specific. That is, each part type is associated with a unique kanban card. Due to this we are required to maintain some inventory of every part type at every process the part type visits. Only then, a signal will be transmitted from the end (when the customer pulls an end product) to all the processes to fill the void created by the customer. If we do not have

an inventory of the part type at some process, the signal transmitted will break at this process, as it cannot respond to the call given by its downstream process. In a high variety environment, the number of part types is large. If we were to implement a Toyota type kanban system, we would have to maintain some inventory of every part type, which is impractical since the part types have fluctuating demand. A product type is made up of a number of components. Usually, there is a degree of commonality between the different product types in terms of their component requirements. If we maintain an inventory of all the product types at the end, then the demand for the components that are part of a product type is simply the demand for that product type. If the demand for product types is highly variable, it introduces variability in the component demand as well [8]. The demand for the components can be aggregated across the product types so that the components can be pulled from a point where the product variety begins. Hence fluctuating demands increase the difficulty of implementing product specific kanbans. Route specific kanbans fit well in the hybrid flow structure. Figure 3 represents a hybrid flow structure, consisting of three levels with two modules in each level. A module consists of a group of machines that are co-located to aid smooth, unidirectional flow of the products that flow through that module. The products flow from level A through level C. Some products could use just two levels, but the flow is always unidirectional. Note that we have a buffer between the different levels. Single piece flow is maintained within the modules. Once the operations within a module are complete, the products are batched and sent to a buffer or a work backlog [8].

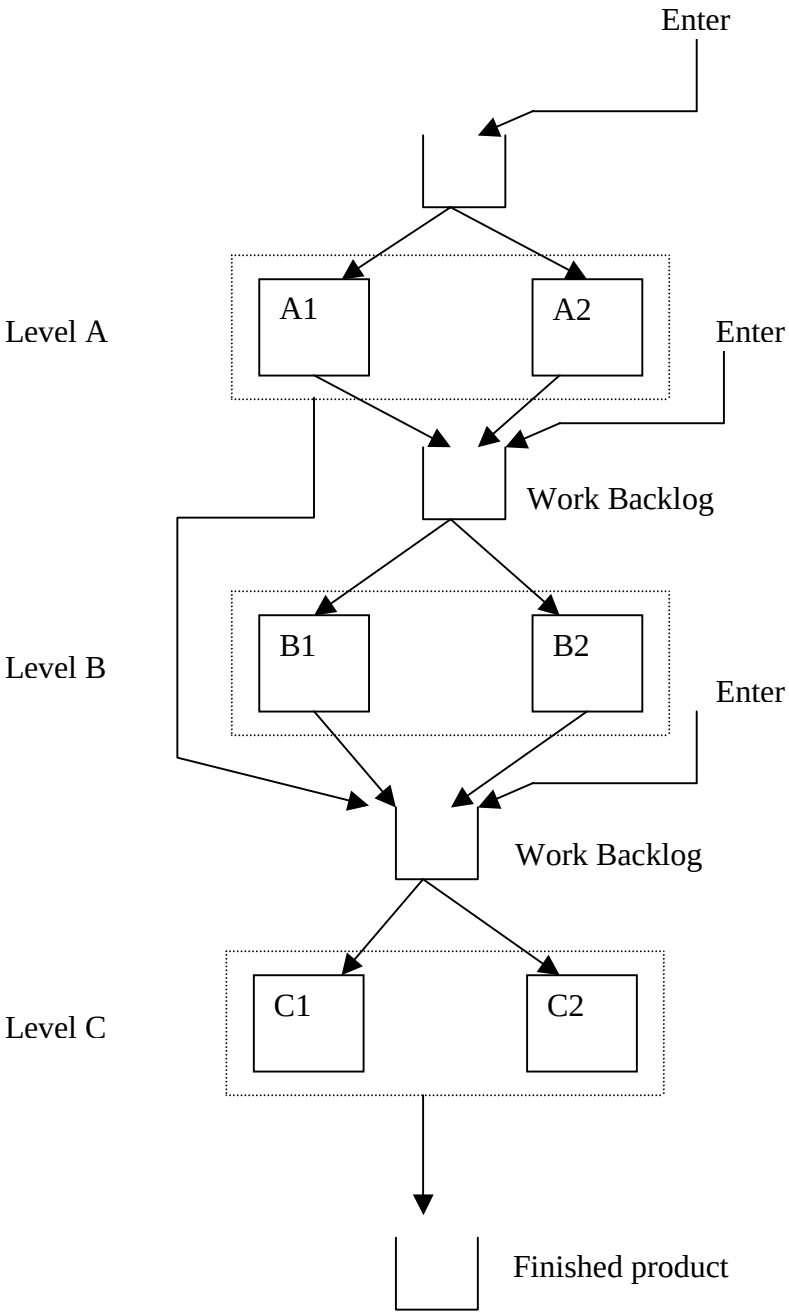


Figure 1.3: A framework for establishing pull in a Hybrid Flow Layout

Let us assume that a product has the operation sequence A1, B2, C1, where A1, B2 and C1 are the modules shown in Figure 1.3. In route specific kanban, as the name

specifies, we associate a kanban card with every possible route that products could take. In this case the route A1, B2, C1 would be associated with a card, say A1B2C1. When a product with the routing A1, B2, C1 is pulled out from the system (from the finished product buffer by the customer), the card associated with that product is returned to the beginning of A1, which will authorize the production of the same product or another product with the routing A1, B2, C1. Note that in this case the product that is launched at the beginning need not be the same as the one that authorizes production since the kanban cards here are route specific and not part specific. Hence we do not have to maintain an inventory of all the part types in the system as opposed to part specific kanban system. In this case we should have a work backlog at the start of every routing that will give the sequence in which the products are to be launched. This sequence is similar to a Master production schedule determined by an MRP system [8]. But the products would be launched only when authorized by the route specific kanban cards. This mechanism caps the in-process inventory within the system. The WIP cap promotes consistency in throughput times through the system, facilitating efforts to specify the build list so that end items are produced in time to meet their due dates. Thus route specific kanbans could be used to implement pull production in a hybrid flow layout. A potential problem of the route specific approach is that the number of routes can be large. This is especially true if the number of levels and modules within the levels is large.

Another type of pull system that would be suitable for the hybrid flow layouts is the CONWIP (Constant Work-in-Process) system [8]. A simple version of CONWIP is defined as follows:

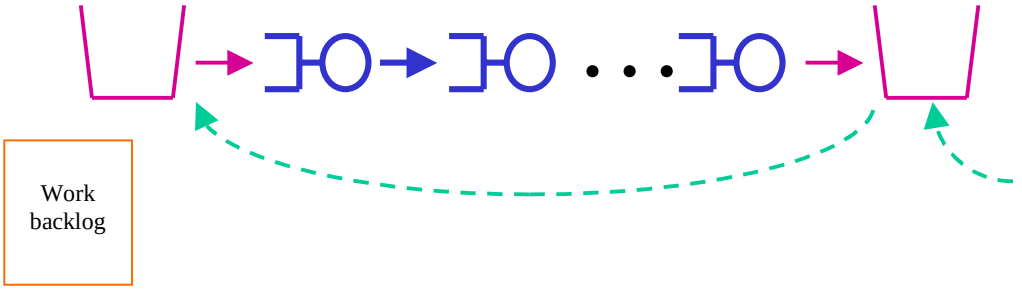


Figure 1.4: Configuration of a CONWIP [8]

The above workstations can be considered to process a number of products. A CONWIP loop spans the series of workstations as denoted by the dotted lines. In front of line, one could observe a work backlog. This work backlog contains the sequence in which the products are to be released into the line. The loop is associated with a specific number of CONWIP cards. Each job has to have a card to be released into the line for production. When a job exits the line, the card associated with that job is brought back to the front. This card is then attached to the next job from the work backlog and released into the line. So a constant number of cards keep circulating in the line, which limits the amount of WIP in the system. The WIP in the system is equal to the number of cards associated with the CONWIP loop. Also, the cards are line specific and not product specific as in the case of traditional kanban. The CONWIP system acts as a push system inside the line, but still the amount of WIP remains a constant as opposed to a traditional push system. Coming back to the hybrid flow structure we could consider each module within a level as a CONWIP loop.

Products hence flow between CONWIP loops. When a product type is running inside a module, it may be not use all the machines in the module. A few machines could be idle. The products flowing through a module have overlapping machine requirements. Hence it would not be possible to run multiple products in a module at the same time, if the products are run in batches through the module. The modules support continuous flow, but the flow between the modules will be in terms of batches. In this case we would associate a CONWIP card with each batch. The work backlog would be used only at the beginning of the first loop. After that the jobs will be pulled automatically by the downstream loops. We have two options [8] for releasing authorization cards once a product batch is completed within a module. (a) The batch could be transported with the card to the beginning of its downstream module. When the card for authorizing the batch in the downstream module is available, the card associated with the upstream module could be returned to the beginning of the upstream module to authorize production. (b) As soon as a batch is completed in a module, the card is returned to the beginning of the module to authorize production. At the same time, the completed batch is also transported to its downstream module. Option (a) is used if the processes in the downstream module are slower than the processes in the upstream module in general. This will prevent WIP explosion in front of the downstream module. Hybrid flow layouts thus support the CONWIP system well.

1.5 Summary

The above discussions clearly show that the Hybrid Flow Layouts promote both flow and pull in the case of high variety low volume manufacturing facilities. This research

focuses on the determination of the module configurations in a hybrid flow layout. The procedure available in literature is discussed in detail in the next chapter. Chapter 3 highlights some limitations of the existing procedure and proposes a new and relatively simple heuristic procedure for forming the modules. The heuristic is tested using the example given in the literature. Chapter 4 describes a case study undertaken at an Electronic Manufacturing facility. The applicability of the heuristic to the case study is outlined in the chapter. The proceeding chapter highlights the limitations of new heuristic and provides some countermeasures for overcoming those limitations. Finally conclusions are drawn based on the work done.

Chapter 2 Procedure for Forming Hybrid Flow Layouts

2.1 Existing procedure

Irani et.al [10] have proposed a heuristic methodology for establishing Hybrid Flow Layouts represented in the form of networks of Layout modules. The input requirement for performing the procedure is the operation sequence for all the products in the facility. Let us consider the following example:

Table 2.1 gives the operation sequence of a group of products.

Product #	Operation sequence
1	1,2,3,4,5,6,7,8,9,10
2	1,2,3,11,4,8,10
3	12,2,13,3,2,9,10
4	12,2,6,3,10
5	12,6,2,3,2,4,10
6	1,2,8,9,2,4,10
7	2,3,5,4,6,7,6,7,10
8	2,3,5,4,6,10
9	1,2,14,4,5,6,9,10
10	1,2,3,4,5,6,7,10
11	12,2,3,9,10
12	1,2,13,3,6,5,9,10
13	1,2,3,5,4,8,6,8,10
14	12,2,3,5,6,2,10
15	1,2,3,4,5,8,6,5,7,10
16	1,2,3,4,5,8,6,5,7,10
17	12,2,3,10
18	1,2,3,5,6,10
19	12,2,3,5,6,9,10
20	12,2,3,8,10
21	1,2,3,4,5,6,7,5,10
22	1,2,5,6,4,9,10
23	12,2,10
24	12,2,3,10
25	12,2,3,5,4,6,9,10

Table 2.1: Operation sequence of the products [10]

Table 2.1 shows that there are 25 different product types and 14 different machine types (named from 1 to 14). An operation sequence defines the order in which a product visits the different machine types in a facility. For example the operation sequence of product number 23 as observed from Table 2.1 is 12,2,10. It means that product number 23 visits the machines 12, 2 and 10 in the same order before exiting the facility.

The procedure for forming layout modules requires that we understand the idea of common sub-strings [10] between operation sequences. For example let us consider the operation sequences of products 1 and 18 found in Table 2.1, namely 1,2,3,4,5,6,7,8,9,10 and 1,2,3,5,6,10 respectively. We could easily observe that the two products have some commonality in their operation sequences. That is, both the products visit the machines 1,2, 3 and the machines 5 and 6 in the same order. A common sub-string is one that consists of a set of machines that are visited in the same order by *more than one* product type. Products 1 and 18 visit machine sets 1,2,3 and 5,6 in the same order as they have been listed in their operation sequences. Hence we designate the machine sets 1,2,3 and 5,6 as common sub-strings. The common sub-strings could be used to group machines that can handle multiple products. Table 2.2 gives the complete list of the common sub-strings for the example shown in Table 2.1. Irani et.al [10] have given the algorithm for identifying the common sub-strings. Also note that if a common sub-string is contained in another common sub-string, the smaller common sub-string will not be included in the list of common sub-strings (Table 2.2. in this case). That is, 1, 2, 3 is a common sub-string, but this is contained

within the common sub-string 1, 2, 3, 4, 5, 6, 7. Hence 1, 2, 3 is not included in Table 2.2.

Number	Common Sub-string
S1	8,9
S2	1,2,3,4,5,6,7
S3	4,8
S4	8,10
S5	3,2
S6	2,13,3
S7	3,10
S8	2,4,10
S9	6,2
S10	2,3,5,4,6
S11	6,7,10
S12	6,10
S13	5,6,9,10
S14	6,5
S15	8,6
S16	1,2,3,5
S17	12,2,3,5,6
S18	2,10

Table 2.2 – Common Sub-strings obtained from Operation sequence [10]

There are 18 common sub-strings in the data given in Table 2.1. The labels S1 to S18 in Table 2.2 represent common sub-strings 1 through 18. If we have 18 modules represented by the common sub-strings in Table 2.2, all the products could be flowed smoothly between the modules. But this would require a significant amount of duplication as some of the machines are found in several of the common sub-strings. We must have a procedure to combine some of the common sub-strings to reduce machine duplication. Irani et.al define a similarity measure called merger coefficient that will enable us to identify and combine similar common sub-strings. *Levenshtein*

distance [11] is a common measure used for comparing two strings. This measure gives the number of substitutions, deletions and insertions necessary to make the two strings equivalent. The larger the Levenshtein distance, the more dissimilar the strings are. Note that it is a measure of similarity based on just the composition of the strings. That is, it does not give importance to the order in which the elements occur in the two strings[10]. But to combine common sub-strings we have to give importance to the order in which the elements (machines in our case) occur in the sub-strings. This is done to ensure that products flow in the forward direction within a module. Irani et.al [10], have developed a measure of similarity that accounts for this, called the *Merger Coefficient.* To calculate merger coefficient between two strings, Irani et.al [10], have defined two additional measures, namely *Merger distance* and *Interruption distance*. The merger distance for the absorption of string *x* into string *y,* denoted by *md(x, y)* has been defined as the smallest number of substitutions and insertions such that string *x* is contained in string *y,* preserving the sequence of the elements in string *x* . Keeping *md(x,y)* fixed, the smallest number of deletions required between two consecutive basic transformations, between two consecutive matching operations, and between two consecutive transformation and matching operation, is defined as the Interruption Distance for the absorption of *x* into *y*, denoted by *id(x, y).* (Irani et.al). The Merger Coefficient *mc(x, y)* between the strings *x* and *y* is then evaluated using the following set of equations.

$$mc(x,y) = mc(y,x) = \begin{cases} 1 - \dfrac{md(y,x) + \dfrac{id(y,x)}{N_x}}{N_y + 1} & if\ N_x > N_y \rightarrow Equation 1 \\ 1 - \dfrac{md(y,x) + \dfrac{id(y,x)}{N_x}}{N_y + 1} & if\ N_x < N_y \rightarrow Equation 2 \\ \max\left(1 - \dfrac{md(x,y) + \dfrac{id(x,y)}{N_y}}{N_x + 1},\ 1 - \dfrac{md(y,x) + \dfrac{id(y,x)}{N_x}}{N_y + 1} \right) & if\ N_x = N_y \rightarrow Equation 3 \end{cases}$$

where N_x and N_y are the lengths of the strings x and y respectively. As can be seen the Merger distance gets more weight than the Interruption Distance when evaluating the Merger Coefficient.

2.2 A discussion on Merger Coefficient

The detailed procedure for determining the Merger Coefficient has not been outlined by Irani et.al [10]. The definition of Merger distance is clear, but the one given for Interruption distance in the literature does not appear to clearly portray the intended procedure. When we calculate the merger distance, we have to make the minimum number of substitutions or insertions to one of the strings and convert that string to the second string. Since we are not allowed to make any deletions, we would be left with an intermediate string after we calculate the merger distance. For example let us consider the strings, (1, 2, 3) and (1, 4, 5, 3). Let *x* represent (1, 2, 3) and *y* represent (1, 4, 5, 3). The merger distance for absorbing *y* into *x* is defined as *md(x, y)*. We should make the minimum number of substitutions or insertions to string *y* and convert it to *x*. By saying that we should convert string *y* to string *x*, we mean that string *y* should have all the elements of string *x* in the same order as they occur in

string *x*. The only element that string *y* needs, for it to contain all the elements of string *x* is element 2. The following possibilities could be used:

1. Substitute element 4 with element 2:

$$(1, 4, 5, 3) \longrightarrow (1, \overset{2}{\cancel{4}}, 5\ 3)$$

2. Substitute element 5 with element 2:

$$(1, 4, 5, 3) \longrightarrow (1, 4, \overset{2}{\cancel{5}}\ 3)$$

3. Insert element 2 between the elements 1 and 4

$$(1, 4, 5, 3) \longrightarrow (1, \mathbf{2}, 4, 5, 3\mathbf{)}$$

4. Insert element 2 between elements 4 and 5

$$(1, 4, 5, 3) \longrightarrow (1, 4, \mathbf{2}, 5, 3\mathbf{)}$$

5. Insert element 2 between elements 5 and 3

$$(1, 4, 5, 3) \longrightarrow (1, 4, 5, \mathbf{2,}\ 3\mathbf{)}$$

Using one of the above five options we could convert string *y* to string *x* by performing the minimum number of substitutions or insertions. Hence the merger distance for the above pair of strings, *md(x, y)* is 1 since we have to make just one substitution or one insertion to convert string *y* to string *x*. The intermediate string(s) left after determining the Merger Distance are (1, 2, 5, 3), (1, 4, 2, 3), (1, 2, 4, 5, 3), (1, 4, 2, 5, 3) and (1, 4, 5, 2, 3). The next step is to determine the Interruption distance, which is the minimum number of deletions required to convert the intermediate string(s) to string *x*. This is carried out as follows:

Consider each of the intermediate strings that resulted in minimum Merger distance, which in this case includes all five intermediate strings.

Intermediate string 1 (1, 2, 5, 3):

If element 5 is deleted from (1, 2, 5, 3), it results in string *x*

(1, 2, ~~5~~, 3) ⟶ (1, 2, 3) = string *x*

Number of deletions made = 1 = Interruption distance *id(x, y)*

Intermediate string 2 (1, 4, 2, 3):

(1, ~~4~~, 2, 3) ⟶ (1, 2, 3) = string *x*

Number of deletions made = 1 = Interruption distance *id(x, y)*

Intermediate string 3 (1, 2, 4, 5, 3):

(1, 2, ~~4~~, ~~5~~, 3) ⟶ (1, 2, 3) = string *x*

Number of deletions made = 2 = Interruption distance *id(x, y)*

Intermediate string 4 (1, 4, 2, 5, 3):

(1, ~~4~~, 2, ~~5~~, 3) ⟶ (1, 2, 3) = string *x*

Number of deletions made = 2 = Interruption distance *id(x, y)*

Intermediate string 5 (1, 4, 5, 2, 3):

(1, ~~4~~, ~~5~~, 2, 3) ⟶ (1, 2, 3) = string *x*

Number of deletions made = 2 = Interruption distance *id(x, y)*

The minimum possible value of Interruption distance is 1.

So, *md(x, y)* = 1 and *id(x, y)* = 1.

N_x = Length of string *x* (1, 2, 3) = 3

N_y = Length of string *y* (1, 4, 5, 3) = 4

The value of Merger Coefficient is obtained by substituting the above values in *Equation 2.*

Merger Coefficient, $mc(x, y) = mc(y, x) = 0.6875$

Merger Coefficient between two strings is a symmetrical measure.

$$\boldsymbol{mc(x, y) = mc(y, x)}$$

But note that:

$$\boldsymbol{md(x, y) \neq md(y, x)}$$

$$\boldsymbol{id(x, y) \neq id(y, x)}$$

Hence, to avoid confusion it is easier to designate the longer string as string *y* and the shorter string as string *x*. In this way one could compute *md(x, y)* and *id(x, y)* and use Equation 2 or vice versa, instead of randomly assigning the strings to string *x* or string *y*. But when the string lengths are equal, all the parameters have to be calculated since *Equation 3* is used for equal string lengths. In that case, it does not matter if we are assigning the string labels *x* and *y* randomly. (See Appendix A for the C code for determining Merger Coefficients).

2.3 Formation of modules using existing procedure

The next step in the procedure for forming layout modules is to determine the Merger Coefficient between every pair of common sub-string listed in Table 2.2. The following matrix gives the Merger coefficients between the common sub-strings for the example described by Irani et.al. [10]

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18
S1	1	0.75	0.67	0.67	0.33	0.5	0.33	0.5	0.33	0.67	0.5	0.33	0.8	0.33	0.67	0.6	0.67	0.33
S2	0.75	1	0.88	0.75	0.88	0.88	0.88	0.83	0.88	0.85	0.88	0.88	0.75	0.88	0.88	0.97	0.85	0.88
S3	0.67	0.88	1	0.67	0.33	0.5	0.33	0.75	0.33	0.83	0.5	0.33	0.6	0.33	0.67	0.6	0.67	0.33
S4	0.67	0.75	0.67	1	0.33	0.5	0.67	0.75	0.33	0.67	0.75	0.67	0.8	0.33	0.67	0.6	0.67	0.67
S5	0.33	0.88	0.33	0.33	1	0.75	0.67	0.75	0.67	0.83	0.5	0.33	0.6	0.33	0.33	0.8	0.83	0.67
S6	0.5	0.88	0.5	0.5	0.75	1	0.75	0.5	0.75	0.83	0.25	0.5	0.4	0.5	0.5	0.8	0.83	0.75
S7	0.33	0.88	0.33	0.67	0.67	0.75	1	0.75	0.33	0.83	0.75	0.67	0.8	0.33	0.33	0.8	0.83	0.67
S8	0.5	0.83	0.75	0.75	0.75	0.5	0.75	1	0.75	0.72	0.5	0.75	0.6	0.5	0.5	0.6	0.67	0.88
S9	0.33	0.88	0.33	0.33	0.67	0.75	0.33	0.75	1	0.83	0.75	0.67	0.8	0.67	0.67	0.8	0.83	0.67
S10	0.67	0.85	0.83	0.67	0.83	0.83	0.83	0.72	0.83	1	0.67	0.83	0.63	0.83	0.83	0.83	0.83	0.83
S11	0.5	0.88	0.5	0.75	0.5	0.25	0.75	0.5	0.75	0.67	1	0.88	0.8	0.75	0.75	0.4	0.67	0.75
S12	0.33	0.88	0.33	0.67	0.33	0.5	0.67	0.75	0.67	0.83	0.88	1	0.9	0.67	0.67	0.6	0.83	0.67
S13	0.8	0.75	0.6	0.8	0.6	0.4	0.8	0.6	0.8	0.63	0.8	0.9	1	0.8	0.8	0.4	0.67	0.8
S14	0.33	0.88	0.33	0.33	0.33	0.5	0.33	0.5	0.67	0.83	0.75	0.67	0.8	1	0.67	0.8	0.83	0.33
S15	0.67	0.88	0.67	0.67	0.33	0.5	0.33	0.5	0.67	0.83	0.75	0.67	0.8	0.67	1	0.6	0.83	0.33
S16	0.6	0.97	0.6	0.6	0.8	0.8	0.8	0.6	0.8	0.83	0.4	0.6	0.4	0.8	0.6	1	0.83	0.8
S17	0.67	0.85	0.67	0.67	0.83	0.83	0.83	0.67	0.83	0.83	0.67	0.83	0.67	0.83	0.83	0.83	1	0.83
S18	0.33	0.88	0.33	0.67	0.67	0.75	0.67	0.88	0.67	0.83	0.75	0.67	0.8	0.33	0.33	0.8	0.83	1

Table 2.3: Matrix of Merger Coefficients for the data in Table 2.2

The above matrix of merger coefficients is then subjected to cluster analysis in order to obtain clusters of common sub-strings. Each common sub-string within each cluster represents a flowline layout. The members of the cluster are then merged so as to produce a acyclic digraph [10]. An acyclic digraph is one that exhibits no cycles or strongly connected components (i.e., back and forth flow) consisting of more than two nodes. This is a characteristic of the material flow networks in the Flowline, Branched Flowline and Patterned Flow modules. (refer [10] for detailed discussion). Figure 2.1 shows the dendogram obtained after Average Linkage Cluster Analysis [12]. A dendogram [12] is a graph that gives the values of the similarity measure, at which the clusters are formed. In this case the similarity measure used is the Merger Coefficient

(shown along the vertical axis). The horizontal axis shows the labels for the common sub-strings.

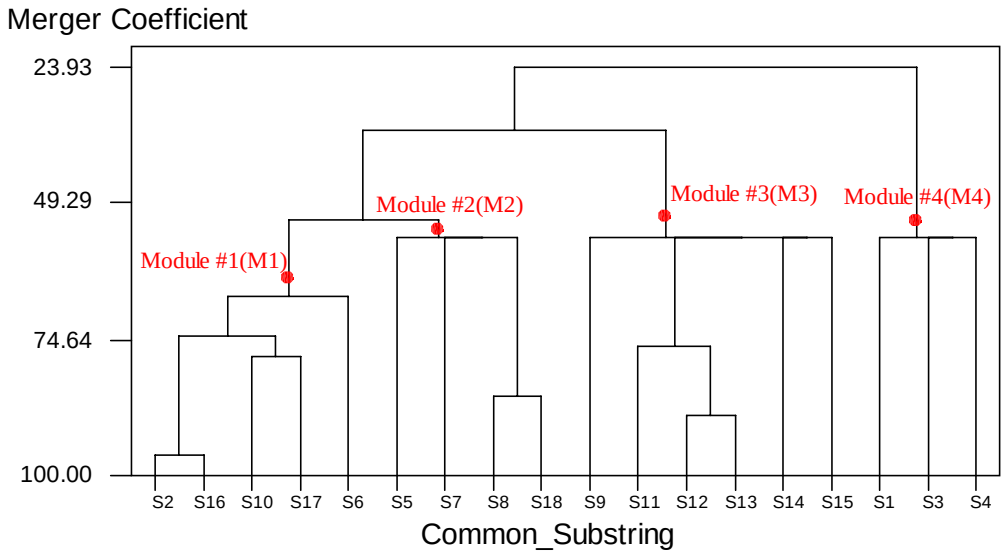


Figure 2.1: Dendogram obtained after cluster analysis [10]

Figure 2.1 shows 4 clusters that would be part of the four modules. Table 2.4 gives the layouts of the modules.

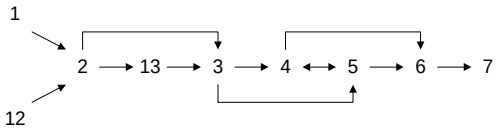
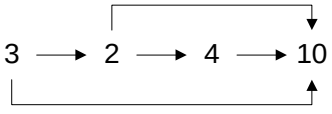
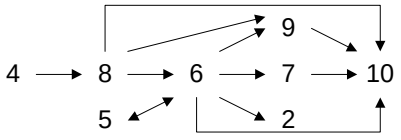
Module #	Cluster of Common_Sub-strings	Acyclic Digraph for the Layout Module
M1	S2, S16, S10, S17, S6	
M2	S5, S7, S8, S18	
M3	S9, S11, S12, S13, S14, S15 S1, S3, S4	

Table 2.4: Structure of the Modules for the case study [10]

The operation sequences of the products are then expressed in terms of the modules as shown in Table 2.5.

Product #	Operation sequence	Module Sequence
1	(1,2,3,4,5,6,7),(8,9,10)	M1, M3
2	(1,2,3),11,(4,8,10)	M1,11,M3
3	(12,2,13,3),2,(9,10)	M1,2,M3
4	(12,2),6,(3,10)	M1,6,M2
5	12,(6,2),(3,2,4,10)	12,M3,M2
6	(1,2),(8,9),(2,4,10)	M1,M3,M2
7	(2,3,5,4,6,7),(6,7,10)	M1,M3
8	(2,3,5,4,6),10	M1,10
9	(1,2),14,(4,5),(6,9,10)	M1,14,M1,M3
10	(1,2,3,4,5,6,7),10	M1,10
11	(12,2,3),(9,10)	M1,M3
12	(1,2,13,3),(6,5),(9,10)	M1,M3,M3
13	(1,2,3,5,4),(8,6),(8,10)	M1,M3,M3
14	(12,2,3,5,6),(2,10)	M1,M2
15	(1,2,3,4,5),(8,6,5),(7,10)	M1,M3,M3
16	(1,2,3,4,5),(8,6,5),(7,10)	M1,M3,M3
17	(12,2,3),10	M1,10
18	(1,2,3,5,6),10	M1,10
19	(12,2,3,5,6),(9,10)	M1,M3
20	(12,2,3),(8,10)	M1,M3
21	(1,2,3,4,5,6,7),5,10	M1,5,10
22	(1,2),(5,6),4,(9,10)	M1,M1,4,M3
23	(12,2),10	M1,10
24	(12,2,3),10	M1,10
25	(12,2,3,4,6),(9,10)	M1,M3

Table 2.5: Operation sequences expressed in terms of Modules [10]

We could observe that in this procedure certain machines do not belong to any of the modules. These have been classified as residual machines. The operation sequence of a product can thus be expressed in terms of modules and residual machines. Module visitations are permitted only if sub-strings with lengths of at least two would be used. If this cannot be accommodated then *residual machines* are used. For example

consider the operation sequence 12,2,3,10. We know that 12,2,3 is a sub-string of module 1 (M1). Machine 10 is found in module 2 (M2) and module 3 (M3). But it does not make sense to use any of these modules since we will be using only one machine. The other machines will be starved since we would be running only one product at a time in a module. So a residual machine 10 is brought into picture and hence the operation sequence is expressed as (12,2,3), 10 and hence M1,10.

2.4 Summary

The procedure described so far for forming layout modules is an effective method for forming hybrid flow layouts. The layout modules promote flow to a great extent. The modules themselves could be treated as CONWIP loops [8] to implement a pull system. But the procedure has some limitations. The next chapter provides some insight on the issues related to the above procedure. A simplified approach is then outlined for forming the layout modules.

Chapter 3 New Procedure for Forming Layout Modules

3.1 Limitations of existing procedure

Table 3.1 shows the structure of the modules for the example discussed in the previous chapter.

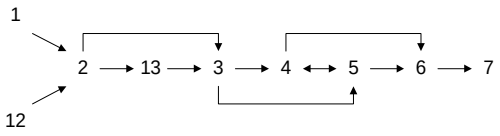
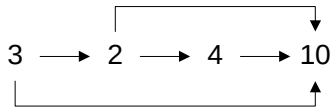
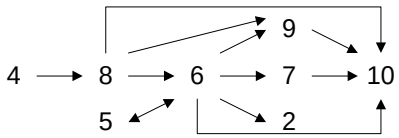
Module #	Cluster of Common_Sub-strings	Acyclic Digraph for the Layout Module
M1	S2, S16, S10, S17, S6	
M2	S5, S7, S8, S18	
M3	S9, S11, S12, S13, S14, S15 S1, S3, S4	

Table 3.1: Structure of the Modules for the case study [10]

One could easily observe from Table 3.1 that some of the processes are found in all the modules. Table 3.2 shows the number of duplicates used for each process in the above module configuration.

Process type	Number / type	Process type	Number / type
1	1	7	2
2	3	8	1
3	2	9	1
4	3	10	2
5	2	12	1
6	2	13	1

Table 3.2: Count of duplicates resulting from existing procedure

The procedure does not take as an input the number of duplicates that were available for each process type. Instead it comes up with a solution that suggests the number of duplicates that are necessary for each process type. It might not be feasible to duplicate certain processes due to the following reasons as well as others:

Cost: Certain processes are costly to duplicate. If the facility under consideration falls under the category of High variety Low volume, the demand for the products may not be constant. In that case, we might have to be flexible with respect to the layout configurations. That is we would have to change the layout based on the demand instead of having a constant configuration. It would prove costly if we start duplicating the processes to create flow everytime the layout is changed.

Nature of the process: There are certain processes, which cannot be brought in a continuous flow line since they are inherent batch processes. If we consider a Water jet machine or a Stamping press they produce products in batches since they have significant setup times and in cases such as Water jet, multiple parts are processed concurrently. Ideally we would want to reduce the setup time to smallest possible number. But until we achieve that, it is not feasible to include them on a continuous flow line. So such processes should be excluded from the operation sequence before starting the analysis. The procedure proposed by Irani et.al [10] does not give consideration to the nature of the processes. If the any of the processes in the example discussed above had been a batch process, then it does not make sense to duplicate that process due to the reasons cited above.

The rationale behind the procedure for forming Layout Modules is the existence of common sub-strings between the operation sequences of the products. The common sub-strings are subjected to cluster analysis using the merger coefficients. That is, similar common sub-strings are grouped together and combined as to form acyclic digraphs, which constitute a module. Table 3.3 gives the list of common sub-strings identified for the example discussed previously:

Number	Common Sub-string
S1	8,9
S2	1,2,3,4,5,6,7
S3	4,8
S4	8,10
S5	3,2
S6	2,13,3
S7	3,10
S8	2,4,10
S9	6,2
S10	2,3,5,4,6
S11	6,7,10
S12	6,10
S13	5,6,9,10
S14	6,5
S15	8,6
S16	1,2,3,5
S17	12,2,3,5,6
S18	2,10

Table 3.3: Common sub-strings obtained from operation sequence [10]

Table 3.3 suggests that the 18 common sub-strings are unique, i.e., no single common sub-string is completely contained in any other common sub-string, although there exists some degree of similarity between them. This degree of similarity is utilized for forming clusters of common sub-strings, wherein members within a cluster are more similar to each other than to the members in another cluster. But, there are some

individual processes that are common even between members of different clusters. For example, let us consider the common sub-strings S2 – 1, 2, 3, 4, 5, 6, 7 and S3 – 4, 8. They belong to separate clusters namely, Module 1 and Module 3 respectively. But still process 4 is common between these two common sub-strings. As a result process 4 is found in both Modules 1 and 3. We have other cases similar to the one discussed above, hence increasing the amount of process duplication. If we intend to reduce or eliminate process duplication, we should have a means to decide as to which module each process should be allocated instead of allocating the process to more than one module. To accomplish this we need to have a finer breakdown of the common sub-strings. The common sub-string S2 – 1, 2, 3, 4, 5, 6, 7 consists of finer sub-strings like (1, 2, 3), (2, 3), (4, 5), (4, 5, 6, 7), (5, 6) and (6, 7). These occur independently as well as with their parent sub-string, namely 1, 2, 3, 4, 5, 6, 7. The frequency of occurrence of these finer sub-strings can be found from the operation sequence of the products. For example the frequency of occurrence of (4, 5) is more than that of (4, 8), which suggests that process 4 fits better in Module 1 than Module 3. But we cannot simply remove a process from a module since it fits well in another module. This is because each module is an acyclic digraph and hence removing a process from that module could result in two modules. This would complicate the procedure. It is clear that the information obtained from the finer sub-strings would help in forming modules with minimal process duplication. So it would be better to form modules with no duplicate processes or minimum number of duplicate processes and then make additions to the module based on the availability of duplicate processes. The following section gives a simplified procedure for forming layout modules that accomplishes this.

3.2 New Procedure

The complete list of common sub-strings for the example discussed so far is given in Table 3.4.

Number	Substring	Frequency
1	2,3	18
2	1,2	12
3	12,2	10
4	1,2,3	8
5	9,10	8
6	5,6	8
7	2,3,5	7
8	3,5	7
9	12,2,3	7
10	4,5	6
11	6,7	5
12	1,2,3,4,5	5
13	2,3,4,5	5
14	3,4,5	5
15	4,5,6	4
16	2,3,5,4	4
17	3,5,4	4
18	5,4	4
19	7,10	4
20	1,2,3,4,5,6,7	3
21	2,3,4,5,6,7	3
22	3,4,5,6,7	3
23	4,5,6,7	3
24	5,6,7	3
25	8,10	3

Number	Substring	Frequency
26	3,10	3
27	2,3,5,4,6	3
28	3,5,4,6	3
29	5,4,6	3
30	4,6	3
31	6,9,10	3
32	6,5	3
33	8,6	3
34	2,3,5,6	3
35	3,5,6	3
36	12,2,3,5	3
37	8,9	2
38	4,8	2
39	3,2	2
40	2,13,3	2
41	13,3	2
42	2,4,10	2
43	6,2	2
44	6,7,10	2
45	6,10	2
46	5,6,9,10	2
47	1,2,3,5	2
48	12,2,3,5,6	2
49	2,10	2
50	1,2,3,4,5,8,6,5,7,10	2
51	12,2,3,10	2

Table 3.4: Exhaustive list of common sub-strings

Table 3.4 also gives the frequency of occurrence of each of the common sub-strings, arranged in descending order. The proposed procedure for forming layout modules consists of three Phases. Phase 1 forms module structures, assuming that there are no duplicate processes available. Phase 2 adds processes that were not added during

Phase 1. At the end of Phase 2, the modules are arranged so that there exists a unidirectional flow between the modules. If this is not possible, Phase 3 is used to where certain processes are duplicated, based on availability to prevent backtracking between the modules. Phase 4 uses the information about the availability of additional duplicate processes to enhance the structures of modules for better flow of products.

Phase 1:

Let, N = Total number of common sub-strings

i = 1, 2, …, N represent the common sub-strings

where, (Frequency (Sub-string 1) >= Frequency (Sub-string 2) >= …

Frequency (Sub-string i) >= …>= Frequency (Sub-string N)

m = Module number

M_m = Module name

For example Table 3.4 gives a list of common sub-strings with their frequency. In this case N = 51. Sub-string 1 refers to (2, 3), Sub-string 2 refers to (1, 2) and so on.

Phase1 involves the following steps:

Initialization:

Let i = 1

m =1

Assign Sub-string i to Module M_m

Step1: $i = i + 1$

Step 2: If $i == N + 1$, exit, else go to Step 3

Step 3: For ($k = 1$ to $k = i-1$)

If (Sub-string i has any element in common with Sub-string k)

Go to Step 1, Else go to Step 4

Step 4: If $k = = i - 1$, $m = m + 1$, Assign Sub-string i to module M_m, go to Step 1

The above steps are executed iteratively and at the end we would be left with a certain number of modules, each of which containing a sub-string. Phase 1 is illustrated using the data in Table 3.1 as follows:

Initially $i = 1$, $m = 1$. Assign Sub-string i, which in our case is (2, 3), to Module M_m or Module M_1. Increment m to 2.

Iteration 1:

$i = 2$.

Sub-string 2 is (1, 2), which has a common element , namely element 2 with Sub-string 1. So we do not add Sub-string 2 to any of the modules and we move to the next iteration.

Iteration 2:

$i = 3$.

Sub-string 3 is (12, 2), which has a common element with Sub-strings 1 and 2. (12, 2) is not added to any of the modules and we move to the next iteration (next sub-string)

Iteration 3:

i = 4.

Sub-string 4 is (1, 2. 3), which has common elements with all of its preceding sub-strings, namely, Sub-strings 1, 2, 3 and 4.

Iteration 4:

i = 5.

Sub-string 5 is (9, 10). This sub-string does not have any element in common with its preceding sub-strings. So (9, 10) is added to Module M_m or M_2 and *m* is incremented to 3. We then move to the next iteration.

Iteration 5:

i = 6

Sub-string 6 is (5, 6). It does not have any element in common with any of its preceding sub-strings, namely Sub-strings 1, 2, 3, 4 and 5. Hence Sub-string 6 is added to Module M_m or M_3 and *m* is incremented to 4. *i* is incremented to 7.

We similarly proceed with the rest of the iterations. That is, we locate every common sub-string in Table 3.4 and verify if that common sub-string has any element in common with any of its preceding sub-strings. If it does not have common elements, it

is added to a new module. If it has common elements even with one of its preceding sub-strings, it is not added to any of the modules. In the illustration made above, we do not encounter any sub-string, for $i > 7$, that does not have common elements with any of its preceding sub-strings. So at the end of Phase 1 we are left with three modules. The rationale behind this procedure is that we are identifying skeletal structures or the nuclei for the modules. The most frequently occurring common sub-string is added to a module at the start of Phase 1. Then every other sub-string is compared with all its preceding sub-strings (this includes the ones that were added to the module(s)) for the occurrence of common elements. The occurrence of common elements suggests that the sub-string has some affinity to the already formed module(s) and thus cannot be added to a new module. When the sub-string being examined does not have common elements with any of its preceding sub-strings, it means that the sub-string consists of elements that do not have a strong connection with the elements of the modules formed previously. But this sub-string is of high frequency since we have the sub-strings arranged in descending order of frequency. Hence we consider it to be a nuclei of a different module. The nuclei of the different modules at the end of Phase 1 represent most frequently occurring common sub-strings.

The modules formed using the above procedure would not contain all the processes that are involved in the operation sequence of the products. So the next part of Phase involves adding those processes to the modules. Each process that has not been added is evaluated using the list of common sub-strings to find the best module that would accommodate that process.

Phase 2:

Let, P = number of processes

j represent the process identification numbers / labels for the processes 1 to P

q represent the product identification number

O_q represent the operation sequence of the qth product

T represent the total number of products

m = total number of modules formed

M_k, k = 1 to m represent the Module names

N = Total number of common sub-strings

i = 1 to N represent the common sub-strings

where, (Frequency (Sub-string 1) >= Frequency (Sub-string 2) >= …

Frequency (Sub-string i) >= …>= Frequency (Sub-string N)

Let j = 1

Step 1: While (j <= P)

If (process j has not been added to any of the modules)

Go to Step 3

Else

$j = j + 1$;

Repeat Step 1

Step 2: If (j > P), Exit

Step 3: For i = 1 to N

If (process j is an element of Sub-string i)

For (k = 1 to m)

If (Sub-string i has common elements with M_k)

Add process j to M_k

$j = j + 1$

Go to Step 1

If (i = = N)

Go to Step 4

Step 4: For q = 1 to T

If (j belongs to O_q)

For k = 1 to m

If (Predecessor element of j in O_q and Successor

element of j in O_q belong to M_k)

Add Process j to M_k,

$j = j + 1$

Go to Step 1

Go to Step 5

Step 5: For k = 1 to m

$count_k = 0$

For q = 1 to T

If (j belongs to O_q)

For k = 1 to m

If (Predecessor element of j in O_q belongs to M_k)

$count_k = count_k + 1$

If (Successor element of j in O_q belongs to M_k)

$count_k = count_k + 1$

Find the value of k (k = 1 to m) for which $count_k$ is the largest (if two or more values are equal choose a k arbitrarily

Add Process j to M_k

$j = j + 1$

Go to Step 1

In this Phase, we select the process types that were not added during Phase 1. For each of those process types, we locate the most frequently occurring common sub-string containing that process type. The process type is added to the Module that has common elements with the located common sub-string. In this way, we ensure that the process types would be added only to the Modules, whose element(s) form a frequently occurring common sub-string with those process types. There are some process types that might not occur in any of the common sub-strings. Steps 4 and 5 of Phase 2 are used for adding those process types. The operation sequence of the products is used for adding the process types not belonging to any common sub-string. Each such process type has a predecessor and/or a successor element in the operation sequence (s) of the product (s) containing that process type. If both the predecessor and successor elements in any of the operation sequence belong to a particular module, the process type is added to that module. This is done to avoid backtracking

of the product defined by that operation sequence. If there is no such operation sequence, the frequency with which the process type occurs with the elements of the different modules is utilized to locate the module that would accommodate the process type in a better fashion. This ensures that the process type has more affinity to the module to which it has been added.

Let us consider the same data that was used to illustrate Phase 1. Processes 1, 4, 7, 11, 12, 13 and 14 were not added at the end of Phase 1. Let us see an example for the iteration that adds processes 1 and 11. These two processes are selected since they illustrate all the Steps of Phase 2.

Process 1:

First locate a common sub-string (from Table 3.4) that contains process 1. In our case it is Sub-string 2 or (1, 2). Next compare this sub-string with the Modules formed during Phase 1. That is, we find the Module that has elements in common with the located sub-string. Module 1 has the elements (2, 3). Hence process 1 is added to Module 1. (Steps 1 through 3 are used)

Process 14:

Process 14 does not occur in any of the common sub-strings listed in Table 3.4. So we go back to the operation sequences and locate the sequence that contains process 14. We find that process 14 occurs only in one operation sequence, namely 1,2,14,4,5,6,9,10. We find that the predecessor element and successor elements of

process 14, namely process 2 and process 4, belong to Modules 1 and 2 respectively. Step 4 does not add process 14 since the predecessor and successor elements belong to different modules. Step 5 is used in this case. So $count_1 = 1$ and $count_2 = 1$. Since both of them are equal we choose one of them arbitrarily, namely $count_1$ which refers to $k = 1$. Process 14 is hence added to Module 1 or M_1.

Phase 3:

In Phase 3, the operation sequences of the products are expressed in terms of the modules that were formed during Phase 2. That is, we convert the operation sequence to module sequence. The direction of product flow should be unidirectional between the modules. If we find any backtracking between the modules, the process / processes responsible for backtracking should be duplicated in one of the modules so as to eliminate backtracking. But this depends on the availability / feasibility of introducing duplicates.

Phase 4:

The layout modules formed so far use the minimum number of duplicates for the processes. Phase 4 uses the information about the availability of additional duplicate processes so as improve the flow. The modules formed are composed of the most commonly occurring sub-strings (forming the core part of the module) that are combined with some additional processes, forming a flowline or a branched flowline [10]. If we have duplicates for the processes represented in the core part of the module, those processes could be duplicated to form a parallel module. This increases

the capacity for running the products whose operation sequences consist of the most frequently occurring common sub-strings. Phase 3 and Phase 4 will become clear when we do the example case study.

3.3: Applying the New Procedure

Table 3.4 gives the list of common sub-strings for the example case study. The four Phases of the new procedure are applied to the example as follows:

Phase 1:

The iterative procedure is applied and at the end of the procedure, the following modules are formed:

Module M_1 $2 \rightarrow 3$

Module M_2 $9 \rightarrow 10$

Module M_3 $5 \rightarrow 6$

Phase 2:

At the end of Phase 1 we find that processes 1, 4, 7, 11, 12, 13 and 14 have not been added to the modules. The iterative procedure described in Phase 2 is applied.

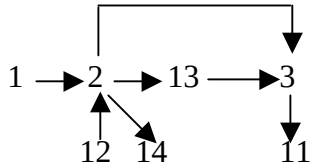
Processes 11 and 14 do not occur in any common sub-string. They are added using Step 5 of Phase 2. Table 3.5 gives the operation sequences for the products.

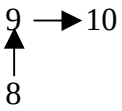
Product #	Operation sequence
1	1,2,3,4,5,6,7,8,9,10
2	1,2,3,11,4,8,10
3	12,2,13,3,2,9,10
4	12,2,6,3,10
5	12,6,2,3,2,4,10
6	1,2,8,9,2,4,10
7	2,3,5,4,6,7,6,7,10
8	2,3,5,4,6,10
9	1,2,14,4,5,6,9,10
10	1,2,3,4,5,6,7,10
11	12,2,3,9,10
12	1,2,13,3,6,5,9,10
13	1,2,3,5,4,8,6,8,10
14	12,2,3,5,6,2,10
15	1,2,3,4,5,8,6,5,7,10
16	1,2,3,4,5,8,6,5,7,10
17	12,2,3,10
18	1,2,3,5,6,10
19	12,2,3,5,6,9,10
20	12,2,3,8,10
21	1,2,3,4,5,6,7,5,10
22	1,2,5,6,4,9,10
23	12,2,10
24	12,2,3,10
25	12,2,3,5,4,6,9,10

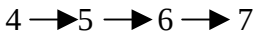
Table 3.5: Operation sequence of the products [10]

At the end of Phase 2, the module configuration is as follows:

Module M_1



Module M_2 

Module M_3 

Phase 3:

Table 3.6 shows the operation sequences expressed in terms of the modules.

Product #	Operation sequence	Module sequence
1	1,2,3,4,5,6,7,8,9,10	M1, M3, M2
2	1,2,3,11,4,8,10	M1. M3, M2
3	12,2,13,3,2,9,10	M1, M2
4	12,2,6,3,10	**M1,M3, M1, M2**
5	12,6,2,3,2,4,10	**M1, M3, M1, M3, M2**
6	1,2,8,9,2,4,10	**M1, M2, M1, M3, M2**
7	2,3,5,4,6,7,6,7,10	M1, M3, M2
8	2,3,5,4,6,10	M1, M3, M2
9	1,2,14,4,5,6,9,10	M1, M3, M2
10	1,2,3,4,5,6,7,10	M1, M3, M2
11	12,2,3,9,10	M1, M2
12	1,2,13,3,6,5,9,10	M1, M3, M2
13	1,2,3,5,4,8,6,8,10	**M1, M3, M2, M3, M2**
14	12,2,3,5,6,2,10	**M1, M3, M1, M2**
15	1,2,3,4,5,8,6,5,7,10	**M1, M3, M2, M3, M2**
16	1,2,3,4,5,8,6,5,7,10	**M1, M3, M2, M3, M2**
17	12,2,3,10	M1, M2
18	1,2,3,5,6,10	M1, M3, M2
19	12,2,3,5,6,9,10	M1, M3, M2
20	12,2,3,8,10	M1, M2
21	1,2,3,4,5,6,7,5,10	M1, M3, M2
22	1,2,5,6,4,9,10	M1, M3, M2
23	12,2,10	M1, M2
24	12,2,3,10	M1, M2
25	12,2,3,5,4,6,9,10	M1, M3, M2

Table 3.6: Operation sequences expressed in terms of modules

Figure 3.1 represents the *From/To* network diagram showing the flow of products between the modules. The numbers above the arrows indicate the frequency of flows between the modules. For example, if we observe the Module sequence in Table 3.6, it could be found that the frequency of occurrence of the sub-sequence (M_1, M_3) is 20 while the frequency of occurrence of (M_3, M_1) is 3, which is shown above the arrows in Figure 3.1. Hence it can be deduced that the majority of the products flow from Modules, M_1 to M_3 to M_2 or M_1 to M_2.

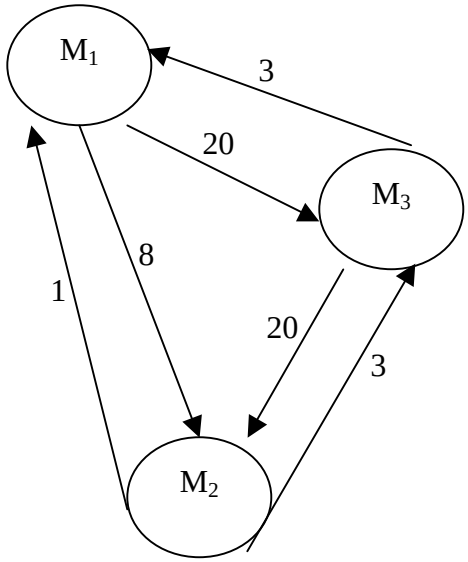


Figure 3.1: From / To graph showing the product flow between modules

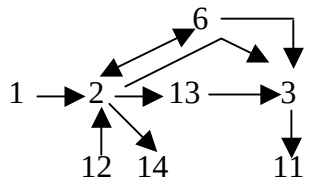
Certain Module sequences in Table 3.6 have been highlighted. These sequences represent backtracking between the modules. If we go through the operation sequences

we can find that the following need to be performed to eliminate backtracking completely:

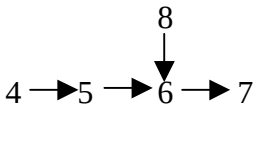
- Process 6 should be added in Module 1
- Processes 2 and 4 should be added to Module 2
- Process 8 should be added to Module 3

At this stage we must check if we have duplicate processes available for the above processes. If they do not exist, it is desirable to duplicate them so as to avoid backtracking. Assuming that we do have duplicate processes, the module structures are changed as follows:

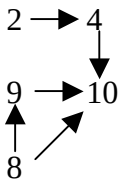
Module M_1



Module M_3



Module M_2



We could now observe that the flow of products is unidirectional.

Phase 4:

Each of the modules formed above consist of a frequently occurring common sub-string and a few other processes that have been added to that common sub-string. It is preferred that those frequently occurring common sub-strings are arranged in the form of a line layout without any process in between them. Module 1 has two frequently occurring common sub-strings namely, 1, 2, 3 and 12, 2, 3. The frequently occurring common sub-strings associated with Modules 2 and 3 are 9, 10 and 4, 5, 6, 7 respectively. The common sub-strings in Modules 2 and 3 have a line layout without any process interrupting the layout. But in Module 1, processes 6 and 13 are placed in between processes 2 and 3. If we have duplicate processes for processes 1, 2, 3 and 12, we could form a parallel module consisting of just those processes. In this way, those products using the just the sequence 12, 2, 3 or 1, 2, 3 in Module 1 could be routed to that parallel module. The other products would use the originally formed module. This helps in creating a smooth flow as well as increases the capacity of the system. The final configuration of the modules, provided we have the necessary duplicate processes, is as shown in Figure 3.2. The configuration shows three levels, A, B and C, with the flow of products being unidirectional between the levels. The number of duplicate processes generated, inside modules by the new procedure in comparison with the Merger Coefficient Algorithm is shown in Table 3.7.

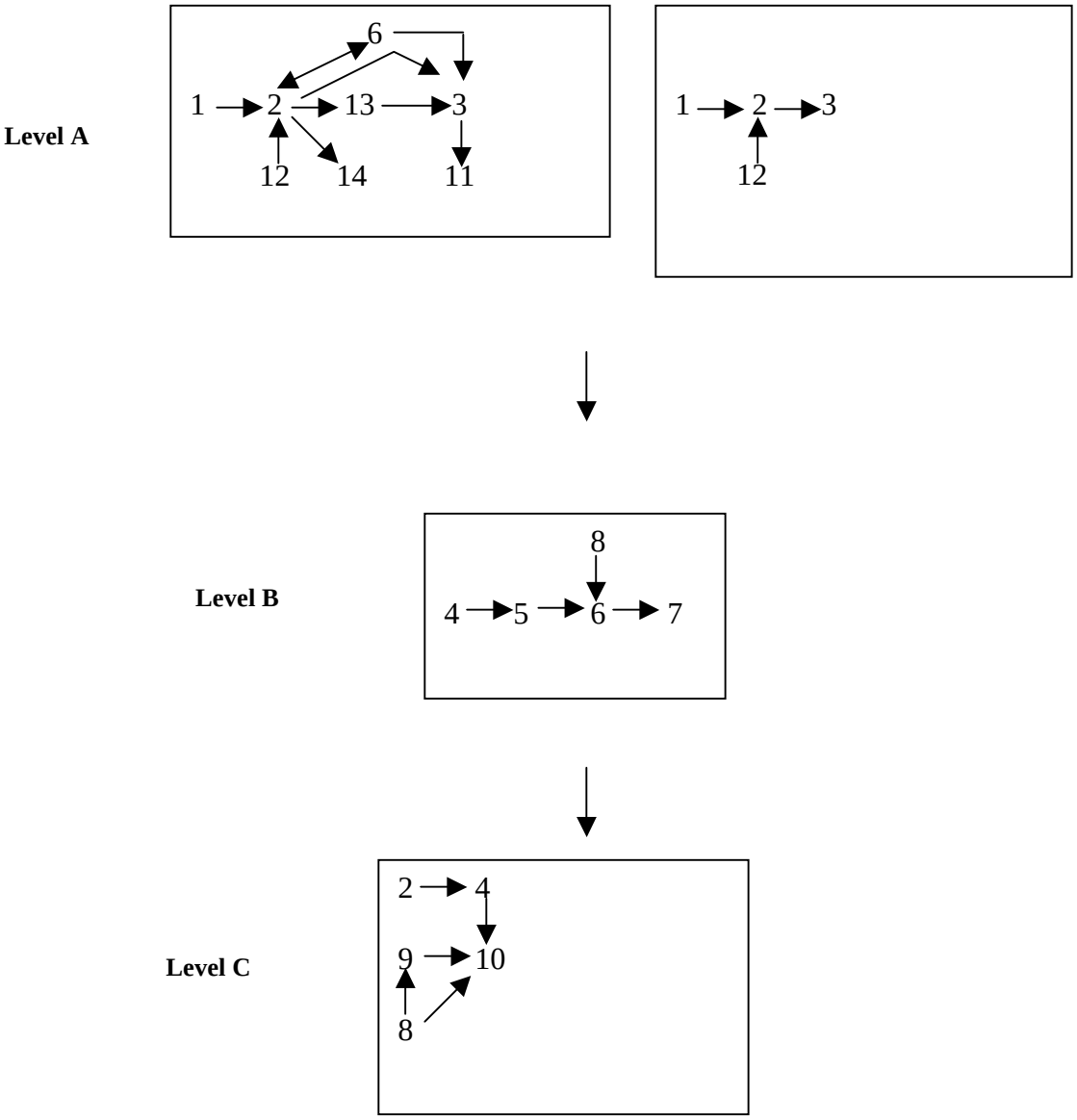


Figure 3.2: Configuration of the modules obtained from the New Procedure

Process Type	Number /type – Merger coeff. Algorithm.	Number/ type – New procedure, end of Phase 2	Number/ type – New procedure, end of Phase 3
1	1	1	1
2	3	1	2
3	2	1	1
4	3	1	2
5	2	1	1
6	2	1	2
7	2	1	1
8	1	1	2
9	1	1	1
10	2	1	1
11	0	1	1
12	1	1	1
13	1	1	1
14	0	1	1

Table 3.7: Comparison of number of duplicates used by the two procedures

3.4 Summary

As disucussed previously, the new procedure uses duplicate processes just to avoid backtracking. Table 3.7 shows that the number of duplicate processes used by the new procedure is lesser than that of the Merger Coefficient algorithm. Also, notice that process types 11 and 14 are not included in any of the modules that were generated using the Merger Coefficient algorithm. This is because, these process types do not occur in any of the common sub-strings. The new procedure accounts for this using Steps 4 to 5 of Phase 2 and hence makes sure that all the process types are represented in a module. The number of duplicates used after Phase 4 of the new procedure is not

used for comparison with the Merger Coefficient algorithm since Phase 4 is diagnostic in nature and depends on the availability of duplicates.

This chapter thus provided a new heuristic procedure for forming a Hybrid Flow Layout in the form of Layout Modules. It also highlighted the flexible aspects of the heuristic in comparison to the Merger Coefficient algorithm. The next chapter provides a real world case study, which is analyzed using the Merger Coefficient algorithm and the new heuristic.

Chapter 4 Comparison of the two approaches using a Real World Case Study

4.1 Electronic Manufacturing case study

This chapter describes the applicability of the procedures discussed so far, to a real world case study. The project was undertaken at an Electronic Manufacturing Plant. The plant manufactures Printed Circuit Board Assemblies. Both Surface Mount Boards and Through-hole Boards are assembled at the plant. It was determined that the majority of the boards had two levels of operations: pre-wave solder operations and post-wave solder operations. The wave-solder process is used for the Through-hole boards. The Surface Mount Boards flow directly from the Surface Mount processes to the post-wave solder processes. They do use the wave-solder process. The flow in the first level is similar for all the boards. They either flow through the surface mount machines and the manual placement (build/pre-build) area or the through hole component insertion machines and the build/pre-build area. Also, the order in which the boards visit the machines and manual workstations in the Surface Mount / Through Hole area was consistent. Hence the interesting part of the layout design will involve just the analysis of the post wave solder operations. The post-wave operation sequences of the products were obtained for analysis (See Appendix B for the operation sequence).

The key post-wave solder operations include the following:

- Clipping

- Breakout
- Post-wave inspection
- Hardware assembly
- In-circuit test
- Functional test
- Rework
- Final QC

Apart from these operations, there are a few other processes like Pot/Coat, which are required by a limited number of boards. These processes are stand-alone batch processes and cannot be included for production flow analysis if the objective is to establish single piece flow cells. The operation sequence listed in Appendix B does not include any of these processes. For example if the sequence is Clipping, Post wave inspection, Poat/Coat, Hardware assembly, Functional Test, Quality inspection, we would split the sequence at Pot/Coat and treat it as two separate sequences, denoted by Clipping, Post wave inspection and Hardware assembly, Functional Test, Quality inspection.

Firstly the algorithm developed by Irani et.al [10] is applied to the operation sequences. Then the new procedure described in the previous chapter is used for layout formation in order to overcome some of the limitations of the first procedure.

4.2 Applying the Existing Procedure to the Case Study

Table 4.1 gives the list of common sub-strings between the operation sequences. They are determined the way required by the first procedure.

Number	Common Sub-string	Number	Common Sub-string
1	5,4,6,4,9,8	**30**	1,2,3,4,7,9,8
2	1,3,5,6,9,8	**31**	2,3,5,7,9,8
3	1,5,3	**32**	1,2,3,5,7,6,9,8
4	5,3,6,9,8	**33**	1,2,3,9,8
5	4,5,4,6,9	**34**	1,2,3,7,9,8
6	1,3,9,4	**35**	3,5,6,4
7	1,3,9,8	**36**	2,4,6,9,8
8	3,8,9	**37**	3,2,4
9	1,3,4,5,6	**38**	2,5,3
10	1,3,5,9,8	**39**	1,3,5,2
11	1,3,5,4,7	**40**	1,2,3,5,4,6,9,8
12	3,5,4,7,9,8	**41**	2,4,3,9,8
13	5,4,9,8	**42**	5,9,4
14	4,8,9	**43**	2,1,3,4,9,8
15	1,3,4,6,9	**44**	1,2,3,4,5,6
16	3,4,6,9,8	**45**	1,3,2
17	3,4,6,9,4	**46**	3,2,5
18	1,3,6	**47**	3,1
19	3,4,5,6,9,8	**48**	1,2,6,9,8
20	1,3,5,4,6,9,8	**49**	7,8,9
21	3,5,8	**50**	2,3,7,4,6,9,8
22	5,3,9	**51**	1,2,3,5,4,7,6,9,8
23	7,6,4,9,8	**52**	2,3,4,5,7
24	1,2,3,6,9,8	**53**	5,1,2
25	6,8,9	**54**	2,4,9,8
26	1,3,5,7	**55**	5,6,9,4
27	1,3,4,7,9,8	**56**	2,5,6,9,8
28	1,2,3,4,6,9	**57**	2,5,9,8
29	1,2,3,5,6,9,8	**58**	9,8
		59	5,9,8
		60	2,3,6,9,4,9,8
		61	4,5,9,8
		62	5,1,6,9,8
		63	4,6,9,8

Table 4.1: Common Sub-strings obtained using the existing procedure

The next step is to determine the matrix of Merger Coefficients between the common sub-strings listed in Table 4.1. **Appendix C** shows the matrix of Merger Coefficients for the above data. The matrix is then subjected to Cluster Analysis (Average Linkage Clustering Method). Figure 4.1 shows the dendogram obtained after Cluster analysis. The common sub-strings within each cluster are then merged as to produce acyclic digraphs, which constitute the module structures.

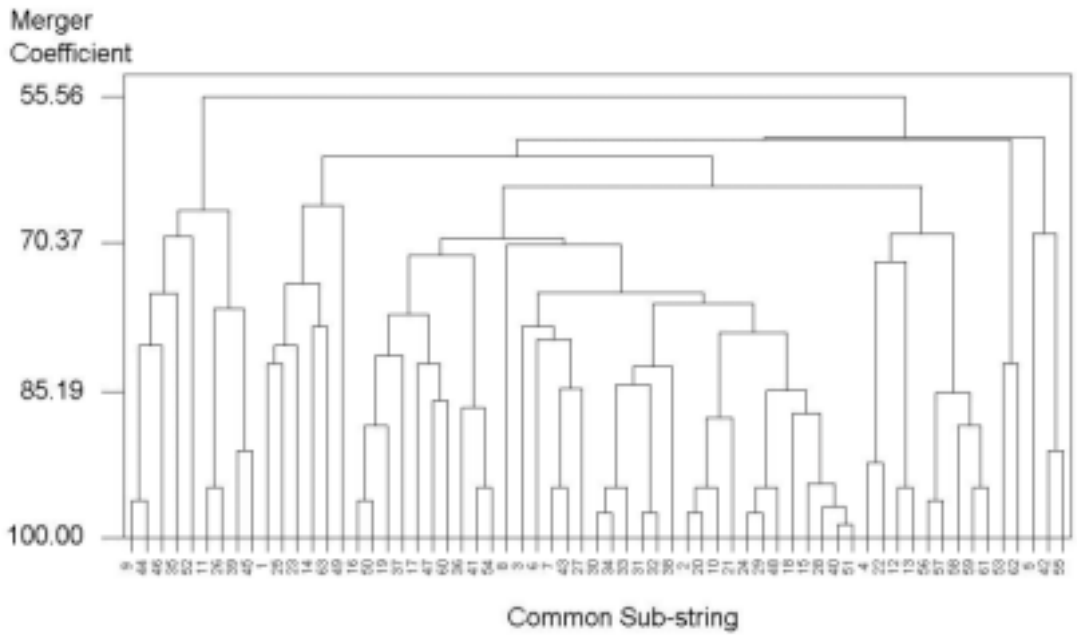


Figure 4.1: Dendogram obtained after cluster analysis of the matrix of Merger Coefficients

At a threshold level of 69% we have the following clusters:

Cluster	Common Sub-strings
1	1,25,23,14,63,49
2	9,44,46,35,52,11,26,39,45
3	16,50,19,37,17,47,60,36,17,47,60,36,41,54,8,3,6,7,43,27,30,34,33,31,32,38,2, 20,10,21,24,29,48,18,15,28,40,51
4	4,22,12,13,56,57,58,59,61
5	53,62
6	5,42,55

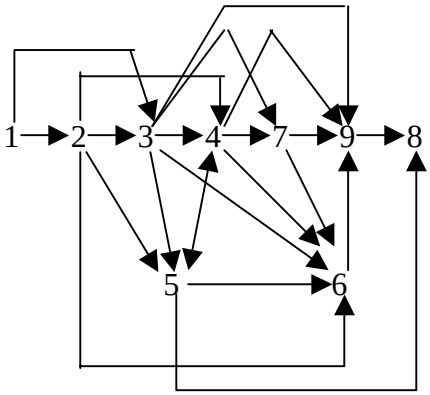
Table 4.2: Clusters of Common Sub-strings

If we go beyond a threshold level of 70 %, the number of clusters increases further. Also note that the common sub-strings within each cluster contain the majority of the processes. Hence an increase in the number of clusters will force an increase in the number of modules, resulting in more process duplication. As we will see later the modules formed using Table 4.2 are not feasible due to a lot of process duplication. The 6 clusters shown in Table 4.2, as observed from Figure 2 are used to form the module structures. The members of a cluster have to be merged to form a module. The module formed that way should be an acyclic digraph [10]. If some of the members cannot be merged, they are retained as separate flowlines. If this procedure is applied to the data shown in Table 1, there would be a lot flowlines [10] since every member

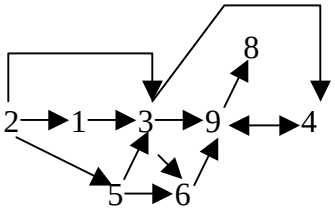
cannot be absorbed in the cluster to form an acyclic digraph. So some of the members are exchanged between clusters to aid the formation of acyclic digraphs. The results below give the structures of the modules along with the common sub-strings that constitute each module. We end up having 7 modules in order for each module to behave like an acyclic digraph.

Modules:

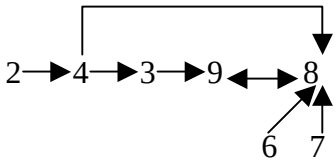
Module 1: (Common sub-strings 2, 5, 7, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21 24, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 40, 44, 51, 52, 54, 56, 57, 58, 59, 61, 63)



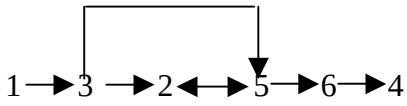
Module 2: (Common Sub-strings 6, 42, 55, 43, 38, 4, 22, 60)



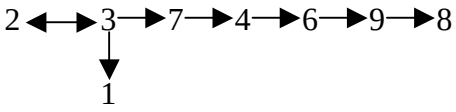
Module 3: (Common Sub-strings 25, 14, 49, 8, 41)



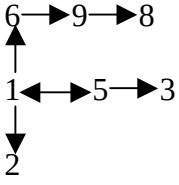
Module 4: (Common Sub-strings 46, 35, 39, 45)



Module 5: (Common Sub-strings: 50, 37, 47)



Module 6: (Common Sub-strings: 3, 53, 62)



Module 7: (Common Sub-strings 1, 23)

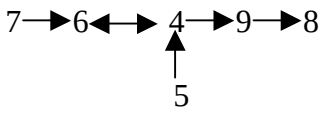


Table 4.3 gives the information about the number of duplicates used for each process type, as observed from the results of the Merger Coefficient Algorithm.

Process Type	Number / type
1	5
2	6
3	6
4	6
5	5
6	7
7	4
8	6
9	6

Table 4.3: Count of duplicates required by the existing procedure

Table 4.4 shows the actual number of duplicates available for process type at the plant under consideration.

Process Type	Number / type available
1	2
2	2
3	2
4	3
5	3
6	4
7	2
8	2
9	2

Table 4.4: Count of duplicates available

It is clear that the algorithm does not provide a reasonable solution for the case study. Let us apply the simplified procedure for forming layout modules discussed in the previous chapter to the data available from the case study.

4.4 Applying the New Procedure to the Case Study

Appendix D gives the exhaustive list of common sub-strings arranged in descending order of their frequency of occurrence in the post-wave operation sequences of the products. This list of common sub-strings is used in Phase 1 and Phase 2 of the new procedure for forming layout modules:

Phase 1:

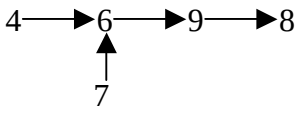
The iterative procedure is used as discussed in the previous chapter and it results in the following structure:

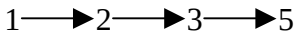
Module 1, M_1 $\quad 9 \longrightarrow 8$

Module 2, M_2 $\quad 2 \longrightarrow 3$

Phase 2:

The processes that were not added during Phase 1 are added during Phase 2 and the result is as follows:

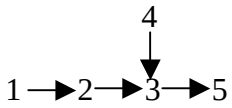
Module 1, M_1 

Module 2, M_2 

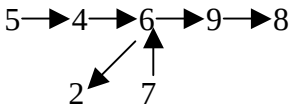
Phase 3:

In this Phase the operation sequences are expressed in terms of the Modules. Appendix 4 provides a Table showing the operation sequences expressed in terms of the modules. The flow of products is from Module 2 to Module 1. There are specific cases where the products would backtrack between the modules. These cases have been highlighted in the Table. The module configuration is changed as follows in order to avoid the backtracking.

Module 2, M_2



Module 2, M_1



As Table 4.4 indicates, we do have duplicates available for processes 2, 5 and 4 and hence the above changes made to the module structures as part of Phase 3 are feasible.

Phase 4:

Phase 4 uses the information available from Table 4 to make further modifications to the module structures so as to improve the flow. In Module 2, the most frequently occurring common sub-strings, as observed from the list of common sub-strings (refer Appendix D for the exhaustive list of common sub-strings) are (2, 3), (1, 3), (1, 2, 3) and (3, 5). The operation sequences (Appendix C) suggest that the first few operations are either characterized by the sub-string (1, 2, 3) or the sub-string (1, 3). So we could have two modules for first set of operations, represented by Level 1, one consisting the sub-string (1, 2, 3) and the other consisting of (1, 3). This will enable to split the products in either one of the cells depending on their operation sequences instead of sending them into one cell with the processes 1, 2 and 3. This is feasible since we do have duplicate processes for 1 and 3. Process 4 is maintained in one module and

process 5 is taken to the second module within Level 1. Looking at Module 1, we see that the most frequently occurring common sub-string is (4, 6, 9, 8). The list of common sub-strings shows that there are certain cases where products flow back and forth between processes 4 and 5, processes 5 and 6, and processes 6 and 7. Taking this into account and considering the availability of duplicates, the Module structures are changed as shown in Figure 4.2. Thus the available duplicate processes are utilized to the best possible extent to improve the module configuration. There is not a defined procedure for performing Phase 4. It depends on the type of the case study under consideration. Phase 1 and 2 give a skeletal structure for the Hybrid Flow Layout, while Phase 3 and 4 help in improving the skeletal structure to promote flow.

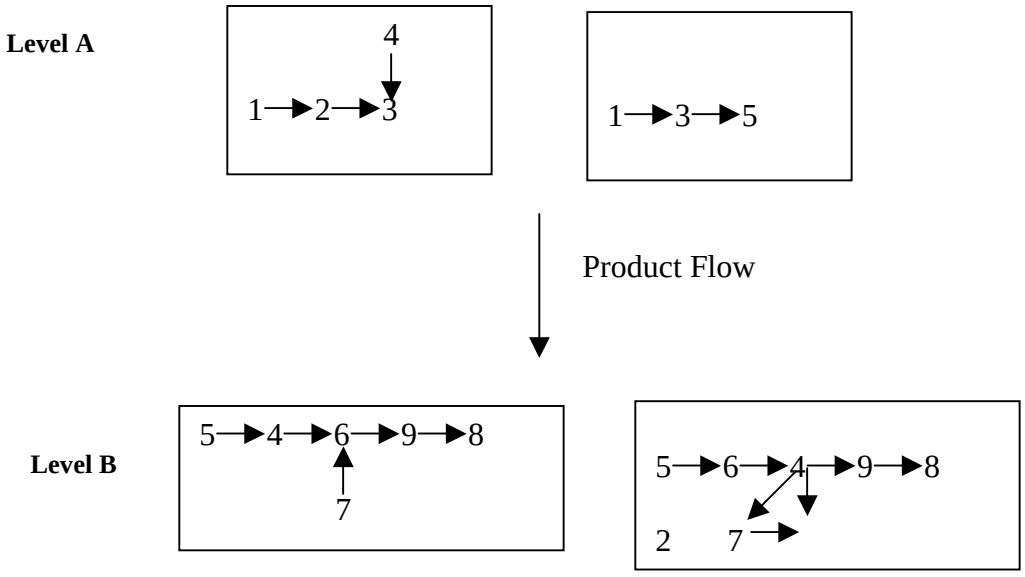


Figure 4.2: Configuration of the modules obtained using the new procedure

4.5 Summary

The new procedure was thus applied to both the literature case and the real world case study. The procedure is more flexible than the Merger Coefficient algorithm in the sense that it is adaptable to the problem studied, whereas the former has to be used as defined irrespective of the nature of the problem. There are some limitations to the existing procedure compared to the Merger Coefficient algorithm, but to a lesser degree. The limitations would be discussed in the next chapter, which also highlights some of the issues that affect generalized procedures for forming layouts.

Chapter 5 Limitations of all Hybrid Flowshop Formation procedures

5.1 Limitations of the new procedure

The previous chapters provided a framework for establishing flow and pull in High Variety Low Volume Manufacturing Facilities using Hybrid Flowlines. Flow is established primarily through the formation of Layout modules. As discussed in the first chapter, a variety of pull production control mechanisms like Route specific kanbans, and CONWIP could be applied to Hybrid Flow Layouts that are configured in the form of Layout modules. Two procedures for forming Layout Modules were presented, one of them excerpted from the literature and the other developed so as to overcome the problem of process duplication. This chapter discusses some of the limitations of the new procedure compared to the Merger Coefficient Algorithm. The chapter also shows how U-shaped cells could overcome these limitations to a good extent. Finally the chapter highlights some of the issues that need to be addressed before implementing the Layout modules formed using generalized procedures.

Figure 5.1 shows one of the Modules formed using the Merger coefficient algorithm for the real world case study. The module structure constitutes an acyclic digraph. The module represents all the possible flows between the processes as long as these flows are part of common sub-strings. For example, the above module shows the flows (2, 3), (2, 4) and (2, 5). These flows are represented in one or the other of the common sub-strings found between the operation sequences.

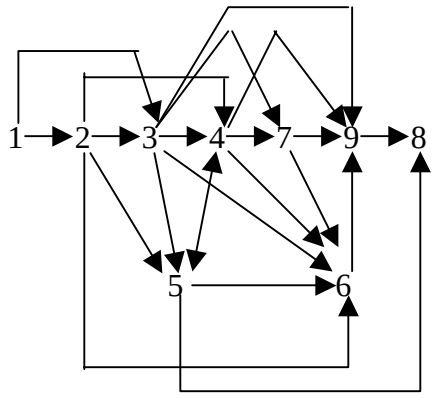


Figure 5.1: Structure of a layout module formed using the Merger Coefficient Algorithm

One of the main criteria of the Merger Coefficient algorithm is that the modules should constitute an acyclic digraph. The rationale for this assumption is that the material flow network in a flowline or a branched flowline module resembles an acyclic digraph [10]. It has been argued that the existence of cycles (backtracking) could be counter-measured by using a bi-directional material handling system that will move the products back and forth between the processes, or by having duplicate processes within the same module so as to avoid the backtrack flows [10]. The merger coefficient algorithm takes the second approach, namely process duplication. It does not employ process duplicates within a module. But instead, backtrack flows are accommodated in a different module ultimately resulting in process duplication across the modules. Also, backtrack flows are represented only if they are part of a common

sub-string. According to Irani et.al [10], if we are to avoid process duplication, bi-directional material handling systems need to be established. If the processes in a flowline or a branched flowline are arranged in a linear fashion, any backtracking of the products would increase the distance traveled by the products hence increasing the difficulty of setting a bi-directional material handling system. This is true is we are to employ material handling systems within the Module. If the operators could handle the products, it is highly undesirable to employ a material handling system, irrespective of whether it is uni-directional or bi-directional. Material handling systems not only require investment, but also result in increased costs through issues such as maintenance and downtimes. Moreover the adaptability of Material Handling Systems to changing conditions is also questionable. Operators are more flexible than the material handling systems. The next section shows how U-shaped cells with operators handling the products, help in overcoming some of problems related to flow within a module.

5.2 Overcoming the limitations of the new procedure using U-shaped cells

One of the modules formed using the new procedure for the real world case study is shown in Figure 5.2.

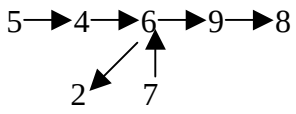


Figure 5.2: Structure of a layout module formed using the New Procedure

The above module also constitutes an acyclic digraph [10]. But it does not represent all the possible flows within that module. This is due to the fact that the process for forming the modules takes into account just the most frequently occurring common sub-strings as opposed to considering all the common sub-strings. This enables to avoid machine duplication. A minimum amount of machine duplication is of course required to avoid backtracking of the products between the modules (Phase 3). If we include all the flows within that module it would look like the one shown in Figure 5.3.

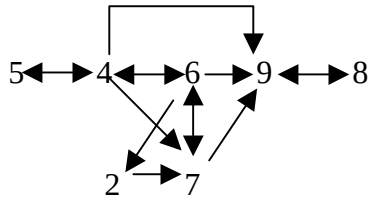


Figure 5.3: Structure of a layout module formed using the New Procedure, showing possible flows between the processes

The above module does not represent an acyclic digraph, since it has cycles as well as two or more strongly connected components. But the structure shown above is not representative of the flow pattern of all the products. A few products flow back and forth between the processes in a module and these few products make the module structure cluttered. The majority of the products flow in the forward direction. If we set up a U-shaped cell, the distances between every pair of processes is reduced to a

great extent and in that case all possible types of flows within the module can be accommodated. Figure 5.4 shows a U-shaped cell for the module shown above.

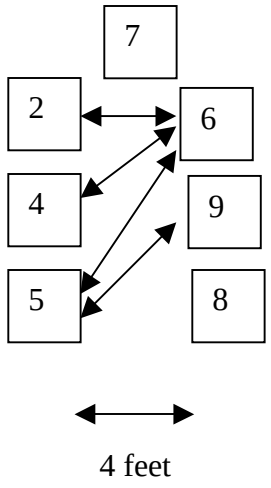


Figure 5.4: A U-shaped cell depicting the flexibility of flows between the processes

It is clear from Figure 5.4 that a U-shaped cell can handle flows between the every pair of processes. The operators in a U-shaped cell carry the products with them from one process to the other. Also, the operators in a U-shaped cell work in cycles. That is each operator is given a pattern of work according to which, they move inside the cell. So, if the pattern of flow for some products is as depicted in Figure 1, with the products having to move back and forth between processes, the work pattern of the operators could be designed to match the flow pattern of those products. So it does not hurt even if flow patterns of certain products resemble a cyclic graph. In our case only a small percentage of the products have a cluttered flow pattern within a module, which the U-shaped cell can thus handle. The majority of the products flow in the

forward direction within the cell. Also, the number of processes within a module, obtained from the new procedure is comparatively lesser than that of the Merger Coefficient algorithm, making it easier to set up a U-shaped cell. In this way, U-shaped cells help in overcoming some limitations of the new procedure. Apart from the above, there are some other issues that need to be addressed before implementing the Layout modules on the shop floor. The next section highlights these aspects.

5.3 Practical issues governing the formation of Layout Modules

The Hybrid Flow Layout consists of Layout Modules, with the products having a unidirectional flow between the modules. Each module is similar to a cell. The processes within the module are arranged in a U-shape. When it comes to production control, pull mechanisms like route specific kanbans or CONWIP are used to control the flow of products between the cells. In a U-shaped cell, it is highly preferable to train all the operators to work on all the processes within that cell. This not only helps in improving the ergonomic aspects through job rotation, but also aids in applying Flexible Line Balance strategies like Circulation [13], Floating worker strategy [8] and Bucket Brigade strategy [8]. With these strategies, the operators within the cell operate a majority of processes with the cell. If a group of processes within the cell / module are different in scope than another group of processes within the same module, the difficulty of cross training increases. In such a case we should consider splitting the module into two modules so that the processes within each module are similar in scope and level of skill required. Another feature of a U-shaped cell is that one

operator controls the first and last processes. This enables to have a better control on the quality of the products produced by the cell. In this way, a new product cannot be started unless a finished product exits the cell. If there is a problem with the finished product, the operator is motivated to find the source of the problem within the cell and hence fix that problem before a new product enters the cell. It is clear that the operators in a U-shaped cell control more than one process and require cross training to enable better line balance and improved quality. If we treat each cell as a CONWIP loop, another issue that needs to be addressed is span of control [8]. It is possible that one person supervises certain group of processes within a module while another group needs to be supervised by a second person. It is preferable to have just one supervisor per cell to aid better management of the cell. So we might have to split modules again to apply the span of control in a better fashion. As an example let us consider the layout formed for the electronic manufacturing case study using the new procedure. The layout is depicted in Figure 5.5

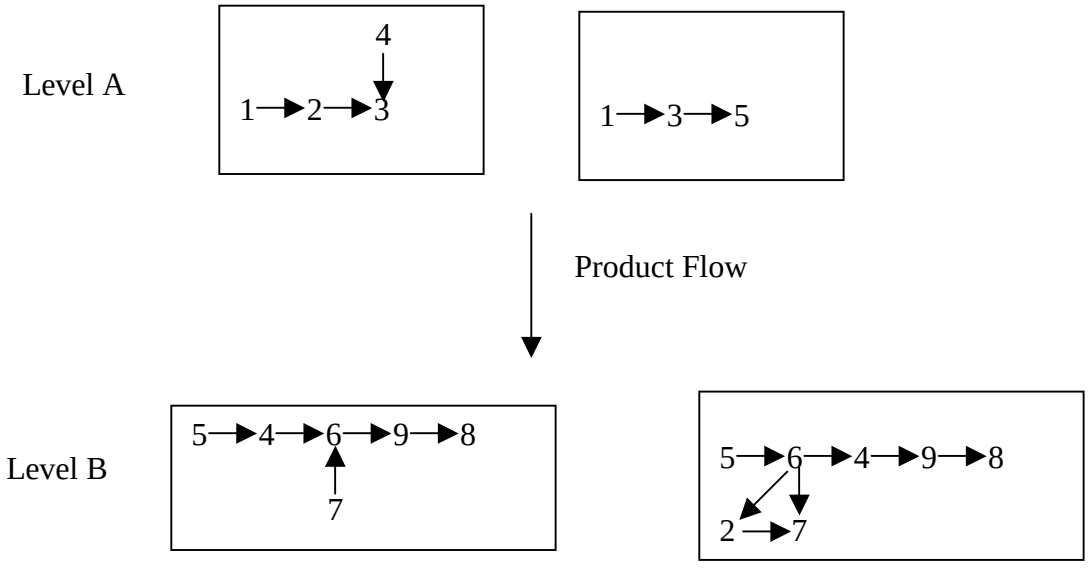


Figure 5.5: Configuration of the modules obtained using the new procedure, for the Electronic Manufacturing Case Study

The processes 9 and 8 found in the modules of Level 2 represent Quality Inspection and Rework respectively. These processes require operators trained to inspect every aspect of a Printed Circuit Board. The operators performing the processes 4, 5, 6 and 7 are not trained to do perform processes 9 and 8. If we include processes 9 and 8 within those modules, the operators performing those processes would be confined to a specific area of the cell, while the other operators would be circulating among the processes 4, 5, 6 and 7. Hence it would be difficult for one operator to control the first and last process since the last process for all the products within that module is either process 9 or process 8. In such a case the configuration shown in Figure 5.6 could be helpful till the operators are cross- trained.

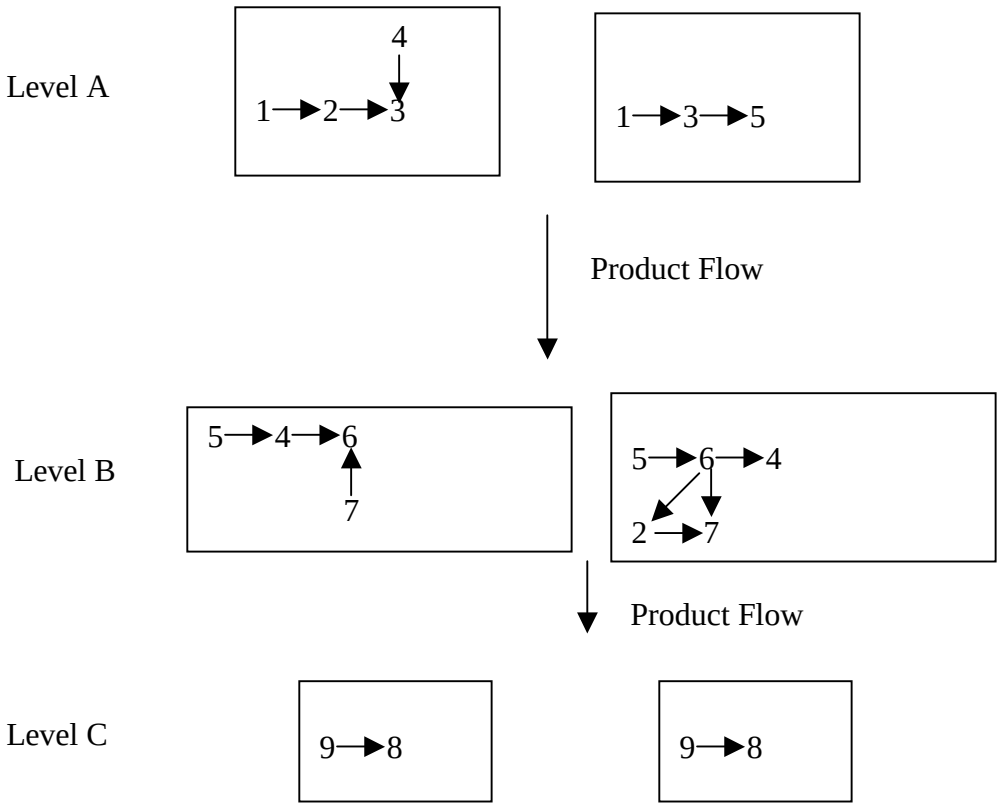


Figure 5.6: Modified structure of the modules for the Electronic Manufacturing Case Study, based on span of control

Thus the number of levels is increased to 3. These are some of the issues that have to be addressed before implementing Layout Modules.

5.4 Summary

So far we have discussed a procedure for forming a Hybrid Flow Layout that will enable in establishing flow and pull in a High Variety Low Volume Manufacturing facility. The limitations of the procedure have been addressed along with some of the issues that are important for a successful implementation of the layout. There are a few other factors that need to be delved in detail to facilitate better understanding of High Variety Low Volume Manufacturing facilities. These factors have not been examined as part of this research as the main objective of the research is just to give a general framework for Production Flow Analysis in High Variety Low Volume Manufacturing facilities. The next chapter provides some concluding remarks on this research, along with the future direction to enhance knowledge in this area.

Chapter 6 Conclusions

6.1 Summary of research

This research highlighted the importance of flow and pull as a means for the elimination of the seven types of manufacturing wastes. Flow and Pull can be established in any type of facility, but the methodology followed for achieving them differs depending on the product demand, variety and volumes in the facility. Flow and Pull as defined in the principles of Lean Manufacturing are more applicable to situations where the product volumes and demands are high. When we have a High Variety Low Volume Manufacturing Facility with varying demands, a different approach has to be pursued for accomplishing flow and pull. Chapter 1 provided a general framework for achieving full and pull in a High Variety Low Volume Manufacturing facility through the idea of Hybrid Flow Layouts. The rest of the chapters provided a detailed description on the procedures for developing a Hybrid Flow Layout characterized by Layout Modules. The procedure available from the literature was analyzed and its limitations were presented. A new procedure was developed to overcome the limitations of the existing procedure. A real world case study was then analyzed using both the procedures to demonstrate the flexibility of the new procedure. There are some limitations even with the new procedure and hence some countermeasures were provided to negate the effect of those limitations. Finally some practical issues that are significant for implementing the Hybrid Flow Layouts were discussed.

There are indeed some limitations to this research. The work focused predominantly on procedures for creating flow in High Variety Low Volume Manufacturing Facilities. Although a general framework for creating pull was discussed, the intricacies of developing a pull production system were not discussed. It embodies another area of research, which is not under the scope of this dissertation. However, the foundation for pull is the existence of flow, which has been addressed in detail.

6.2 Future Direction

There are a lot of things that would be of interest for extending this research. Layout Modules provide a structured approach for routing the products between modules in a unidirectional fashion. As we saw before, some products are routed through all the modules, while some products may not use some modules. If the modules are treated as CONWIP loops, wherein we operate only one product at a time within the loop, it would be of interest to associate a utilization factor to the modules since a majority of the products entering a module do not use all the processes within that module. This can lead to splitting the modules, to free up some processes that could be used to run a different product. But again, when the number of modules increases the amount of batching also increases, resulting in increased lead times. A procedure could be developed so that a balance would be struck between the utilization factor and the lead times of the products. Another extension of the research would be to examine the effects of applying pull production procedures like Route specific kanbans or CONWIP, on the Work-in-process levels and lead times in a Hybrid Flow Layout.

APPENDIX A

C program for determining the Merger Coefficients

Note: To execute the program we must first get a file that contains the common sub-strings between all pairs of operation sequences. (labeled as *stringfile2.txt* in the program). If there is a common sub-string (1, 2, 3), it is entered as 123 in the text file. Each common sub-string is entered in a new line in the text file. If we have a common sub-string like (1, 2, 3, 10), then an alphabet is used to substitute processes, which are denoted by more than 1 digit. So the common sub-string 1, 2, 3, 10 is entered as 123a where the alphabet 'a' represents process '10'.

```
#include<stdio.h>
#include<malloc.h>
#include<string.h>
#include<math.h>

typedef struct{
        int positions[100];
        int number;
}sequences;



int function(int *,sequences *,int ,int *,int *,int );
float manipulation(char *, char *);
int findmaximumascendingsequence(int *, int , sequences *, int *);

main()
{
        FILE *fp=NULL;
        FILE *fp1=NULL;


        int i=0;
        int j,k;
        float MC, MC1;
```

```
        char os1[100],os2[100],string[100][25],osbuffer[100];


        fp = fopen("h:/stringfile2.txt", "r");
        fp1 = fopen("h:/abc.txt", "w");
        //memset(string, 0, sizeof(int)*2500);
        if(fp==NULL)
        {
                printf("hi\n");
                exit(0);
        }
        while(!feof(fp))
        {
                fscanf(fp, "%s", string[i]);
                i=i+1;
        }
        i=i-1;
printf("Hi\n");
printf("i= %d\n", i);
fprintf(fp1, "i= %d\n", i);
        for(j=0;j<i;j++)
                fprintf(fp1, "%s\n", string[j]);



        for(j=0;j<i;j++)
        {

                for(k=0;k<i;k++)
                {
                        if(k!=j)
                        {


                                if(strlen(string[j]) < strlen(string[k]))
                                {
                                        memset(os1,0,100);
                                        memset(os2,0,100);
                                        strcpy(os1,string[j]);
                                        strcpy(os2,string[k]);
                                        MC = manipulation(os1, os2);
                                        printf("MC = %0.2f", MC);
                                        fprintf(fp1, "%0.2f, ", MC);
                                }
                                if(strlen(string[j]) > strlen(string[k]))
```

```
                {
                        memset(os1,0,100);
                        memset(os2,0,100);
                        strcpy(os1,string[k]);
                        strcpy(os2,string[j]);
                        MC = manipulation(os1, os2);
                        printf("MC = %0.2f", MC);
                        fprintf(fp1, "%0.2f, ", MC);
                }
                if(strlen(string[j]) == strlen(string[k]))
                {
                        memset(os1,0,100);
                        memset(os2,0,100);
                        strcpy(os1,string[j]);
                        strcpy(os2,string[k]);

                        MC = manipulation(os1, os2);
                        memset(os1,0,100);
                        memset(os2,0,100);
                        strcpy(os1,string[k]);
                        strcpy(os2,string[j]);
                        MC1 = manipulation(os1, os2);
                        if(MC > MC1)
                        {
                                printf("MC = %0.2f", MC);
                                fprintf(fp1, "%0.2f, ", MC);
                        }
                        else
                        {
                                printf("MC1 = %0.2f", MC1);
                                fprintf(fp1, "%0.2f, ", MC1);
                        }
                }
        }
        else
        {
                MC = 1;
                fprintf(fp1, "%0.2f, ", MC);
        }

    }

                fprintf(fp1, "\n");
}
fclose(fp);
```

```
	fclose(fp1);


	}

float manipulation(char *os1, char *os2)
{



	int pos[100],i,j;
	int numseq,h=0;
	int MD;
	int ID;
	float MC, MD1, ID1;
	int idgare[100];
	int tempval[100];
	int temptemp;
	sequences seq[1000];
	char *temp,*temp1;


	for(i=0;i<(strlen(os1));i++)
	{
		temp = strchr(os2,os1[i]);
		if(temp != NULL)
		{
			pos[i] = temp-os2;
		}
		else
			pos[i]=99;
	}

	printf("\nPositions: ");
	for(i=0;i < (strlen(os1));i++)
	{
		printf("%d ",pos[i]);
	}


	findmaximumascendingsequence(pos, strlen(os1), seq, &numseq);
	for(i=0;i<strlen(os1);i++)
	{
		if(pos[i] == 99)
			h++;
```

```
	}

	for(i=0;i<numseq;i++)
	{
		printf("\nPosition combination %d: ",i+1);
		for(j=0;j<seq[i].number;j++)
		{
			printf("%d ",seq[i].positions[j]);
		}
	}

if(h<strlen(os1))
	MD = strlen(os1) - seq[0].number;
else
	MD = strlen(os1);


		for(i=0;i<numseq;i++)
		{
			idgare[i] = 0;
			for(j=0;j<seq[i].number-1;j++)
			{
				tempval[i] = seq[i].positions[j+1]-seq[i].positions[j];
				temp = strchr(os1,os2[seq[i].positions[j]]);
				temp1 = strchr(os1,os2[seq[i].positions[j+1]]);
				temptemp = temp1-temp;
				if((tempval[i] - temptemp) >0)
				idgare[i] += tempval[i] - temptemp;
			}


		}


		ID=idgare[0];
		for(i=0;i<numseq;i++)
		{
			if(idgare[i] < ID)
			{
				ID = idgare[i];
			}
		}

		ID1 = (float)(ID);
		MD1 = (float)(MD);
```

```
                MC = 1 - (float)(MD1 + (ID1/((float)(strlen(os1))))) /
(float)(((float)(strlen(os2))) + 1);



        printf("\nThe value of MD is: %d and value of ID is: %d\n", MD, ID);


        return MC;

}


int findmaximumascendingsequence(int *pos, int len, sequences *seq, int *numseq)
{
        int i;
        int max;
        int count;
        int j=0,n=0, m=0;

        sequences temp[1000];


        temp[n].positions[0]=pos[0];
        temp[n].number=1;
        j=n+1;

        function(pos, temp, j, &n, &m, len);

        max=temp[0].number;
        for(i=1;i<m;i++)
        {
                if(temp[i].number > max)
                {
                        max = temp[i].number;
                }
        }

        /*To store all the equal occurences in the structure*/

        *numseq=0;
```

```
	for(i=0;i<m;i++)
	{
		if(temp[i].number == max)
		{
			seq[(*numseq)].number=max;
			for(j=0;j<max;j++)
			{
				seq[(*numseq)].positions[j]=temp[i].positions[j];
			}
			(*numseq)++;
		}
	}


	return 0;
}

int function(int *pos,sequences *temp,int j,int *n,int *m,int len)

/* To find all possible ascending values for each value in the array "pos" and storing
those values in the structure. */

{
	int i, l=0;
	int k,f=0, max,s,count=1;

	printf("n value %d", (*n));
	max = pos[(*n)];
	printf("\n j value: %d", j);

	   for(s=((*n)+1);s<(j-1);s++)
		{
			if((pos[s]>max && pos[s]<pos[j]) && pos[s]!=99)
			{
			max = pos[s];
			temp[(*m)].positions[count]=max;
			temp[(*m)].number++;
			count++;
			}

		}
	for(;j<len;j++)
	{
		if(pos[j]>max && pos[j]!=99)
		{
```

```
                max=pos[j];
                temp[(*m)].positions[count] = max;
                temp[(*m)].number++;
                count++;
        }
        else
        {
                if((f==0 && pos[j]>pos[(*n)]) && pos[j]!=99)
                {
                        f=j;
                }
        }
}

if(f>0)
{
        (*m)++;
        j=f;
        temp[(*m)].positions[0] = pos[(*n)];
        temp[(*m)].number = 1;
        function(pos, temp, j, n, m, len);
}
if(f==0)
{
        (*n)++;
        (*m)++;
        if((*n) < (len-1))
        {
                j=(*n)+1;
                temp[(*m)].positions[0]=pos[(*n)];
                temp[(*m)].number = 1;

                function(pos, temp, j, n, m, len);
        }
        else
        {
                for(i=0;i<(*m);i++)
                {
                        printf("\n Combination %d :",  i+1);
                        for(f=0;f<temp[i].number;f++)
                        {
                                printf("%d", temp[i].positions[f]);
                        }
                }
                return 0; }}}
```

APPENDIX B

Operation sequence for the case study

Number	Operation sequence	Number	Operation sequence
1	5,4,6,4,9,8	41	1,3,5,8,9
2	1,3,5,6,9,8	42	**1,4,5,3,9,8**
3	1,3,4,6,4,9,8	43	1,3,5,9,8
4	1,5,3,6,9,8	44	**3,4,5,9,8**
5	1,2,3	45	2,3
6	**4,5,4,6,9**	46	1,2,3,5,7,6,4,9,8
7	1,3,9,4	47	1,2,3,5,4,9,8
8	4,6,9,8	48	1,3,5,4,7,6,4,9,8
9	1,3,9,8	49	**3,6,2,4,9**
10	1,3,8,9	50	1,2,3,6,9,8
11	1,3,9,5,8	51	2,1,3,6,8,9
12	**1,3,4,5,6,4**	52	2,3,6,9,8
13	4,6,4,9,8	53	4,9,8
14	1,3,5,6	54	1,3,5,4,6
15	9,4	55	4,6,9,8
16	4,6,9,8	56	3,4,9
17	1,3,5,6,9	57	4,9,8
18	4,6,9,8	58	3,2,3
19	5,9,8	59	1,3,5,7,9,8
20	1,3,5,9	60	1,3,4,7,9,8
21	1,3,5,9,8	61	1,3,5,7,6,9,8
22	**1,3,4,5,7,9,8**	62	4,9,8
23	1,3,5,4,7,9,8	63	1,2,3,4,6,9,8
24	**1,3,4,5,6**	64	1,2,3,5,6,9,8
25	6,9,4	65	1,2,3,4,7,9,8
26	4,6,4,9,8	66	1,2,3,5,7,9,8
27	5,4,9,8	67	1,2,3,5,7,6,9,8
28	5,4,8,9	68	3,5,6,9,8
29	1,3,4,6,9,8	69	1,2,3,9,8
30	1,3,4	70	1,2,3,7,9,8
31	4,9,8	71	1,3,4
32	1,3,5,4,6,9	72	4,6,9,8
33	4,9,8	73	1,3,5,6,4
34	1,3,4,6,9,4	74	4,9,8
35	4,9,8	75	1,3,5
36	1,3,9,4,6,9,8	76	6,9,8
37	1,3,6,9,8	77	1,3,5,6
38	**1,3,4,5,6,9,8**	78	4,6,9,8
39	1,3,5,4,6,9,8	79	1,2,3,9,8
40	5,4,6,4,6,9,8	80	1,2,3,8,9

Operation sequence (Cont…)

Number	Operation sequence	Number	Operation sequence
81	5,4,9,8	121	5,7,9,8
82	2,3,5,7,9,8	122	2,3,5,7,9,8
83	2,3,5,7,6,9,8	123	1,2,3,7,4,6,9,8
84	3,2,4,6,9,8	124	1,2,3,5,4,7,6,9,8
85	1,2,3,7,9,4	125	2,5,6,9,8
86	4,6,9,8	126	2,3,5,6,9,8
87	1,2,5,3,9	127	1,2,3,1,6,9,8
88	1,6,9,8	128	1,3,5,2,7,9,8
89	1,3	129	1,2,3,4,5,7,4,6,9,8
90	2,4,6,9,8	130	5,1,2,3,6,9,8
91	1,2,3,5,6,8,9	131	2,6,9,8
92	1,3,5,2,9,8	132	1,2,3,5,6,4
93	1,3,5,4,6,9,8	133	4,6,9,8
94	1,2,3,5,4,6,9,8	134	2,3,4,5,7,6,9,8
95	1,3,9,4	135	1,2,3,5,7,6,9,4,9,8
96	4,9,8	136	2,3,6,9,4,9,8
97	2,4,3,9,8	137	3,5,9,8
98	1,6,9,8	138	2,3,4,9,8
99	5,9,4	139	5,6
100	4,9,8	140	6,9,8
101	2,1,3,4,9,8	141	1,2,3,5,9,4
102	4,3,9,8	142	4,9,8
103	1,2,3,4,5,6,9,8	143	2,3,7,4,6,9,8
104	1,2,3,4,5,4,6,9,8	144	1,2,3,5,4,6,9,8
105	1,2,3,5,4,7,9,8	145	1,2,3,4,5,6
106	1,2,3,6,9,8	146	2,3,7,9,8
107	1,2,3,5,8,6,9	147	1,2,3,6,9,8
108	5,3	148	1,3,2,6,9,8
109	4,3,6,9,8	149	5,6
110	1,2,3,5,6,9,8	150	6,9,8
111	1,3,2,5,6,9,8	151	5,4,6,9,8
112	2,3,6,9,8	152	1,3,5,6,9,8
113	1,2,3,7,9,8	153	5,1,6,9
114	3,1,2,6,9,8	154	1,2,3,4,6,9,4,9,8
115	1,7,8,9	155	1,2,3,5,7,6,9,8
116	1,2,3,4,8,9	156	3,2,4,9,8
117	1,3,4,7,6,9,8	157	5,1,6,9
118	4,6,9,8	158	1,3,4,7,9,8
119	1,2,3,5,4,7,8,9	159	1,3
120	5,6,9,8	160	4,2,6,9,8

Operation sequence (Cont…)

Number	Operation sequence	Number	Operation sequence
161	3,5,2,4,9,8	193	2,3,6,9,4,9,8
162	5,6,9,4	194	2,5,6,9,8
163	6,9,8	195	2,5,9,8
164	1,2,3,5,4,7,6,9,8	196	2,5,6,9,8
165	5,9,4	197	2,5,9,8
166	9,6,9	198	4,5,9,8
167	5,9,8	199	5,9,8
168	1,3,5,4,6,9,8	200	2,5,9,8
169	5,4,6,4,9,8	201	5,2,6,9,8
170	1,3,5,9,8	202	9,8
171	1,3,9,8	203	9,8
172	1,2,3,4,7,9,8	204	5,9,8
173	1,3,4	205	5,9,8
174	7,6,9,8	206	5,1,6,9,8
175	1,2,3,5,6,9,8	207	4,5,9,8
176	1,2,3	208	4,5,9,8
177	5,4,9,8	209	2,5,9,8
178	3,2,5,9,8	210	9,8
179	2,1,3,4,9,8	211	5,1,6,9,8
180	1,5,3,4,6,9,8	212	9,8
181	2,4,3,9,8	213	5,6,9,4
182	5,1,9,8	214	4,6,9,8
183	2,5,3,6,9,8	215	4,6,9,8
184	2,5,6,9,8	216	5,9,8
185	5,1,2,6,9,8	217	9,8
186	2,6,9,8	218	5,4,9,8
187	2,5,9,8	219	5,9,8
188	9,8	220	9,8
189	5,9,8	221	9,8
190	2,5,6,9,8	222	9,8
191	2,3,6,9,4,9,8	223	5,4,9,8
192	2,3,6,9,4,9,8		

KEY:

1 – Clipping

2 – Break out

3 – Post wave inspection

4 – Hand Solder

5 – Hardware Assembly

6 – Functional Test

7 – In-circuit Test

8 – Rework

9 – Final Quality Inspect

APPENDIX C

Matrix of Merger Coefficients for the Case Study

Note: This is a 63 x 63 matrix. Hence only a sample data is included.

Sub-string number →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	1	0.33	0.29	0.17	0.17	0.43	0.29	0.29	0.43	0.37	0.49	0.33	0.07	0.14	0.31	0.17	0.31	0.29
2	0.33	1	0.14	0.14	0.29	0.21	0.07	0.19	0.14	0.03	0.29	0.31	0.14	0.29	0.14	0.14	0.29	0.05
3	0.29	0.14	1	0.17	0.33	0.2	0.2	0.5	0.17	0.17	0.17	0.29	0.4	0.75	0.17	0.33	0.33	0.33
4	0.17	0.14	0.17	1	0.33	0.38	0.21	0.17	0.5	0.33	0.67	0.29	0.21	0.33	0.37	0.2	0.37	0.17
5	0.17	0.29	0.33	0.33	1	0.5	0.5	0.33	0.37	0.57	0.5	0.29	0.21	0.17	0.33	0.33	0.33	0.33
6	0.43	0.21	0.2	0.38	0.5	1	0.2	0.2	0.17	0.21	0.17	0.29	0.6	0.4	0.17	0.33	0.25	0.2
7	0.29	0.07	0.2	0.21	0.5	0.2	1	0.2	0.33	0.04	0.33	0.25	0.4	0.4	0.25	0.25	0.42	0.2
8	0.29	0.19	0.5	0.17	0.33	0.2	0.2	1	0.33	0.17	0.33	0.24	0.4	0.25	0.22	0.22	0.22	0.5
9	0.43	0.14	0.17	0.5	0.37	0.17	0.33	0.33	1	0.37	0.33	0.43	0.5	0.33	0.2	0.37	0.5	0.11
10	0.37	0.03	0.17	0.33	0.57	0.21	0.04	0.17	0.37	1	0.33	0.2	0.17	0.33	0.37	0.37	0.53	0.17
11	0.49	0.29	0.17	0.67	0.5	0.17	0.33	0.33	0.33	0.33	1	0.14	0.33	0.33	0.37	0.53	0.57	0.17
12	0.33	0.31	0.29	0.29	0.29	0.29	0.25	0.24	0.43	0.2	0.14	1	0.04	0.14	0.31	0.17	0.31	0.29
13	0.07	0.14	0.4	0.21	0.21	0.6	0.4	0.4	0.5	0.17	0.33	0.04	1	0.2	0.38	0.21	0.38	0.6
14	0.14	0.29	0.75	0.33	0.17	0.4	0.4	0.25	0.33	0.33	0.33	0.14	0.2	1	0.17	0.17	0.33	0.75
15	0.31	0.14	0.17	0.37	0.33	0.17	0.25	0.22	0.2	0.37	0.37	0.31	0.38	0.17	1	0.17	0.17	0.06
16	0.17	0.14	0.33	0.2	0.33	0.33	0.25	0.22	0.37	0.37	0.53	0.17	0.21	0.17	0.17	1	0.17	0.22
17	0.31	0.29	0.33	0.37	0.33	0.25	0.42	0.22	0.5	0.53	0.57	0.31	0.38	0.33	0.17	0.17	1	0.22
18	0.29	0.05	0.33	0.17	0.33	0.2	0.2	0.5	0.11	0.17	0.17	0.29	0.6	0.75	0.06	0.22	0.22	1
19	0.31	0.17	0.29	0.14	0.14	0.29	0.25	0.24	0.14	0.2	0.43	0.31	0.14	0.19	0.17	0.03	0.17	0.24
20	0.13	0.02	0.13	0.13	0.13	0.13	0.09	0.21	0.15	0.05	0.13	0.13	0.03	0.13	0.03	0.03	0.15	0.08
21	0.33	0.1	0.5	0.22	0.33	0.4	0.2	0.33	0.22	0.06	0.17	0.14	0.33	0.5	0.33	0.28	0.33	0.5
22	0.24	0.14	0.25	0.06	0.22	0.2	0.2	0.33	0.33	0.17	0.33	0.19	0.2	0.5	0.28	0.28	0.28	0.5
23	0.14	0.29	0.5	0.37	0.5	0.5	0.33	0.33	0.67	0.5	0.67	0.29	0.17	0.17	0.5	0.33	0.5	0.33
24	0.45	0.17	0.14	0.14	0.43	0.21	0.07	0.14	0.31	0.17	0.46	0.48	0.29	0.29	0.17	0.14	0.29	0.05
25	0.14	0.14	0.75	0.17	0.17	0.4	0.4	0.25	0.33	0.33	0.5	0.29	0.4	0.25	0.17	0.17	0.17	0.5
26	0.43	0.14	0.2	0.5	0.5	0.4	0.4	0.4	0.21	0.17	0.04	0.18	0.6	0.6	0.33	0.5	0.5	0.2
27	0.43	0.29	0.14	0.31	0.43	0.14	0.07	0.19	0.29	0.17	0.14	0.17	0.18	0.14	0.14	0.14	0.29	0.14

APPENDIX D

Complete List of Sub-strings for the Case Study

Number	Sub-string	Frequency	Number	Sub-string	Frequency
1	9,8	170	41	3,6,9,8	10
2	6,9	96	42	3,4,5	10
3	6,9,8	79	43	6,9,4	10
4	2,3	58	44	3,5,7	10
5	1,3	51	45	2,3,6,9	10
6	3,5	49	46	7,6,9	10
7	1,2	45	47	5,4,6,9	9
8	1,2,3	42	48	8,9	9
9	4,6	39	49	7,6,9,8	9
10	4,9	36	50	1,2,3,4	9
11	4,9,8	34	51	3,9,8	8
12	4,6,9	32	52	3,5,6,9	8
13	5,6	29	53	2,3,5,7	8
14	3,4	29	54	6,4,9,8	7
15	4,6,9,8	28	55	3,5,6,9,8	7
16	5,4	26	56	1,3,5,4	7
17	5,9	26	57	5,4,6,9,8	7
18	1,3,5	23	58	5,7,6	7
19	2,3,5	23	59	1,2,3,5,4	7
20	5,9,8	22	60	2,3,5,4	7
21	5,6,9	19	61	2,4	7
22	1,2,3,5	19	62	3,5,4,6	7
23	9,4	18	63	1,6,9	7
24	7,9	17	64	2,6,9,8	7
25	5,6,9,8	16	65	5,1	7
26	3,6	16	66	4,6,4	6
27	7,9,8	16	67	1,3,5,6	6
28	1,3,4	15	68	3,4,6	6
29	4,5	15	69	5,3	6
30	3,6,9	14	70	1,3,9	6
31	5,7	14	71	3,5,9	6
32	3,5,4	14	72	5,7,9,8	6
33	2,5	14	73	3,5,4,7	6
34	5,4,6	13	74	5,4,7	6
35	3,5,6	13	75	4,7,9,8	6
36	3,9	13	76	5,4,9,8	6
37	7,6	12	77	3,5,4,6,9	6
38	6,4	11	78	3,5,7,6	6
39	4,7	11	79	2,3,6,9,8	6
40	2,3,4	11	80	3,2	6

Complete List of Sub-strings for the Case Study (Cont…)

Number	Sub-string	Frequency	Number	Sub-string	Frequency
81	5,7,6,9	6	121	1,2,3,4,5	4
82	2,3,5,6	6	122	1,2,3,5,4,7	4
83	2,3,7	6	123	2,3,5,4,7	4
84	3,7	6	124	2,3,6,9,4,9,8	4
85	2,5,6,9,8	6	125	5,1,6,9	4
86	6,9,4,9,8	6	126	5,4,6,4	3
87	2,5,9,8	6	127	1,3,5,6,9	3
88	4,6,4,9,8	5	128	1,3,4,6	3
89	3,4,5,6	5	129	1,3,9,4	3
90	4,5,6	5	130	5,8	3
91	3,4,6,9	5	131	1,3,4,5,6	3
92	1,3,5,4,6	5	132	5,6,4	3
93	3,5,4,6,9,8	5	133	1,3,5,9,8	3
94	1,2,3,5,7	5	134	3,4,5,7	3
95	2,3,5,7,6	5	135	4,5,7	3
96	3,4,7	5	136	3,4,6,9,8	3
97	5,7,6,9,8	5	137	1,3,5,4,6,9,8	3
98	3,5,7,6,9	5	138	3,5,4,7,6	3
99	1,2,3,5,6	5	139	5,4,7,6	3
100	1,6,9,8	5	140	2,4,9	3
101	2,3,4,5	5	141	2,1,3	3
102	1,3,4,5	4	142	1,3,4,7	3
103	1,3,5,9	4	143	1,2,3,5,6,9,8	3
104	3,5,9,8	4	144	2,3,5,7,9,8	3
105	1,3,5,4,6,9	4	145	2,3,5,7,6,9,8	3
106	4,5,9,8	4	146	1,2,3,5,7,6,9	3
107	1,2,3,5,7,6	4	147	1,2,3,7,9	3
108	4,7,6	4	148	2,3,7,9,8	3
109	1,2,3,6,9,8	4	149	3,5,2	3
110	3,4,9	4	150	4,3,9,8	3
111	3,5,7,9,8	4	151	5,9,4	3
112	3,4,7,9,8	4	152	3,4,9,8	3
113	3,5,7,6,9,8	4	153	4,7,6,9,8	3
114	2,3,5,6,9,8	4	154	7,4,6,9,8	3
115	2,3,5,7,6,9	4	155	5,4,6,4,9,8	2
116	2,3,7,9	4	156	1,3,5,6,9,8	2
117	3,7,9	4	157	1,5,3	2
118	1,2,3,7	4	158	5,3,6,9,8	2
119	5,2	4	159	4,5,4,6,9	2
120	4,3	4	160	1,3,9,8	2

Complete List of Sub-strings for the Case Study (Cont…)

Number	Sub-string	Frequency	Number	Sub-string	Frequency
161	3,8,9	2	182	2,4,6,9,8	2
162	1,3,5,4,7	2	183	3,2,4	2
163	3,5,4,7,9,8	2	184	2,5,3	2
164	4,8,9	2	185	1,3,5,2	2
165	1,3,4,6,9	2	186	1,2,3,5,4,6,9,8	2
166	3,4,6,9,4	2	187	2,4,3,9,8	2
167	1,3,6	2	188	2,1,3,4,9,8	2
168	3,4,5,6,9,8	2	189	1,2,3,4,5,6	2
169	3,5,8	2	190	1,3,2	2
170	5,3,9	2	191	3,2,5	2
171	7,6,4,9,8	2	192	3,1	2
172	6,8,9	2	193	1,2,6,9,8	2
173	1,3,5,7	2	194	7,8,9	2
174	1,3,4,7,9,8	2	195	2,3,7,4,6,9,8	2
175	1,2,3,4,6,9	2	196	1,2,3,5,4,7,6,9,8	2
176	2,3,4,6,9	2	197	2,3,4,5,7	2
177	1,2,3,4,7,9,8	2	198	5,1,2	2
178	1,2,3,5,7,6,9,8	2	199	2,4,9,8	2
179	1,2,3,9,8	2	200	5,6,9,4	2
180	1,2,3,7,9,8	2	201	5,1,6,9,8	2
181	3,5,6,4	2			

KEY:

1 – Clipping

2 – Break out

3 – Post wave inspection

4 – Hand Solder

5 – Hardware Assembly

6 – Functional Test

7 – In-circuit Test

8 – Rework

9 – Final Quality Inspect

APPENDIX E

Operation Sequence expressed in terms of Modules

Note: The ones that have been highlighted show the existence of backtracking, which is fixed in Phase 3.

Number	Operation sequence	Module sequence	Number	Operation sequence	Module sequence
1	5,4,6,4,9,8	M2, M1	31	4,9,8	M1
2	1,3,5,6,9,8	M2, M1	32	1,3,5,4,6,9	M2, M1
3	1,3,4,6,4,9,8	M2, M1	33	4,9,8	M1
4	1,5,3,6,9,8	M2, M1	34	1,3,4,6,9,4	M2, M1
5	1,2,3	M2	35	4,9,8	M1
6	**4,5,4,6,9**	M1, M2, M1	36	1,3,9,4,6,9,8	M2, M1
7	1,3,9,4	M2, M1	37	1,3,6,9,8	M2, M1
8	4,6,9,8	M1	38	**1,3,4,5,6,9,8**	M2,M1,M2,M1
9	1,3,9,8	M2, M1	39	1,3,5,4,6,9,8	M2, M1
10	1,3,8,9	M2, M1	40	5,4,6,4,6,9,8	M2, M1
11	1,3,9,5,8	M2, M1	41	1,3,5,8,9	M2, M1
12	**1,3,4,5,6,4**	M1, M2, M1	42	**1,4,5,3,9,8**	M2,M1,M2,M1
13	4,6,4,9,8	M1	43	1,3,5,9,8	M2, M1
14	1,3,5,6	M2, M1	44	**3,4,5,9,8**	M2,M1,M2,M1
15	9,4	M1	45	2,3	M2
16	4,6,9,8	M1	46	1,2,3,5,7,6,4,9,8	M2, M1
17	1,3,5,6,9	M2, M1	47	1,2,3,5,4,9,8	M2, M1
18	4,6,9,8	M1	48	1,3,5,4,7,6,4,9,8	M2, M1
19	5,9,8	M2, M1	49	**3,6,2,4,9**	M2,M1,M2,M1
20	1,3,5,9	M2, M1	50	1,2,3,6,9,8	M2, M1
21	1,3,5,9,8	M2, M1	51	2,1,3,6,8,9	M2, M1
22	**1,3,4,5,7,9,8**	M2,M1,M2,M1	52	2,3,6,9,8	M2, M1
23	1,3,5,4,7,9,8	M2, M1	53	4,9,8	M1
24	**1,3,4,5,6**	M2,M1,M2,M1	54	1,3,5,4,6	M2, M1
25	6,9,4	M1	55	4,6,9,8	M1
26	4,6,4,9,8	M1	56	3,4,9	M2, M1
27	5,4,9,8	M2, M1	57	4,9,8	M1
28	5,4,8,9	M2, M1	58	3,2,3	M2
29	1,3,4,6,9,8	M2, M1	59	1,3,5,7,9,8	M2, M1
30	1,3,4	M2, M1	60	1,3,4,7,9,8	M2, M1

Operation Sequence expressed in terms of Modules (Cont…)

Number	Operation sequence	Module sequence	Number	Operation sequence	Module sequence
61	1,3,5,7,6,9,8	M2, M1	101	2,1,3,4,9,8	M2, M1
62	4,9,8	M1	102	**4,3,9,8**	M1, M2, M1
63	1,2,3,4,6,9,8	M2, M1	103	**1,2,3,4,5,6,9,8**	M2,M1,M2,M1
64	1,2,3,5,6,9,8	M2, M1	104	**1,2,3,4,5,4,6,9,8**	M2,M1,M2,M1
65	1,2,3,4,7,9,8	M2, M1	105	1,2,3,5,4,7,9,8	M2, M1
66	1,2,3,5,7,9,8	M2, M1	106	1,2,3,6,9,8	M2, M1
67	1,2,3,5,7,6,9,8	M2, M1	107	1,2,3,5,8,6,9	M2, M1
68	3,5,6,9,8	M2, M1	108	5,3	M2
69	1,2,3,9,8	M2, M1	109	**4,3,6,9,8**	M1, M2, M1
70	1,2,3,7,9,8	M2, M1	110	1,2,3,5,6,9,8	M2, M1
71	1,3,4	M2, M1	111	1,3,2,5,6,9,8	M2, M1
72	4,6,9,8	M1	112	2,3,6,9,8	M2, M1
73	1,3,5,6,4	M2, M1	113	1,2,3,7,9,8	M2, M1
74	4,9,8	M1	114	3,1,2,6,9,8	M2, M1
75	1,3,5	M2	115	1,7,8,9	M2, M1
76	6,9,8	M1	116	1,2,3,4,8,9	M2, M1
77	1,3,5,6	M2, M1	117	1,3,4,7,6,9,8	M2, M1
78	4,6,9,8	M1	118	4,6,9,8	M2, M1
79	1,2,3,9,8	M2, M1	119	1,2,3,5,4,7,8,9	M2, M1
80	1,2,3,8,9	M2, M1	120	5,6,9,8	M2, M1
81	5,4,9,8	M2, M1	121	5,7,9,8	M2, M1
82	2,3,5,7,9,8	M2, M1	122	2,3,5,7,9,8	M2, M1
83	2,3,5,7,6,9,8	M2, M1	123	1,2,3,7,4,6,9,8	M2, M1
84	3,2,4,6,9,8	M2, M1	124	1,2,3,5,4,7,6,9,8	M2, M1
85	1,2,3,7,9,4	M2, M1	125	2,5,6,9,8	M2, M1
86	4,6,9,8	M1	126	2,3,5,6,9,8	M2, M1
87	1,2,5,3,9	M2, M1	127	1,2,3,1,6,9,8	M2, M1
88	1,6,9,8	M2, M1	128	1,3,5,2,7,9,8	M2, M1
89	1,3	M2, M1	129	**1,2,3,4,5,7,4,6,9,8**	M2,M1,M2,M1
90	2,4,6,9,8	M2, M1	130	5,1,2,3,6,9,8	M2, M1
91	1,2,3,5,6,8,9	M2, M1	131	2,6,9,8	M2, M1
92	1,3,5,2,9,8	M2, M1	132	1,2,3,5,6,4	M2, M1
93	1,3,5,4,6,9,8	M2, M1	133	4,6,9,8	M2, M1
94	1,2,3,5,4,6,9,8	M2, M1	134	2,3,4,5,7,6,9,8	M2, M1
95	1,3,9,4	M2, M1	135	1,2,3,5,7,6,9,4,9,8	M2, M1
96	4,9,8	M1	136	2,3,6,9,4,9,8	M2, M1
97	2,4,3,9,8	M2, M1	137	3,5,9,8	M2, M1
98	1,6,9,8	M2, M1	138	2,3,4,9,8	M2, M1
99	5,9,4	M2, M1	139	5,6	M2, M1
100	4,9,8	M1	140	6,9,8	M1

Operation Sequence expressed in terms of Modules (Cont…)

Number	Operation sequence	Module sequence	Number	Operation sequence	Module sequence
141	1,2,3,5,9,4	M2, M1	183	2,5,3,6,9,8	M2, M1
142	4,9,8	M1	184	2,5,6,9,8	M2, M1
143	2,3,7,4,6,9,8	M2, M1	185	5,1,2,6,9,8	M2, M1
144	1,2,3,5,4,6,9,8	M2, M1	186	2,6,9,8	M2, M1
145	**1,2,3,4,5,6**	M2,M1,M2,M1	187	2,5,9,8	M2, M1
146	2,3,7,9,8	M2, M1	188	9,8	M1
147	1,2,3,6,9,8	M2, M1	189	5,9,8	M2,M1
148	1,3,2,6,9,8	M2, M1	190	2,5,6,9,8	M2, M1
149	5,6	M2, M1	191	2,3,6,9,4,9,8	M2, M1
150	6,9,8	M1	192	2,3,6,9,4,9,8	M2, M1
151	5,4,6,9,8	M2, M1	193	2,3,6,9,4,9,8	M2, M1
152	1,3,5,6,9,8	M2, M1	194	2,5,6,9,8	M2, M1
153	5,1,6,9	M2, M1	195	2,5,9,8	M2, M1
154	1,2,3,4,6,9,4,9,8	M2, M1	196	2,5,6,9,8	M2, M1
155	1,2,3,5,7,6,9,8	M2, M1	197	2,5,9,8	M2, M1
156	3,2,4,9,8	M2, M1	198	**4,5,9,8**	M1, M2, M1
157	5,1,6,9	M2, M1	199	5,9,8	M2, M1
158	1,3,4,7,9,8	M2, M1	200	2,5,9,8	M2, M1
159	1,3	M2	201	5,2,6,9,8	M2, M1
160	**4,2,6,9,8**	M1, M2, M1	202	9,8	M1
161	3,5,2,4,9,8	M2, M1	203	9,8	M1
162	5,6,9,4	M2, M1	204	5,9,8	M2, M1
163	6,9,8	M1	205	5,9,8	M2, M1
164	1,2,3,5,4,7,6,9,8	M2, M1	206	5,1,6,9,8	M2, M1
165	5,9,4	M2, M1	207	**4,5,9,8**	M1, M2, M1
166	9,6,9	M1	208	**4,5,9,8**	M1, M2, M1
167	5,9,8	M2, M1	209	2,5,9,8	M2, M1
168	1,3,5,4,6,9,8	M2, M1	210	9,8	M1
169	5,4,6,4,9,8	M2, M1	211	5,1,6,9,8	M2, M1
170	1,3,5,9,8	M2, M1	212	9,8	M1
171	1,3,9,8	M2, M1	213	5,6,9,4	M2, M1
172	1,2,3,4,7,9,8	M2, M1	214	4,6,9,8	M1
173	1,3,4	M2, M1	215	4,6,9,8	M1
174	7,6,9,8	M1	216	5,9,8	M2, M1
175	1,2,3,5,6,9,8	M2, M1	217	9,8	M1
176	1,2,3	M2, M1	218	5,4,9,8	M2, M1
177	5,4,9,8	M2, M1	219	5,9,8	M2, M1
178	3,2,5,9,8	M2, M1	220	9,8	M1
179	2,1,3,4,9,8	M2, M1	221	9,8	M1
180	1,5,3,4,6,9,8	M2, M1	222	9,8	M1
181	**2,4,3,9,8**	M2,M1,M2,M1	223	5,4,9,8	M2, M1
182	5,1,9,8	M2, M1			

KEY:

1 – Clipping

2 – Break out

3 – Post wave inspection

4 – Hand Solder

5 – Hardware Assembly

6 – Functional Test

7 – In-circuit Test

8 – Rework

9 – Final Quality Inspect

REFERENCES

[1] Taiichi Ohno, 1998, Toyota Production System, Productivity Press, Portland.

[2] Yasuhiro Monden, 1993, Toyota Production System, Industrial Engineering and Management Press.

[3] Jon C. Yingling, 2001, Course notes – Principles and Practices of Lean Manufacturing, University of Kentucky, Lexington, KY

[4] John. L. Burbridge, 1997, Production Flow Analysis for Planning Group Technology, Clarendon Press.

[5] Shahrukh. A. Irani, 1999, Handbook of Cellular Manufacturing Systems, Wiley – Interscience.

[6] King, J.R. 1980, Machine-Component grouping in production flow analysis: An approach using rank order clustering algorithm, International Journal of Production Research, Vol. 18(2), pp 213-232.

[7] Daita, S.T.S., Irani, S.A., and Kotamraju, S, 1999, Algorithms for production flow analysis, International Journal of Production Research, 37 (11), 2609-2638.

[8] Wallace J. Hopp and Mark L. Spearman, 2001, Factory Physics, Irwin McGraw-Hill.

[9] Shukla, J, 1995, Hybrid and Progressive Cellular Layouts for Jobshops, Master of Science thesis, Department of Mechanical Engineering, University of Minnesota, Minneapolis, MN.

[10] Irani, S. A. & Huang, H (2000), Custom Design of Facility Layouts for Multi-Product Facilities using Layout Modules, IEEE Transactions on Robotics and Automation, 16(3), 259-267.

[11] Michael Gilleland, Levenshtein Distance in three flavours, www.merriampark.com.

[12] Anderberg, M.R, 1973, Cluster Analysis for Applications, Academic Press, New York, NY.

[13] Kelly Andrew Wise, 2001, Adapting traditional and u-line assembly line-balancing problems to allow for the inclusion of circulating workers in shared stations via heuristic and optimization methods, Master of Science Thesis, Department of Mechanical Engineering, University of Kentucky, Lexington, KY

VITA

Mukund Narayanan was born in Coimbatore, India on 16th January 1979. He earned his high school diploma from Coimbatore, India in 1996. He earned his Bachelor's degree with honors in the field of Mechanical Engineering from Birla Institute of Technology and Science, India in 2000. He joined the Master's program in Manufacturing Systems Engineering at the University of Kentucky in Fall semester of 2000. He worked as a Research Assistant under Dr.Jon C Yingling at the Center for Robotics and Manufacturing Systems during his Master's degree.