

**Job Sequencing & WIP level determination in a cyclic  
*CONWIP* Flowshop with Blocking**

by

Nipun Palekar

Proposal submitted to the Faculty of the  
Virginia Polytechnic Institute and State University  
in partial fulfillment of the requirements for the degree of

Masters in Science  
in  
Industrial and Systems Engineering

Approved:

---

**Dr. Subhash C. Sarin**

---

Dr. P. Koelling

---

Mr. Tim W. Joseph

# **Job Sequencing & WIP level determination in a cyclic *CONWIP* Flowshop with Blocking**

by

**Nipun P. Palekar**

Dr. Subhash C. Sarin, Chairman  
Industrial and Systems Engineering

## **ABSTRACT**

A CONWIP (Constant Work-In-Progress) system is basically a hybrid system with a PUSH-PULL interface at the first machine in the line. This research addresses the most general case of a cyclic CONWIP system by incorporating two additional constraints over earlier studies namely; stochastic processing times and limited intermediate storage. One of the main issues in the design of a CONWIP system is the WIP level 'M', to be maintained. This research proposes an iterative procedure to determine this optimal level. The second main issue is the optimization of the line by determining an appropriate job sequence. This research assumes a 'permutational' scheduling policy and proposes an iterative approach to find the best sequence. The approach utilizes a controlled enumerative approach called the Fast Insertion Heuristic (FIH) coupled with a method to appraise the quality of every enumeration at each iteration. This is done by using a modified version of the Floyd's algorithm, to determine the cycle time (or Flow time) of a partial/full solution.

The performance measures considered are the Flow time and the Interdeparture time (inverse of throughput). Finally, both the methods suggested for the two subproblems, are tested through computer implementations to reveal their proficiency.

## ACKNOWLEDGEMENTS

This is a major milestone in my life, for which I first wish to thank my parents, without whom this would not have been possible. Their unconditional love and constant support over the years is something that I cannot thank them enough for. I also thank them for having faith in me, and my capabilities, and the advice they offered to me over all these years.

My deepest appreciation goes to my advisor, the Chairman of my committee, Professor Subhash C. Sarin, for his valuable assistance, guidance, and encouragement in bringing this research work to a successful completion. I highly commend him for his character and academic achievements. He showed extraordinary patience and was available for advice, pretty much 24 hours a day.

I am also grateful to my dissertation committee members, Dr. Koelling and Mr. Tim Joseph, for their advise and help. I appreciate Dr. Koelling for his flexibility and Mr. Tim Joseph who treated me like a friend over the past 1 year I have worked with him on a project and now on this rsearch. I consider both of them as my good friends whose friendship I will always cherish.

Last, but not least, I thank all my colleagues and friends that I have acquired throughout my years at Virginia Tech – Karl-Johan Nyberg, Xiaomei Zhu, Saleel Gadgil, Saleel Limaye, Aditya Gadre, Greg Beskow, Laurent Matthey, John Tester, Amit Kabnurkar, and Elise Caruso - to name a few. Special thanks are due to Xiaomei Zhu and Aditya Gadre, who helped me a lot in the coding of my algorithms and Johan for giving me company and support. I would also like to thank the Faculty and the Staff in the Department of Industrial and Systems Engineering and the team at Ericsson, Lynchburg. I also appreciate the help and understanding Lovedia Cole and Kathy Buchanan has showed to me.

I am honored to have been a part of such a talented and dedicated staff during my years at Tech. I will cherish this period greatly for the rest of my life.

# Table of Contents

<b>ABSTRACT .....</b>	<b>2</b>
<b>ACKNOWLEDGEMENTS.....</b>	<b>3</b>
<b>TABLE OF CONTENTS.....</b>	<b>4</b>
<b>CHAPTER 1: INTRODUCTION.....</b>	<b>10</b>
1.1 BACKGROUND AND MOTIVATION.....	10
1.2 PUSH AND PULL SYSTEMS – DEFINITION AND DIFFERENCES.....	12
1.3 EFFECT OF VARIABILITY .....	16
1.4 IMPORTANCE OF BOTTLENECK: A COMPARISON OF CONWIP WITH KANBAN.....	18
1.5 ROBUSTNESS OF A CONTROL SYSTEM: CONWIP VS. PUSH.....	20
1.6 PEOPLE ISSUES .....	22
1.7 TRADE-OFF BETWEEN HIGH LINE UTILIZATION AND WIP .....	23
1.8 CONCLUDING REMARKS.....	25
<b>CHAPTER 2: PROBLEM BACKGROUND AND LITERATURE REVIEW</b> <b>.....</b>	<b>26</b>
2.1 PROBLEM DEFINITION.....	28
2.1.1 <i>Problem Statement</i> .....	28
2.1.2 <i>Scope and Assumptions</i> .....	28
2.2 OUTLINE OF THESIS .....	30
2.3 STOCHASTIC DISTRIBUTION – WHY EXPONENTIAL?? .....	31
2.4 PERFORMANCE MEASURES .....	33
2.5 LITERATURE REVIEW .....	34
2.5.1 <i>Study of CONWIP systems and terminologies used</i> .....	34
2.5.2 <i>Use of simulation in related research</i> .....	36
2.5.3 <i>Use of Queuing theory models and Mean Value Analysis to similar problems</i> .....	37
2.5.4 <i>Sequencing policy at the first machine</i> .....	41
2.5.4.1 Analytical and Heuristic Approaches .....	41
2.5.4.2 Controlled Enumerative Approaches.....	44
2.5.4.3 Sequencing problem for added constraints such as CONWIP control, cyclic flow and finite intermediate buffers.....	46
<b>CHAPTER 3: PRELIMINARIES: CONCEPTS AND TERMINOLOGIES</b> <b>OF CONWIP IN VIEW OF THE PROBLEM AT HAND.....</b>	<b>48</b>
3.1 CONCEPTS .....	48
3.1.1 <i>One Product Case</i> .....	48
3.1.2 <i>Multi-product case</i> .....	50

3.1.3 <i>Conceptual Bottleneck Machine (CBN)</i> .....	51
3.2 ANALYSIS OF PRELIMINARY TEST RESULTS .....	52
3.2.1 <i>Common Data</i> .....	52
3.2.2 <i>Experiment description</i> .....	53
3.2.3 <i>Definitions Used and Results</i> .....	53
3.3 MIP PROGRAMMING FORMULATION.....	56
<b>CHAPTER 4: MODIFIED MEAN VALUE ANALYSIS ALGORITHM (MMVA) .....</b>	<b>60</b>
4.1 INTRODUCTION.....	60
4.2 MEAN VALUE ANALYSIS ALGORITHM .....	62
4.3 CASCADING BOTTLENECKS .....	65
4.3.1 <i>A Method to determine the approximate value of Blocking time</i> .....	72
4.4 A LOWER BOUND ON $M^*$ .....	78
4.5 DESCRIPTION OF THE EXPERIMENTS CONDUCTED AND THE OBSERVATIONS MADE.....	80
4.6 CONCLUSIONS .....	89
<b>CHAPTER 5: SEQUENCING MULTIPLE JOB TYPES IN A FIXED INTERMEDIATE STORAGE CONWIP SYSTEM.....</b>	<b>90</b>
5.1 INTRODUCTION.....	90
5.2 COMPLEXITY OF THE SEQUENCING PROBLEM .....	92
5.3 A NETWORK FLOW INTEGER PROGRAMMING MODEL OF THE PROBLEM TO DETERMINE THE CRITICAL PATH AND OTHER RELATED CONCEPTS .....	94
5.3.1 <i>The Concept of Critical Path or Flow time for a permutation, and the definition of                 variables</i> .....	94
5.3.2 <i>An IP Model to determine the optimal sequence</i> .....	98
5.4 AN IP MODEL TO DETERMINE THE LENGTH OF CRITICAL PATH FOR A GIVEN SEQUENCE AMONG JOBS.....	100
5.4.1 <i>Model for an Open System with Zero Buffers</i> .....	100
5.4.2 <i>A Model to determine critical path for a System with Zero Buffer Between Machines                 and CONWIP Level of M</i> .....	101
5.5 FIH IN ITS SIMPLEST FORM (AS SUGGESTED BY NAWAZ ET AL (1983)) .....	115
5.6 MODIFIED FLOYD'S ALGORITHM FOR DETERMINATION OF THE CRITICAL PATH.....	116
5.6.1 <i>Determination of Cycle Length</i> .....	121
5.6.2 <i>CONWIP Level &amp; Cyclic Sequence Considerations in the Network Representation</i> .....	122
5.6.3 <i>Computational Complexity of the method to determine the best sequence</i> .....	124
5.7 EXPERIMENTAL STUDY TO TEST THE QUALITY OF THE SEQUENCES GENERATED.....	125
5.7.1 <i>Experiments Design</i> .....	125
5.7.2 <i>Results and analysis of the experiments</i> .....	127
<b>CHAPTER 6: SUMMARY AND CONCLUSIONS .....</b>	<b>132</b>
6.1 SUMMARY AND CONCLUSIONS .....	132
6.2 FUTURE RESEARCH .....	135
<b>BIBLIOGRAPHY .....</b>	<b>137</b>

<b>APPENDICES .....</b>	<b>141</b>
APPENDIX A: VISUAL BASIC PROGRAM TO DETERMINE THE BEST (CLOSE TO OPTIMAL) WIP LEVEL M .....	142
APPENDIX B: TIME MATRICES USED IN THE TESTING OF MMVA (UNIFORM[5 – 25]).....	148
APPENDIX C: C ++ PROGRAM TO DETERMINE THE “BEST” SEQUENCE USING FIH AND .. MMVA .....	152
APPENDIX D: TEST EXCEL SHEETS FOR 3, 4, 5 JOB CASES .....	168
<b>VITA .....</b>	<b>173</b>

## List of Figures

Fig 1.1.	Release triggers of push and pull production systems	13
Fig 1.2.	An illustration of a CONWIP system	15
Fig 1.3.	Relative robustness of CONWIP and pure push systems (As mentioned in “Factory Physics [2nd ed]	21
Fig 1.4.	Variation of Flow time and Cycle time with WIP level M	24
Fig 2.1.	The Density function of job processing time observed in practice	31
Fig 3.1.	Variation of IT and FT with CONWIP level M	55
Fig 3.2.	Job i and j on the Gantt chart	57
Fig 4.1.	Schematic depiction of Events at a buffer under consideration on a time line	72
Fig 4.2.	Snapshots in time of the arrival and departure events at a buffer	73
Fig 4.3.	Depiction of Methodology to generate processing times values during simulation from the pre-defined exponential distributions	82
Fig 4.4.	Comparison plots of simulation vs. MMVA outputs for CONWIP level of 7 and time matrix # 1	86
Fig 4.5.	Comparison plots of simulation vs. MMVA outputs for CONWIP level of 14 and time matrix # 1	86
Fig 4.6.	Comparison plots of simulation vs. MMVA outputs for CONWIP level of 35 and time matrix # 1	87
Fig 5.1.	Network flow representation for make span calculation of a permutation schedule $\sigma$ , for a zero buffer capacity case	95
Fig 5.2.	Network flow representation for make span calculation of a permutation schedule $\sigma$ , showing the Epochs and the critical path using a color scheme	101
Fig 5.3.	Schematic graph of a typical open flow shop Gantt chart	103
Fig 5.4.	Schematic graph of a typical flow shop Gantt chart with CONWIP level condition	106
Fig 5.5.	Gantt chart showing the concept of Epochs in a CONWIP system	110
Fig 5.6.	A directed graph showing nodes i, j, and k	116
Fig 5.7.	Schematic diagram showing the pivot manipulations in a Floyd’s matrix	118
Fig 5.8.	Flow chart showing additions to the Floyd’s Heuristic	120





## List of Tables

Table 2.1	Variation of flow time over M and sequence	26
Table 3.2.3-1	Comparison between Different Production Control Policies	54
Table 3.2.3-2	Variation of IT and FT with CONWIP level M	55
Table 4.5.1.	Intermediate storage sizes used in the experiment	81
Table 4.4-1.	Comparison between the simulation and MMVA results for CONWIP level of 7	83
Table 4.4-2.	Comparison between the simulation and MMVA results for CONWIP level of 14	84
Table 4.4-3	Comparison between the simulation and MMVA results for CONWIP level of 35	85
Table 5.7.1-1	Summary of the Experiments Performed	126
Table 5.7.2-1	Summary of results of experiments of comparison between job selection Strategies	127
Table 5.7.1-2	Flow times for the stochastic case, obtained using simulation	130
Table 5.7.1-3	Processing time distribution extremes with deviation level of 10% ( $\pm 5\%$ )	130
Table 5.7.1-4	Processing time distribution extremes with deviation level of 5% ( $\pm 2.5\%$ )	130
Table 5.7.1-5	Processing time distribution averages	131

# Chapter 1: Introduction

## 1.1 *Background and Motivation*

Modern day high volume manufacturing and assembly lines are excellent examples of Transfer lines and automated flexible flow lines. Such lines are capital intensive and thus must be utilized to their full extent. As a result, a lot of research has been done to develop production control rules, to obtain a tighter control of production parameters such as throughput, machine utilizations and work-in-progress. Many systems, which control the flow of material in assembly lines, exist and a lot of research has been done on them since the widespread use of automation. These systems are also described as Order Release Mechanisms. Many criteria are used to optimize such a system. Trade-off between high line utilization and WIP, is one such commonly encountered trade-off. Some authors discuss the minimization of *total* WIP (Spearman et al. 1989), while others aim at minimizing the location as well as the quantity of WIP.

The importance of the work in this area stems from the fact that Material Requirement Planning (MRP) and Manufacturing Resources Planning (MRP-II) were the first structured methods to be developed for Production Planning and Control (PPC) and their disadvantages are now started to become apparent in modern day optimized customer driven settings. To this day, these methods are deployed commonly in industry. MRP-based methods provide a timely plan for the acquisition of raw materials and their processing, based on the Bill of Materials (BOM), and the procurement and production Lead Times of the end products (those highest in the MRP hierarchy). MRP-II based methods further accommodate the limited capacity of the available resources, ensuring that the MRP timely plans would be feasible.

Another important fact is that, in factories, the WIP levels between machines have capacity limits. This is mainly due to the limited physical space available to store the parts temporarily. Other reasons could be found in semiconductor or electronic manufacturing industries. Some parts need to be heated to a fixed predetermined temperature before they can be worked on the machine down the line. This means that if the queue before a machine down the line is too big, the average wait time for the parts would be too high and the parts would cool off to a temperature below the lower acceptable limit. This limitation has been overlooked in the research carried out in this area. The queue capacities between the machines or stations are typically considered to be infinite. However, this research considers limited intermediate (between machines) buffers and thus is more in tune with the real world systems.

The last but most important factor, which differentiates this research from the others, is the consideration of processing times as stochastic variables. This is very true in actual production settings where processing times are not deterministic and more so if one of the operations in the line is a manual operation or requires some level of manual interference. This variance could even be the sole cause of failure of perfectly planned MRP-based system, which is very highly sensitive to any system internal variations.

Next to set the stage, we describe the various production control systems, their differences, analogies, advantages and disadvantage over each other, with more importance given to a CONWIP system, as it is the operational system under consideration in this research.

## 1.2 *Push and Pull systems – Definition and differences*

Virtually, all descriptions of just-in-time (JIT) systems make use of the terms *push* and *pull* production systems. In the following paragraph, we present the two systems by offering a formal definition at the conceptual level. The reason for superiority of pull systems over push systems is also explained. By separating the concepts of push from pull in their specific implementations, we observe that most real-world systems are actually hybrids or mixtures of the “pure push” and “pure pull” systems. The CONWIP system, considered in this research, is just such a synthesis.

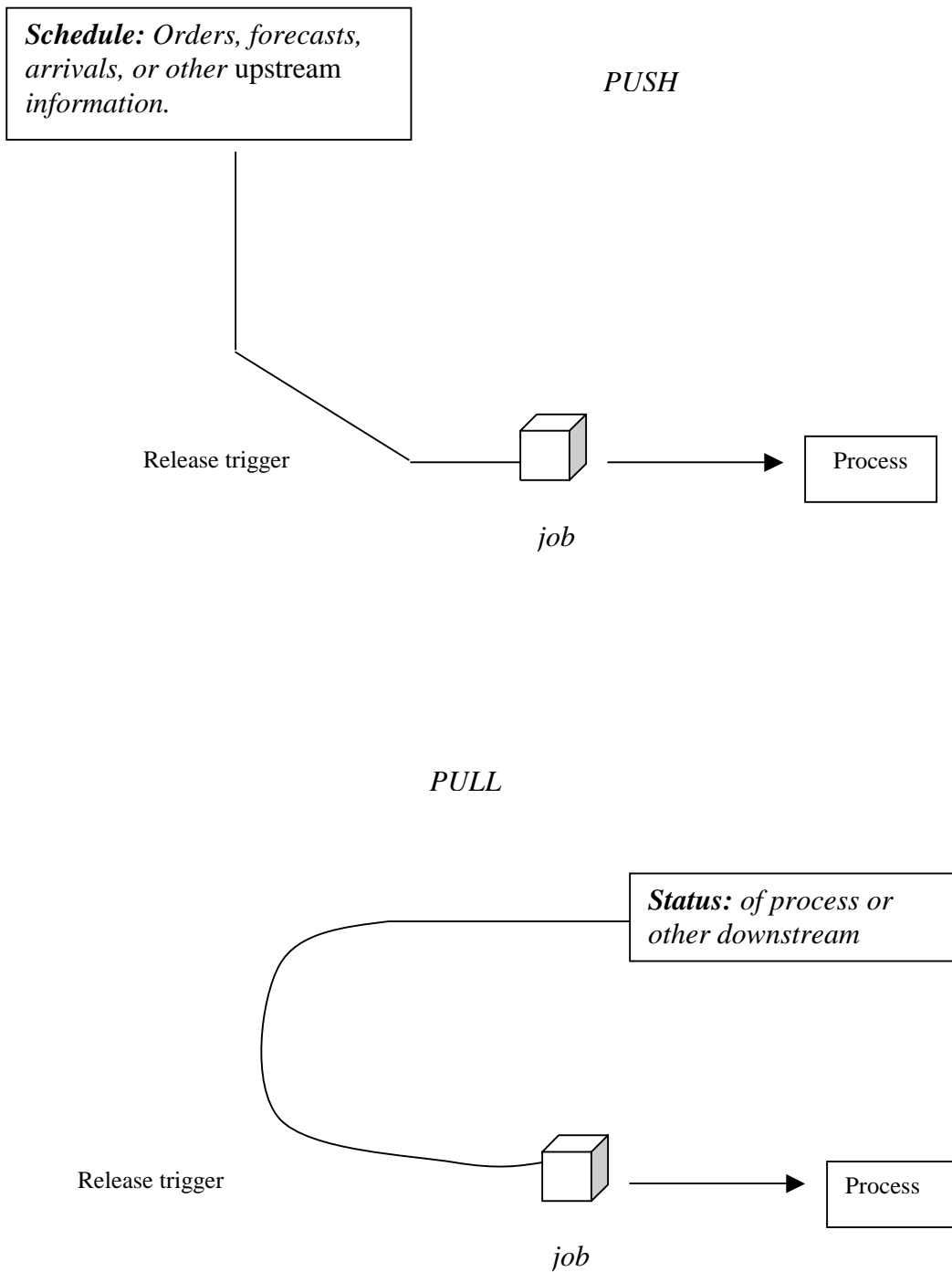
The father of JIT, Taiichi Ohno (Ohno 1988) used the term *pull* in a very general sense. The following words are quoted from his research:

*Manufacturers and workplaces can no longer base production on desktop planning alone and then distribute, or push, them onto the market. It has become a matter of course for customers, or users, each with a different value system, to stand in the frontline of the marketplace and, so to speak, pull the goods they need, in the amounts and at the time they need them.*

Wallence J. Hopp and Mark L. Spearman in their book “Factory Physics” [2<sup>nd</sup> ed], define push and pull system as follows:

**Definition:** *A **push** system schedules the release of work based on demand, while a **pull** system authorizes the release of work based on the system status.*

Figure 1.1 contrasts these two methodologies:



**Fig 1.1:** Release triggers of push and pull production systems (As mentioned in Chapter 10 of *Factory Physics*)

A push system does not accept feedback from the system factors themselves. It releases a job in the production process when called for by an external and predetermined schedule. The time of release is not modified in accordance with what happens internally in the system. Thus, if a machine or a processing station fails and is down for some time, parts start to pile up till the machine is repaired. However, if that station had sent a signal, which was interpreted by the push system, the schedule could be modified to avoid or at least reduce the blocking. On the contrary a pull system releases a job onto the shop floor only when a signal generated by the line status calls for it.

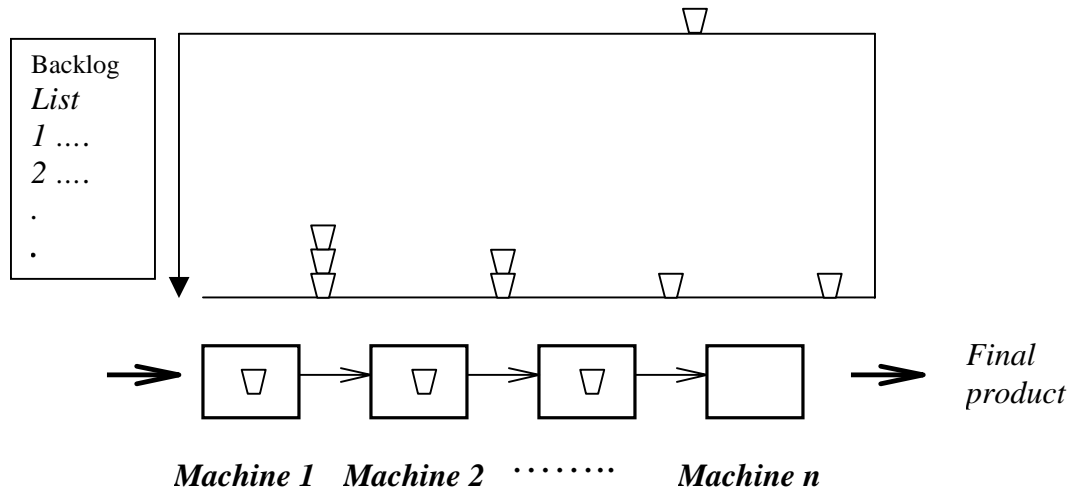
Another useful way to think about the distinction between push and pull systems is that push systems are inherently *make-to-order* while pull systems are *make-to-stock*. That is, the schedule that drives a push system is driven by orders (or forecasts), but not by the system status. The signals that authorize releases in a pull system are voids in a stock level somewhere in the system. Viewed in this sense, the base stock model, which triggers orders when the stock drops below a specified level, is a pull system. A MRP system, which releases orders into the system according to a schedule based on customer orders, is a push system. Most real-world systems such as the CONWIP system (which is a push-pull system) are of course a mixture of these pure systems. Hybrid systems such as the push-pull system gives better control under certain conditions.

One such idea introduced by Spearman et al (1990), was to set a predetermined WIP level for a pull system and was called the CONWIP (Constant Work In Progress) system. This research deals with issues related to this type of control system.

All the differences explained in the preceding paragraph can thus be summarized into the following statement:

*Push systems control throughput and observe WIP whereas Pull systems control WIP and observe throughput.*

The operation of a CONWIP system is depicted in the Figure 1.2



**Fig. 1.2:** An illustration of a CONWIP system

In any production system, there are a finite number of products to be manufactured on the line (in reality, this could be a very large number). If the jobs, which were already sequenced or which entered the system are deleted from the job set of all parts planned to be manufactured, the jobs left constitute a backlog list. One of the aims of this research is to sequence the jobs on this list. Any CONWIP system can be considered as consisting of a finite number of containers shown schematically in the above figure. The containers, shown schematically, are of a fixed capacity (lot size or transfer size; if different from lot size) or as assumed in the present research, of size one. The first machine has an ever-available reservoir of jobs, which are loaded on machine 1 as soon as its status is idle. A constant number of containers circulate in the system, thus maintaining a constant WIP. After leaving the system at machine  $m$ , the containers reenter the system at machine 1 with a new job or set of jobs in them. The intermediate storage buffer capacities limit the number of such containers (or jobs), which can be stored between machines.

### 1.3 *Effect of Variability*

In a given production shop, all production related parameters need not be deterministic and constant. For example, the processing time of any machine in the production line may vary due to many extraneous and internal factors. If the line throughput is optimized based on the deterministic values, a small change in the values will disrupt the plan and more than often, this is the case in reality. Another common example of variability is the demand of products. Static scheduling rules based on forecasted demand, lose their effectiveness when there is a deviation of the actual demand from the forecasted value. If WIP levels are high, parts must be released to the plant floor well in advance of their due dates. Due to the fact that customer orders become less certain as planning horizon is increased, the inherent variability of far fetched future demands, has an influence on the system output reliability and thus degrades the performance of the system if the WIP level is still maintained the same.

Also, high WIP levels impede priority or scheduling changes, as parts may have to be moved out of the line to make way for a high-priority part. A pull system releases work as late as possible based on the systems internal inputs, which ensures that the releases are based on firm customer orders to the greatest possible extent.

The key to keeping customer service high is a predictable flow through the line. In other words, we need to have **low cycle variability**. If the cycle time variability is low, then we know with a high degree of precision how long it will take a job to get through the plant. This allows us to quote more accurate due dates or shipping dates. Low cycle time variability also helps in quoting shorter lead times to customers. If cycle time is 10 days plus or minus 5 days, then we will have to quote a 15-day lead-time to ensure high quality of service. On the other hand, if cycle time is 10 days plus or minus 2 days, then a quote of 12 days will suffice for the same level of service.

According to Spearman and Hopp (“Factory Physics [2<sup>nd</sup> ed]”), kanban achieves less variable cycle times than does a pure push system. Since cycle time increases with WIP level (by Little’s Law), and kanban prevents WIP explosions, it also prevents cycle time explosions. However, note that the reason for this, again, is the WIP cap – not the pulling at each station.



Hence, any system that caps WIP prevents high explosions in WIP, and hence cycle time, that can occur in a pure push system.

For more explanation about the effect of variability on system behavior, refer to section 1.4. In order to create a model robust enough to allow limited variability, this research assumes stochastic processing times at all the machines. The algorithm proposed will be tested for such a non-deterministic system, to appraise the quality of the solution. The first part of the problem considered in this research, is finding the value of the optimal WIP level and the proposed solution algorithm uses the stochasticity to its benefit.

#### **1.4 Importance of Bottleneck: A Comparison of CONWIP with kanban**

In Eliyahu Goldratt's book "The Goal: A Process of Ongoing Improvement (1984)", the importance of a bottleneck in a factory is described through an analogy to a troop of boy scouts out for a march. One of the scouts, who is carrying an extra-heavy backpack, walks more slowly than the rest, so a gap keeps opening between him and the scouts in front. This is then connected to how inventory masses up in front of a slow machine in the factory.

But this is less than half the story. In a column of marching soldiers, the problem is not a slow marcher falling behind. Each soldier carries the same weight, so the line is balanced, and there is no pronounced bottleneck. The problem is variability amplification: If the first soldier for some reason speeds up a little bit, the second soldier will see a gap open in front of him, and take this as a signal to speed up, as well. But, he will have to speed up more than the first soldier did, in order to catch up with him. When he has caught up, he then needs to slow down again to avoid bumping into the one in front.

Now, the third soldier sees a gap opening up even faster than the second one did, so he has to speed up by even more, and has to slow down more abruptly when he has closed the gap. This way, the small change in speed amplifies down the line like a whiplash, and the poor guy at the end of the line will alternate between running flat out and marching in place.

This is what occurs in a kanban line. The last machine in the line tries to track the demand process, but adds some noise to it due to process variability. The second last machine tries to track the input process of the last machine, but adds some more noise. This amplifies the noise upstream, so the first machine in the line will alternate between working at capacity and waiting for something to be taken out of its output buffer. To get rid of the problem, one has to eliminate all process variability, such as machine failures and operation time variability. This can be time-consuming and expensive.

How do soldiers counteract this age-old problem? Very simply as follows: If the soldiers are recruits, they get the attention of a very loud drill sergeant that yells out the cadence. More seasoned soldiers will be singing a marching song as they go along, and any infantry outfit has a

large supply of these songs. Both of these techniques have the effect of distributing the proper cadence to every soldier in the line, simultaneously.

This is what the CONWIP control does. It passes the demand information, without any noise, to the first machine on the line. All downstream machines know that any part arriving in their input buffer can be worked on, so they hear the signal, too.

But marching soldiers do not close their eyes and march blindly. Even if they receive the proper cadence, they will still be watching the distance to the marcher in front. If the gap widens, they will take longer strides, and if it narrows, they will shorten their steps. This way, the marchers act on two types of information at once: The global information flow that determines the overall speed, and the local information that is used for minor adjustments.

This is also the way the hybrid policy works: The CONWIP control gives a global information flow (like the drill sergeant), and the kanban control gives a local flow of information (like watching the distance to the guy in front). In the hybrid policy, the global information flow from the demand process is supplemented by the local information from the buffer levels. This attains the advantages of CONWIP control, while using the strengths of kanban control to cancel its disadvantages.

## 1.5 Robustness of a control system: CONWIP vs. Push

So far, we have talked at length about analogies and differences between CONWIP and kanban methodologies, both being variations of Pure Pull methodology. In this section, we discuss a very important notation namely the “Robustness” of a system, with respect to CONWIP and Push methodologies.

Wallace J. Hopp and Mark L. Spearman in their book “Factory Physics[2<sup>nd</sup> ed]” define the law of robustness of a system as follows:

The profit function:

$$\textit{Profit} = p\textit{TH} - hw \tag{1.5.1}$$

where,

$p$  is the marginal profit per job

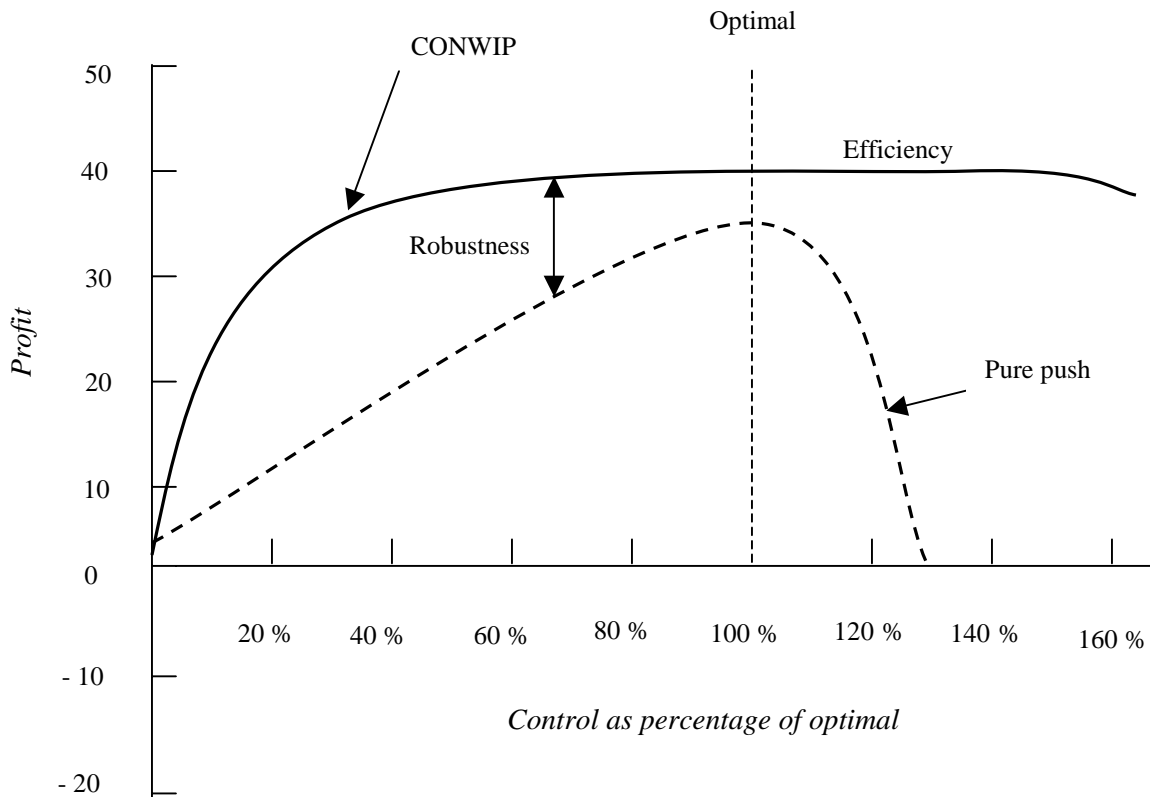
$\textit{TH}$  is the throughput rate

$h$  is a cost for each unit of WIP (this includes the costs for increased cycle time, decreased quality, etc.)

$w$  is the average WIP level

In a CONWIP system, throughput will be a function of WIP, that is,  $\textit{TH}(w)$ , and we will choose the value of  $w$  to maximize profit. In the push system, average WIP is a function of release rate  $w'(\textit{TH})$ , and we will choose the value of  $\textit{TH}$  that maximizes profit.

Now, the CONWIP robustness law is concerned with what happens if  $w$  is chosen at a sub optimal level in the CONWIP system or  $\textit{TH}$  is chosen at a sub optimal level in the push system. Since WIP and throughput are measured in different units, we can measure the sub optimality in terms of percentage error. The curves of Profit functions vs. percentage deviation from optimal are given below:



**Fig 1.3:** Relative robustness of CONWIP and pure push systems (As mentioned in “Factory Physics [2<sup>nd</sup> ed]

It is a normal human tendency to try to drive the throughput as high as possible. The above graph shows that this would have disastrous effects on profit of the Pure Push system as the curve is quite steep around 100 % throughput level. The CONWIP system, on the other hand, is controlled by setting the easily observable parameters of WIP level. This combined with the flatness of the profit curve in the vicinity of the optimum, means that achieving a profit close to the optimum level will be much easier than in the push system. We use this fact to our advantage by using an approximate algorithm to address the issues of optimal CONWIP level. This methodology is time tested and widely accepted due to its underlying principle.

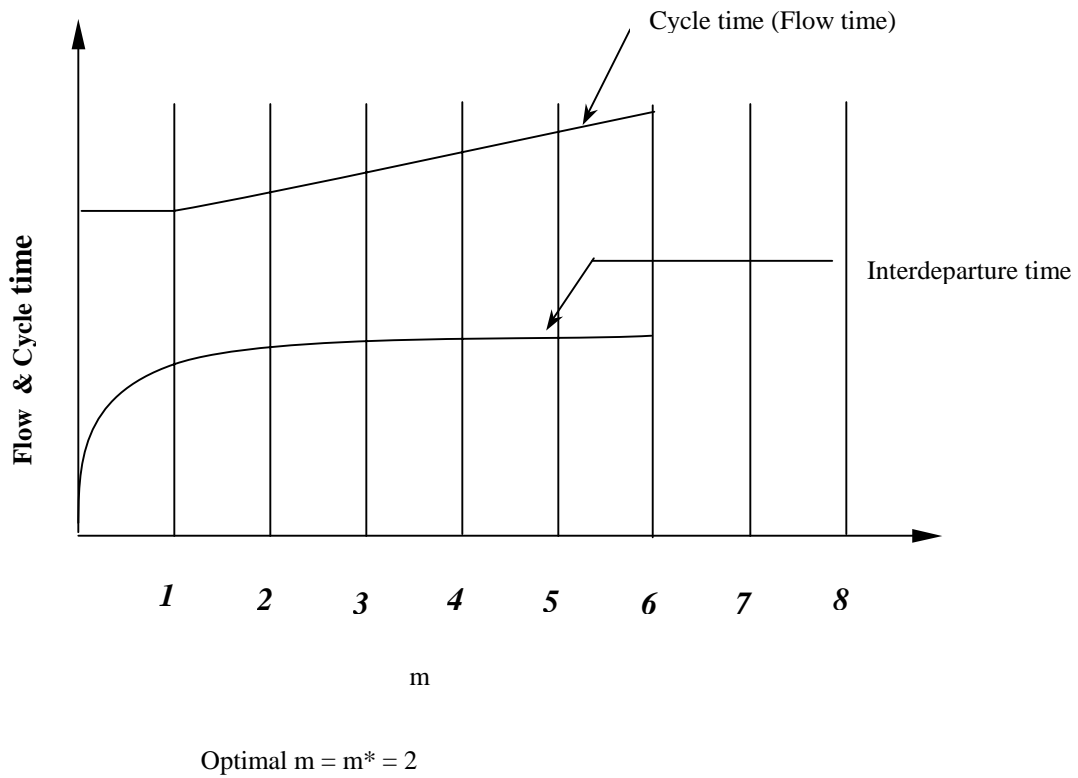
## **1.6 People Issues**

Finally, we complete our comparison of CONWIP and kanban with two people-related observations. One should note that a kanban system is a pure pull system for all workstations. However, a CONWIP system is a pull system only for the first station, but not for the stations that follow. All these stations are allowed to process products as soon as all the resources needed are available. In a pull system, operators must wait for a need for the product somewhere down the line and when they do receive this signal, they are expected to produce as soon as possible to replenish the starvation. This induces an operator's stress called the "pacing" stress. In a CONWIP system, this stress will only be induced at the first station, whereas the stations down stream do not experience this stress. This is known to be a main reason of operator dissatisfaction.

The second most discussed issue in the literature is that pulling at each station, in a kanban line, may foster a closer relationship between operators of adjacent workstations. Since operators must pull needed parts in a kanban system, they will communicate with the operators of the upstream machines. This provides an opportunity to check parts for quality problems and to identify and discuss any problems pertaining to the production rate.

## **1.7 Trade-off between high line utilization and WIP**

One always strives for a high service level of predictability. Thus, it is a normal human tendency to carry a high WIP in the system, as it is the easiest way to achieve that goal. However, WIP is locked capital and should be avoided in any case. In other words, only that amount of WIP should be carried, which affects the system performance such as throughput. Thus, one of the most important parameter, which should be carefully chosen, is the WIP level. This is more relevant for a CONWIP system as the whole advantage of this system depends upon the carefully chosen WIP level. If the WIP level is chosen very high, the flow time is *unnecessarily* high and does very less to aid in maintaining a high throughput. However, if the WIP is very low, the throughput is directly affected in a detrimental fashion and the system cannot produce to its capacity due to the dearth of products in the system. A high WIP increases the mean and variance of flow time thereby resulting in long lead times, poor forecasting and late feedback. Thus, generally, we want as small a WIP as possible that allows us to approach the maximum throughput of the system. The results of a simulation study, performed to study the effect of WIP in a CONWIP system on interdeparture time and cycle time (Flow time), are shown in Figure 1.4:



**Fig 1.4:** Variation of Flow time and Cycle time with WIP level  $m$

The plot clearly shows that after the optimal  $m$  (WIP level), the interdeparture time remains unchanged, while the Cycle time (Flow time) increases from its value at optimal.



## **1.8 Concluding Remarks**

This chapter throws light on the relevant issues related to control of a CONWIP system. Even though the detailed knowledge of all these factors is not required for the understanding of this research, they provide the reader with a good base to appreciate the topic. Some of these issues such as the Robustness of a CONWIP system to WIP level are used to our benefit in the methodology proposed. This research is the first to incorporate the condition of variability (stochastic processing times) in a limited intermediate storage CONWIP line with blocking. Finally, issues such as Peoples issues, though not in the domain of this problem have to be considered with as much seriousness as the factors considered in this research.

## Chapter 2: Problem Background and Literature Review

As mentioned in Chapter 1, the problem that we address is two-pronged. Both the WIP level  $M$ , and the sequence in which the jobs are released into the system impact its performance. This is illustrated in the Table 2.1:

**Table (2.1):** *Variation of flow time over  $M$  and sequence*

		M = 2		M = 3		M* = 4		M = 5	
ABC	Job #	FT	IT	FT	IT	<b>FT</b>	<b>IT</b>	FT	IT
		A	14	17	14	14	<b>20</b>	<b>13</b>	23
	B	12	17	12	14	<b>16</b>	<b>13</b>	21	13
	C	7	17	12	14	<b>16</b>	<b>13</b>	19	13
	<b>Total</b>	<b>33</b>		<b>38</b>		<b>52</b>		<b>63</b>	
CBA	C	14	17	14	14	<b>20</b>	<b>13</b>	23	13
	B	9	17	13	14	<b>17</b>	<b>13</b>	19	13
	A	10	17	11	14	<b>15</b>	<b>13</b>	20	13
		33		38		<b>52</b>		62	

The table shows the variation of flowtime over both  $M$  and sequence. For each sequence, the values of Flow time and Interdeparture time is noted for each job for every level of WIP ( $M$ ). The last row is the total of these times. Thus, the optimal sequence, which results in the least flow time, could be anywhere in this matrix. In this case however, the  $M^*$  is the same for both

the sequences (since value of flowtime is the same and equals 52), which might not always be the case.

Thus, the problem that we address pertains to the determination of the optimal values of both  $M$  and the sequence in which to release the jobs into the system, while considering finite buffer capacity in front of each station except the first one on the line. Thus, any feasible solution can be stated mathematically as flowtime or cycletime (henceforth referred to as FT or CT respectively) =  $f(M, \text{seq.})$  or throughput.

This chapter defines the problem statement precisely and describes the scope of the research. It also lists the premises on which the research is based. As mentioned in the earlier chapter, this research assumes stochastic variables to represent processing times of the jobs on the machines. Thus, an important consideration in the design of the system is the proper choice of a probability distribution function (pdf). Many researchers use Exponential distribution due to its ease of use and simplicity to understand and also because only one parameter is required to define the distribution. However, an Erlang distribution with a low mean added to a high constant is more suited to manual lines as described in this chapter. An equally important issue in any research is to define the performance measures, which would be used to determine the quality of the proposed solution. A couple of performance measures are defined in this chapter.

Finally a summary of an exhaustive literature review is presented, which is divided into four parts from the basic study of a CONWIP system and its advantages to the use of queuing theory, and finally a brief description of a few in the multitude of flow shop-sequencing heuristics as used to address the CONWIP sequencing issues.

## 2.1 Problem definition

### 2.1.1 Problem Statement

*This research attempts to find optimal solutions to the following two questions, which are cardinal in the design and control of a CONWIP flow line production system:*

- *What constant work in progress (WIP) should be maintained?*
- *What is the optimal sequencing policy at the first machine, at this optimal WIP level?*

*Thus the main aim is to find the optimal combination of CONWIP level  $M$  and input sequence, so as to minimize the FT or CT and to maximize the throughput.* The above two questions have to be answered for a general case of a flow line, with finite intermediate buffers (leading to the occurrence of “Blocking”), and stochastic processing times.

### 2.1.2 Scope and Assumptions

We have mentioned in the earlier chapters that the objectives that we have coined for this research pertain to both the WIP level selected, as well as the sequence in which the jobs are to be released into the system. Greco and Sarin (1996) studied this issue for a CONWIP system, with deterministic processing times and unlimited buffer capacities between stations. A heuristic to sequence the jobs in the backlog list was proposed. Their heuristic was a modification of the existing procedure employed in cases of static flow shops, to minimize the makespan.

This research attempts to determine the following:

- The optimal value (or very close to optimal value as the curve in Chapter 1 indicates that CONWIP is relatively insensitive to minor deviations from the optimal value) of  $m$ .
- A lower bound on the value of  $M^*$  (optimal  $m$ ; to help achieve the above).
- The optimal sequence of the jobs in the Backorder list in order to minimize both  $M$  and makespan (or Flow or Cycle time).

This research is based on the following assumptions:

- 1) Production line type: Mixed models (single item flow, no setup between models). Serial production line.
- 2) Size of models: Large (3 job families or more)
- 3) Arrivals: Static (all items are available at  $t = 0$ )
- 4) Number of workstations: Large (3 or more)
- 5) Type of workstation: No parallel machines; serial flow line
- 6) Job release policy: Cyclic; Pull for the first machine and push for all the other machines in the line.
- 7) Processing times: Stochastic; Exponentially distributed
- 8) Homogeneity: Non-homogeneous (for each model)
- 9) Work station reliability: 100 % Reliable
- 10) Buffer size: Finite and constant.
- 11) Production control methodology: CONWIP with  $n$  jobs in the system. (For initial study the following system types were tried:
  - i. Open system with infinite buffer capacity
  - ii. Buffers with a finite capacity
  - iii. CONWIP system.(A comparison of results corroborated the previously published results of similar experiments, indicating superior control of CONWIP system over other methods. These experiments are however irrelevant to this research and hence are not included.)
- 12) Buffers queuing discipline: FCFS
- 13) Due-Dates: None (Made to Stock)
- 16) System state: All parameters are measured after the system has reached its steady state.

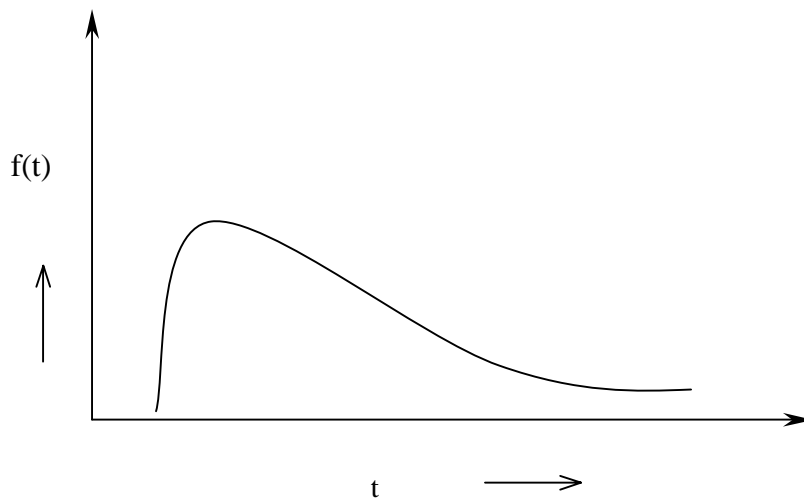
## **2.2 Outline of thesis**

The outline of the thesis is as follows. Chapter 1 provides the required background and discusses various issues related to the CONWIP system. This chapter brings out the relevance of this type of control in a modern day manufacturing environment. Chapter 2 gives a brief outline of the problem under study, and also states succinctly, the assumptions made. This is followed by a detailed and exhaustive literature review, which is divided into four relevant. Chapter 3 presents relevant analysis of the CONWIP system. A MIP formulation is presented, which generates additional insights of the problem. The iterative method to determine the value of  $M^*$  is presented in chapter 4. The computer code is included in Appendix A, and the results of experimentation are mentioned in Appendix B. Chapter 5 addresses the issue of finding the best permutational sequence at the input. Appendix C and D present the computer code used to test the algorithm and the results of the code respectively. Finally, chapter 6 presents conclusions and recommendations for further research.

### 2.3 Stochastic distribution – Why exponential??

It is very important to choose a distribution, which captures the actual process as close as possible. The deviation of the actual sample from the real system should be within certain tight limits of the corresponding values generated from the fitted distribution.

The stochastic models studied in the literature usually assume very special processing time distributions. The exponential distribution, for example, is a distribution that has been studied at length. In reality, processing times usually are not distributed exponentially. Studies in the past have shown that the density functions in practice are as shown in Figure 2.1:



**Fig 2.1:** The Density function of job processing time observed in practice

One can think of this curve to be a mixture of an Erlang ( $k$ ) distribution and a constant value. That the processing time may have this distribution is highly plausible especially for manual operations. One can imagine that there is a certain minimum time that is needed to perform the task to be done. Even the fastest worker cannot finish the task before this constant portion of the time. The Erlang part of the time captures the variability involved with manual operation. For automated operations, however, the time is more or less deterministic, with little or no variation. However, in this research, we choose the exponential distribution, as it is the most commonly used distribution for many applications. Also, it is the most researched and is

easy to use, which simplifies the solution especially of difficult NP-hard problems such as this one.

Another important issue with stochastic data is the correlation involved. Successive processing times on the same machine may tend to be positively correlated in practice. However, in this research, we assume the processing time values of the jobs to be independent. When the down times of the machines are considered, they can be added to the processing times. Thereby resulting in an increase in the processing time variability.



## 2.4 Performance measures

Operational parameters need to be estimated to enable us to predict job completion times in order to aid the management in the optimization of monetary criteria, customer service and system design. In this research, we estimate two low level operational parameters:

- Maximum throughput/ Minimum interdeparture time
- Minimum WIP/ Minimum flow time

The mean interdeparture time is the inverse of the mean throughput rate and therefore may be used in its place. Flow time (cycle time) is another important parameter that characterizes the system performance and is thus of interest to us. There is always a trade-off between these two parameters when the system is sub-optimal, after which the flow time increases without any decrease in throughput with increase in WIP. This point is the optimal CONWIP level. This trade-off is clearer from the inverse correlation between these two parameters as given by Little's law applied to the whole line.

$$M = \frac{(\text{FT or CT})}{\text{Interdeparture time}} \quad (2.4.1)$$

## 2.5 Literature Review

CONWIP control strives to maintain a constant work-in-progress (Spearman et al. 1990). When the present WIP level is reached, no new job is allowed to enter the system until a job leaves. Although the CONWIP system of control is not new to industry, not much research has been reported in this context, which addresses the issues at hand.

The literature review is divided into four categories:

- Study of the CONWIP system and the advantages of using a CONWIP system over other existing systems with a view to build a good base, introduce common terms in this context and coin some common premises for this research.
- Shed some light on a few approaches, presented in past research, on this topic such as the use of simulation to analyze the behavior of the system due to the difficulty in obtaining analytical results, and relevant methodologies used in related research.
- Discuss the use of Queuing theory models and Mean Value Analysis to similar types of problems, and
- Discuss the importance of a sequencing policy at the optimal (or good) value of the CONWIP level.

### 2.5.1 Study of CONWIP systems and terminologies used

CONWIP, as kanban, operates as a pull system, i.e. the start of a batch is triggered by the completion of another batch. In order to achieve this method of control, CONWIP utilizes a closed production system approach. Another way to look at a CONWIP system, as mentioned in chapter 1, is to consider a fixed number of containers (or cards) that traverse a circuit, which constitutes the entire production line. Each container is then sent back to the beginning of the line where it waits in queue to receive another batch of items. During each container's cycle, all items in the container are of the same type. The amount of material loaded in the container, is equal to a predetermined transfer lot size.

It was shown by Spearman et al. (1990) for a single production line, that in the cases of variable processing times and/or unreliable machines, the kanban policy tends to build up WIP upstream of the bottleneck machine. However, by maintaining higher (but still, constant) WIP level, the system results in higher bottleneck utilization and as a consequence, a higher throughput.

There are several factors, which motivate the use of CONWIP over kanban. Some of these as alluded to in Spearman's paper are:

- Large product mixture (which affects the repetitiveness of the production system)
- Long setup times
- Process variability
- Unbalanced workload

Thus, it is not purely due to high variability that CONWIP is preferred over kanban. In fact, CONWIP addresses all these issues by relaxing one of the cardinal constraints for kanban system of control. Kanban aims at keeping a minimum WIP level, and where possible, zero. However a CONWIP system allows a low constant inventory, which minimizes the flow time, while maintaining a high throughput. Dar-El, Herer and Masin (1998), list the following advantages of CONWIP over kanban system:

- It is very robust regarding changes in the production environment.
- It easily handles the introduction of new products and changes in the product mix
- It copes with flow shop operations with large set-up times and permits a large product mix
- CONWIP systems yield larger throughput than kanban systems for the same number of containers (maximum inventory), even for systems with yield losses.

Spearman et al (1990) has shown, that this inherent difference between CONWIP and kanban, of maintaining a constant level of inventory, has resulted in rendering the CONWIP model more robust in terms of system performance due to process variability. Kanban assumes negligible setup times and this condition is more often untrue than true in actual manufacturing setups. CONWIP ensures faster response to market demand, because some work has already been carried out and incoming orders need not necessarily be processed from the very beginning

(unlike Kanban), thus allowing setup times not to be negligible. In fact, this is the very reason that CONWIP is more suited to handle a large product mix than does kanban.

Research carried out by Bonvik, C.E. Couch and Gershwin (1997) at MIT compares the performances of different production line mechanisms. The performance of a four-machine tandem production line was studied under different control policies such as kanban, Minimal blocking, CONWIP and hybrid kanban-CONWIP. The main performance measures considered were service level and amount of WIP. The research basically used simulation to test various strategies and compared their results statistically. The CONWIP and hybrid policies gave better response to changes in demand rate.

Dar-El, Herer, Masin (1999) performed a study on CONWIP closed production control system and developed estimates for the level of WIP for four important performance measures. The model assumes finite mean and variances of processing time distributions. Detail insights were generated from the analytical model developed, into how a CONWIP production system operates.

### *2.5.2 Use of simulation in related research*

Another interesting method employed has been the use of simulation along with scheduling rules to determine the release policy for a CONWIP system. The problem alluded to in the paper by Graves and Milne, addresses the issue of when to authorize work centers to produce or, conversely, authorize to remain idle. This problem is a little different from the problem on hand (of this research), but is nevertheless important to know, as it is an ingenious methodology to deal with the issue of order release, rather than ordering of jobs at an individual work center. The manufacturing set up considered by Grave and Milne bears similarity to this research from the fact that the processing times at the work centers are probabilistic in nature. This heuristic releases jobs only if their estimated times are sufficiently small. Waiting times are estimated using simulation. Finally, simulation is also used to examine the performance of the heuristic.

Due to the complexity of the analytical methods for multistage production lines, the analytical results are limited to two – or three – station lines under some restrictive assumptions. Thus, many approximate decomposition solutions procedures of the performance measures have received considerable attention. The study carried out by Tan and Yeralan (1997), uses an iterative decomposition scheme, which does not alter the station parameters such as the breakdown, repair and service rates. At every iteration, the input and the output processes of the station model are matched to the most recent solutions of the adjacent stations. The processing and repair times are assumed to be deterministic. Due to the fact that the single station models have closed form solutions with respect to the buffer capacity, the computational effort associated with the decomposition method is independent of the buffer capacities. The issue of convergence, typically associated with such approximate iterative methods, is also addressed.

### *2.5.3 Use of Queuing theory models and Mean Value Analysis to similar problems*

Queuing network theory has rapidly progressed since the fifties. More general classes of networks turned out to have a product-form solution. The generalizations include multiple customer classes or product types, queuing policies (FIFO, LIFO etc.) and general distributions of service time distributions. The pioneering work in this area was presented by Jackson (1957). Jackson presented a situation with  $n$  ‘departments’, each being a system with a finite capacity and a Poisson input distribution and exponential processing times. He proved in this study that if the mean arrival rates at the various departments are properly defined, then the result is a steady-state distribution, of the waiting-line lengths (at the departments) that are independent and are exactly like those of the ‘ordinary’ multiserver systems that they resemble.

Dubois D (1982) presented a mathematical model of a flexible manufacturing system with limited in-process inventory. Unlike earlier approaches, consisting of closed queuing networks, Dubois proposed an open queuing network with limited amount of inprocess customers. He mentions that for such queuing networks, the product form of state probabilities is valid and the normalization constant can be easily obtained.

Chandy M. K. and Sauer C. H. (1978) explain the trade of between the cost of building and solving a model to solve the problem of Optimization of Computing Systems and the credibility of the model. He shows that queuing theory can be successfully be used to address this problem resulting in a sufficiently credible approximate solution.

Though Mean Value Analysis (MVA) existed before it was first used to address issues in a stochastic manufacturing problem, Riser (1980) was first to find solutions to queuing networks with product-form solutions in terms of mean queue size, mean waiting time and throughput. This new analysis lead to simpler algorithms, which have better numerical behavior than the ones previously proposed. Riser pointed out that little attention was paid to a strange property of the product-form solution, namely, the fact that from the many parameters, necessary to specify a network, much fewer entered into its solution. His research was based on the following two intuitively appealing principles:

- A customer upon arrival into the system sees, in the long term, the same closed system as the one with himself removed (one less customer).
- Little's law applies to both the entire system as well as each queue individually.

For the single class system, they define the following variables:

$K$	Number of customers
$N$	Number of queues
$\zeta_i$	Mean service time of queue $i$
$t_i$	Mean waiting time of queue $i$ (including service)
$n_i$	Mean queue size of queue $i$ (including customer in service)
$\lambda(K)$	Throughput of the chain, with $K$ customers in the system

The first principle stated above can be restated in mathematical terms as follows: The probability to see state  $k$  upon customer arrival in  $S(k)$  is the same as the long term equilibrium probability of  $k$  in  $S(k-1)$ .

Or

$$t_i(k) = \tau_i + \tau_i n_i(k-1) \quad (2.5.3-1)$$

The second principle, stated above can be expressed in mathematical form as follows:

$$\lambda(k) = k / \sum_{i=1}^N t_{i(k)} \quad (2.5.3-2)$$

$$\text{and } n_i(k) = \lambda(k) * t_{i(k)} \quad (2.5.3-3)$$

Though Riser proposed the MVA algorithm, he did so intuitively, and did not prove its correctness mathematically. Due to its simplicity, it was computationally much less demanding than a convolution algorithm, and also avoided the problems of floating point overflow/underflow, which were inherent with earlier algorithms.

Hildebrant (1980) used Riser's MVA methodology to address the issue of scheduling flexible manufacturing systems. The algorithm used was essentially the same as proposed by Riser, but was extended to more than one server at each processing station. He showed that queuing methods yield good solutions for the purpose of balancing work among resources. It is not necessary that they predict system performance exactly. Relative accuracy seems sufficient. He also proved that this algorithm was much more efficient computationally, when compared to the non-linear programming model he developed as a part of his research.

Another heuristic method based on Mean Value Analysis for flexible manufacturing systems performance measures was suggested by Cavaille and Dubois (1982), just after Riser's work was published. Researchers started using queuing theory as a cheap and quick but not very accurate method as an alternate for expensive simulation techniques. An interesting contribution of this research is the critical discussion about whether MVA can be used to solve FMS related problems.

Suri (1982) studied the robustness of analytical-models for non-classical discrete event systems. An important point, which the author made, was about the use of the "Homogeneous Service Times" (HST) assumption. HST has been criticized as being restrictive since, for analysis of non-classical discrete event systems. His results proved that performance measures

were in fact less sensitive to such violations. It thus further explained the robustness of queuing models.

As mentioned, Hildebrant, (1980) introduced in the MVA algorithm, a variable  $\varepsilon$  as a correction term to improve its accuracy. He, however, does not mention the reason behind the expression used to get the value of this correction term. Schweitzer and Seidmann (1986) did research to get a more accurate value of this term. The approximate MVA treatment by Riser and Lavenberg considers mean queue lengths as seen by an arriving customer, in a closed network with product form solution, equal to the ergodic mean queue lengths in the same closed network from which the arriving customers has been removed. Their research describes properties characterizing the original approximation, which the authors hope to result in providing useful guidelines, in order to create better algorithms.



#### 2.5.4 Sequencing policy at the first machine

Sequencing and scheduling is concerned with the *optimal allocation of resources to activities over a period of time, which could be infinite or finite*. Of obvious practical importance, it has been the subject of extensive research since the early 1950's and an impressive amount of literature has been created. Thus, any discussion of the available material is bound to be selective. In the following subsections we cover relevant research done since this problem was first studied. The terminology 'flow shop' is used to describe a serial production system in which all jobs flow along the same route. A more general case would however be when some jobs are not processed on some machines. In other words these jobs simply flow through the machines, on which they are not processed at, but without having to spend any time on them. To generalize the situation, we can assume that all the jobs have to flow through all the machines but have a zero processing time at the machines, which are not in the routing matrix. The static flow shop-sequencing problem denotes the problem of determining the best sequence of jobs on each machine in the flow shop. The class of shops, in which all the jobs have the same sequence on all the machines, is called 'Permutation' flow shop. Thus, in this case, the problem is then be that of sequencing the jobs only on the first machine, due to the addition of an extra constraint of same job sequence at each machine. Ironically this problem is a little harder to address than the more general case, even though this might seem as a small part (sub problem) of the general case. Various objectives can be used to determine the quality of the sequence, but the majority of the research considers the minimization of makespan (i.e., the total completion time of the entire list of jobs) as the primary objective. Other objectives that can be found in the literature of flow shops are flow time related (e.g., minimal mean flow time), due-date related (e.g., minimal maximum lateness), and cost related (e.g., minimal total production cost).

##### 2.5.4.1 Analytical and Heuristic Approaches

The earliest analytical results for flow shop sequencing are due to Johnson (1954). He has shown that, in a two-machine flow shop, an optimal sequence can be constructed as follows:

Given the pair  $\{a_i, b_i\}$ , the processing time of each lot  $i$  on each machine  $j$  ( $j=1,2$ ), the optimal sequence on both the machines is the one in which the following condition is satisfied for any two lots  $i,k$ :

$$\text{if: } \min\{a_i, b_k\} \leq \min\{a_k, b_i\} \text{ then lot } i \text{ precedes lot } k \quad (2.5.4.1-1)$$

The above condition is known as Johnson's rule. One way to implement the rule is to first arrange the lots with  $a_i \leq b_i$  (i.e., for which machine 2 is dominant) in increasing order of  $a_i$  and then to arrange the remaining lots in decreasing order of  $b_i$ . In other words, the lots are sequenced according to a shortest processing time (SPT) rule on  $M_1$  and a longest processing time (LPT) rule on  $M_2$ .

An immediate result of the above solution is that the same sequence is utilized for both the first and the second machines, i.e., the sequence need not be modified. As mentioned earlier this family of sequences is termed a "permutation sequences". Although permutation sequences need not be optimal in general  $m$ -machine flow shops, it has become a tradition to assume identical processing sequence on the machines and to look for the best permutation sequence (Lawler et al, 1993). Conway et al (1967) have observed that, due to the inverse property of the makespan objective, there exists an optimal sequence which is identical on  $M_1$  and  $M_2$  as well as on  $M_{m-1}$  and  $M_m$ . Hence, for the three-machine flow shop, there must exist an optimal permutation sequence.

Unfortunately, Johnson's rule cannot be generalized to yield optimal sequences for flow shops with more than two machines. The three-machine flow shop problem is strongly NP-hard as shown by Gary, Johnson & Sethi (1976). Special problems of three machines in which one machine dominates the others can be reduced to equivalent two-machine problems. The property of dominance in general can be stated in mathematical terms as:

$$\min_{j \in \{1, \dots, n\}} p_{i,j} \geq \min_{j \in \{1, \dots, n\}} p_{i+1,j} \quad (2.5.4.1-2)$$

Consider a special case when one machine has very little load assigned to it, compared to the neighboring machines. Then, no jobs will ever wait in queue at this machine. That is, the moment a job is released from the machine immediately upstream, it will start its processing on this machine. Thus, intuitively, it appears that this machine serves as a buffer or a waiting room for the busier neighboring machine downstream. Thus heuristics are typically used to address this typically combinatorial problem. Next we present some well-known heuristics and methodologies for the flow shop problem.

Page (1961) was the first to introduce the concept of a slope index in prioritizing jobs. Palmer (1965) then adopted this idea and proposed the following slope index to be utilized for job sequencing in a general  $m$ -machine flow shop:

$$s_i = (m - 1) \cdot p_{i,m} + (m - 3) \cdot p_{i,m-1} + (m - 5) \cdot p_{i,m-2} + \dots - (m - 5) \cdot p_{i,3} - (m - 3) \cdot p_{i,2} - (m - 1) \cdot p_{i,1} \quad (2.5.4.1-3)$$

Then, a permutation schedule is constructed using the job ordering:

$$s_{[1]} \geq s_{[2]} \geq \dots \geq s_{[n]}$$

Where  $s_i$  is the slope index of job type  $i$ . The sequence is constructed in the decreasing order of the slope indices of the lots. In the spirit of Johnson's SPT(1)-LPT(2) rule, Palmer's rule gives priority to jobs having higher tendency to progress from short times to long times. However, for  $m=2$ , Palmer's rule does not coincide with Johnson's rule and therefore does not guarantee optimality even for the two-machine case.

Campbell et al (1970) proposed a simple heuristic extension of Johnson's rule to the  $m$ -machine flow shop problem. This extension is known in the literature as the CDS heuristic. The heuristic constructs at most  $(m-1)$  different sequences from which the best sequence is chosen. Each sequence corresponds to the application of Johnson's rule on a new two-machine problem that is derived from the original problem in the following manner:

The new processing time of job  $i$  on the first machine is:  $p'_{i1} = \sum_{j=1}^k p_{ij}$

The new processing time of job  $i$  on the second machine is:  $p'_{i2} = \sum_{j=1}^k p_{i,m-j+1}$

For each  $k = 1, 2, \dots, m-1$  a (possibly) different sequence is generated. The best makespan from the corresponding schedules is identified and the respective sequence is chosen. Notice that for  $m=2$ , the CDS heuristic reduces to Johnson's rule and is thus optimal.

Gupta (1971) investigated cases in which Johnson's rule is optimal for three-machine flow shops. He proposes the following slope index for the general  $m$ -machine flow shop based on his analysis:

$$s_i = \frac{e_i}{\min_{1 \leq k \leq m-1} \{p_{ik} + p_{i,k+1}\}} \quad (2.5.4.1-4)$$

where:

$$e_i = \begin{cases} 1 & \text{if } p_{i1} < p_{i,m} \\ -1 & \text{if } p_{i1} \geq p_{i,m} \end{cases}$$

Several other heuristics have also been proposed. A different type of approach in this region is Controlled Enumeration and few of its variations are discussed in the following section.

#### 2.5.4.2 Controlled Enumerative Approaches

A branch and bound procedure was developed by Ibnall and Schrage (1965) and a similar one independently by Lomnicki (1965). The job sequence was constructed in a forward direction in proceeding down the branching tree. For each node on the tree, a lower bound on the makespan associated with the completion of the corresponding partial sequence  $\sigma$  is obtained by considering the work remaining on each machine. A variety of extensions and refinements have been developed for the branch and bound algorithm. The major modifications were the refinement of bounds and use of job and machine based bounds, which identified the dominant jobs and machines.

Recent trends use the more efficient search techniques such as simulated annealing and genetic algorithms. Osman and Potts (1989) use the random search simulated annealing technique to solve the problem. Simulated Annealing is motivated by an algorithm from statistical thermodynamics developed by Metropolis et al. that simulates the cooling at material in a heat bath known as annealing. Approximately 30 years after this initial formulation, the Metropolis algorithm was modified and applied to discrete optimization problems by Kirkpatrick et al [76]. They showed that a discrete optimization algorithm could be created by randomly searching the neighborhood of the current solution for a new solution via a neighborhood function and computing the change in the objective function. An inferior solution is accepted with a certain probability calculated by the following formula:

$$P\{\text{Accepting solution } j \text{ as new solution from old solution } i\} = \begin{cases} \exp\left(\frac{-\Delta_{ij}}{t_k}\right), \Delta_{ij} > 0 \\ 1, \Delta_{ij} \leq 0 \end{cases}$$

where  $t_k$  is the temporary parameter at iteration  $k$ , such that  $t_k > 0$  for all  $k$  and  $\lim_{k \rightarrow \infty} t_k = 0$ . Later Das H., et al (1990) used this technique to schedule a serial multiproduct batch process under the assumption of permutation schedule. They propose four versions of the Simulated Annealing algorithm based on two move acceptance criteria, the Metropolis algorithm and the Glaube algorithm, and two annealing schedules, the exponential schedule and the Aarts and Van Laarhoven schedule.

Matsuo et al (1988) used Simulated Annealing to schedule a general job shop. They developed a controlled search Simulated Annealing method, which utilizes a good initial solution, relatively small search neighborhood, and acceptance probability of interior solution that are independent of the change in the objective function. This method was demonstrated to outperform the best heuristic methods available.

Another type of enumerative approach is the use of tabu search. Nowicki E. (1997) proposed a tabu search approach to minimize the make span of permutation flow shop with limited capacity buffers between machines. They showed that this algorithm was able to achieve excellent results for problems up to 200 jobs and 20 machines.

#### 2.5.4.3 Sequencing problem for added constraints such as CONWIP control, cyclic flow and finite intermediate buffers

Pinedo, Shenker and Wolf (1988) consider an assembly line with  $m$  stations in series having finite capacity buffers. Production requirements are defined for each class of jobs. They state that the length of the critical path around a cylinder formed by a single MPS (Minimal Part Sequence) equals the cycle time. A network flow model to seek additional insight into the problem by dualing the 'longest path around a cylinder' is proposed. They show that similarity between the longest path problems to determine the critical path to the minimum flow problem. A general maximum cut plane approach is then utilized to solve the dual problem. Finally, a 'profile fitting' heuristic is proposed to sequence the jobs within an MPS.

Selcuk, Panagiotis and Kiran (1992) address the non-preemptive flow shop-scheduling problem for makespan minimization. The problem is modeled as a min max problem. Using the property for a directed graph that the solution to a max min problem is always lower than that of a max min problem, a lower bound is suggested. A game theoretic interpretation is also suggested due to the type of objective function, which is analogous to that of a two person zero sum game. However, this approach is not effective due to the combinatorial nature of the problem of constructing a game theory problem. A branch and bound scheme is employed to address the problem.

Kamoun and Sriskandarajah (1993) study the complexity of scheduling jobs in a repetitive manufacturing system. Problems of finding schedules in a flow shop; open shop and job shop are studied where the sequence is cyclic (repetitive). Rock (1984) has proved that a  $F3|no\text{-}buffer, no\text{-}wait|C_t$  problem is strongly NP-hard. This is very relevant to this research as a general flow shop with limited buffers can be converted to an equivalent problem with no buffers. Due to the complexity of the acyclic case, they mention that the corresponding cyclic scheduling problem is also NP-hard in the strong sense.

Song and Lee (1998) develop a petri net model for a general cyclic shop with blocking. They propose a sequential buffer control policy that restricts a job to enter the input buffer of the next machine in a specified sequence. It is shown that the scheduling model of a cyclic shop with finite buffers can be transformed into a scheduling model of a cyclic shop with no buffer that can be modeled as a timed marked graph. They also present a mixed integer-programming model to obtain a deadlock-free schedule that minimizes the cycle time.

In one of the most recent studies, Logendran and Sriskandarajah (2000), attempt to minimize weighted tardiness in a two-machine scheduling problem with blocking and anticipatory setups. An algorithm based on tabu search technique is developed. A simple heuristic based on the earliest due date (EDD) rule is employed to identify as initial solution that can be used to trigger the application of tabu search-based algorithm to finally find the best solution.

# Chapter 3: Preliminaries: Concepts and Terminologies of CONWIP in view of the problem at hand

## 3.1 *Concepts*

### 3.1.1 One Product Case

The simplest case of a CONWIP system is the one that involves only one job family. When the processing times are deterministic, the system will pass through a transient stage and, then, finally reach a steady stage, when the various system parameters such as throughput, flow time etc reach a constant value. This value remains the same for as long as the system exists and the input remains constant. In the case of stochastic processing times, the same effect is observed though it is not so apparent. However, the trend of both the throughput and flow time plot is identical to that for the deterministic case. This fact was found by extensive simulation runs performed as a part of this research. The results of these experiments are included at the end of this Chapter.

Our first concern now is to find the optimal level of WIP. As discussed in Chapter 1, that the throughput increases as WIP level ( $M$ ) is increased but only to a certain point after which the throughput remains constant even though  $m$  is increased. For the values of  $M$ 's before this point, the flow time remains at a constant level, after which it increases at a linear rate. Any increase in flow time, after the throughput reaches steady state, causes unnecessary wait. We say that the



system is running at an optimal level when the minimal value of M ( $M^*$ ), gives optimal throughput.

The value of  $M^*$  for a single product case, where  $t_i$  is the processing time at station  $i$  and  $t_b$  is the processing time at the bottleneck station (highest  $t_i$ ) is as follows: Hopp & Spearman (1991)

$$M^* = \frac{\sum_{i=1}^N t_i}{t_b} \quad (3.1.1-1)$$

When  $m^*$  is a fractional value, it may be rounded to the next higher integer. For  $N$  machines, it is apparent from the formula that the value of  $m^*$  would range from 2 to  $N$ . For lower values of  $m^*$  the bottleneck station is more dominant than when the value of  $M^*$  is higher. Obviously, for a given value of  $N$ , the system is perfectly balanced.

Little's law holds true for a CONWIP system and is stated mathematically for this case as follows:

$$M = \frac{FT}{IT} \quad (3.1.1-2)$$

where FT is the flow time and

IT is the Interdeparture time.

It should be noted that this law is applicable only until  $M$  reaches  $M^*$ . After  $M^*$ , the value of CT remains constant. However, the value of FT increases and thus the above equation is no longer valid.

### 3.1.2 Multi-product case

In this case, there are a number of families, which need to be scheduled in the backlog list. A backlog list contains jobs at the start of station 1, which need to be sequentially fed into the system, taking into consideration the CONWIP level. If the product whose individual bottleneck (IBN) is at a machine somewhere down the line is first scheduled and is then followed by a product whose IBN is at a machine different from the IBN of the first job, then we say that the bottleneck has “shifted”. This property can be utilized to our advantage by scheduling products such that the proceeding job gives the IBN of the earlier job enough time to recuperate and the queue build up to subside. This way, the same throughput can be achieved with a lower level of WIP. Thus, it is very important to schedule the jobs in the Backlog list appropriately. Greco and Sarin (1996) have proved that a mixed sequence always gives a higher throughput rate than long sequential runs of individual products. This is, however, only true when the CBN (Conceptual Bottleneck machine, defined in Section 3.1.3) station is not the bottleneck for each individual product. When the processing time of one of the products at the CBN is not the highest processing time of the product on all the stations, then the throughput rate that is governed by the CBN is superior to that of unmixed sequences.

Consider a two-product case. The processing time at the CBN station of both the products is determined by the following formula:

$$tcbn = \max_j \sum_{i=1}^2 t(i, j) \quad (3.1.2-1)$$

If the bottlenecks for the products are at different stations then:

$$tcbn \leq tb(1) + tb(2) \quad (3.1.2-2)$$

Consequently, the following inequality must be satisfied:

$$\frac{2}{tcbn} \geq \frac{2}{tb(1) + tb(2)} \quad (3.1.2-3)$$

The first term in Equation 3.2.1-3 is an upper bound on the throughput level for a mixed sequence. The second term is applicable to the unmixed case. Thus, the above expression confirms that mixing the two products can increase throughput with the exception when all the IBNs are at the CBN.

### 3.1.3 Conceptual Bottleneck Machine (CBN)

The case considered in this research is that of multiple product classes. Each product class has its own bottleneck station and will form a queue only in front of that station. Thus, we define a machine as the bottleneck for the entire problem, whose addition of processing time of all classes is the highest. This machine is called the “Conceptual Bottleneck Machine” or CBN. Here, we assume that each station has the same capacity, or in other words can work on only one product at a time. We have discussed the ideal WIP level in the system in Chapter 1. Under the ideal case, the CBN should work continuously, without a queue before it or in any other part of the system.

### 3.2 Analysis of Preliminary Test Results

The main objective of this study is to observe using simulation, variations if any, in the flow time and the interdeparture time, due to changes in the sequence in which the jobs are dispatched at the first machine and the CONWIP level  $M$ . The results of these experiments would prove that an improvement is infact possible, and also would give an idea of the magnitude of improvement to be expected. The results of experiments to prove the superiority of a CONWIP system over variations of open systems are also discussed.

#### 3.2.1 Common Data

For all simulation runs, the following jobs were considered:

2 jobs of type A

1 jobs of type B

1 jobs of type C

The number of machines for all the runs were assumed to be 5 as this problem would reflect any changes in the sequence due to its relatively large size.

It was also assumed that all the products were available at the first machine at time  $t = 0$ . All the processing times are stochastic following an Exponential distribution. The following matrix gives the averages of processing time distributions of all the jobs at the machines:

	Machine 1	Machine 2	Machine 3	Machine 4	Machine 5
Job type A	1	15	1	5	1
Job type A	1	15	1	5	1
Job type B	4	5	1	2	13
Job type C	15	6	2	1	1

### 3.2.2 Experiment description

The possible sequences for the above mentioned job types are:

1A1A1B1C

1A1A1C1B

1A1B1A1C

For each of the above-mentioned sequences, the following cases were studied:

- No constraints on the limit on the number of parts in the system at any time (or no CONWIP level).
- No CONWIP level and a total intermediate storage size of 5
- No CONWIP level and a total intermediate storage size of 10
- No CONWIP level and a total intermediate storage size of 15
- A CONWIP system with a level of 5

The first experiment shows the superiority of a CONWIP system over other the first 4 types of systems listed above. The second experiment shows the variation of Flowtime and Interdeparture time with the CONWIP level  $M$ , the sequence maintained the same.

### 3.2.3 Definitions Used and Results

Flow time: The time interval between the instant a product enters the first machine to the instant it leaves the last machine.

Interdeparture time: The time interval between the instant a product leaves the last machine in one cycle to the instant it leaves the same machine in the succeeding cycle.

The Table 3.2.3-1 summarizes the results of the first experiment:

**Table 3.2.3-1: Comparison between Different Production Control Policies**

**IT = Interdeparture time and FT = Flow time**

Q = size of the system

		Open					Q = 5				
		1	2	3	4	Avg. of 1,2,3 &4	1	2	3	4	Avg. of 1,2,3 &4
<b>IT</b>	2A1B1C	42.02	42.11	42.03	42.03	42.05	42.93	43.01	42.95	42.90	42.94
	2A1C1B	43.18	43.03	43.06	43.06	43.08	42.00	42.02	41.98	41.97	41.99
	1A1B1A1C	39.64	39.66	39.69	39.69	<b>39.67</b>	41.60	41.30	41.32	41.59	<b>41.45</b>
<b>FT</b>	2A1B1C	2542.5 1	2557.02	2560.32	2561.68	2555.38	65.97	79.05	78.88	71.92	73.95
	2A1C1B	2798.8 8	2815.62	2809.53	2819.35	2810.84	69.27	80.46	64.05	80.59	73.59
	1A1B1A1C	50.20	52.49	51.35	55.68	<b>52.43</b>	70.68	75.84	66.41	70.42	<b>70.84</b>

		Q = 10					Q = 15				
		1	2	3	4	Avg. of 1,2,3 &4	1	2	3	4	Avg. of 1,2,3 &4
<b>IT</b>	2A1B1C	41.11	41.23	41.31	41.13	41.20	42.68	42.67	42.67	42.68	42.67
	2A1C1B	40.72	40.84	40.63	40.66	<b>40.71</b>	39.47	39.47	39.48	39.47	<b>39.47</b>
	1A1B1A1C	41.67	41.53	42.00	41.93	41.78	41.48	41.46	41.47	41.46	41.47
<b>FT</b>	2A1B1C	116.45	127.55	122.58	120.68	121.81	177.39	185.57	177.02	175.89	178.97
	2A1C1B	118.86	123.87	112.85	123.72	<b>119.83</b>	154.70	163.22	145.35	158.66	<b>155.48</b>
	1A1B1A1C	117.49	123.02	120.47	125.06	121.51	163.19	170.66	159.74	164.42	164.50

		CONWIP M = 5				
		1	2	3	4	Avg. of 1,2,3 &4
<b>IT</b>	2A1B1C	42.70	42.66	42.65	42.63	42.66
	2A1C1B	43.84	43.96	43.81	43.83	43.86
	1A1B1A1C	39.64	39.66	39.62	39.69	<b>39.65</b>
<b>FT</b>	2A1B1C	49.49	63.15	55.01	50.06	54.43
	2A1C1B	54.89	69.28	43.44	61.18	57.20
	1A1B1A1C	50.20	52.49	51.35	55.67	<b>52.43</b>

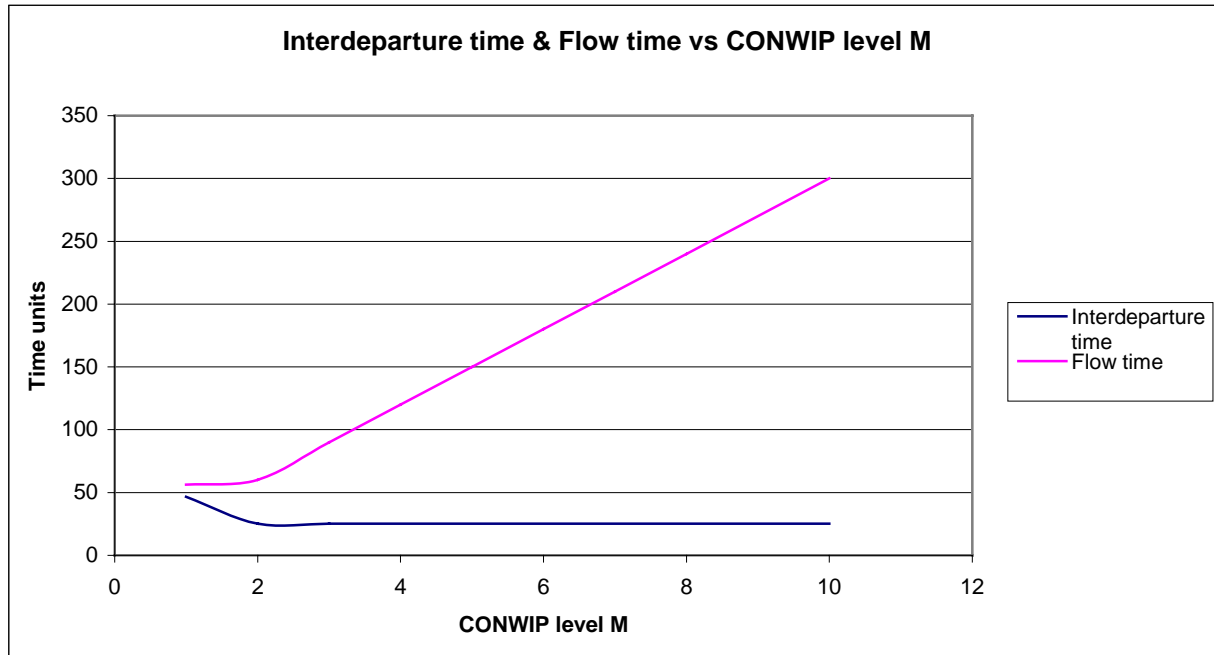
The values of Flow time and Interdeparture time for a CONWIP system are always lower than the corresponding values for an open system with the same system size. Though this is not an

exhaustive study to make such a claim, such in-depth studies have been done and mentioned in the literature review.

The second experiment shows the variation of these parameters with CONWIP level M. The following table summarizes the results of the second experiment for a CONWIP level of 5:

**Table 3.2.3-2:** Variation of IT and FT with CONWIP level M

Sr. No.	IT	FT
1	46.5	56.2
2	25.26	60
3	25.26	90
4	25.26	120
5	25.26	150



**Figure 3.1:** Variation of IT and FT with CONWIP level M

The graph shows that in this case the optimal level of WIP ( $M^*$ ) is 2. For values of M above 2, the flow time increases linearly but the throughput remains the same. Thus the point where the Interdeparture time curve just becomes horizontal is the optimal point.

### 3.3 MIP programming formulation

To generate useful insights, the problem was modeled as a Mixed Integer Programming problem (MIP). To achieve this, and to take into consideration the complexity of the problem, the following two conditions were relaxed:

- Stochastic processing times to avoid the probability terms.
- Limited buffer capacities for queues in-between machines.

The following is the MIP formulation:

$P_{ij}$  → Processing time of job  $j$  on machine  $i$

$P_{i(k)}$  → Processing time of the job in the  $k^{\text{th}}$  slot on machine  $i$

$X_{jk}$  → equals 1 if the job  $j$  is the  $k^{\text{th}}$  job in the sequence

0 otherwise

$I_{ik}$  → denotes the idle time on machine  $I$  between the processing of the job in the  $k^{\text{th}}$  position and  $(k+1)^{\text{th}}$  position

$W_{ik}$  → Waiting time of the job in the  $k^{\text{th}}$  position in between machines  $i$  and  $i+1$

$n$  → Total number of jobs that need to be scheduled

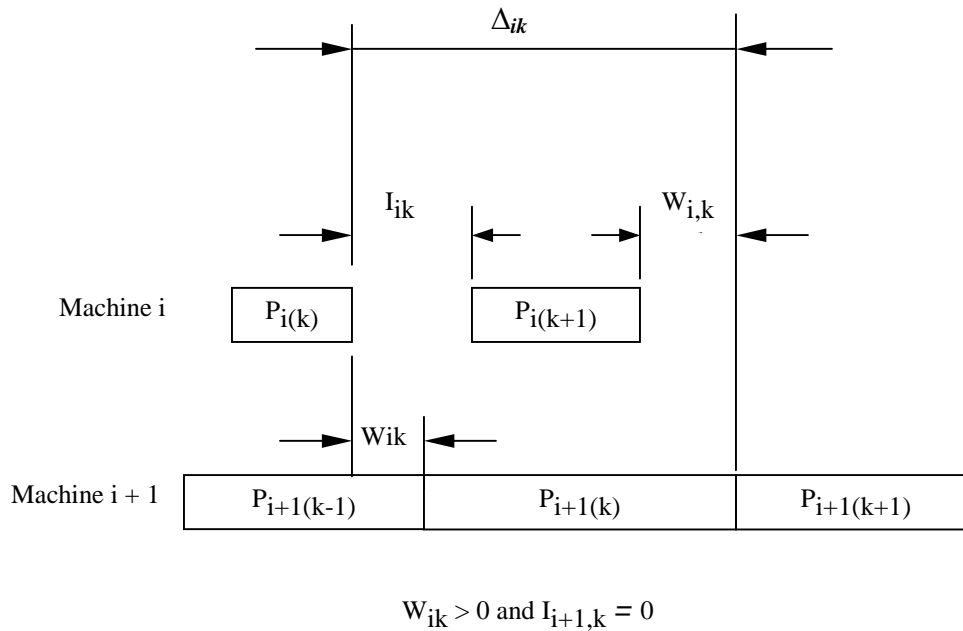
$m$  → CONWIP level to be maintained

$M$  → Total number of machines

$f$  → CONWIP cycle number;  $f = 1, \dots$  Higher integer  $(\frac{n}{m})$



Refer to Figure 3.1 for the time sequence graph:



**Fig 3.2:** Job  $i$  and  $j$  on the Gantt chart

Before we start building the model, it is important to note that on all machines, from 1 to  $M-1$ , both or either of the variables  $I_{jk}$  and  $W_{ik}$  can be a finite value. However, it is only on the last machine that  $W_{ik} = 0$  and  $I_{jk}$  can have a finite value.

Also, the objective of minimizing the makespan is equivalent to minimizing the total idle time on the last machine, machine  $M$ . The total idle time on machine  $M$  can be further subdivided into idle time that must occur before the job in the first position reaches the last machine and the second part being the idle time between successive jobs on the last machines. This fact is reflected in the objective function of the model.

## MIP Formulation:

*Objective function:*

$$z = \min\left(\sum_{i=1}^{m-1} \sum_{j=1}^n X_{ji} P_{ij} + \sum_{j=1}^{n-1} I_{Mj}\right)$$

*Constraints:*

$$\sum_{j=1}^n X_{jk} = 1, \quad k = 1, \dots, n \quad (\text{Only one job to position } k)$$

$$\sum_{k=1}^n X_{jk} = 1, \quad j = 1, \dots, n \quad (\text{Job } j \text{ to be assigned exactly to only 1 position or slot})$$

$$I_{ik} + \sum_{j=1}^n X_{jk+1} P_{ij} + W_{ik+1} - W_{ik} - \sum_{j=1}^n X_{jk} P_{i+1,j} - I_{i+1,k} = 0, \quad k = 1, \dots, n-1; i = 1, \dots, M-1$$

*CONWIP condition:*

$$\sum_{i=1}^{M-1} \sum_{j=1}^n X_{ij} P_{ij} + \sum_{j=1}^{q-1} I_{Mj} + \sum_{k=1}^q \sum_{j=1}^n X_{jk} P_{Mj} = \sum_{k=1}^{q+m} \sum_{j=1}^n X_{jk} P_{1j} + \sum_{j=1}^{q+m-1} I_{1j} + \sum_{l=1}^{q+m} W_{1l}$$

$$X_{jk} = 1, 0; \quad I_{ik}, W_{ik}, P_{ij}, n, m, M \geq 0$$

Though we do not use the formulation directly in this research, as the complexity prohibits its use, especially when the problem is large, we do gain some insights, which could prove helpful in the methodology to be proposed. One of the observations made is that, to decrease the

summation of idle times and waiting times at all machines, the following difference has to be minimized:

$$r_{(i,k)} = P_{i+1(k-1)} - P_{i(k)} \text{ for } k = 2 \text{ to } m \quad (3.2-1)$$

For a simple Flow shop problem, Stinton & Smith (1982), suggest a methodology using a formulation as a Traveling Salesman Problem (TSP) with a cost matrix of  $r_{(i,k)}$ 's. They argue that a low value of cost elements in the matrix is preferable since this indicates the likelihood of a better fit between any two jobs. However, this argument is myopic in the fact that it does not consider all the jobs at once. Thus, the solution obtained by this method need not be optimal, though a good one.

# Chapter 4: Modified Mean Value Analysis Algorithm (MMVA)

## 4.1 Introduction

In this Chapter, the concepts of Chapter 3 are further expanded. The Modified version of the Mean Value Analysis Algorithm is proposed and implemented on the Flow Shop problem on hand (with finite CONWIP level and finite intermediate buffers).

As alluded to earlier, the problems of finding the optimal CONWIP level ( $M$ ) and the optimal sequence cannot be separated and solved to optimality independently. Thus, there is a need to devise a methodology wherein the two can be dealt with appropriately, to get a close-to-optimal solution. In this chapter, we describe a method, which insulates the problem of finding the CONWIP level  $M$ , from that of finding the optimal sequence. It uses to its advantage, the following facts about the system:

- Processing times are stochastic
- Processing times are exponentially distributed and Interarrival time at the first machine follows Poisson distribution.
- The system has reached the state of dynamic equilibrium.

A method to determine a close-to-optimal sequence among jobs is presented in Chapter 5. The algorithm developed to determine 'M', is a modification of the MVA algorithm originally proposed by Reiser (1979) and then later modified by Hildebrant (1980). In order to solve the system analytically, the flow line needs to be broken into single machine stations, which can then be treated individually using standard queuing theory formulae. We develop an iterative

decomposition algorithm, which results in values of arrival rates at each buffer/machine and the probabilities of having no products at any machine and its buffer. Using these values, the values of average waiting times at each buffer, and finally, the average blocking time can be computed by a separate approximate method, which follows a proposed iterative algorithm. This Chapter also describes the concept of “Cascading Bottlenecks” and proves two results, which are used in the algorithm. Finally, the results of experimentation, to reflect the accuracy of the solution obtained and hence the reliability of the proposed method, are presented. A unique testing methodology is suggested and used, which is expected to be more reliable than that used by Hildebrant and Reiser (1980). The methodology is more in tune with the assumptions of the MVA algorithm and hence provides a superior testing scheme.

## 4.2 Mean Value Analysis Algorithm

As mentioned in the Literature review, Mean Value Analysis was first presented, without proof, by M Riser (1979). Richard R. Hildebrant (1980) further refined it for a FMS application. Due to the strong relevance of the algorithm presented by Hildebrant to this research, it is first presented in this section before it is further modified to include the additional features assumed in this research over that of Hildebrant's. The Mean Value Analysis is based on the following two intuitively appealing principles:

- A customer upon arrival, into the system, sees the same closed system with himself removed (one less customer) in long term.
- Little's law applies to the entire system as well as to each queue individually.

We need the following notation to present the Hildebrant's algorithm

$n_i$	Population size of part type $i$
$\vec{n} = (n_1, n_2, \dots, n_n)$	Population Vector
$t_{i,k}$	Service time of part type $i$ at machine $k$
$q_k$	Equilibrium mean queue size at machine $k$
$q_{i,k}$	Equilibrium mean number of part type $i$ at machine $k$
$\tau_{i,k}$	Equilibrium mean waiting time (including service time) of part $i$ at machine $k$
$\tau_k$	Mean waiting time at machine $k$ .
$\lambda_i$	Throughput of part type $i$
$M_i$	Set of machines part type $i$ visits for processing

*MVA Algorithm:*

*Step 1: Initial:*  $q_{i,k} = n_i / |m_i|$  for all  $k$  in  $m$ ,

$$q_{i,k} = 0 \text{ for all } m \text{ not in } m$$

*Step 2:* Repeat steps (3) through (6) until a suitable convergence criterion is met.

*Step 3:* Determination of the approximation factor:  $\epsilon_{i,k}^j = \frac{q_{j,k}(\bar{n})}{\sum q_{j,k}(\bar{n})}$ ;  $i = j$   
 $= 0$ , otherwise

*Step 4:* Mean Value Equation:

$$\tau_{i,k} = t_{i,k} + \sum_j t_{i,k} (q_{i,k} - \epsilon_{i,k}^j)$$

*Step 5:* Little's Result for parts

$$\lambda_i = n_i / \sum_k \tau_{i,k}$$

*Step 6:* Little's Result for machines

$$q_{i,k} = \lambda_i / \tau_{i,k}$$

Hildebrant (1980) analyzed the above-mentioned MVA Algorithm for its accuracy and reliability. To reduce the inherent variability associated with exponential distributions, Hildebrant used uniformly distributed processing times between 1 and 29 on a 4-machine problem, which processed an unlimited number of 10 job types. The processing times were chosen from the uniform distribution and were fixed during simulation runs. The results for the simulation runs were averaged over 50 runs and were compared with those obtained using the above-mentioned MVA Algorithm. The results showed significant differences between the numbers obtained by these procedures, reminding us of the fact that these algorithms are incapable of resulting in an exact solution. Due to the complexity of the problem at hand, further complicated by the stochasticity involved, analytical methods are impossible to resort to. However, the results obtained by the above MVA algorithm, show the same inherent trend as those of the simulation runs.

The above algorithm assumes infinite size buffers between machines. To adapt it to the finite intermediate storage case on hand, some modifications need to be made to incorporate the

effect of blocking. We describe this in the next section. This modification leads to an inclusion of an extra term in step 4 of the MVA. The rest of the algorithm is essentially the Little's law applied to the jobs and machines and, thus, remains unchanged. We utilize the MVA to iterate towards a good value of  $\tau_{i,k}$ . Our goal is to find  $\sum_i \sum_k \tau_{i,k}$ , which is the total flow time of a cycle for all the product types or classes. This term is the average blocking time of a part at a machine when the buffer queue at the immediately succeeding machine in the line is full. In order to find this mean blocking time, the concept of "Cascading Bottlenecks" is used.



### 4.3 Cascading Bottlenecks

Due to the finite capacity of the immediately succeeding machine (node)  $k+1$ , server  $k$  will get blocked from time to time. The server remains blocked until a service completion occurs at node  $k+1$ . It may also get blocked for longer duration, if the customer who just completed its service at node  $k+1$ , gets blocked due to the queue in front of node  $k+2$ . In general, when node  $k$  gets blocked, nodes  $k+1$  to  $j$ ,  $j \geq k + 1$  are already blocked. In this case, when a customer completes its service at node  $j+1$  and given that it does not get blocked, nodes  $k$  and  $j$  become unblocked and each blocked customer moves to the next node. This blocking effect, due to the downstream nodes, has to be incorporated into the service mechanism of each node, when studied in isolation. Another important point to be noted is that the first node is always assumed to be saturated (i.e. never empty). This is a common assumption when studying production systems and it implies that there is an unlimited supply of raw material at the first node. Similarly, the last machine is never blocked, as jobs are always free to leave the system when they are finished processing on the last machine.

We next present the modified algorithm and develop necessary results. Before we proceed, some additional notations are introduced. Other notations are defined later, before they are used.

$P_k(x)$	Probability of having $x$ parts at a station $k$ (in the processor plus in queue)
$\phi_k$	Capacity of station $k$
$\lambda_k$	Average arrival rate at station $k$
$\mu_k$	Processing rate at station $k$
$m$	Number of stations in the system
$\xi_0$	Instant in time during which part $i$ leaves machine $k$ to be the last job in the queue at machine $k+1$
$\xi_{int}$	Instant in time during which part $i+1$ leaves machine $k$
$\xi_q$	Instant in time during which the first part on machine $k+1$ is finished processing and part $i$ moves one position ahead in queue $k+1$ to accommodate part $i+1$

The rest of the variables remain the same as those used in Hildebrant's algorithm.

As explained earlier, it is desired to develop an expression to determine the blocking time term as close to the actual expression as possible, for inclusion in step 4 in Hildebrant's Algorithm. Each station (a machine with a buffer preceding it) is studied in isolation, so that standard one-machine formulae can be used to determine the variables associated with each station. However, due to the dynamics of the system and the interrelation of various variables among different machines, the stations have to be mathematically separated, to allow individual analysis.

For a M/M/1:FCFS/N/ $\infty$  model, the probability of having 'n' customers at a station is given by the following expression (Operations Research by Hira & Gupta, 1996; all standard formulae used in this chapter are taken from this book and are hence not referred from this point on to avoid repetition):

$$P_n = \frac{1-\rho}{1-\rho^{N+1}} * \rho^n \quad (n \leq N) \quad (4.3-1)$$

where N = size of the single server system. Thus, the probability of having  $\phi_k$  parts at machine (station, machine and node, are used synonymously in this research) is given by the following equation:

$$P_k(\phi_k) = \frac{1-\rho_k}{1-\rho_k^{\phi_k+1}} * \rho_k^{\phi_k} \quad (4.3-2)$$

Now, the probability of having  $\phi_k + 1$ ,  $\phi_k + 2$ ,  $\phi_k + 3$ ,  $\phi_k + 4$  or in general  $\phi_k + n$  parts is zero due to the limitation on the number of jobs at a station due to the intermediate storage size of  $\phi_k$ .

Thus, the probability of having  $n \geq \phi_k$  is

$$P_k(\phi_k) = \frac{1-\rho_k}{1-\rho_k^{\phi_k+1}} * \rho_k^{\phi_k} \quad (4.3-3)$$

where  $\rho_k = \frac{\lambda_k}{\mu_k} =$  % utilization of each machine

We thus needed to find the value of  $\rho_i$  for each machine, in order to determine the value of  $P_k(\phi_k)$ .

Note that, for a serial line, the value of  $\lambda_k$  is not always equal to  $\mu_{k-1}$ . Thus, to calculate the value of  $P_k(\phi_k)$ , we need to determine the value of  $\lambda_k$ .

Here, we propose an iterative procedure to obtain the value of  $\lambda_k$ . This procedure is explained next.

As we defined earlier, the arrival process is Poisson and service times are exponential. If a part cannot be processed due to the finite capacity of node  $i$ , it is considered lost. Thus, those jobs that are served during the time the server 1 is blocked are treated as lost jobs from the point of view of the entire system. The percentage of time station  $i$  is busy (i.e. either serving or blocked) is  $1 - P_i(0)$ . Consequently it follows that:

$$\lambda_i = \mu_{i-1}(1 - P_{i-1}(0)) \tag{4.3-4}$$

We formally state and prove this result next:

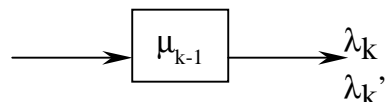
**Statement:**

If  $\lambda_i$  is the arrival rate of the jobs at machine  $i$ , and  $\mu_i$  is its processing rate, and if the probability of having zero parts at station  $i$  is  $P_i(0)$ , then the arrival rate at machine  $i$  is given by the following expression:

$$\lambda_i = \mu_{i-1}(1 - P_{i-1}(0))$$

**Proof:**

Consider a black box representation of the system as shown below:



Let  $\mu_{k-1}$  be the processing rate at machine k-1 and  $\lambda_k$  be the ideal output rate of machine k-1. Let  $\lambda_k'$  be the actual or as observed, output rate of machine 'k-1'. We assume in the definition of  $\lambda_k$ , that the machine k does not run "dry", which means that no capacity of machine is lost due to parts not being available for processing.

If the machine runs dry, for some period of time, the output rate  $\lambda_1$  drops say to a value  $\lambda_1'$ . We call this output rate the "effective" output rate.

Let:

$\xi_1$ : time of departure of job 1 from machine k-1

$\xi_2$ : time of departure of job 2 from machine k-1

$P_0(k-1)$ : probability of having 0 jobs at station k-1

$t_{k-1}$ : actual processing time at machine k-1

$t_k'$ : effective processing time at machine k

Now, we consider two cases for an exhaustive analysis:

Case I:  $\lambda_0 > 0$  and Case II:  $\lambda_0 = 0$

For any general case, we can write the following equation:

$$\frac{1}{\xi_2 - \xi_1} = \underbrace{[1 - P_0(k-1)]}_{\text{Case I}} \underbrace{\mu_{k-1}}_{\text{Case II}} + [P_0(k-1)]$$

"Ideal output rate" or output rate of the machine, before machine 0

$$\therefore \frac{1}{\xi_2 - \xi_1} = [1 - P_0(k-1)] \frac{1}{t_{k-1}} + [P_0(k-1)] \frac{1}{\infty}$$

Since  $t_k' = \xi_2 - \xi_1 =$  effective interdeparture time

$$\frac{1}{t_k'} = [1 - P_0(k-1)] \frac{1}{t_{k-1}}$$

Thus,

$$\lambda_k' = [1 - P_0(0)] \frac{1}{t_{k-1}}$$

$$\text{or } \lambda_k' = [1 - P_0(0)] \mu_{k-1}$$

QED

**Corollary:**

Consider the first two machines. According to our assumption, all jobs are available at the first machine at the start of the scheduling period. Thus, the first machine never runs dry or, in other words, the probability of there being no job at the first machine is zero.

Mathematically stating:

$$P_1(0) = 0$$

$$\text{Thus } \lambda_2 = \mu_1[1 - 0] \text{ or } \lambda_2 = \mu_1$$

Let  $\bar{\mu}_k$  be the effective service rate with blocking and let  $T_r$  be the average throughput for the entire system. Then,

$$\bar{\mu}_k = \frac{T_r}{1 - P_k(0)} \tag{4.3-5}$$

We assume that a server can process the job and then can check the queue size in the following machine to confirm its blocked status. Thus, processor  $k$  serves as an additional unit capacity buffer for machine  $k+1$ . Consequently, we analyze the system as a series of  $M/M/1/m_{k+1}$  queues in isolation.

The following is a standard formula for a single server queuing system:

$$1 - P_k(0) = \frac{\bar{\rho}_k * (1 - \bar{\rho}_k^{\phi_k + 1})}{1 - \bar{\rho}_k^{\phi_k + 2}} \tag{4.3-6}$$

$$\text{where } \bar{\rho}_k = \lambda_k / \bar{\mu}_k$$

Combining equations (4.3-4), (4.3-5) and (4.3-6), we obtain the following equation:

$$\bar{\rho}_k = \frac{\mu_{k-1}[1 - P_{k-1}(0)]}{T_r/[1 - P_k(0)]} \quad (4.3-7)$$

We know, by assumption, that the first machine is never starved and the last machine is never blocked. Thus, we get the following two equalities, out of which, the second one was proved earlier, as a corollary:

$$\bar{\mu}_m = \mu_m$$

$$\lambda_2 = \mu_1$$

The following algorithm is proposed to determine the values of  $\rho_k$ 's

#### ALGORITHM

*Step 1:* Assume some starting value of  $T_r^0$ . A good value might be  $\min_k \frac{1}{\mu_k}$

*Step 2:* Knowing  $P_1(0) = 0$ , we solve equations (4.3-6) and (4.3-7) simultaneously to find the value of  $P_2(0)$ . We continue this process till we get the value of  $P_m(0)$ .

Notes: The values of  $T_r$  and  $\mu_{k-1}$  are known in equation (4.3-7). Substituting these values in the equation, results in a relationship between  $P_k(0)$  and  $\bar{\rho}_k$ . This value of  $\bar{\rho}_k$ , in terms of  $P_k(0)$ , can be substituted in equation (4.3-6), to obtain a polynomial equation in  $P_k(0)$ . This equation can then be solved using numerical techniques using standard software such as Mathematica™.

*Step 3:* Using  $P_m(0)$ , we compute a new value of  $T_r$  say  $T_r^1$ . If  $T_r^1 - T_r^0 > \varepsilon$ , then go to *step 1*, else *stop*.

Notes: The value of  $\lambda_i$  is found from Equation (4.3-4). The value of  $P_m(0)$  can be used separately in Equation (4.3-6), to obtain the new value of  $\bar{\rho}_m$ . Using these two values, the value of  $\bar{\mu}_m$  can be calculated which can then be used in Equation (4.3-5) to obtain the value of  $T_r$  for the next iteration.

At the end of the algorithm, we have a set of values of  $P_k(0)$ 's for all the machines.

Using the set of  $P_k(0)$  from the last iteration, we then calculate the values of  $\lambda_k$  from Equation 4.3-4.

Consequently, for each machine, we calculate the following:

$$\rho_k = \frac{\lambda_k}{\mu_k} \quad \text{and then values of } P_k(m_k)\text{'s}$$

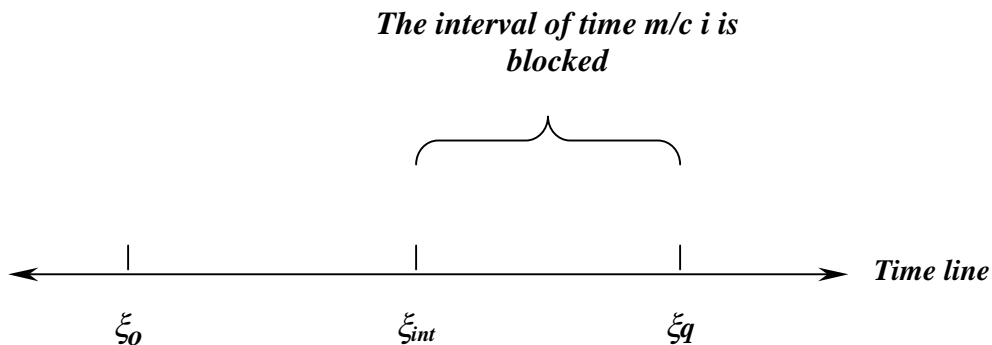
We have thus so far studied the system as 'm' queues in isolation and have incorporated all the properties of a serial queue system in it to compensate for the behavioral dynamics of the various parameters. The next step is to find the blocking time value, which will be added to step 4 of the original MVA algorithm.

### 4.3.1 A Method to determine the approximate value of Blocking time

The following is the method proposed to determine the value of Blocking time:

The variables  $\xi_0$ ,  $\xi_{int}$ ,  $\xi_q$  were defined at the start of section 4.3. We study the case from the instance, just before blocking occurs due to the immediate down the line station getting full, to the instance the block is cleared. Consider a pair of stations  $k$  and  $k + 1$ . Let the second station in the pair be one job short of reaching its capacity, and let the first job just enter the first machine. After finishing processing on the first machine, the first job enters the second machine. Now the second station has reached its capacity, thus blocking the first machine. The second job after processing on the first machine leaves the first station, but cannot enter the second station because it is full. It thus waits in between the two stations for a job in the machine of the second station to be processed. The time instance the second job leaves the first machine is designated by  $\xi_{int}$ . After some time interval, the second machine finishes processing the job and all the jobs in its queue move one slot ahead in its queue. The second job can now enter the queue of the second machine in the last position, unblocking the first machine in the process. This time instance is called  $\xi_q$ .

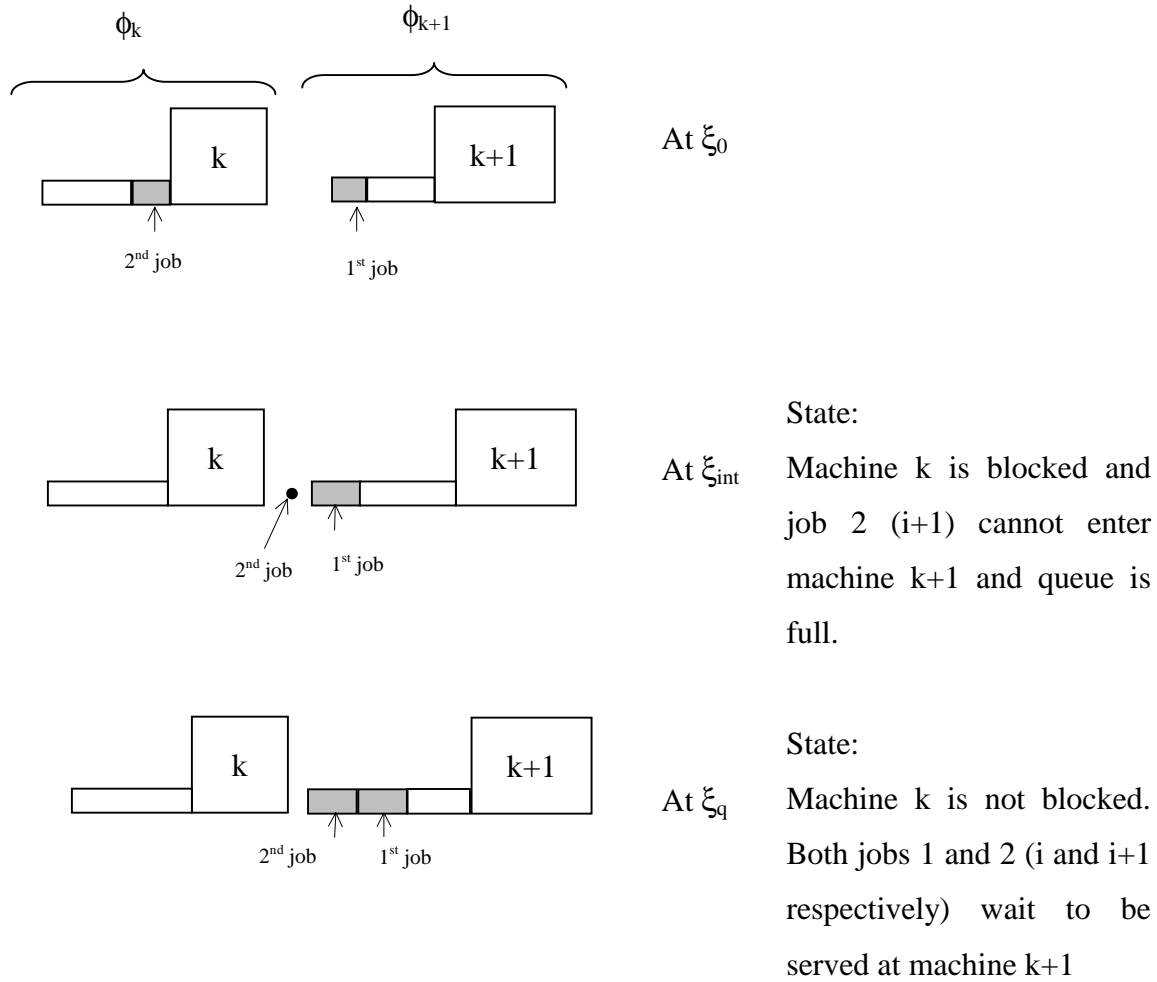
Figures 4.1 and 4.2 further clarify the significance of the variables.



**Fig 4.1:** Schematic depiction of Events at a buffer under consideration on a time line



The following three figures are snapshots, in time of the system, just as the blocking occurs. Thus, each figure is the state just after an event, which occurs at the time instance mentioned on the right hand side.



**Fig 4.2:** Snapshots in time of the arrival and departure events at a buffer

The time period over which machine  $k$  is blocked is  $\max\{0, \xi_q - \xi_{int}\}$  and thus depends on whether  $\xi_{int}$  lies on the left or the right side of  $\xi_q$ . If  $\xi_{int}$  lies on the right of  $\xi_q$ , machine  $i$  is not blocked.

Now, for interval  $(\xi_0, \xi_{int})$ , the following equation holds:

$$(\xi_{\text{int}} - \xi_0) [1 - P_k(0)] = t_k \quad (4.3-8)$$

where  $t_k$  is the average processing time at station  $k$ , as defined earlier

This is because  $(\xi_{\text{int}} - \xi_0)$  may consist of idle or empty (starvation) time of station  $k$ , when there is no part in the queue or processor  $k$ .

For interval  $(\xi_q, \xi_0)$ , the following statement holds true:

**Statement:**

For the time interval over which the blocking occurs and is then cleared due to the finishing of process of the second job at the second machine, the variables  $\xi_q, \xi_0, \xi_k$  are as defined earlier, the following equality holds true:

$$(\xi_q - \xi_0 - t_k)[1 - P_k(0)] = 0.5 * \left[ \frac{\tau_{k+1} - t_{k+1}}{\phi_{k+1}} \right] [P_{k+1}(\phi_k + 1)] \quad (4.3-9)$$

We provide an intuitive argument for this equality:

The left-hand side of the equality (LH) represents the expected time job 2 or  $(i+1)$  has to wait to enter the queue of the second machine. During the same time interval, a job on machine  $k + 1$  finishes processing, thus allowing the last job in the queue of machine  $k+1$ , job 1 or  $(i)$ , to move forward a slot in the queue. Assuming a straight-line projection of the processing time, a job at machine  $k + 1$ , takes  $0.5 * \left[ \frac{\tau_{k+1} - t_{k+1}}{\phi_{k+1}} \right]$  amount of time to finish processing, with a probability of

$[P_{k+1}(\phi_k + 1)]$ . This is the right hand side of the equality (RH)

More formally, to justify equation (4.3-9), we present the following argument:

1. Expected time job 2 is waiting after processing at machine k =  $\xi_q - \xi_0 - t_k$ .
2. As  $\tau_{k+1}$  is the equilibrium mean waiting time (including the service time) of a part s at service center k+1, the average time in queue =  $\tau_{k+1} - t_{k+1}$
3. Let us assume that the product moves in the queue towards the machine at a constant rate.

Thus, time taken by the product to shift a slot =  $\frac{\tau_{k+1} - t_{k+1}}{(\phi_{k+1} - 1)/2}$ . Note that the denominator is  $(\phi_{i+1} - 1)/2$  and not  $\phi_{i+1}/2$  as there are only  $\phi_{i+1} - 1$  products in queue. Also, if this time is assumed to satisfy approximately a straight-line slope, the denominator should be  $(\phi_{i+1} - 1)/2$  or average queue size. Other more accurate methods of estimating the waiting time could be found by detailed study.

Now, we can write:

Job 2 waiting time, after processing at machine k	*	Probability that machine i is not idle during the wait	=	Expected waiting time of job 1 at machine k+1 to move one slot in the queue	*	Probability of having k+1 full with $\phi_{k+1}$ parts in queue
---	---	--	---	---	---	---

Thus from points 1, 2, and 3 and the above equality, we can write the following:

$$(\xi_q - \xi_0 - t_k)[1 - P_k(0) - P_k(1)] = \left[ \frac{\tau_{k+1} + t_{k+1}}{(\phi_{k+1} - 1)/2} \right] [P_{k+1}(\phi_{k+1})]$$

The term in each bracket of this equality corresponds to the term mentioned in the descriptive equation, mentioned immediately above it.

Now, blocking time =  $\xi_q - \xi_{int} = (\xi_q - \xi_0) - (\xi_{int} - \xi_0)$

$$\therefore \text{Waiting time} = \tau_{k+1} [1 - P_{k+1}(0)] - t_k [1 - P_k(0)]^{-1}$$

From the iterative procedure mentioned earlier, we know the value of  $P_k(0)$ .

Using  $\lambda_k$  from the iterative procedure, we can find  $\tau_{k+1}$  according to the following standard formula for a (M/M/1:FCFS/ $\phi_k/\infty$ ) model:

$$\tau_k = \frac{L_k}{\lambda_k}, \text{ where } L_k = \frac{\bar{\rho}_k^2 - \phi_k \bar{\rho}_k^{\phi_k+1} + (\phi_{k-1}) \bar{\rho}_k^{\phi_k+2}}{(1 - \bar{\rho}_k)(1 - \bar{\rho}_k^{\phi_k+1})} \quad \text{and } \lambda_k = \frac{\bar{\rho}_k \tau_k}{1 - P_k(0)}$$

We now determine the mean blocking time of different machines (i's) using  $(t_q - t_0 - t_i)$  as the waiting time of machine k due to  $\phi_{k+1}$  parts at machine k+1.

Thus,

Probability of k being blocked = (probability of only machine k+1 having  $\phi_{k+1}$  parts) +  
(probability of only machine k+1 having  $\phi_{k+1}$  and machine  
k+2 having  $\phi_{k+2}$  parts) + ...

Hence,

Blocking time for machine k ( $BT_k$ ) is:

$$\begin{aligned} BT_i &= P_{i+1}(m_{i+1}) * (\xi_q - \xi_{int})_{i,i+1} * (1 - \Pr < m_i) + P_{i+1}(m_{i+1}) * P_{i+2}(m_{i+2}) (\xi_q - \xi_{int})_{i+1,i+2} * (1 - \Pr < m_i) + \\ &P_{i+1}(m_{i+1}) * P_{i+2}(m_{i+2}) * P_{i+3}(m_{i+3}) (\xi_q - \xi_{int})_{i+2,i+3} * (1 - \Pr < m_i) + \dots \\ &+ \prod_{x=1}^{N-i} P_{i+x}(m_{i+x}) (\xi_q - \xi_{int})_{N-1,N} * (1 - \Pr < m_i) \end{aligned} \tag{4.3-10}$$

where,

$(1 - \Pr < \phi_k)$  is the probability of machines other than k+1 having less than  $\phi_k$  parts in their queues. In other words, only machine k + 1 is full in the whole line, and  $(\xi_q - \xi_{int})_{k,k+1}$  is the interval of time machine k is blocked due to machine k + 1 having  $\phi_{k+1}$  jobs in its queue.

This blocking time is then added in step 4 of the MVA algorithm. Steps 5 and 6 remain unchanged, as they are simply Little's law applied to parts and to machines. Finally,  $\sum_k \sum_i \tau_{i,k}$

gives us the value of flow time, which needs to be minimized.

It is important to note that, since the MVA method is only an "approximate" method proposed initially by Reiser without proof, there is a possibility that during an iteration,  $\sum_i q_{i,k}$  become

greater than  $m_i$ . In such a case,  $\sum_i q_{i,k}$  is scaled down to  $\phi_k$  and the procedure is continued. Once

a queue is scaled or gets full, it is not included in any future calculations. This is due to the fact that the changes in values of the queue lengths are fairly monotonic, which means that if they are increasing, they do not decrease in the next iteration. Thus, once a queue length takes a value above the buffer size, there is no chance of missing any solution with a queue length lower than its capacity.

#### 4.4 A Lower Bound on $M^*$

It should be noted that the MMVA method is only a descriptive method and not a prescriptive one as related to the problem of finding  $M^*$ . In other words, it describes the system parameters in the closest possible way only when the ‘Job Population Size’ defined as  $n_j$  in the algorithm is known. This variable is nothing but the CONWIP level in case of a closed queuing system. Thus we need to start with a good value of  $M$ , which is as close to the actual  $M^*$  as possible. Greco and Sarin (1996) suggested a lower bound on  $M$  for a CONWIP system with unlimited intermediate buffers. They assume that the sequence is known in advance and propose the following formula to determine the value of  $M^*$  for a pair of jobs  $i$  and  $i'$ :

$$LB(M^*) = 1 + \frac{\max(tr(i, i'))}{tcb(i)} \quad (4.4-1)$$

where  $tr(i, i') = \sum_j^{B-1} t(i', j) + \sum_{j=B+1}^N t(i, j)$

$B$  is the conceptual bottleneck machine

$tcb(i)$  is the summation of processing times of all job types at machine  $i$

Job  $i$  triggers the job  $i'$  to enter the system

$t(i, j)$  is the processing time of job  $i$  on machine  $j$

Since the sequence of the jobs is yet unknown, we use a sequence, which results in the lowest value of  $M$  [ $LB(M^*)$ ], so that we can test only for values above  $M$ , in the MMVA. We can observe, from the second expression in equation (4.4-1), that a job pair for which, the job leaving the (Combined Bottleneck) cbn machine has the lowest summation of processing times on all the machines downstream of cbn and the job entering the system, the least summation of processing time on the machines upstream of cbn, so that the value of  $tr(i, i')$  will be the lowest. Thus, we select two jobs, which satisfy the above condition, to find the value of  $LB(M^*)$ . We then use this  $M$  value as a starting value in our method. Since limited size intermediate buffers can only add to increased blocking over a similar system with unlimited buffer, for which formula (4.4-1) was proposed, the matrix  $tr(i, i')$  would be lower and consequently the value of  $LB(M^*)$  would be lower. Even though the quality of  $LB(M^*)$  will deteriorate, for the finite

intermediate storage case, it is still the actual lower bound on  $M^*$ . The following example explains the selection of these two jobs.

Example:

Consider the following processing time matrix:

	J1	J2	J3	J4	Total	
M1	2	4	4	5	15	
M2	1	3	4	5	13	
<b>M3</b>	<b>6</b>	<b>1</b>	<b>7</b>	<b>7</b>	<b>21</b>	<b>cbn</b>
M4	2	3	5	3	13	
Sub total for job leaving	2	3	5	3		
Sub total for job entering	3	7	8	10		

Thus Machine 3 is the CBN designated as B.

The leaving job is selected as J1 as it has the least processing time after the cbn. The entering job is chosen as J2 as J2 is the second lowest after job J1 which we already chose as the entering job.

Thus the value of  $tr(i, i') = 7 + 2 = 9$  and  $tcb(i) = 21$

Hence, the value of  $M^*$  is found using the formula as follows:

$$LB(M^*) = 1 + \frac{7 + 2}{21} = 1.428 \text{ or } 2$$

All values from 2 are then used successively as value of  $n_s$ , in the MMVA till the point where the interdeparture time reaches a constant and FT (or CT) starts rising at a constant rate. This M is the best CONWIP level. For a more detailed explanation of the concepts used in this method, refer to Greco, M.P, 1996, Sequencing Policy for a CONWIP Production System, M.Sc. Thesis, Virginia Polytechnic Institute and State University, VA, USA.

#### **4.5 Description of the Experiments Conducted and the Observations Made**

It should be noted that the method proposed in Section 4.3 is only a descriptive one and does not result in a value of  $M^*$ . However, given a value of  $M$ , the method can be used to determine the value of Flow time. The proposed method of determining  $M^*$  starting from the lower bound value of  $M$ , determined by using the method presented in section 4.4, and increasing it successively in the MMVA, until the interdeparture time reaches steady state and FT (or CT) starts to increase, is next tested to determine the quality of the solution obtained. The solution quality is tested by comparing it to the optimal solution value obtained from simulation. The quality is inversely proportional to the deviation from the optimal.

The details of the experiments conducted are as follows:

- Number of jobs: 7; Number of Machines: 10
- Distribution for processing time: Uniform between 5 and 25
- Number of replications: 30 with Ran Generator number 1, 2, 3,.. , 30. Thus 30 replications were performed for each of the five time matrices and for each of the three CONWIP levels.  
Notes: Taylor II™ has 100 different seeds, which can be used to generate random numbers during run time. The use of a specific seed, results in the same random numbers. Each of this seed is given a number from 1 through 100 called simply as the Random Number Generator Number.
- CONWIP levels tested ( $M$ ): 7, 14, 35 (below  $M^*$ , near  $M^*$ , and well above  $M^*$  respectively)
- Buffer sizes: Summarized in the table (4.5.1):

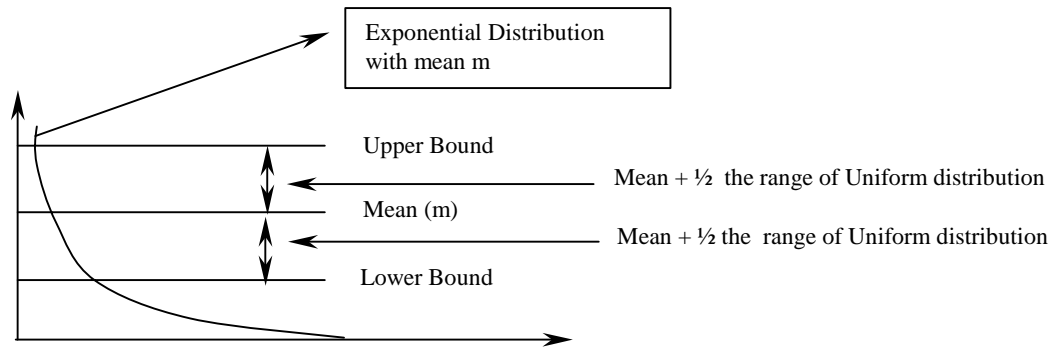


**Table (4.5.1): Intermediate storage sizes used in the experiment**

Buffer number (# 1 is between machines 1 & 2)	Size in units
1	4
2	3
3	2
4	5
5	2
6	8
7	4
8	20
9	5

To find the optimal solution, a simulation model was built using the Taylor II<sup>TM</sup> simulation software (software courtesy of Ericsson, Lynchburg) developed by F&H Simulations Inc. The model used the random generator inherent in Taylor II<sup>TM</sup>. The results were analyzed using an Excel worksheet and compared to the results obtained from the modified MVA algorithm. The MMVA was coded in Visual Basic Ver. 6.0 on a Win NT station. Each simulation run was allowed 20 cycles to warm up. This number was determined from experimentation, in order to ensure steady state.

A more realistic testing scheme is employed here, than that used by Hildebrant (1980). Hildebrant used uniform distribution, as compared to exponential distribution, which MVA assumes. The reason for this was to curb the high random deviation of the exponential distribution. The random deviations of an exponential distribution increase proportional to the value of the mean and may even take extreme values of zero to infinity in very rare instances. However, here we assume a smaller range for the uniform distribution to generate the means for the exponential distributions. The runs however used a “truncated” or “bound” exponential distribution generated by Taylor II as depicted in Figure 4.3.



**Fig 4.3:** *Depiction of Methodology to generate processing time values during simulation from the Pre-defined exponential distributions*

The values of processing times, thus generated, produce a reliable testing scheme, more in tune with the MVA assumptions.

The 30 observations for each output metric (avg. queue size & avg. queue wait) are averaged into 3 groups of 10 each. This is done merely for presentation purposes for lack of space. The standard deviation however, was taken separately over all the 30 readings.

For an exhaustive study the three CONWIP levels chosen namely (7, 14, 35), represent values below, near and above  $M^*$ .

The time matrices generated in Excel for the simulation are given in the Appendix B. A summary of the output data, which highlights the difference between the optimal simulation results and the MMVA, is presented below. The following four parameters were chosen to test the accuracy of the method:

- Average total queue size
- Average total wait time at the buffers
- Percentage standard deviation of average total queue size
- Percentage standard deviation of the average total wait time at buffers

These parameters for each time matrix and every CONWIP level are summarized in the following tables. The last two columns are the totals of the percentage differences in Average Queue and Average Waits.

**Table (4.4-1): Comparison between the simulation and MMVA results for CONWIP level of 7**

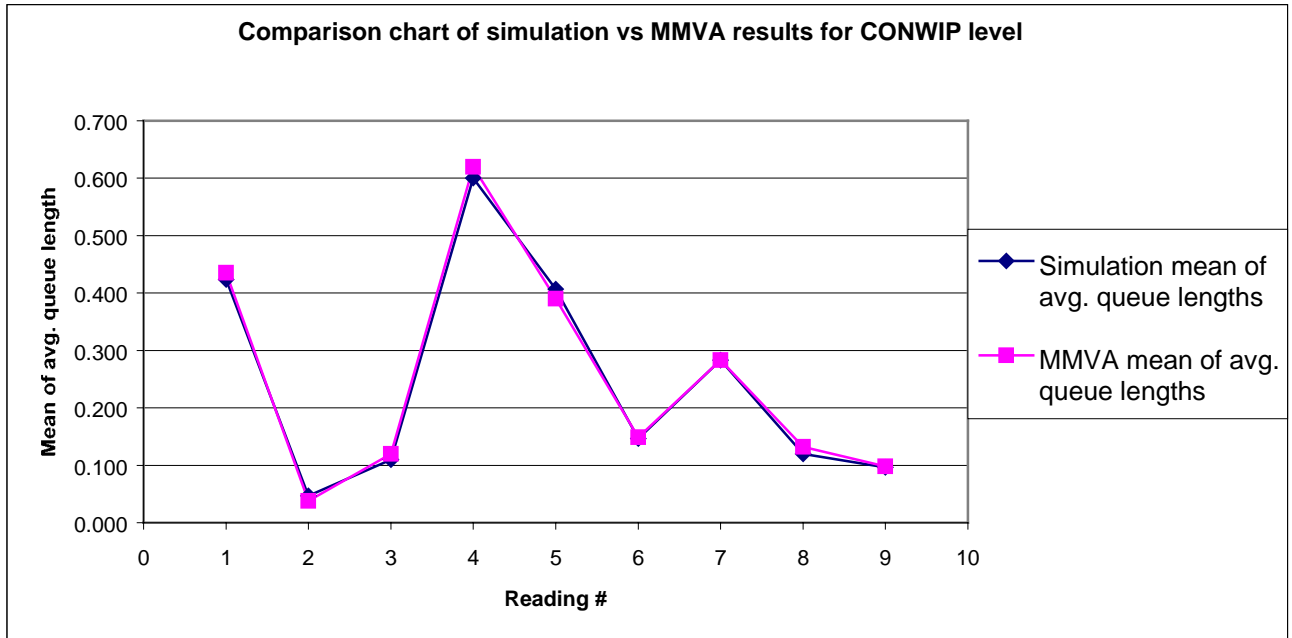
	Sr. No.	Differences in Means of		Percentage Differences in		Sum of Mean % error in Queue length	Sum of Mean % error in Wait time
		Avg. Queue	Avg. Wait	Avg. Queue	Avg. Wait		
<b>CONWIP level m: 7 Time matrix #: 1</b>	1	-0.012	0.880	-2.76	6.04		
	2	0.009	-0.073	18.57	-4.30		
	3	-0.010	-0.423	-9.09	-10.78		
	4	-0.020	2.473	-3.33	11.89		
	5	0.017	1.663	4.10	11.87		
	6	-0.002	0.417	-1.59	8.22		
	7	0.000	-0.410	0.12	-4.18		
	8	-0.012	-1.157	-10.00	-28.40		
	9	-0.001	0.480	-1.38	14.24	-5.36	4.62
<b>CONWIP level m: 7 Time matrix #: 2</b>	10	0.007	2.107	1.46	10.65		
	11	0.011	-1.407	3.21	-10.11		
	12	0.005	-0.097	10.00	-5.00		
	13	0.015	-0.920	9.36	-15.13		
	14	-0.020	1.190	-6.90	10.40		
	15	-0.012	0.007	-7.06	0.10		
	16	0.050	4.010	7.81	15.87		
	17	-0.002	0.443	-1.25	5.97		
	18	-0.030	-1.103	-13.64	-12.92	3.00	-0.17
<b>CONWIP level m: 7 Time matrix #: 3</b>	19	-0.002	-0.060	-3.85	-3.85		
	20	0.007	-0.167	7.69	-5.41		
	21	-0.005	-0.163	-3.64	-3.21		
	22	-0.207	4.927	-20.81	14.25		
	23	-0.002	-0.060	-1.79	-1.90		
	24	0.004	-0.330	8.13	-17.28		
	25	-0.013	0.317	-3.69	2.65		
	26	-0.019	5.434	-2.31	18.67		
	27	0.011	-0.153	22.00	-8.39	1.74	-4.47
<b>CONWIP level m: 7 Time matrix #: 4</b>	28	0.010	1.813	2.50	12.54		
	29	-0.110	-1.083	-15.71	-4.27		
	30	0.001	-0.100	1.82	-7.25		
	31	0.004	0.180	2.65	3.09		
	32	0.004	0.780	1.59	7.85		
	33	-0.005	0.173	-4.21	3.66		
	34	-0.021	4.280	-3.07	17.38		
	35	0.012	-0.403	6.25	-5.89		
	36	0.008	-1.503	5.71	-29.61	-2.47	-2.50
<b>CONWIP level m: 7 Time matrix #: 5</b>	37	-0.040	-1.617	-13.33	-14.65		
	38	0.000	-0.133	-0.20	-2.16		
	39	-0.013	0.473	-7.14	6.81		
	40	0.007	0.397	6.25	10.08		
	41	0.020	0.290	8.33	3.25		
	42	0.053	3.353	10.00	16.87		
	43	-0.050	-1.400	-9.80	-7.39		
	44	-0.011	1.043	-2.72	6.69		
	45	0.011	-0.623	7.33	-11.16	-1.28	8.33

**Table (4.4-2): Comparison between the simulation and MMVA results for CONWIP level of 14**

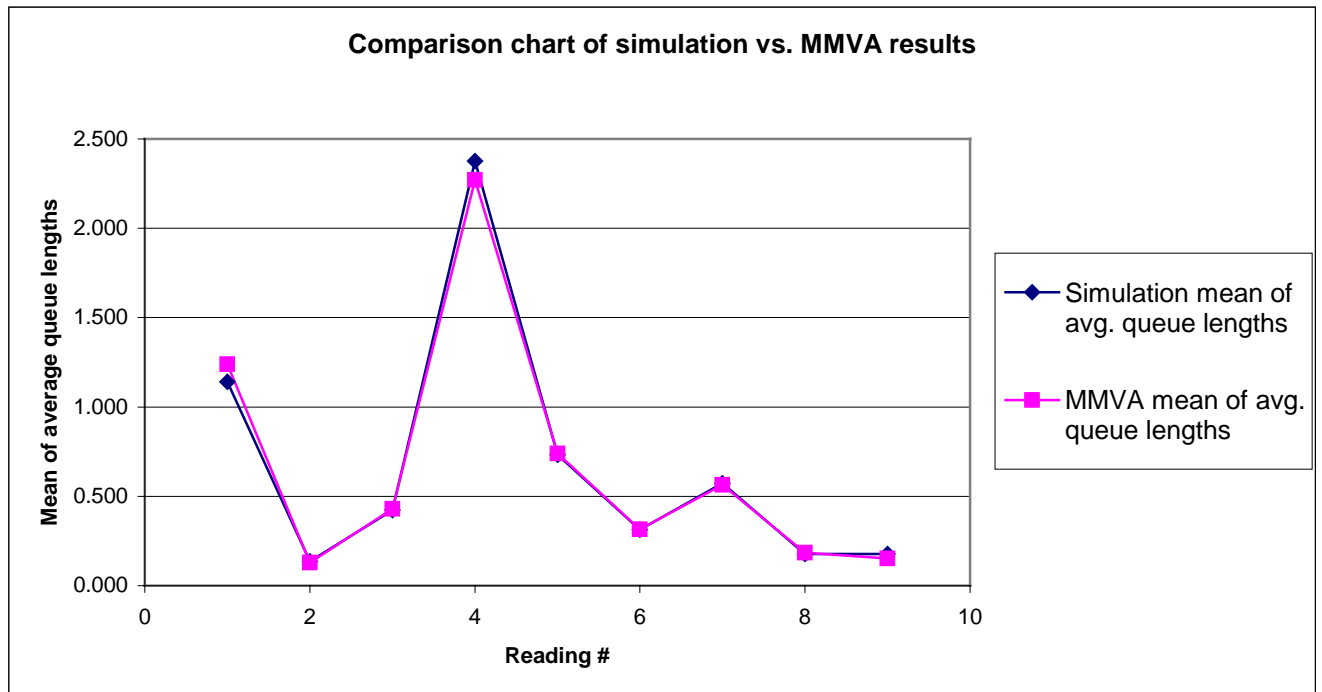
	Sr. No.	Differences in Means of		Percentage Differences in		Sum of Mean % error in Queue length	Sum of Mean % error in Wait time
		Avg. Queue	Avg. Wait	Avg. Queue	Avg. Wait		
<b>CONWIP level m: 14</b> Time matrix #: 1	91	-0.100	3.787	-8.77	12.58		
	92	0.008	-0.633	5.61	-17.51		
	93	-0.007	-1.053	-1.57	-9.32		
	94	0.107	-2.020	4.49	-3.20		
	95	-0.007	0.837	-0.91	4.30		
	96	-0.002	0.093	-0.53	1.12		
	97	0.009	2.270	1.63	14.89		
	98	-0.008	-0.550	-4.72	-11.53		
	99	0.024	-0.110	13.40	-2.31	<b>8.62</b>	<b>-11.00</b>
<b>CONWIP level m: 14</b> Time matrix #: 2	100	-0.017	9.303	-0.96	18.01		
	101	0.070	-4.133	8.24	-16.27		
	102	0.005	0.120	4.38	3.75		
	103	-0.010	-2.900	-1.60	-14.94		
	104	-0.040	-2.207	-6.56	-12.08		
	105	-0.003	-2.610	-0.33	-8.71		
	106	-0.070	-6.823	-3.61	-11.68		
	107	-0.009	3.220	-1.95	24.14		
	108	-0.037	2.117	-7.67	14.42	<b>-10.07</b>	<b>-3.36</b>
<b>CONWIP level m: 14</b> Time matrix #: 3	109	-0.007	-0.383	-8.00	-16.91		
	110	-0.017	-0.117	-6.33	-1.64		
	111	0.017	1.027	3.29	7.61		
	112	0.247	14.277	9.96	21.44		
	113	0.020	0.027	12.45	0.61		
	114	-0.009	0.480	-6.50	13.26		
	115	-0.107	-5.233	-13.62	-24.84		
	116	-0.030	-13.370	-1.12	-18.33		
	117	0.003	0.233	3.23	8.41	<b>-6.64</b>	<b>-10.39</b>
<b>CONWIP level m: 14</b> Time matrix #: 4	118	-0.007	9.193	-0.36	18.23		
	119	-0.103	-10.207	-6.68	-24.24		
	120	0.000	0.150	0.00	7.77		
	121	0.014	2.117	2.38	12.85		
	122	-0.010	-1.007	-1.64	-6.05		
	123	0.063	-1.050	13.97	-8.47		
	124	-0.117	-2.770	-8.08	-7.01		
	125	0.040	-1.177	9.30	-9.98		
	126	-0.007	0.763	-1.94	8.12	<b>6.95</b>	<b>-8.77</b>
<b>CONWIP level m: 14</b> Time matrix #: 5	127	0.057	-4.050	7.59	-19.99		
	128	-0.010	-0.405	-2.27	-3.39		
	129	-0.067	0.517	-18.87	5.35		
	130	0.020	1.077	5.56	10.79		
	131	0.043	-2.363	7.83	-15.57		
	132	0.270	-13.950	11.84	-22.31		
	133	-0.053	3.240	-4.15	9.31		
	134	-0.003	4.443	-0.36	13.02		
	135	-0.045	1.757	-10.84	15.41	<b>-3.67</b>	<b>-7.37</b>

**Table (4.4-3): Comparison between the simulation and MMVA results for CONWIP level of 35**

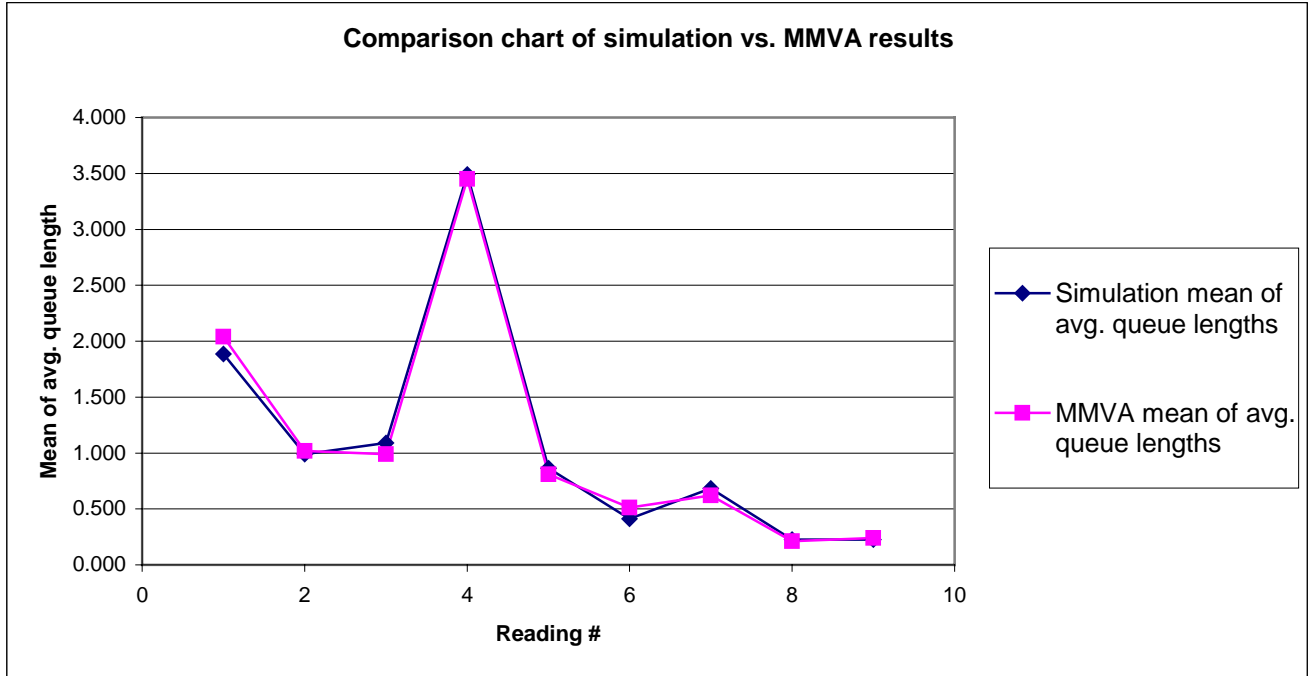
	Sr. No.	Differences in Means of		Percentage Differences in		Sum of Mean % error in Queue length	Sum of Mean % error in Wait time
		Avg. Queue	Avg. Wait	Avg. Queue	Avg. Wait		
<b>CONWIP level m: 35</b> Time matrix #: 1	189	-0.153	4.500	-8.13	9.54		
	190	-0.030	-1.647	-3.03	-6.61		
	191	0.100	0.920	9.17	3.35		
	192	0.040	-12.097	1.15	-13.75		
	193	0.053	-4.363	6.18	-19.88		
	194	-0.100	1.263	-24.11	12.03		
	195	0.063	1.763	9.27	10.14		
	196	0.014	-0.500	6.03	-8.59		
	197	-0.013	-0.020	-5.88	-0.35	<b>-9.36</b>	<b>-14.12</b>
<b>CONWIP level m: 35</b> Time matrix #: 2	198	-0.067	-13.273	-1.92	-14.79		
	199	-0.100	1.840	-7.58	5.35		
	200	-0.050	1.063	-13.51	10.96		
	201	0.173	7.393	9.67	15.75		
	202	0.137	1.130	12.93	4.07		
	203	0.083	-7.440	2.72	-9.20		
	204	0.193	8.633	7.63	12.83		
	205	-0.067	-1.457	-8.40	-6.90		
	206	0.083	-2.327	9.33	-9.74	<b>10.87</b>	<b>8.33</b>
<b>CONWIP level m: 35</b> Time matrix #: 3	207	-0.073	-1.893	-4.74	-5.11		
	208	0.090	-6.367	4.35	-12.75		
	209	-0.247	3.453	-15.29	8.83		
	210	0.203	18.127	4.58	16.77		
	211	-0.012	0.423	-5.00	7.46		
	212	0.044	0.990	15.65	14.39		
	213	-0.043	0.510	-3.65	1.75		
	214	0.867	-48.427	10.25	-23.29		
	215	0.002	0.160	1.14	4.42	<b>7.27</b>	<b>12.48</b>
<b>CONWIP level m: 35</b> Time matrix #: 4	216	0.347	6.497	9.72	7.15		
	217	-0.007	-9.097	-0.36	-19.26		
	218	-0.007	-0.193	-7.14	-7.87		
	219	-0.060	1.460	-7.69	7.26		
	220	0.080	1.227	11.94	7.13		
	221	0.047	1.913	6.80	10.81		
	222	0.090	-10.263	5.59	-24.64		
	223	-0.067	-2.370	-11.63	-15.95		
	224	-0.073	2.953	-16.79	25.90	<b>-9.57</b>	<b>-9.47</b>
<b>CONWIP level m: 35</b> Time matrix #: 5	225	0.183	-5.657	5.80	-7.35		
	226	0.137	3.123	6.52	6.07		
	227	0.127	3.537	8.58	9.71		
	228	-0.613	-13.240	-15.62	-13.60		
	229	0.100	-1.980	5.75	-4.56		
	230	0.507	-26.167	7.55	-15.56		
	231	0.087	-1.447	4.91	-3.23		
	232	-0.307	5.780	-16.11	11.95		
	233	-0.060	1.810	-13.33	15.81	<b>-5.97</b>	<b>-0.77</b>



*Fig 4.4: Comparison plots of simulation vs. MMVA outputs for CONWIP level of 7 and Time matrix # 1*



*Fig 4.5: Comparison plots of simulation vs. MMVA outputs for CONWIP level of 14 and Time matrix # 1*



**Fig 4.6:** Comparison plots of simulation vs. MMVA outputs for CONWIP level of 35 and Time matrix # 1

The following observations can be made from the above data and plots:

- The MMVA gives good results for the two metrics defined, namely, average queue lengths at buffers and Average wait times at buffers, with a slightly better result for the average queue length.
- The maximum deviation of the sum of percent deviation of average queue is about 11 % and that for average wait is about 15 %. However individual percent deviations show a large deviation, as high as 25 %. This alludes to the fact that the proposed method is only an approximate method. This magnitude of error is not unexpected as reported by Hildebrant (1980) and Reiser (1979). In spite of such errors, this method is still widely favored due to the absence of closed form solutions of this complex problem.
- Though the average percent error increases as the CONWIP level increases, the increase is marginal and could be due to some magnification of the randomness associated with the processing times. Also the percentage error near  $M^*$  ( $M = 14$ ), is strictly below 10%, which is the best obtained so far for a problem of this type and complexity.

- As expected the standard deviation of the average increases as the value of the corresponding mean of average increases.

As mentioned in Chapter 2, Spearman proved that a small deviation in the selected CONWIP level does not make any significant difference in the value of the profit function due to the inherent flatness of the curve near the optimal value of  $M$ . Thus, we use to our benefit, the ease of implementation of this method without sacrificing much on the accuracy of throughput of the system.



## **4.6 Conclusions**

A new solution to queuing networks, which are known to be complex due to their size and intricate behavioral dynamics, is proposed. Such networks do not possess a closed product form solution, which could be found by analytical methods. In spite of these difficulties, the solution obtained by this method, is entirely in terms of mean queue size, mean waiting time and throughput. This new analysis leads to a simpler algorithm with a better numerical behavior than the previous ones and also reduces the computational complexity as compared to the convolution algorithms. Thus, no joint distributions of product form or normalizing constants are involved. As pointed out by Reiser (1980), MVA also avoids problems of floating point overflow/underflow inherent in the earlier algorithms.

An approximate yet fairly good and reliable performance of this method does not affect the quality of the final objective significantly due to the flatness of the profit function curve around the optimal value of CONWIP level ( $M^*$ ).

# Chapter 5: Sequencing multiple job types in a fixed intermediate storage CONWIP system

## 5.1 Introduction

The previous chapter alluded to the issue of finding the optimal (or a near optimal) CONWIP level. Having the right level maintains a constant flow of parts to the combined bottleneck station, and ensures that it is utilized to its capacity. The second part of the problem is to find the optimal sequence with a low WIP (which in theory need not be the lowest CONWIP level achievable, to guarantee 100% utilization). This chapter starts with a discussion on the complexity of the sequencing problem at the first machine, and then a Network Flow Integer Program model is presented. Finally, a fast enumeration and appraisal technique is used to generate a good quality sequence. As will be explained, the problem has already been proven to be NP-hard. It is observed that such problems can often be simplified or at least some useful insights can be generated, by modeling it as a network flow problem. Various network flow algorithms can then be used to solve the problem or at least to reduce it to a problem of manageable size. Following this concept, the problem was divided into two sub-problems as follows:

- Generation of possible partial feasible sequences, which can then be rated for their quality and then build-up successively to sequence all the jobs in the set.
- Testing the quality of the sequences generated to reject the lesser effective partial sequences.

The first part of the first-subproblem problem is NP-hard while its second part consists of determining the optimal sequence from the partial job sets in the presence of the existing conditions such as a fixed CONWIP level and limited intermediate storage. The second sub-

problem is a critical path problem, and belongs to the class P problems, which can be solved in polynomial time. Thus, it is obvious that the overall quality of the solution will depend upon the capability of generating an optimal sequence, especially on the assumption it is based upon, which is that if a partial sequence is better in comparison to others in the partial job set, an addition of a job in the right slot retains that superiority. Since this assumption is not always true, the method does not always result in the optimal solution.

For the first sub-problem, this research uses an enumerative approach called Fast Insertion Heuristic devised by Nawaz et al (1983). According to Lawler et al (1993), the fast insertion heuristic (FIH) and the less efficient simulated annealing algorithm of Osman and Potts (1989), are few of the most effective procedures for the flow shop scheduling problem.

## 5.2 Complexity of the sequencing problem

Practical experience indicates that some scheduling problems are easier to solve than the others. Scheduling algorithms have been known for decades for some types of problems, which are capable for solving instances of large size problems. However, for some types of problems called NP-hard, the best algorithms strain to cope with only a handful of jobs. A computational problem can be viewed as a function  $f$  that maps each input  $x$  in some given domain to an output  $f(x)$  in some given range. Although there may be many ways to represent the input domain for a particular problem, these specifics are largely unimportant. We would be interested in studying the time required to compute  $f(x)$  as a function of the length of encoding of the input  $x$ , denoted  $|x|$ . Generally speaking, if a problem can be solved by a branching scheme of polynomial depth, then the problem is said to belong to the NP (nondeterministic polynomial) class. A problem is NP-complete, if it is in NP and every problem in NP reduces to it. All other problems can be reduced to class P, which have simpler solutions.

Large flow shop problems have been long known to be NP-hard. In fact the F3|Cmax problem was shown to be strongly NP-hard by Garey, Johnson and Sethi (1976). The problem at hand only considers permutation schedules. We know that for a general F|Cmax problem, permutation schedules are not necessarily optimal. However, due to their simplicity to execute and ease to model (as the whole problem of sequencing at each machine can be shortened to that of only on the first machine), they have been widely used and studied.

Since we convert the finite intermediate storage problem to a no-wait problem, it would be relevant and interesting to know that a F3|no wait|Cmax problem has been proven to be strongly NP-hard by Röck (1984). Since the makespan problem is NP-hard in the strong sense, the corresponding cyclic scheduling problem is also NP-hard in the strong sense as pointed out by Karmoun and Sriskandarajah (1992). They also prove that a F3|no-buffer (1,2), no-wait (2,3)|Cmax problem for a cyclic shop is NP-hard in the strong sense. Thus, it is obvious that addition of constraints such as stochastic processing times would only make the problem more difficult to address and thus NP-hard in the strong sense. A network flow based IP model is developed for the problem of scheduling the jobs, and is presented in Section 5.3.2. The

subsequent two sections pertain to the second part of the problem, which is to determine the length of the critical path. Again, Network Flow models are developed to generate useful insights, which further help to address the sub-problem.

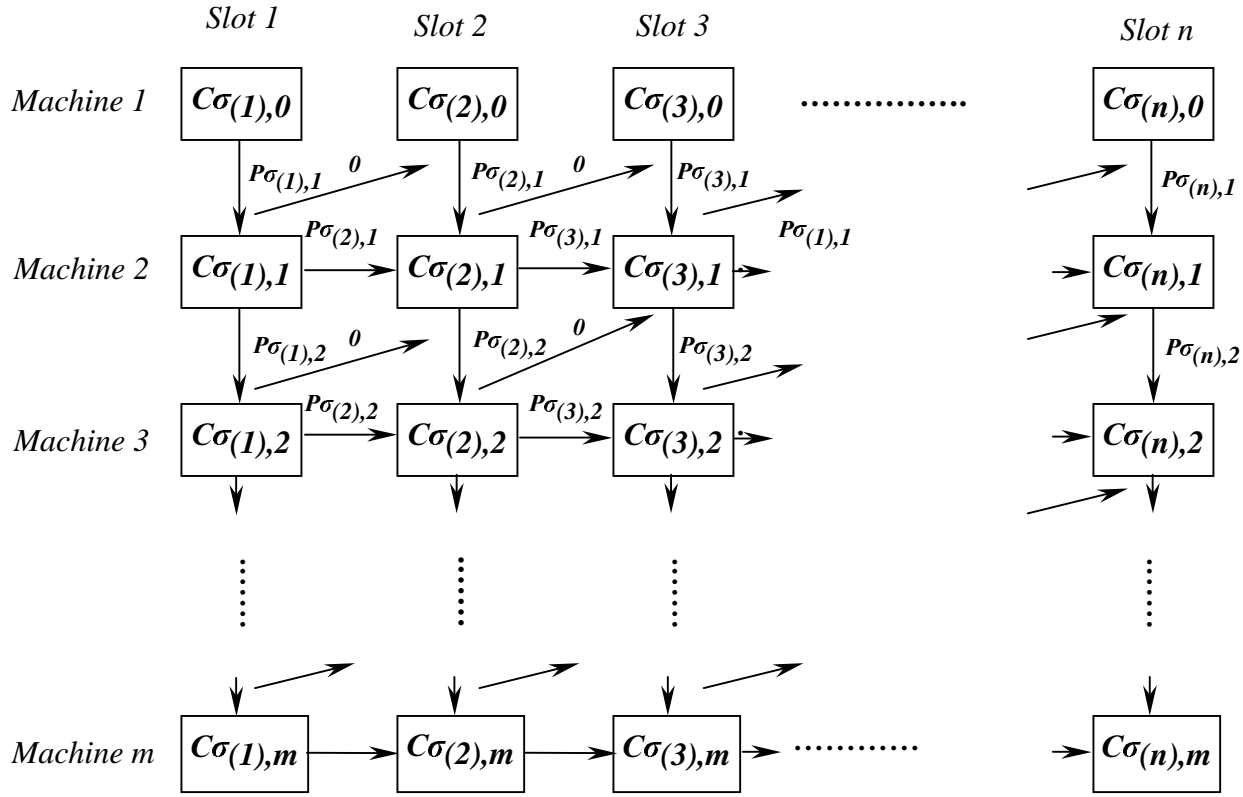
### 5.3 A Network Flow Integer Programming Model of the Problem to determine the critical path and Other Related Concepts

#### 5.3.1 The Concept of Critical Path or Flow time for a permutation, and the definition of variables

The following variables are used in the sequel:

$\sigma$	A permutation in the set of permutation schedules $\sigma_1, \sigma_2, \sigma_3, \dots$ Possible number of $\sigma$ 's (or $ S $ ) = $n!$ for a M machine, n job problem
$\sigma(i)$	$i^{\text{th}}$ job in permutation $\sigma$ or the job in the $i^{\text{th}}$ slot
$X_{jik}$	Indicator variable = 1, if job k is the $i^{\text{th}}$ job on machine j in the permutation $\sigma$ = 0, otherwise
$C^{\sigma(i),j}$	Completion time of the $i^{\text{th}}$ job in $\sigma$ on machine j
$P^{\sigma(i),j}$	Processing time of the $i^{\text{th}}$ job in $\sigma$ on machine j
$L(\sigma)$	Length of critical path for permutation $\sigma$

We can assume, without loss of generality, that a general flow shop with finite intermediate storage between machines can be converted to a shop with no storage between machines. This can be achieved by replacing buffers in the original system, by machines with unit capacity and a zero processing time for all the jobs. Thus, we construct a network only for the zero buffer case. This network is for the open shop (no CONWIP level) and a static schedule, which means that there is no cycling. This simplified model acts as a stepping stone towards the model developed for the problem on hand.



**Fig 5.1:** Network Flow Representation for Makespan Calculation of a Permutation Schedule  $\sigma$ , for a Zero Buffer Capacity case

Refer to the network in Figure 5.1. The nodes represent the completion times of the job  $\sigma(i)$  on machine  $j$  for a given permutation,  $\sigma$ . Directed arcs are defined from each node  $(C\sigma_{(i),j})$  towards  $(C\sigma_{(i+1)j})$ ,  $(C\sigma_{(i),j+1})$  and  $(C\sigma_{(i+1),j-1})$ . Arcs from nodes  $(C\sigma_{(i),j})$  to  $(C\sigma_{(i+1),j-1})$  have zero weights associated with them. Arcs from nodes  $(C\sigma_{(i),j})$  to  $(C\sigma_{(i),j+1})$  have corresponding weights of  $P\sigma_{(i),j+1}$  associated with them and those to  $(C\sigma_{(i+1),j})$  have corresponding weights of  $P\sigma_{(i+1),j}$ . Thus, the above network is a directed graph. The problem of finding the makespan value or flow time corresponding to a schedule is analogous to the determination of the maximum path on the above-mentioned acyclic graph from node  $(C\sigma_{(1),0})$  to  $(C\sigma_{(n),m})$ .

For a zero buffer flow shop, the following recursive relationship specifies the completion times,  $C\sigma_{(i),j}$ , corresponding to the node  $\sigma_{(i),j}$ , in the above network:

$$C\sigma_{(i),j} = \max \{ \max \{ C\sigma_{(i),j-1}, C\sigma_{(i-1),j} \} + P\sigma_{(i),j}, C\sigma_{(i-1),j+1} \} \quad (5.3.1-1)$$

for  $1 \leq i \leq n$  and  $1 \leq j \leq m$

For any given permutation (or schedule  $\sigma$ ), we can find the critical path. The critical path is the path such that the summation of the processing times of the jobs on that path is higher than that of the jobs on any other path. The length of the critical path corresponding to a schedule is the value of the flow time for that schedule. All permutations [ $\sigma \in \{S\}$ ,  $|S| = n!$ ] will have their own critical paths and values of flow times. The minimum among these flow times (or the critical path lengths) is the flow time of the optimal schedule. This is explained in a more mathematical fashion in the following paragraph.

The length of any path from  $(C\sigma_{(1),0})$  to  $(C\sigma_{(n),m})$  can be written as follows:

$$L(\sigma) = \sum_{j=1}^{w_1} P_{\sigma_{(1),j}} + \sum_{j=w_1}^{w_2} P_{\sigma_{(2),j}} + \dots + \sum_{j=w_{n-1}}^m P_{\sigma_{(n),j}} \quad (5.3.1-2)$$

where integers  $w_1, w_2, \dots, w_{n-1}$  define the path. Also  $1 \leq w_1 \leq m$ ;  $w_1 - 1 \leq w_2 \leq m \dots$

For example, referring to Figure 5.2, the critical path as shown in brown color can be defined mathematically by the following values of  $w$ 's .

$w_1 = 2, w_2 = 2, w_3 = 3, w_4 = 3$  and so on.

The max of  $L(\sigma)$ , among all permutations in set  $S$ , is the critical path.

Now, if  $w_2 = w_1 - 1$  or the path follows the arc from  $(C\sigma_{(i),j})$  to  $(C\sigma_{(i+1),j-1})$ , then

$$\sum_{j=w_1}^{w_2} P_{\sigma_{(i),j}} = 0 .$$

We still have not defined as to which jobs would be on this critical path. This problem, rather, is reverse of what was just explained. The jobs are first assigned to specific slots, resulting in a fixed schedule, the critical path of whose network graph is the value of the



objective function or Flow time. The indicator variable  $X_{jik}$  does this function by assigning the job to slots on the critical path. Consequently, we can write the equation for  $L(\sigma)$ , in terms of  $X_{jik}$  as follows:

$$L(\sigma) = \sum_k \sum_{j=1}^{w_1} P_{\sigma(1),j} X_{j1k} + \sum_k \sum_{j=1}^{w_2} P_{\sigma(2),j} X_{j2k} + \dots \quad (5.3.1-3)$$

or

$$L(\sigma) = \sum_{k=1}^n \sum_{i=1}^n \sum_{l=1}^m \sum_{j=w_{l-1}}^{w_l} P_{\sigma(i),j} X_{jik}$$

This is the length of a path on the network for a specific sequence or permutation. Thus, the maximum of this value among all paths for a particular sequence  $\{ \max_w L(\sigma) \}$  is the critical path (or Flow time), which we would like to minimize. The problem of finding the optimal sequence can be formulated as an IP problem as shown in the following section.

### 5.3.2 An IP Model to determine the optimal sequence

*Objective Function:*

$$\text{Min}_{\sigma} L(\sigma) \text{ where } L(\sigma) = \max_w \left\{ \sum_{k=1}^n \sum_{i=1}^n \sum_{l=1}^m \sum_{j=w_{l-1}}^{w_l} P_{\sigma(i),j} X_{jik}(\sigma) \right\}$$

or

$$\text{Min}_{\sigma} \left\{ \text{Max} \sum_{k=1}^n \sum_{i=1}^n \sum_{l=1}^m \sum_{j=w_{l-1}}^{w_l} P_{\sigma(i),j} X_{jik}(\sigma) \right\} \text{ where } X_{jik}(\sigma) \text{ defines the sequence for a particular}$$

permutation  $\sigma$

*Constraints:*

The following constraint states that the length of any path is smaller than the longest path (or the critical path) for a given  $\sigma$

$$\sum_{k=1}^n \sum_{i=1}^n \sum_{l=1}^m \sum_{j=w_{l-1}}^{w_l} P_{\sigma(i),j} X_{jik}(\sigma) \leq L(\sigma) \text{ for all } w\text{'s and } \sigma\text{'s}$$

Also, one slot can hold one job

$$\sum_k X_{jik} = 1 \text{ for all } i,j$$

Job k can be assigned to only one slot i

$$\sum_i X_{jik} = 1 \text{ for all } j,k$$

Only one job can occupy a slot on a machine

$$\sum_j X_{jik}(\sigma) = 1 \text{ for all } i,k$$

$$X_{j,i,k} \in (0,1) \text{ for all } j, i, k$$

Unfortunately, this IP problem cannot be solved in Polynomial time. However, the sub-problem of finding the critical path for any permutation  $\sigma$  is of the order of  $O(nm)$  as shown by Lawler (1976) and can thus be solved in polynomial time.

Thus, the above problem is separated into the following two sub-problems:

- Find the sequence for which the critical path has the least value. This problem cannot be solved in polynomial time and thus a fast heuristic or a search method needs to be employed to solve it. An enumerative approach, called Fast Insertion Heuristic and devised by Nawaz et al (1983) is used to this end.
- Determine the critical path. This is done by a network flow technique, which exploits the specific properties of the network.

We use the concept of scheduling epochs in modeling the problem of finding the critical path of a permutation. A schedule can be as divided into groups of jobs such that the groups follow a strict precedence constraint relationship. We can say that, virtually, any job in the set of jobs which form one epoch, has to be finished, before any job in the set of the next epoch can be started, though this is not necessarily true on the time scale. This virtual precedence relationships, act as constraints in the IP model to determine the critical path. The following two sections, present a mathematical definition of these epochs. The concept is built using the following two stages, for ease of understanding:

- Open system with intermediate storage of size zero.
- Closed system with a CONWIP level of  $M$ , and intermediate storage size of zero.

## 5.4 An IP Model to Determine the Length of Critical Path for a given sequence among jobs

### 5.4.1 Model for an Open System with Zero Buffers

The following variables are defined:

Let  $P_{ji}$  be the processing time of job  $i$  on machine  $j$  and

$X_{ji}$       Indicator variable  
              = 1, if job  $i$  on machine  $j$  is on the critical path  
              = 0, otherwise

#### Model:

*Objective function:*

Max  $L$ , where

$$L = \sum_{j=1}^m \sum_{i=1}^n P_{ji} X_{ji}$$

*Constraints:*

$$X_{j,i} = X_{j-1,i} + X_{j+1,i-1} + X_{j,i-1} \quad \text{for all } j,i$$

$$\text{and } X_{1,1} = 1 \text{ and } X_{m,n} = 1$$

$$X_{ji} \in \{0,1\} \text{ for all } i,j$$

In the following sub-section, we present a formulation to determine the critical path for a closed system with the CONWIP level of size  $M$ .

### 5.4.2 A Model to determine critical path for a System with Zero Buffer Between Machines and CONWIP Level of M

#### CONCEPT OF SCHEDULING EPOCHS

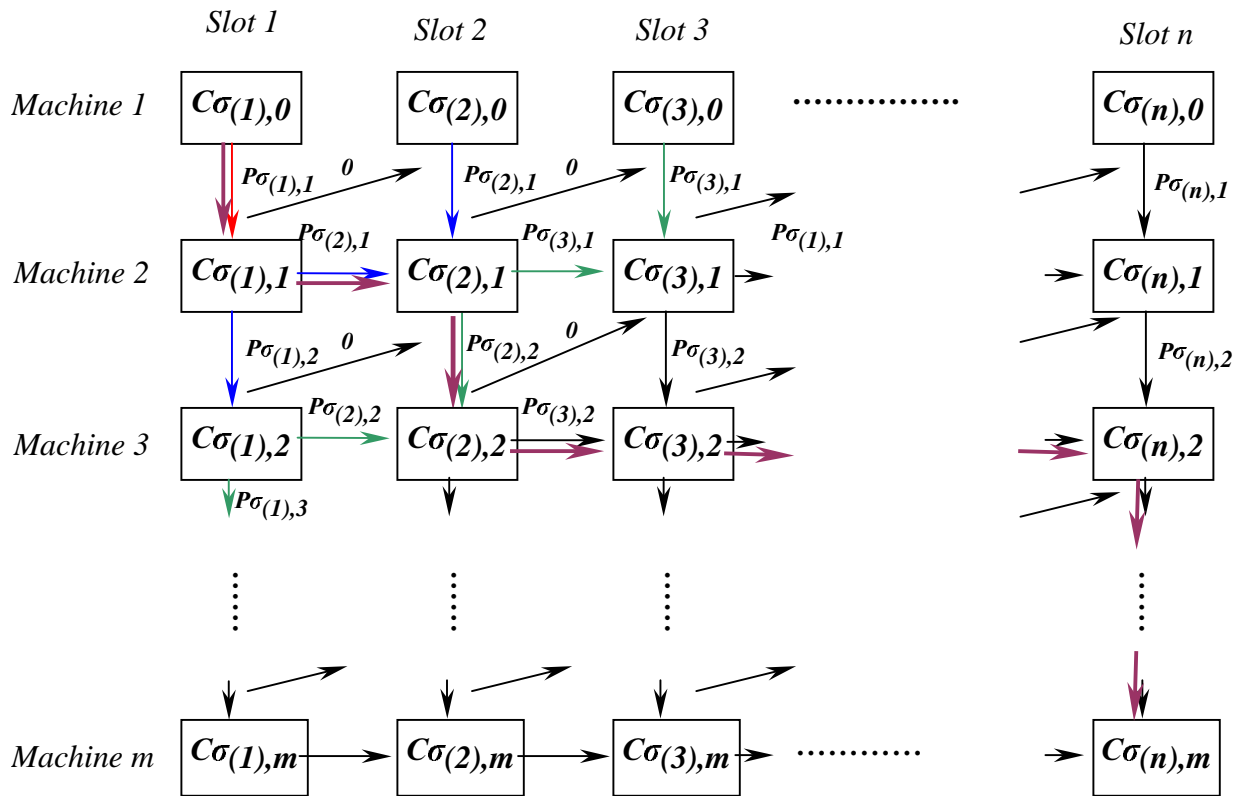
Refer to the Network representation shown in Figure 5.1. We define the Epochs as follows:

Epoch 1 (Red):  $P_{\sigma(1),1}$

Epoch 2 (Blue):  $P_{\sigma(1),2}, P_{\sigma(2),1}$

Epoch 3 (Green):  $P_{\sigma(1),3}, P_{\sigma(2),2}, P_{\sigma(3),1}$

These Epochs are shown as colored heavy lines on the network in Figure 5.2.



**Fig 5.2:** Network Flow Representation for Makespan Calculation of a Permutation Schedule  $\sigma$ , showing the Epochs and the critical path using a colored scheme

The critical path is shown in a heavy brown line, and runs from the first to the last node. These Epochs can be viewed as progressive boundaries between the start and end nodes, and which thus have to be cut enroute the last node from the first one. It can be easily noted from the above depiction of Epochs and the critical path that we have to follow only one of the arcs on all the Epochs.

The following two statements formally define the concept of scheduling epoch.

Statement 1:

*A schedule can be divided into segments or epochs, which for the  $M \geq m$  case can be thought of as time slots demarked by vertical lines joining the end points of successive jobs on all machines.*

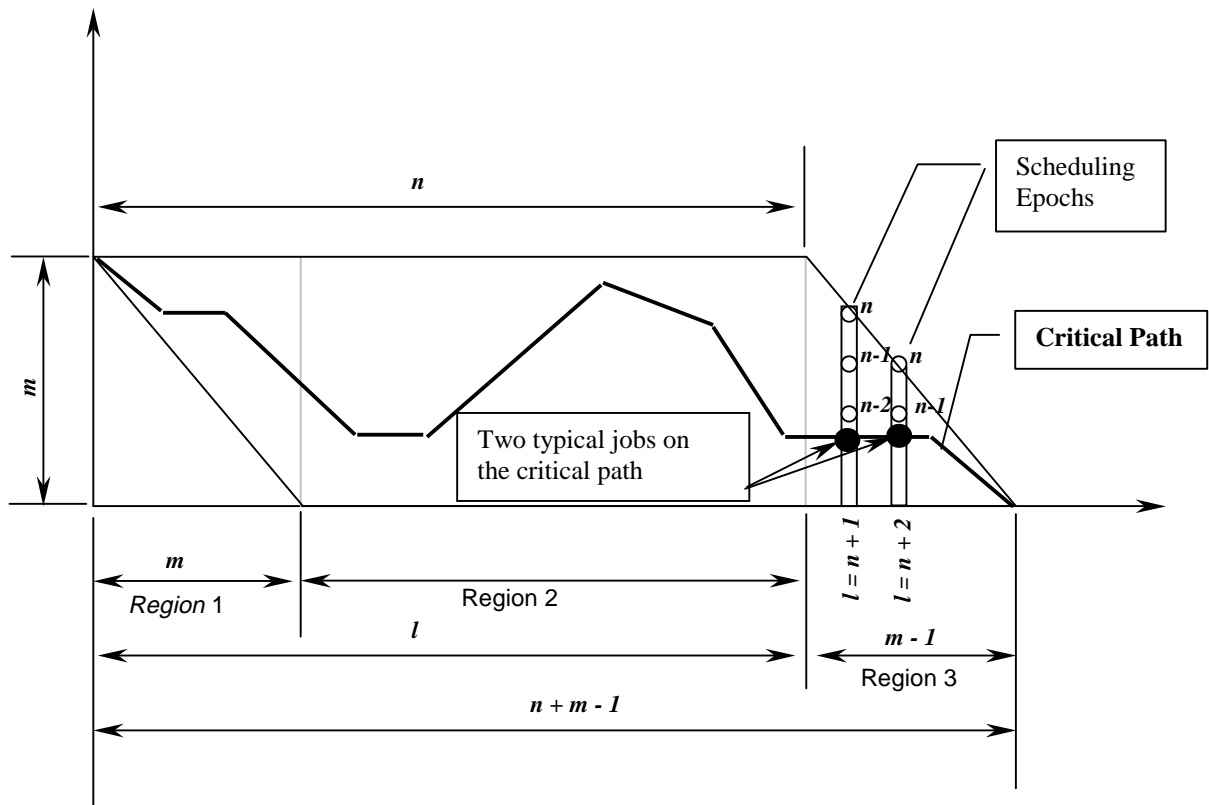
Statement 2:

*Only one of the jobs in every scheduling epoch can be on the critical path.* This statement can be written in mathematical form for a noncyclic (or acyclic) sequence as follows:

$$\sum_{k=1}^l X_{k,l-k+1} = 1 \text{ for all values of } l = 1 \text{ to } m \quad (5.4.2-1)$$

$$\sum_{k=1}^m X_{k,l-k+1} = 1 \text{ for all values of } l = m+1 \text{ to } n \quad (5.4.2-2)$$

Any sequence in the case of an open flow shop can be depicted schematically as shown in Figure 5.3. The schedule can be viewed in terms of vertical columns (or epochs) defined as follows. The first job on machine 1 constitutes the first vertical column. The second vertical column consists of the second job on the first machine and the first job on the second machine. The third vertical column consists of the first job on the third machine, the second job on the second machine, and the third job on the first machine, and so on. This was shown in color scheme earlier.



**Fig 5.3:** Schematic Graph of a Typical Open Flow shop Gantt chart

The scheduling Epochs are vertical slots as shown in Region 3. As can be observed, the scheduling epochs can be differentiated into three distinct types, as labeled in Figure 5.3. These groups of similar epochs are called “Regions”. All the jobs in the epochs belonging to a region can be represented by a common mathematical equality in the form of a constraint in the IP model, to determine the critical path. Thus a Region is a collection of separate epochs with a similar form of mathematical equation defining them, which are clubbed together only for modeling purposes. An example is the Region defined by a common equation (5.4.2-3). The variable  $i$  defined earlier is the index of these Regions, whereas  $l$  is the index of the individual epochs. We can divide the whole graph into 3 distinct regions based on the configurations of the scheduling epochs. Also, as stated earlier, only one job of each epoch can be on the critical path.

We now need to mathematically derive an expression, to determine the length of the critical path. Let us consider Region 1. Refer to Figure 5.3. On the X-Axis, Region1 extends until m. The following equation states that at least one job in this period is on the critical path.

For Region 1:

$$\sum_{k=1}^l X_{k,l-k+1} = 1 \text{ for all values of } l = 1 \text{ to } m \quad (5.4.2-3)$$

Now, there can be cases of multiple critical paths, in which case, more than one job in a period could be on the critical path. However such critical paths result in identical objective function values. Thus, we can neglect one path altogether and still claim that only one job in each path is on the critical path.

Similarly, for Region 2 and 3, we can write the following equations:

Region 2:

$$\sum_{k=1}^m X_{k,l-k+1} = 1 \text{ for all values of } l = m+1 \text{ to } n \quad (5.4.2-4)$$

Region 3:

$$\sum_{k=l-n+1}^m X_{k,l-k+1} = 1 \text{ for all values of } l = n+1 \text{ to } n+m-1 \quad (5.4.2-5)$$

In the determination of the critical path, we assume that the sequence is known. In other words, the job types in all the epochs are known.

The following variables are used in the sequel:

m	Number of machines (actual # of machines + Summation of buffer sizes)
n	Number of jobs in the sequence
M	CONWIP level
l	Index of Epoch; (explained in the sequel)
k	Job number in a scheduling epoch
P <sub>k l</sub>	Processing time of the job type k in epoch l



- $X_{k,l}$  =1, if job type k, in scheduling epoch l, is on the critical path  
=0, otherwise
- i An integer representing Index of the region type (e.g.  $i^{\text{th}}$  region of type (2) etc). This is explained in detail later but is defined here with other variables to facilitate easy reference.

The following two cases arise: (i)  $M \geq m$  and (ii)  $M < m$ . The first case implies that there are atleast as many jobs in the system as the number of machines. Since we are considering the case of zero intermediate buffers, this implies that  $M \geq m$  is not possible and hence we need to consider only the  $M = m$  case. The second case implies that we have less number of jobs in the system than m.

*Case I:  $M = m$*

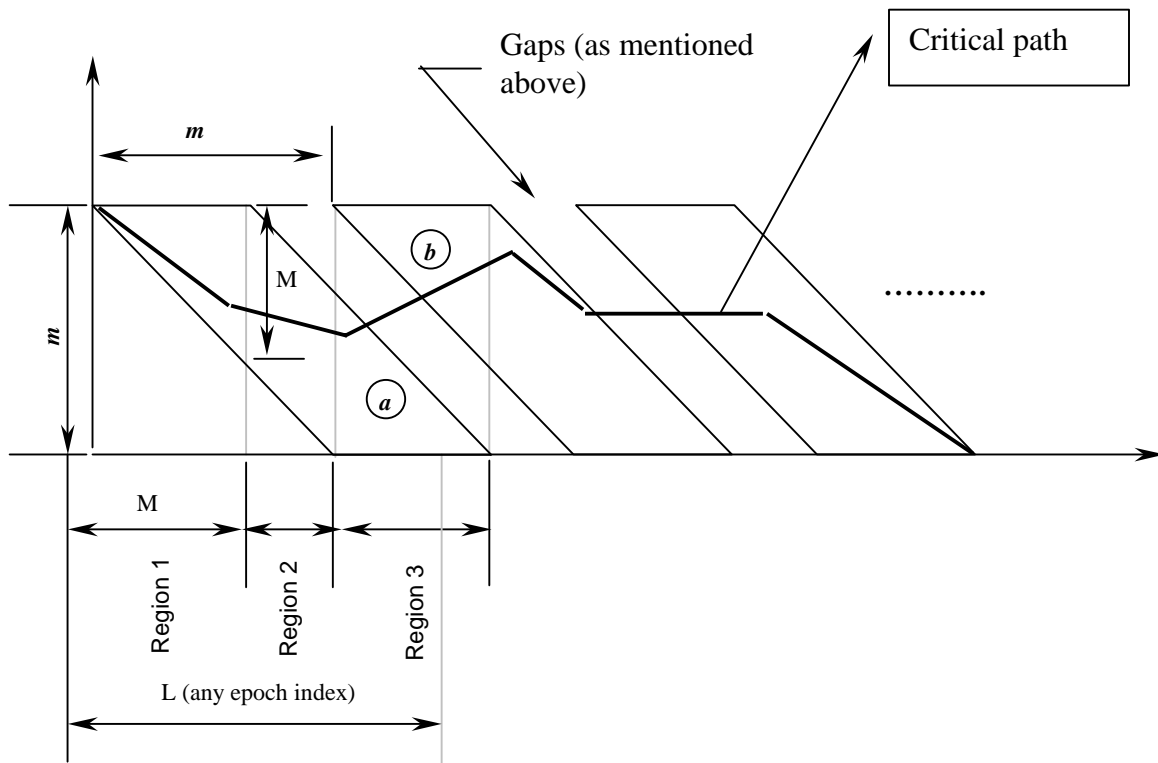
This case results in an open shop situation where the CONWIP level condition is relaxed

This can be explained as follows:

Refer to Figure 5.1. Due to the manner of constructing arcs in the network presented in Figure 5.1 to capture completion times of the jobs processed in a known fixed sequence, the only way possible to move from node  $(C\sigma_{(i),j})$  to  $(C\sigma_{(i+1),j-1})$  is along the diagonal arc with zero weight. Thus, to move from node  $(C\sigma_{(i),m})$  to node  $(C\sigma_{(i+M),0})$ , 'M' nodes, along the diagonal path to the right, need to be traversed. In case  $M \geq m$ , node  $(C\sigma_{(i+M),0})$  will be reached in m steps using the diagonal arcs and no additional arcs need to be drawn in the network to establish the CONWIP precedence constraint. Thus, the original network remains unchanged. This follows from the fact that; a system with 'm' machines (under the case of no buffer) can hold at most m jobs at any time. Now, if  $M \geq m$ , then the  $(M+1)^{\text{th}}$  job cannot enter before the first job leaves, due to the insufficient capacity (m) of the system.

Case II:  $M < m$

In this case, we have to consider the extra arcs added due to the CONWIP constraint. The critical path may include the arcs joining the nodes  $(C\sigma_{(i),m})$  and  $(C\sigma_{(i+M),0})$ . We thus have to make sure that the  $(i+M)^{\text{th}}$  job on the first machine, is not in the same Epoch as the  $i^{\text{th}}$  job on the last machine. We thus need to push the  $(i+M)^{\text{th}}$  job on the first machine, by one Epoch. The need of this modification is explained by the fact that we want to hold statement 2, mentioned earlier, true. This will create a gap in the Epoch on the first machine. We follow the same procedure for the  $(i+M+1)^{\text{th}}$  job on the second machine and so on creating a slanting gap between regions as shown in the schematic Gantt chart in Figure 5.4 below. For an example, refer to Figure 5.5. The entire sequencing area can be divided into the following four types of regions (marked on the Figure):



**Fig 5.4:** Schematic Graph of a Typical Flow Shop Gantt chart with CONWIP level condition

Each region spans a few or even a single epoch. Region 3 can be further divided into region 3a and 3b.

It can be noticed that region 1 is same as region 1 for the  $m < M$  case. Thus the same formula applies here, which is restated as follows:

$$\sum_{k=1}^l X_{k,l-k+1} = 1 \text{ for all values of } l = 1 \text{ to } m \quad (5.4.2-6)$$

We now device formulae, for only the first parallelogram, and region 3b. Then we use those results, to generalize to any parallelogram or region anywhere in the long sequence.

Region2:

$$\sum_{k=l-M+1}^l X_{k,l-k+1} = 1 \text{ for all values of } l = M + 1 \text{ to } m \quad (5.4.2-7)$$

Regions 3b:

$$\sum_{k=1}^{l-m} X_{k,l-k-m+M+1} = 1 \text{ for all values of } l = n + 1 \text{ to } n + m - 1 \quad \left. \vphantom{\sum_{k=1}^{l-m}} \right\} \text{ For all } l = m + 1 \text{ to } m + M \quad (5.4.2-8)$$


Region 3a:

$$\sum_{k=l-M+1}^m X_{k,l-k+1} = 1 \text{ for all values of } l = n + 1 \text{ to } n + m - 1$$

We test these results on any typical sequence. The Gantt chart of this sequence is as shown in the Figure 5.5

For Region 1:  $l = 3$

$$\sum_{k=1}^3 X_{k,4-k} = X_{1,3} + X_{2,2} + X_{3,1} = 1$$

  
 Jobs in one Epoch

For Region 2:  $l = 4$

$$\sum_{k=4-3+1}^4 X_{k,5-k} = \sum_{k=2}^4 X_{k,5-k} = X_{2,3} + X_{3,2} + X_{4,1} = 1$$

For Region 3:

We have Region 3 = Region 3a + Region 3b

$$\sum_{k=1}^{l-m} X_{k,l-k-m+M+1} + \sum_{k=l-M+1}^m X_{k,l-k+1} = 1 \quad \text{for all } l = m + 1 \text{ to } m + M$$

For  $l = 5$

$$\sum_{k=1}^1 X_{k,5-k} + \sum_{k=3}^4 X_{k,6-k} = 1$$

or

$$X_{1,4} + X_{3,3} + X_{4,2} = 1$$

For  $l = 6$

$$\sum_{k=1}^2 X_{k,6-k} + \sum_{k=4}^4 X_{k,7-k} = 1$$

or

$$X_{1,5} + X_{2,4} + X_{4,3} = 1$$

Now, as defined earlier in the variable definitions,  $i$  is an integer, which denotes the number of regions of a particular type before the next region of the same type.

Thus  $i = 4$  for region 3 would be the fourth region of type 3 in succession.

We now extend these results to any  $i^{\text{th}}$  region of any type as follows:

Region 1:

$$\sum_{k=1}^l X_{k,l-k+1} = 1 \quad \text{for all } l = 1 \text{ to } m \quad (5.4.2-9)$$

Region 2:

$$\sum_{k=l-(i-1)m-M+1}^{l-(i-1)m} X_{k,l-(i-1)m-k+1+iM} = 1 \quad \text{for all } l = (i-1)m + M + 1 \text{ to } im \quad (5.4.2-10)$$

Region3:

$$\sum_{k=1}^{l-im} X_{k,l-k-im+iM+1} + \sum_{k=l-(i-1)m-M+1}^m X_{k,l-k-(i-1)m+(i-1)M+1} = 1 \quad \text{for all } l = im + 1 \text{ to } im + M \quad (5.4.2-11)$$

Region 4:

$$\sum_{k=l-\lfloor \frac{n}{M} \rfloor_{m+1}}^m X_{k,l-k-\lfloor \frac{n}{M} \rfloor_{m+1}} = 1 \quad \text{for all } l = im + M + 1 \text{ to } n + m - 1 \quad (5.4.2-12)$$

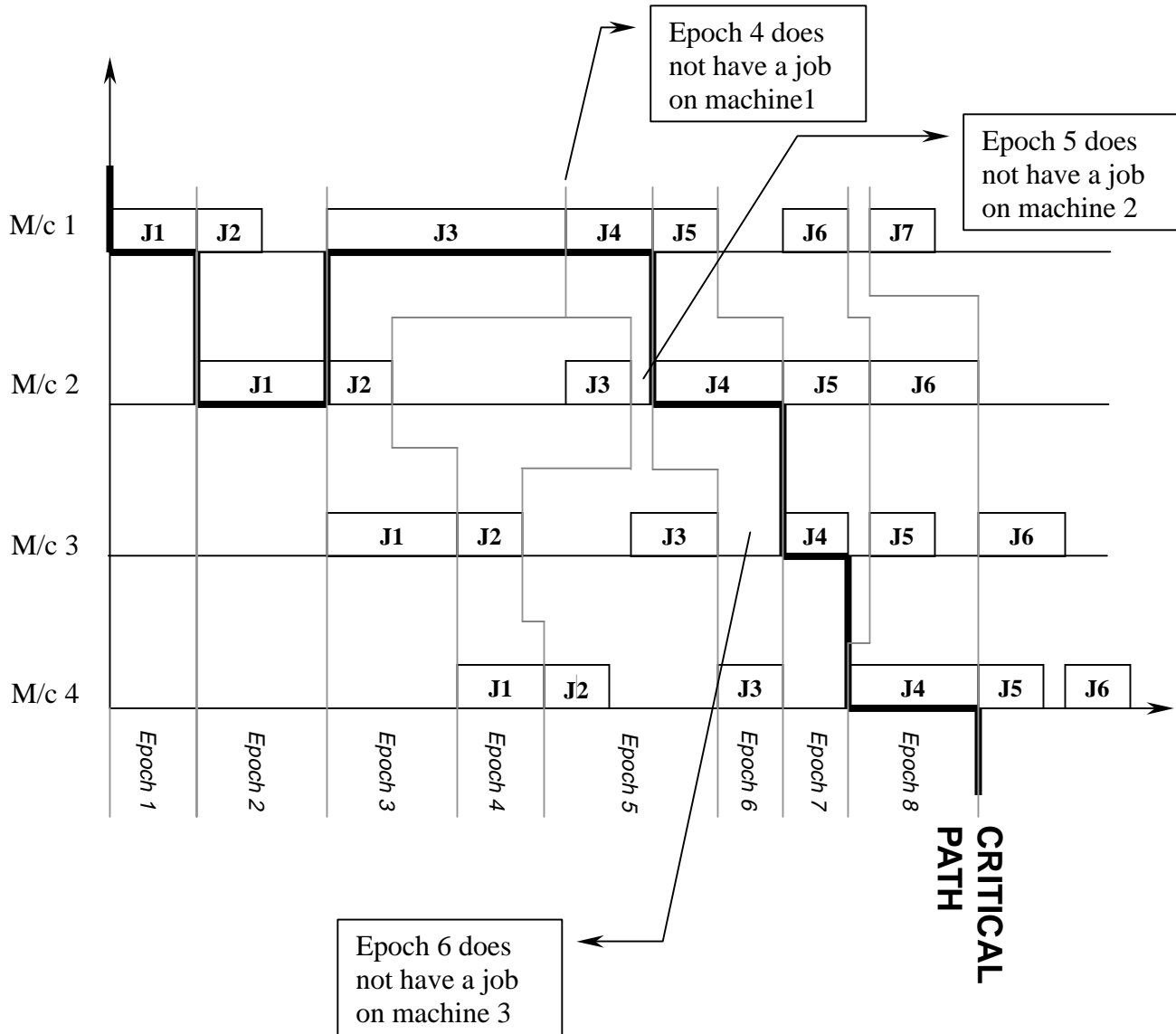
Regions 1 and 4 will appear only once in any sequence. These conditions are true if and only if statement 2 is true. For the case of  $m \geq M$ , we can support the statement both theoretically and by means of an example, that this statement is false. We then add a term to these equations to restore their validity.

Supportive Explanation:

Consider the network shown in Figure 5.5. If  $m = 4$  and  $M = 3$ , then job 4 on machine 1 is pushed in the slot 5. Thus, the fourth Epoch has no job on machine1 as shown on the network. The arc from  $(C\sigma(4),0)$  to  $(C\sigma(4),1)$  [and from  $(C\sigma(3),1)$  towards  $(C\sigma(4),1)$ ] is no longer in the epoch. Also, we have to join node  $(C\sigma(4),0)$  to node  $(C\sigma(1),3)$  by an arc of zero weight to consider the precedence of  $(C\sigma(1),3)$  before  $(C\sigma(4),0)$ . To generalize, arcs joining the nodes

$(C\sigma_{(i),m})$  and  $(C\sigma_{(i+M),0})$  have to be added to the network and should also be included in the constraints. Thus our statement 2 is proved wrong.

The following example makes it clearer.



**Fig 5.5:** Gantt chart showing the concept of Epochs in a CONWIP system

We can clearly see that no job in epoch 4 is on the critical path. Thus, we need to generalize the Statement 2 as, either one on slot 4 or job 3 on machine 1 is on the critical path.

Thus, the revised Statement 2 states:

*At most one of the jobs in every scheduling epoch can be on the critical path. In case none of the jobs of an epoch on a machine lies on the critical path, then a job from the previous epoch on that machine should be on the critical path.*

For any general case we need to add the following 1,0 integer to the equations:

$$X_{l-iM, iM} = 1$$

This, however, would not affect region 1 and 4. The equations for region 2 and 3 would now be as follows:

Region 1:

$$\sum_{k=1}^l X_{k, l-k+1} = 1 \quad \text{for all } l = 1 \text{ to } m$$

Region 2:

$$\sum_{k=l-(i-1)m-M+1}^{l-(i-1)m} X_{k, l-(i-1)m-k+1+iM} + X_{l-iM, iM} = 1 \quad \text{for all } l = (i-1)m + M + 1 \text{ to } im$$

Region3:

$$\sum_{k=1}^{l-(im)} X_{k, l-k-im+iM+1} + \sum_{k=l-(i-1)m-M+1}^m X_{k, l-k-(i-1)m+(i-1)M+1} + X_{l-iM, iM} = 1 \quad \text{for all } l = im + 1 \text{ to } im + M$$

Region 4:

$$\sum_{k=l-\left\lfloor \frac{n}{M} \right\rfloor_{m+1}}^m X_{k, l-k-\left\lfloor \frac{n}{M} \right\rfloor_{m+1}} = 1 \quad \text{for all } l = im + M + 1 \text{ to } n + m - 1$$

The above equations can be inserted into the model presented in section 5.4.1 to represent the CONWIP system with finite intermediate buffers and cyclic flow of multiple classes of jobs. Though this model cannot be solved in Polynomial time, it provides us with interesting insights concerning the issue of finding the critical path for any permutation. The complete model is as presented below:



*Objective function:*

Max L, where

$$L = \sum_{k=1}^m \sum_{l=1}^{n+m-1} P_{kl} X_{kl}$$

*Constraints (constructed in earlier sections for all 4 types of regions):*

Region 1:

$$\sum_{k=1}^l X_{k,l-k+1} = 1 \quad \text{for all } l = 1 \text{ to } m$$

Region 2:

$$\sum_{k=l-(i-1)m-M+1}^{l-(i-1)m} X_{k,l-(i-1)m-k+1+iM} + X_{l-iM,iM} = 1 \quad \text{for all } l = (i-1)m + M + 1 \text{ to } im$$

Region3:

$$\sum_{k=1}^{l-(im)} X_{k,l-k-im+iM+1} + \sum_{k=l-(i-1)m-M+1}^m X_{k,l-k-(i-1)m+(i-1)M+1} + X_{l-iM,iM} = 1 \quad \text{for all } l = im + 1 \text{ to } im + M$$

Region 4:

$$\sum_{k=l-\lfloor \frac{n}{M} \rfloor_{m+1}}^m X_{k,l-k-\lfloor \frac{n}{M} \rfloor_{m+1}} = 1 \quad \text{for all } l = im + M + 1 \text{ to } n + m - 1$$

$X_{ji} \in \{0,1\}$  for all  $i,j$

The following sections of this chapter are devoted to the explanation of the actual method proposed to solve the whole sequencing problem. As mentioned in the earlier sections, a Fast Insertion Heuristic (FIH) along with a modified form of Floyd's algorithm is used. We later try different modifications of the FIH, to determine the best strategy to choose the order in which the jobs should enter the partial job set.

## 5.5 *FIH in its simplest form (As suggested by Nawaz et al (1983))*

### The Fast Insertion Heuristic (FIH) for flow shop sequencing problems

Step 1. For each job  $i$ , compute:

$$T_i = \sum_{j=1}^m p_{ij}$$

Step 2. Arrange the jobs, in a LIST, in decreasing order of  $T_i$ .

Step 3. Pick the first two jobs on the LIST.

Find the best partial sequence of these two jobs, and make it the 'current sequence'.

Set  $i=3$

Step 4. Pick the job in the  $i$ -th position on the LIST.

Insert it in all possible positions in the 'current sequence' and evaluate the sequence.

Make the best resultant partial sequence the 'current sequence'.

Step 5. If  $i \leq n$ , then set  $i = i + 1$  and go to Step 4.

To select the best partial sequence in the FIH, a modified version of Floyd's Heuristic is employed. It can be observed from the network that a large percentage of arcs between the nodes have a weight of zero. This fact can be used to our benefit to simplify the amount of calculations, by using the Floyd's method, in which successive matrix manipulations to be performed are simplified. The FIH algorithm is mentioned in the following section.

## 5.6 Modified Floyd's Algorithm for determination of the critical path

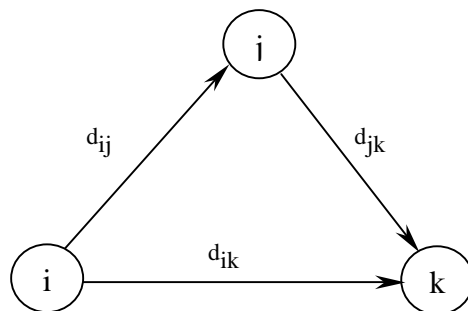
We first explain the Floyd's algorithm and its underlying principle.

Floyd's algorithm is a more generalized algorithm compared to Dijkstra's because it determines the shortest route between *any* two nodes in the network. We thus need to modify the algorithm to result in the length of longest path between any two nodes. The algorithm represents an  $nm$ -node network as a square matrix with  $nm$  rows and  $nm$  column. Entry  $(i,j)$  of a matrix gives the distance  $d_{ij}$  from node  $i$  to node  $j$ , which is finite if  $i$  is linked directly to  $j$ , and infinite otherwise.

The Floyd's algorithm is based on a simple intuitive logic. It states that if the travel to a node from its preceding node can be made shorter by traveling via another node, which is linked to the preceding node, it is always advisable to travel via the extra node so that the travel distance is minimum. This can be stated mathematically as follows:

Given three nodes  $i$ ,  $j$  and  $k$  as shown in the figure 5.6, with the connecting distances shown on three arcs, it is shorter to reach  $k$  from  $i$  passing through  $j$  if

$$d_{ij} + d_{jk} < d_{ik}$$



**Fig 5.6:** A Directed Graph Showing Nodes  $i$ ,  $j$ , and  $k$

In such a case, it is optimal to replace the direct route  $(i,k)$  by the sum of routes  $(i,j)$  and  $(j,k)$ . A systematic method to exhaust all routes joining every node set  $i,j,k$ , is to first form a matrix  $D_0$ . This matrix is formed by the distances between all the possible pairs of nodes in the network. A

row and a column is chosen in such a way that they intersect at a diagonal element. A triple operation exchange is then applied to the chosen nodes using the following steps:

(These steps are as mentioned in the book Operations Research, An Introduction by Hamdy A. Taha [6<sup>th</sup> edition])

**Step 1:** Define the starting distance matrix  $D_0$  as explained earlier. The diagonal elements are marked with (-) to indicate that they are blocked. Set step number  $k$  equal to 1.

**General step  $k$ :** Define row  $k$  and column  $k$  as pivot row and pivot column. As explained earlier, row  $k$  and column  $k$  intersect at a diagonal element. Apply the triple operation to each element  $d_{ij}$  in  $D_{k-1}$  (i.e. on all elements in  $D_{k-1}$ , which are not on the diagonal and not on the selected row and column  $k$ ), for all  $i$  and  $j$ . If the condition

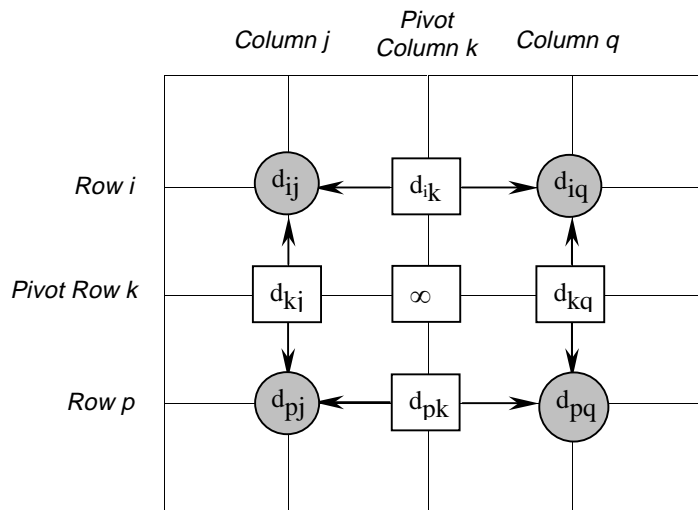
$$d_{ik} + d_{kj} < d_{ij}, \quad (i \neq k, j \neq k, \text{ and } i \neq j)$$

is satisfied, make the following change:

*Create new matrix  $D_k$  by replacing  $d_{ij}$  in  $D_{k-1}$  with  $d_{ik} + d_{kj}$*

*Step  $k$  of the algorithm can be more readily explained by representing  $D_{k-1}$  as shown in figure 5.7. The intersection of row  $k$  and the column  $k$  defines the pivot element. Row  $i$  represents any general row 1, 2, ...,  $k-1$  and row  $p$  represents any row  $k+1, k+2, \dots, n$ . Similarly, column  $j$  represents any column 1, 2, ...,  $k-1$  and column  $q$  represents any column  $k+1, k+2, \dots, n$ . The triple operation can be applied as follows:*

*If the sum of the elements on the pivot row  $k$  and the pivot column (shown by squares) is greater than the associated intersection element (shown by circles), then it is optimal to replace the intersection distance (or the values in the circles) by the sum of the pivot distances (values in the squares).*



**Fig 5.7:** Schematic Diagram showing the Pivot Manipulation in a Floyd's Matrix

After  $n$  steps the longest route between any two nodes  $i$  and  $j$  can be determined as the entry  $d_{ij}$  in the matrix  $D_n$ .

This algorithm gives the shortest path between any two nodes. It is, however, desired to find the longest path between the first and the last node of our processing time network. Thus, we need to modify the algorithm. The modification pertains to additional checks at *step k*. The modified algorithm, is as follows.

Modified Floyd's Algorithm:

**Step 0:** Define the starting distance matrix  $D_0$  as given subsequently. The diagonal elements are marked with (-) to indicate that they are blocked. Set  $k = 1$ .

**General step k:** Define row  $k$  and column  $k$  as pivot row and pivot column. Apply the triple operation to each element  $d_{ij}$  in  $D_{k-1}$ , for all  $i$  and  $j$ . The if conditions are as follows (they are given in a programmer friendly way to help in its coding and understanding):

If ( $i \neq k, j \neq k, \text{ and } i \neq j$ )

If  $d_{ik} \neq \infty$  and  $d_{kj} \neq \infty$

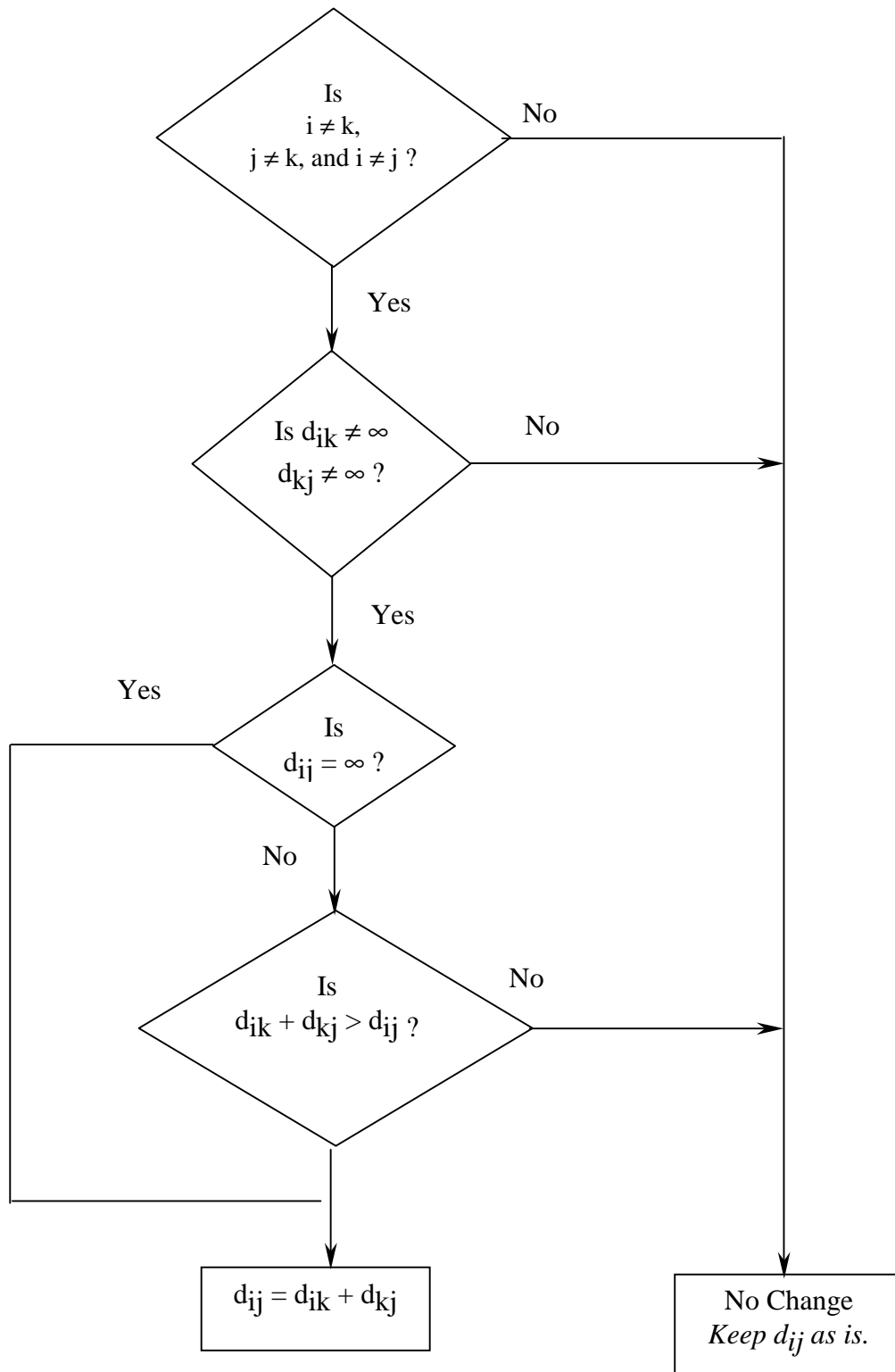
If  $d_{ij} = \infty$

or  $d_{ij} \neq \infty$  and  $d_{ik} + d_{kj} > d_{ij}$

then make the following change:

*Create  $D_k$  by replacing  $d_{ij}$  in  $D_{k-1}$  with  $d_{ik} + d_{kj}$*

These additions can be shown in the form of a flow chart as follows:



**Fig 5.8:** Flow Chart Showing the Additions to the Floyd's Heuristic



The FIH heuristic calls the Modified Floyd's algorithm (MFA) in each iteration  $i$  number of times, if  $i$  is the number of jobs in the partial job set at that iteration. In other words, the  $i^{\text{th}}$  job to be introduced in the permanent set have  $i$  possible slots it can be introduced. For each of such possible combination, the MFA is called to return the value of the critical path. In the implementation of the MFA, the following points need to be highlighted, as they are further modifications to suit this problem:

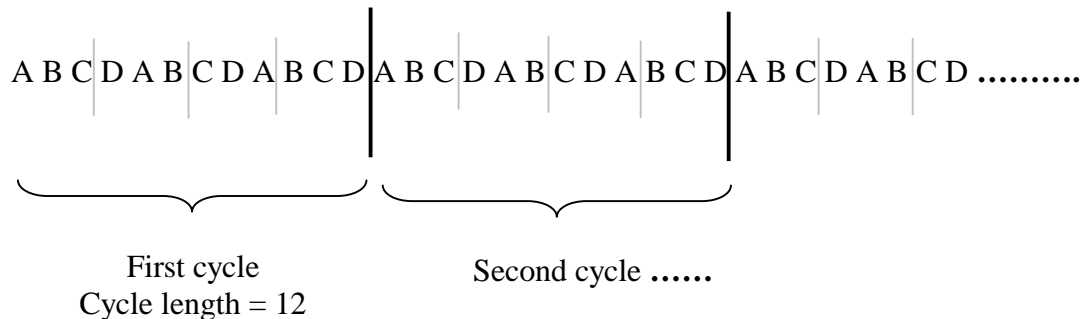
The critical path passes from the first job on machine 1 to the last job on the last machine.

The last step is to include the cyclic nature of the sequencing policy. To that end, it is necessary to study the entire cycle rather than just one set of jobs. At times depending upon the value of CONWIP level  $M$  and the number of jobs in the partial job set, more than one set of jobs might need to be considered to study one complete cycle. The following explains this fact.

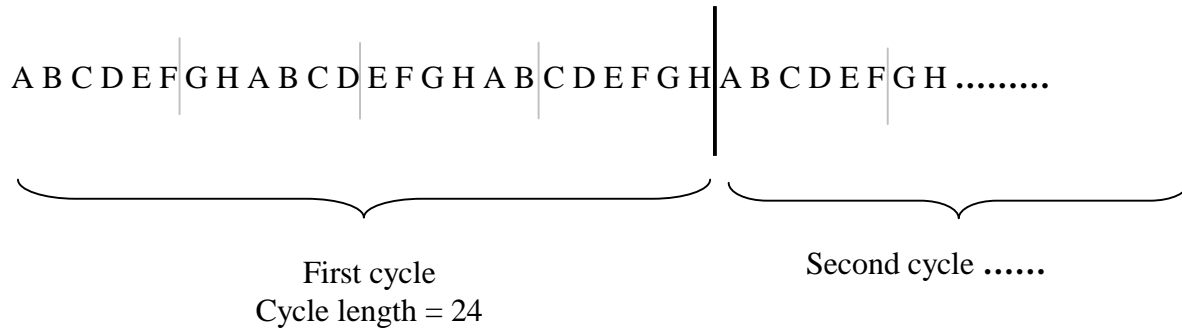
### 5.6.1 Determination of Cycle Length

**Statement:** *The cycle length, which needs to be considered in the determination of the critical path, is the Least Common Multiplier of the CONWIP level  $M$  and the number of job types  $n$ , in the partial job set.*

To illustrate, let there be 4 jobs in the partial job set and the CONWIP level is 3. Then the length of a cycle is  $LCM(4, 3) = 12$ . Thus, if the job types are A, B, C, D, E then the following insert shows how a cycle is determined:



Now consider a case when the cycle length is the actual LCM of the two and not their product. For example, if the number of jobs are 8 and CONWIP level is 6 then the cycle length is  $\text{LCM}(8, 6) = 24$  and not 48 as shown below



The critical path now spans over a network, covering jobs multiple times and not just once as in one MPS. Thus, the critical path length would be the flow time which could be defined as the time interval between the instant the job type A enters the first machine to the instant the job type A enters again 4 MPS later. The number of jobs at any iteration in the FIH is, however, the same, which is  $\text{LCM}(M,n)$ . The partial sequence corresponding to the shortest path length is selected and the new job is fixed in relation to the other jobs in the set to form the optimal partial job set at that iteration.

### 5.6.2 CONWIP Level & Cyclic Sequence Considerations in the Network Representation

Another important issue to be noted is the inclusion of the CONWIP level in the network representation in addition to the fact that the sequence is cyclic. Extra arcs have to be added to the network to include these conditions. The nodes  $C\sigma_{(i),m}$  on the last machine  $m$  for any job  $i$  are joined to nodes  $C\sigma_{(i+M),0}$  on the zero<sup>th</sup> machines. The zero<sup>th</sup> machine nodes represent the starting times of the jobs on the first machine and are so named only for convenience even though there is no zeroth machine physically. All these arcs have a weight of zero. These establish the precedence constraints between the start of  $(i+M)$ <sup>th</sup> job to the end of the  $i$ <sup>th</sup> job on the last machine. These arcs not only span over a single MPS but also between the corresponding

jobs oversuccessive MPS for an entire cycle length, which is  $M$ , MPS sets. It is important to note that the network should be first augmented to include the cyclic condition and then should be followed by the addition of arcs to include the CONWIP condition.

### 5.6.3 Computational Complexity of the method to determine the best sequence

We have thus far included all the unique conditions, which this research assumes. The computational complexity of this method is limited by the FIH heuristic, since the MFA only adds a finite number of computations at every iteration, depending upon the matrix size (or the iteration number). At any iteration  $i$ , the number of computations for finding the length of the critical path by the MFA is  $(\psi - 1)^2$ , where  $\psi = (\text{Number of Machines}) \times (\text{Number of Jobs})$ . There are  $n$  such iterations, where  $n$  is the number of jobs to be sequenced. Newaz et al (1983) mention in their research that the complexity of the FIH heuristic is of the order of  $O(n^2)$  since, at each iteration, at most  $n$  positions are considered and this step is repeated  $(n-2)$  times. It can further be shown that the number of enumerations in the algorithm is:

$$\frac{n.(n+1)}{2} - 1$$

Thus the use of MFA does not change the total complexity and is of the order of  $O(n^2)$ .

## **5.7 Experimental Study to test the Quality of the Sequences Generated**

The methodology proposed to determine the backlog sequence consists of two steps. The first step deals with the generation of partial sequences and the second one pertains to the selection of the best solution from among the possible ones at any iteration. Several partial solutions are generated using the modified FIH heuristic developed by Nawaz et al (1983). Selection of an appropriate solution from among them is accomplished by finding the critical path length between the end nodes of the network, which is a measure of the flow time. This is done using the Modified Floyd's algorithm. Although the method of generating the sequences is only an approximate one, the method of calculating the critical path length is optimal. Thus, the accuracy of the entire problem hinges upon the quality of the solution to the first problem of generating good partial sequences.

The final sequence suggested by the proposed procedure need not be optimal. An experimental study was undertaken to determine the quality of the sequences obtained. The solution generated by the proposed heuristic was compared with the corresponding figures generated by another method, and whose solutions are known to be reliable. To assure a fair comparison, various factors, which affect the final solution, have to be considered. The effects of each of these factors have to be studied by quarantining their effects on the problem solution. In this study, we compared the solution obtained by the proposed procedure with the optimal sequence obtained via enumeration and evaluation using simulation.

### **5.7.1 Experiments Design**

The Modified Fast Insertion Heuristic and Modified Floyd's Algorithm (MFIH + MFA) heuristic was coded using C++ and run on a Windows NT platform. The actual code for one variation of the heuristic is included in Appendix C. The output was displayed for analysis in the form of text files. Two such typical output files that show the sequential building of the best sequence and the Floyd matrix at every iteration are included in the Appendix D.

The simulation models were built using the Taylor II simulation software, courtesy of Ericsson, Lynchburg. The performance measure used for comparison was the flow time (or CT). From the preliminary study done at the inception of this research, it was determined that the following factors affect the final solution value and thus have to be carefully studied to consider their effect.

- The CONWIP level M
- Problem size (number of job types)
- Level of stochasticity

It is not possible to study the effect of all these input factors on the quality of the flow time when a high level of stochasticity in the processing times is involved. This is due to the fact that the exact optimal solution cannot be determined in a stochastic case. Such studies would involve probabilities of achieving a close to optimal or optimal solutions, and would muddle the issue of appraisal of the closeness of the flow time to the optimal value obtained by enumeration and evaluation using simulation. Thus, it was found appropriate to test the effectiveness of the first two factors using deterministic time values to determine a level of accuracy, and to study the third factor separately.

The Table 5.7.1-1 summarizes the experiments performed with the deterministic processing times:

**Table (5.7.1-1):** *Summary of the Experiments Performed*

Problem size	CONWIP level		
	2	4	5
3	2	4	5
4	3	4	5
5	-	4	5
7	-	4	6

## 5.7.2 Results and analysis of the experiments

### *Experiment 1:*

The FIH heuristic as suggested initially chose the jobs to enter the partial job set according to the summation of the processing times of all the jobs on different machines. However, to fine-tune the heuristic, it was decided to employ different strategies to select the entering job. These are as mentioned below:

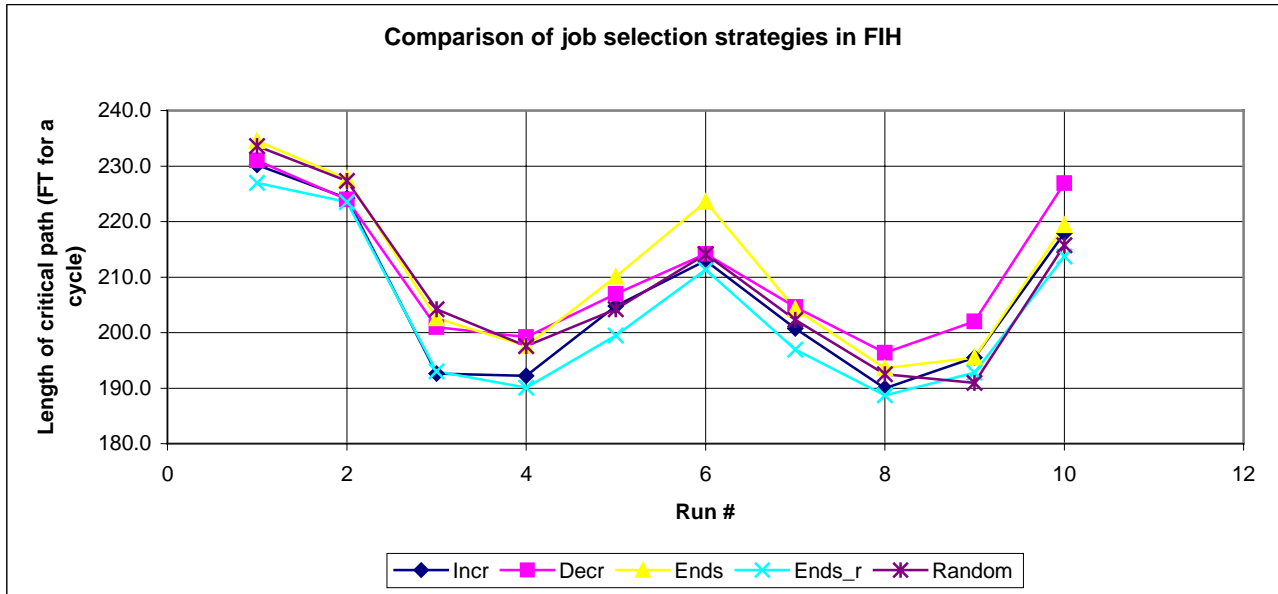
- Incr: This strategy is the same as that proposed by Newaz (1983).
- Decr: This strategy is exactly opposite to the one proposed and selects the jobs in decreasing order of the summation of processing times of the jobs on the machines.
- Ends: This strategy orders the jobs just as the Incr strategy. However the jobs are chosen as the first job as the first to enter, last job as the second to enter, second job as the third to enter and so on.
- Ends\_r: This was an exactly opposite to the Ends strategy.
- Random: No specific rule. The jobs were chosen in the order they appeared in the time matrix.

The Table 5.7.2-1 shows results of experiments performed to determine the optimal strategy. Ten runs were made with different processing time matrices, and all the heuristics were run for every time matrix.

**Table (5.7.2-1):** *Summary of results of experiments of comparison between job selection strategies*

Method	Length of critical path or Flow time of a cycle									
	Run # 1	Run # 2	Run # 3	Run # 4	Run # 5	Run # 6	Run # 7	Run # 8	Run # 9	Run # 10
Incr	230.2	224.2	192.6	192.2	204.8	213.0	200.8	190.0	195.5	217.9
Decr	231.0	224.0	201.0	199.2	207.0	214.2	204.6	196.4	202.0	226.9
Ends	234.5	227.7	202.6	197.6	210.1	223.6	204.3	193.6	195.5	219.4
Ends_rev	227.0	223.5	193.0	190.1	199.5	211.4	197.0	188.7	192.8	213.7
Random	233.6	227.3	204.2	197.6	204.2	214.1	202.3	192.5	191.0	215.8

The critical path values are plotted in Figure 5.9 for easy comparison.



**Figure 5.9:** Graph of Flow time resulting from different job selection strategies

The graph clearly shows the superiority of the Ends and Ends\_r selection strategy. Thus, only these two strategies were used in future experiments, where the actual quality of the heuristic was tested by comparing it with an enumerative simulation scheme. To ensure that the limited size of the first experiment did not result in the superiority of the selected strategies, all the strategies except Random were used in several other cases of the second experiment but not all.

*Experiment 2:*

The second experiment determines the quality of the proposed method, by comparing the value of flow time of the sequence suggested by the heuristic, to that obtained by the enumerative simulation scheme. Deterministic processing times are assumed in this experiment. Averages of the processing time distributions are generated using a uniform distribution ranging from 5 to 25. To facilitate fair comparison between runs, the same time averages were used for all the runs.

The optimal solution was obtained by running the simulation model for each possible sequence, and by determining the least flow time. The methodology proposed in this research



returns the value of the length of the critical path, which is a measure of Flow time. However, since the Flow time as we have defined and used in all the earlier chapters, is for only one MPS, and not M (CONWIP level) MPSs, we cannot compare the two. Thus, the value of flow time for the sequence generated by the heuristic is found by running that sequence using a simulation model. This flow time was compared with the optimal value of flow time obtained by complete enumeration using the same simulation model. The percentage deviation from the optimal was calculated for all these cases, for comparison.

The following points were noted:

- The heuristic was optimal in almost all the cases.
- The probability of the heuristic resulting in an optimal solution is higher for smaller size problems than for bigger size problems. This is corroborated by the results of the experiments carried out with 3 machines, which result in all optimal solutions while the one with the 4 jobs gave optimal values for all except 3 cases. The five-job experiment resulted in 2 non-optimal solutions even when the experiment size was small. This is, however, expected from any heuristic solution.
- The heuristic gave an optimal solution in one case of the experiment with 7 jobs and a very close to optimal solution for the second case again with 7 jobs, but a higher CONWIP level.
- The largest deviation obtained was of 4.016 % in the 5-job case. The average percentage error was less than one percent.

Refer to Appendix D for the detailed results of the experiments for the 3, 4 and 5 job case.

### *Experiment 3: Stochastic Processing Times*

It is assumed that in reality, the deviation of the processing times from their respective averages does not exceed 10%. Thus, two levels of stochasticity were chosen, namely, 5% and 10%. The 4-job, 4-machine case was used for this study. Ten replications of each deviation level were run; with Random numbers generated from generators 1 through 50 in Taylor II<sup>TM</sup>. The processing times for the above experiment were obtained from a uniform distribution between 5

through 25. Tables (5.7.2-5) and (5.7.2-6) show the times for both levels of deviation. The processing time distribution averages is shown in the Table (5.7.2-4).

**Table (5.7.2-2): Processing time distribution averages**

	J1	J2	J3	J4
	Job type A	Job type B	Job type C	Job type D
M1	15	9	16	12
M2 (CBN)	8	22	23	11
M3	6	5	11	13
M4	11	7	9	12

**Table (5.7.2-3): Processing time distribution extremes with deviation level of 5% ( $\pm 2.5\%$ )**

		Job type A	Job type B	Job type C	Job type D
M1	Upper bound	15.375	9.225	16.4	12.3
	Lower bound	14.625	8.775	15.6	11.7
M2	Upper bound	8.2	22.55	23.575	11.275
	Lower bound	7.8	21.45	22.425	10.725
M3	Upper bound	6.15	5.125	11.275	13.325
	Lower bound	5.85	4.875	10.725	12.675
M4	Upper bound	11.275	7.175	9.225	12.3
	Lower bound	10.725	6.825	8.775	11.7

**Table (5.7.2-4): Processing time distribution extremes with deviation level of 10% ( $\pm 5\%$ )**

		A	B	C	D
M1	Upper bound	15.75	9.45	16.8	12.6
	Lower bound	14.25	8.55	15.2	11.4
M2	Upper bound	8.4	23.1	24.15	11.55
	Lower bound	7.6	20.9	21.85	10.45
M3	Upper bound	6.3	5.25	11.55	13.65
	Lower bound	5.7	4.75	10.45	12.35
M4	Upper bound	11.55	7.35	9.45	12.6
	Lower bound	10.45	6.65	8.55	11.4

Table (5.7.2-3) gives the values of the Flow time obtained using simulation for the two chosen deviation levels.

**Table (5.7.2-5):** Flow times for the stochastic case, obtained using simulation

% deviation	Sr. No.	ABCD	<i>BACD (optimal)</i>	ACBD	ADBC	CBAD	BDCA	Rand gen. no.
<b>5%</b>	1	52.165	50.872	54.121	51.982	52.691	52.802	1
	2	52.232	50.822	54.312	51.891	52.721	51.821	10
	3	52.412	50.851	54.246	51.872	52.735	51.796	15
	4	52.183	50.843	53.942	51.914	52.729	51.799	20
	5	52.318	50.831	54.001	51.941	52.696	51.83	25
	6	52.162	50.871	54.041	51.894	52.719	51.793	30
	7	52.212	50.869	54.295	51.911	52.728	51.823	35
	8	52.191	50.858	54.213	51.9	52.722	51.83	40
	9	52.213	50.867	54.247	51.881	52.689	51.823	45
	10	52.241	50.881	54.381	51.894	52.728	51.825	50
<b>10%</b>	1	52.167	50.838	54.004	51.985	52.735	51.796	1
	2	52.132	50.873	53.955	51.957	52.692	51.821	10
	3	52.053	50.852	54.11	51.921	52.842	51.723	15
	4	52.101	50.799	53.874	52.042	52.728	51.591	20
	5	52.069	50.824	54.251	52.221	52.759	51.924	25
	6	52.11	50.88	54.521	51.871	52.592	51.841	30
	7	52.121	50.782	53.87	51.724	52.812	51.728	35
	8	52.067	50.871	53.921	52.121	52.652	51.49	40
	9	52.131	50.821	54.12	51.692	52.91	51.816	45
	10	52.991	50.869	53.891	52.012	52.731	51.732	50

As can be observed from the Table (5.7.2-3), the maximum of the flow times for the 10 replications for the best sequence is less than the minimum for other sequences. If this were not true, it would be necessary to perform statistical tests, like the F-test to determine if the differences between the flow times are significant. If they were significant, then it would be necessary to choose the minimum to determine the optimal. The proposed method also resulted in BACD as the optimal sequence.

# Chapter 6: Summary and Conclusions

## 6.1 *Summary and Conclusions*

This research work has been motivated by the need to develop efficient and practical solutions to the problem of achieving a tighter production control of a CONWIP system. Two important issues were addressed in this research, for such a CONWIP flow shop. These are as follows:

- Determination of the optimal CONWIP level  $M$ . As discussed in the first few Chapters and also in the literature, a lower than optimal work in progress results in an underutilized system, with a low throughput. On the other hand, a higher than optimal WIP, does not increase the throughput any higher than the capacity of the system, but results in a higher flow time. Thus, just enough WIP level should be maintained so that the throughput is up to the system capacity.
- Determination of the optimal sequence of job types .The ordering of the CONWIP backlog list, which specifies the sequence, affects the performance of the system and, therefore, is important to determine for system effectiveness.

Benefits of the study have been evaluated with respect to the following two, commonly used parameters:

- Total flow time, of all job types through the system. It is shown in earlier studies, that this is the same as the total cycle time.
- Interdeparture time of jobs, measured at the last machine.

As explained earlier, the choice of the optimal WIP level, ensures optimal throughput or in other words, the interdeparture time. Thus, the second problem is evaluated only in terms of the flow time, since the problem of the determination of the flow time is studied only after the optimal CONWIP level is known from the solution of the first part. For each performance parameter, the percentage deviation from the optimal value is provided as an indication of the solution quality.

This research is different than the studies undertaken in the past due to the collective consideration of the following additional constraints, which make the system under study closer to an actual production environment:

- Stochastic processing time
- Finite CONWIP level
- Cyclic sequencing of jobs. A job cycle is repeated for a large number of times, as in normal flow line production set up.

Chapter 1 describes the background and motivation for the research. The basic concepts are introduced and control systems such as PULL and PUSH are compared with the CONWIP system. The Chapter also describes other relevant CONWIP control related issues, which are cardinal in its implementation

The problems of finding the optimal CONWIP level ( $M^*$ ) and the optimal sequence have been described in the literature review, which is included in Chapter 2. The literature provides an extensive body of knowledge on the use of the queuing theory to address FMS production control issues as well as the general flow shop sequencing issue. However, both these problems have not been studied in the light of the additional constraints mentioned above. Although earlier research has studied the effect of these constraints on the above mentioned performance parameters, all the factors were not considered together in one model.

First introduced by Reiser (1979) and later modified by Hildebrant (1980), we use a modified form of the “Mean Value Algorithm” (MMVA), to address the issue of the determination of the optimal CONWIP level. The problem is modeled as a closed queuing network problem and is described in Chapter 4. Such networks do not possess a closed product

form solution, which can be determined by analytical methods. In spite of these difficulties, the solution obtained by MMVA, is entirely in terms of mean queue size, mean waiting time and the throughput. This new analysis leads to a simpler algorithm with a better numerical behavior than the previous ones and also reduces the computational complexity as compared to the convolution algorithms. Past studies have used queuing theory to study a FMS system, but have not considered the limitations on inter-station storage capacities. The presence of such a limitation may result in blocking of stations, due to the succeeding station having reached its storage capacity. If this station is blocked for a long enough time, the phenomenon called “Cascading Bottlenecks” may occur and is described in the Chapter 5. Thus, there is a need to include a blocking time term in the MMVA. An iterative procedure to approximate this time is proposed. The MMVA gives good results for the two metrics defined, namely, average queue lengths at buffers and Average wait times at buffers, with a slightly better result for the average queue length. The percentage error was observed to increase slightly with CONWIP level ( $M$ ), and the size of the problem (number of jobs and machines). Even for a CONWIP level, almost twice the optimal level, the error was within 10%. Thus the results obtained are of good quality, especially considering the inherent complexity of the problem, which is further enhanced by the stochasticity in the processing times.

The problem of determining the optimal sequence is modeled as an IP program. The concept of scheduling epochs is introduced and used to build an IP model. Insights generated from the model reveal that the problem at hand can be broken down into two sub-problems, namely that of generating good quality partial sequences, and that of determination of the flow time of these partial feasible solutions. The first part of the problem is NP-hard, as explained in Section 5.2. The second part however, can be solved to optimality in polynomial time. To address the first part, a modified version of the Fast Insertion Heuristic (MFIH), originally proposed by Newaz et al (1983), is suggested. Different methods of choosing the entering job type were tried, and finally two (Ends & Ends<sub>r</sub>) were selected on the basis of the superior quality of solution they consistently resulted in. A modified version of the Floyd’s algorithm (MFA) was used to determining the flow time (length of the critical path). The inclusion of all the constraints in the determination of the critical path is what sets this research apart, from previous studies. The computational complexity of this method is limited by the MFIH heuristic,

since the MFA only adds a finite number of computations at every iteration, and is of the order of  $O(G*n^2)$ , where  $1 \leq G \leq n$ . An unique experimental scheme to test the method was implemented, wherein processing time values were generated from a bounded exponential distribution. The largest deviation obtained was 4.016 % in the 5-job case. The average percentage error from the optimal was less than one percent.

## 6.2 Future Research

Potential ideas for areas of future research:

- Expand the MMVA+MFA algorithm to include stochastic processing times and breakdowns.
- Study the effects of having a smaller CONWIP system within a larger CONWIP system. The idea is explained further as follows:
  - Balance the line by grouping the stations according to capacity.
  - Device a method to obtain an optimal CONWIP level for each of these clusters for better control.

The idea behind the CONWIP system of control as mentioned in Section 1.4, is that the whiplash effect in a flow line produced due to variability in processing times, is curtailed or atleast reduced by the tighter control allowed by maintaining a fixed WIP level. The same principle can be extended further to smaller parts of the line, by maintaining a predetermined WIP level among clusters. The throughput rate of the clusters is defined by their capacities and cannot be controlled, thus leading to the whiplash effect between these clusters. This loss of control can be offset by the tighter control obtained by the introduction of the CONWIP level between them.

- Incorporate the effects of sequence dependent set-up times.
- Device a better method to determine the approximation term ( $\epsilon_{i,k}^j$ ) used in the MMVA algorithm.
- Perform similar study for a more general case of a non-serial production system.
- Study the effect of buffer capacities on line performance and propose a method to design the capacities to optimize throughput.

The potential benefits of a CONWIP system of control, over other methods, are clearly shown to be significant. The positive results presented here and the great potential of improvement, warrants additional study of the CONWIP method of control. The advent of e-business creates a highly customer driven front end for a company, which can only be sustained by an equally agile production system with tighter control.



## Bibliography

- Baker K. R., 1974, Introduction to Sequencing & Scheduling, John Wiley & sons, NY.
- Baskett F, Chandy M. K., Muntz R., Palacios F., 1975, Journal of the Association for Computing machinery, Vol-22, N-2, pp 248-260.
- Bonvik A. M., Couch C. E., and Gershwin S. B., 1997, A Comparison of Production-line Control Mechanisms, Int. J. Prod. Res., Vol-35, N-3, pp 789-804.
- Buzen J. P., 1978, Operational Analysis: An Alternative to Stochastic Modeling, Proc. Int. Conf. Perf. Comp. Install. Holland.
- Buzen J.P., Denning P.J., 1978, Operational Analysis of Queuing Network Models, Computing Surveys, Vol-10, N-3, pp 225-261.
- Campbell H. G., Dudek R. A., and Smith M. L., 1970, A Heuristic Algorithm for the n job, m machine Sequencing problem, Mngt.Sci., 16B, pp 630-637.
- Cavaille J-B, Dubois D., 1982, Heuristic Methods Based on Mean-Value for Flexible Manufacturing Systems Performance Measures, IEEE Conference on Decision and Control, France.
- Chandy M. K., Sauer C. H., 1978, Approximate Methods for Analyzing Queuing Networks Models of Computing Systems, Computing Surveys, Vol-10, N-3, pp 281-317.
- Crama, Y., Flippo, O.E., Klundert, J. and Spieksma, F.C.R., 1997, The Assembly of Printed Circuit Boards: A Case with Multiple Machines and Multiple Board Types, Euro. J. Oper. Res., V-98, N-3, pp.457-472.
- Dar-El E. M., Herer Y. T., Masin M., 1999, CONWIP-based Production Lines With Multiple Bottlenecks: Performance and Design Implications, IIE Trans. Vol-31, pp 99-111.
- Dubois D., 1982, A Mathematical Model of a Flexible Manufacturing System With Limited In-Process Inventory, Euro. J. Oper. Res., Vol-14, pp 66-78.
- Goldratt, E. and Cox, J., 1984, The Goal: A Process of Ongoing Improvement, North River
- Greco, M.P, 1996, Sequencing Policy for a CONWIP Production System, M.Sc. Thesis, Virginia Polytechnic Institute and State University, VA, USA.

Gupta, G.N.D., 1971, A Functional Heuristic Algorithm for the Flow-shop Scheduling Problem, *Oper. Res. Qtrly.*, V-22, N-1, pp 39-47.

Heavey, C., 1993, The Throughput Rate of Multistation Unreliable Production Lines, *Euro. H, Oper. Res.*, V-68, N-1 pp.69-89.

Hildebrant R. R., 1980, Scheduling Flexible Machining Systems Using Mean Value Analysis, *IEEE*, Albuquerque, NM, pp 701-706.

Hopp W. J., Spearman M. L., 1991, Throughput of a Constant Work in Progress Manufacturing Line Subject to Failures, *Int. J. Prod. Res.*, Vol-29, N-3, pp 635-655.

Hopp W. J., Spearman M. L., Factory Physics: Foundations of Manufacturing Management [2<sup>nd</sup> ed.], Irwin Mc Graw-Hill.

Hwang J., Singh M. R., 1998, Optimal Production Policies for Multiple-Stage Systems with Setuo Costs and Uncertain Capacities, *Management Science*, Vol-44, N-9, pp 1279-1294.

Jackson J. R., 1963, Job Shop Like Queuing Systems, *Management Sci*, Vol-10, pp 131-142.

Jackson, J., R., 1957, Network of waiting Lines, *Naval Research Quarterly*, pp 518-521.

Johnson, S.M., 1954, Optimal Two Stage Production Schedules with Set Up Times Included, V-1, N-1, pp.61-68.

Kalir, A. and Arzi, Y, 1998, Optimal Design of Flexible Production Lines with Unreliable Machines and Infinte Buffers, *IIE Trans.*, V-30, N-4, pp.391-399.

Kamoun, H. and Sriskandarajah, C., 1993, The Complexity of Scheduling Jobs in Repetitive manufacturing Systems, *Euro. J. Oper. Res.*, V-70, N-3, pp.350-364.

Karabati, S. Kouvelis, P. and Kiran, A.S., 1992, Games, Critical Paths and Assignment Problems in Permutation Flow Shops and Cyclic Scheduling Flow Line Environments, *J. Opl. Res. Soc.*, V-43, N-3, pp.241-258.

King, J.R. and Spachis, A.S., 1980, Heuristics for Flow-shop Scheduling, *Prod. Res.*, V-18, N-3, pp.345-357.

Kirkpatrick, S., Gelatt Jr. C.D., Vecchi, M. P., 1983, Optimization by Simulated Annealing, *Science* 220, p.671

Lawler, E.G., Lenstra, J.K., Rinnooy Kan, A.H.G., and Shmoys, D.B., 1989, Sequencing and Scheduling: Algorithms and Complexity, Report BS-R8909, Centre of Mathematics & Computer Science, Amsterdam.

Matsuo, H., Suh, C.J. and Sullivan, R. S. 1988, A Controlled Search Simulated Annealing Method for the General Jobshop Scheduling Problem, Working Paper 03-44-88, Graduate School of Business, University of Texas, Austin.

McCromick, S.T., Pinedo, M.L., Shenker, S. and Wolf, B., 1989, Sequencing in an Assembly Line with Blocking to Minimize Cycle Time, *Oper. Res. Soc.*, V-37, N-6, pp.925-935.

Nawaz, M., Ensore Jr, E.E., and Ham, I., 1983, A Heuristic Algorithm for the n-job, m-machine Flow Shop Sequencing Problem, *Omega*, V-11, pp. 91-95.

Nowicki, E. 1997, A Permutation Flowshop with Buffers: A Tabu Search Approach, *Euro. J. Oper. Res.*, 116, pp.205-219

Page, E.S., 1961, An Approach to Scheduling of Jobs on the Machines, *J. Royal Stat. Soc.*, V-23, pp. 484-492.

Palmer, D.S., 1965, Sequencing Jobs through a Multi-Stage Process in the Minimum Total Time Quick Method, *Oper. Res. Qtrly.*, pp. 101-107.

Papadimitriou C. H., and Yannakakis M., 1980, Flow shop Scheduling with Limited Temporary Storage. *J. Assoc. Comput., Mach.* 27, pp 533-549.

Pinedo, M., 1995, *Scheduling Theory: Algorithms and Systems*, Perntice Hall Inc.

Reiser M., 1979, Mean Value Analysis of Queueing Networks. A New Look at an Old Problem, *Proceedings 4<sup>th</sup> International Symposium on Modeling and Performance Evaluation of Computer Systems*, Vienna, pp 63-77.

Sarin, S.C., and Greco, M., 1997, Job Sequencing and Material Flow Control in a Closed Production System, *IIE Trans.*, In press.

Schweitzer P. J., Seidmann A and Shalev-Oren, 1985, The Correction Terms in Approximate Mean Value Analysis, *Operations Research letters*, Vol-4, N-5, pp 197-200.

Song, J. and Lee, T. 1998, Petri Net Modeling and Scheduling for Cyclic Job Shops with Blocking, *Computers Ind. Engng.* V-34, N-2, pp.281-295

Spearman M. L., Woodruff D. L., and Hopp W. J., 1990, CONWIP: A Pull Alternative to Kanban, *Int. J. Prod. Res.*, Vol-28, N-5, pp 879-894.

Spearman, M.L., 1991, An Analytic Congestion Model for Closed Production Systems with IFR Processing Times, *Mgmt. Sci.*, V-37, N-8, pp.1015-1029.

Spearman, M.L., Woodruff, D.L. and Hopp, W.J., 1990, CONWIP: A Pull Alternative to Kanban. *Int. J. Prod. Res.* V-28, N-5, pp. 879-894.

Stewart W. J., 1979, A Direct Numerical Method For Queuing Network, Performance of Computer Systems, pp 89-102.

Stinson, J.P. and Smith, A.W., 1982, A Heuristic Programming Procedure for Sequencing the Static Flowshop, Int. J. Prod., Res. V-20, N-6, pp.753-764.

Suri R., Olsder G. J., 1980, Time-Optimal Control of Parts-Routing in a Manufacturing System With failure-Prone Machines, 19<sup>th</sup> IEEE Conference on Decision & Control, Albuquerque, NM.

Tan B., Yerlan S., A Decomposition Model for Continuous Material Flow Production Systems, Int. J. Prod. Res., Vol-35, N-10, pp 2759-2772.

Taylor II Manuals I-VII, F&H Simulations, <http://www.taylorii.com/>.

Vouros, G.A. and Papadopoulos, H.T. 1998, Buffer Allocation in Unreliable Production lines Using a Knowledge Based System, Computers Ops. Res., V-25, N-12, pp.1055-1067.

## **Appendices**

## **Appendix A: Visual Basic Program to determine the best (close to optimal) WIP level M**

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// VISUAL BASIC PROGRAM FOR DETERMINING THE "BEST" WIP LEVEL M      //
//                                                                    //
// Author: NIPUN P.PALEKAR                                           //
// Date: JUNE 10, 2000                                              //
// Notes: 1. The code calls Mathematica, which has to be run manually. //
//        2. The number of iterations are limited to 500 or till the //
//           difference is reached, which ever is reached earlier.    //
//        3. The Following module is just one among 4, and thus will //
//           not run independently.                                    //
//        4. The entry form prompts the user to enter the values of //
//           input parameters                                         //
//                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

```

Dim temp As Double
Dim isOpen As Boolean
Dim str As String
Dim Ans As String

```

```

Function MVAMain()

```

```

    For i = 0 To Jobs - 1
        Queue(i, 0) = ns(i) / Machines
    Next i
    For i = 1 To Machines - 1 'for loop to copy the columns
        For j = 0 To Jobs - 1
            Queue(j, i) = Queue(j, 0)
        Next j
    Next i

```

```

Dim qRowAdd() As Double 'qColumnadd() is a column matrix (addition of rows in the queue matrix)
ReDim qRowAdd(0 To Jobs - 1, 0)
Dim Eps() As Double
ReDim Eps(0 To Jobs - 1, 0 To Machines - 1)
Dim temp() As Double
ReDim temp(0 To Jobs - 1, 0 To Machines - 1)
Dim add As Double
Dim oldQueue() As Double
ReDim oldQueue(0 To Jobs - 1, 0 To Machines - 1)
ReDim MQLen(0 To Machines - 1) As Double
Dim Epsilon As Double
Dim diffLarge As Double
Dim diff As Double
diffLarge = 0
Iter = 0

```

```

ReDim BT(0 To Machines - 1)

```

```

'initially the queue is not full
For i = 0 To Machines - 1
    isFull(i) = False
Next i

Declarations and Initialization ends here
'=====

'Step 3, 4 of the MMVA
Do

'Calculate qrowadd (which is the row addition of the matrix)
For i = 0 To Jobs - 1
    For j = 0 To Machines - 1
        qRowAdd(i, 0) = qRowAdd(i, 0) + Queue(i, j)
    Next j
Next i

```

```

    For s = 0 To Jobs - 1
        For m = 0 To Machines - 1
            add = 0
            Epsilon = 0
            For j = 0 To Jobs - 1

                If j = s Then
                    Epsilon = Queue(j, m) / qRowAdd(j, 0)
                Else
                    Epsilon = 0
                End If

                temp(j, m) = Queue(j, m) - Epsilon
                add = add + TimeMat(j, m) * temp(j, m)
            Next j
            Tau(s, m) = TimeMat(s, m) + add + BT(m)
        Next m
    Next s

```

```

'Step 5 of the MMVA

For s = 0 To Jobs - 1
    tempadd = 0
    For m = 0 To Machines - 1
        tempadd = tempadd + Tau(s, m)
    Next m
    Lambda(s, 0) = ns(s) / tempadd
Next s

'Equate transfers the elements of matrix oldQueue()into Queue()
Call Equate(Queue(), oldQueue(), Jobs, Machines)

```

Dim Lambdatemp ' value of a element in the column matrix of lambda

```

Lambdatemp = 0
For row = 0 To Jobs - 1
  Lambdatemp = Lambda(row, 0)
  For col = 0 To Machines - 1
    If isFull(col) = False Then
      Queue(row, col) = Tau(row, col) * Lambdatemp
    End If
  Next col
Next row

For i = 0 To Machines - 1 ' Initialize the MQLen to array of zeros
  MQLen(i) = 0
Next i

For i = 0 To Machines - 1 ' Calculate MQLen as sigma Queue(j,i)
  For j = 0 To Jobs - 1
    MQLen(i) = MQLen(i) + Queue(j, i)
  Next j
Next i

' Checking Queue length and calculate the scaling factor
For i = 0 To Machines - 1
  If isFull(i) = True Then
    If isCalled = False Then
      Call Mathematica
      isCalled = True ' flag if the mathematica module is called for machine i
    End If
    Call Blk_Time(i)
  End If
  If MQLen(i) > BuffSize(i) Then
    isFull(i) = True
    Qfactor(i) = BuffSize(i) / MQLen(i)
  Else
    isFull(i) = False
    Qfactor(i) = 1
  End If
Next i

' multiply by the scaling factor
For i = 0 To Machines - 1
  For j = 0 To Jobs - 1
    Queue(j, i) = Queue(j, i) * Qfactor(i)
  Next j
Next i
'=====

Iter = Iter + 1
diffLarge = 0
diff = 0
For i = 0 To Jobs - 1
  For m = 0 To Machines - 1
    diff = (Queue(i, m) - oldQueue(i, m))

```



```

    Next m
  Next i

Loop While Iter < 500 `diffLarge > Threshold or limit number of iterations to 500

'----- Write the outputs to an output file -----
opfile = FreeFile
Open "C:\WINNT\Profiles\npalekar\Desktop\Visualbasic\OUTPUT.TXT" For Output As opfile

Dim tauAdd As Double

Print #opfile, "Average Waiting Time at Machine i", vbCrLf
For i = 1 To Machines - 1
  tauAdd = 0
  For j = 0 To Jobs - 1
    tauAdd = tauAdd + Tau(j, i)
  Next j
  tauAdd = tauAdd / (Jobs ^ 2)
  Print #opfile, CStr(tauAdd), " "
Next i
Print #opfile, vbCrLf, vbCrLf

Print #opfile, "Average Queue Length at Machine i", vbCrLf
For i = 1 To Machines - 1
  Print #opfile, CStr(MQLen(i) / Jobs)
Next i

Close opfile
MsgBox "Program ends", vbInformation, "Message Box"

End Function

'function to call mathematica algorithm
Public Function Mathematica()

  Max = 0
  For i = 0 To Machines - 1
    If Max < Mu(i) Then Max = Mu(i)
  Next i

  Tcur = 1 / Max
  Told = 0
  Dim strAns As String

  Diffint = 0

  Do
    A(0) = 1
    P(0) = 0
    B(0) = 1

    For i = 1 To Machines - 1
      temp = Mu(i - 1) * B(i - 1) / Tcur
      tempr2 = temp ^ (BuffSize(i) + 2)

```

```

tempfile = FreeFile
Open "E:\UsersE\thesis\MATHIN.TXT" For Output As tempfile
Dim eqn As String
eqn = CStr(temp2) + "*x^" + CStr(BuffSize(i) + 2) + "+" + CStr(temp2) + "*x^" + CStr(BuffSize(i) + 1) _
      + "+" + CStr(-temp + 1)
Print #tempfile, eqn
Close tempfile

'Call Mathematica software and open a blank Kernel
If temp2 <> 0 Then
    Call Shell("E:\Program Files\Mathematica\mathematica.exe ", 1)
Else
    A(i) = 1
End If

'Time delay for mathematica (Can be used instead of the existing manual method)

For k = 0 To 100000
    ' For j = 0 To 10000
    ' Next j
Next k

Open "E:\UsersE\thesis\mathout.txt" For Input As tempfile
Ans = "0"

'-----
Do
    Line Input #tempfile, str
    Trim (str) 'remove white spaces
    pos = InStr(1, str, "I", vbTextCompare)
    Dim length As Integer
    length = Len(str)
    If pos = 0 Then 'real answer
        pos = InStr(1, str, ">", vbTextCompare)
        Ans = Right(str, length - pos)
    End If
Loop While (Not EOF(tempfile))
'-----

If Ans = "" Then Ans = "1"

A(i) = CDBl(Ans)
If A(i) > 1 Then A(i) = 1
If A(i) <= 0 Then A(i) = 1

Close #tempfile

P(i) = 1 - A(i)
B(i) = A(i)
Next i

Told = Tcur
Tcur = Mu(Machines - 1) * A(Machines - 1)

Diffint = Tcur - Told      Nipun added this statement (apr16)

```

```

    If Diffint < 0 Then Diffint = -Diffint

    Loop While ((Diffint) > 0.1)
End Function

Public Function Blk_Time(ByVal mc_no As Integer)

'=====
' calculation of blocking time

For i = 0 To Machines - 2
    t1_t0(i) = 1 / (Mu(i) * (1 - P(i)))
    Lambda1(i) = Mu(i) * variables.A(i)
    rho(i) = Lambda1(i) * variables.A(i) / Tcur
    L(i) = (rho(i) ^ 2 - MQLen(i) * (rho(i) ^ (MQLen(i) + 1)) + _
        (MQLen(i) - 1) * (rho(i) ^ (MQLen(i) + 2))) / _
        ((1 - rho(i)) * (1 - rho(i) ^ (MQLen(i) + 1)))
    Tau1(i) = L(i) / Lambda1(i)
    Pn(i) = ((1 - rho(i)) * rho(i) ^ MQLen(i)) / (1 - rho(i) ^ (MQLen(i) + 1))
Next i

For i = 0 To Machines - 2

    tq_t0(i) = ((0.5 * Pn(i + 1) * (Tau1(i + 1) - (1 / Mu(i + 1)) / BuffSize(i + 1))) / A(i)) + (1 / Mu(i))

    WT(i) = tq_t0(i) - t1_t0(i)

    If WT(i) < 0 Then
        WT(i) = -WT(i)
    End If

Next i

Dim P1 As Double
P1 = 1
For j = 0 To mc_no - 1
    BT(mc_no - 1 - j) = P1 * WT(mc_no - 1 - j)
    P1 = P1 * Pn(mc_no - j)
Next j
End Function

```

**Appendix B: Time matrices used in the testing of MMVA (Uniform[5 – 25])**

**Time Matrix no. 1**

		<b>Machines</b>									
		1	2	3	4	5	6	7	8	9	10
<b>Job Type</b>	<b>1</b>	5.9	9.0	24.0	14.3	15.5	12.9	22.7	9.2	16.2	12.1
	<b>2</b>	18.6	12.2	11.0	14.1	9.0	22.8	9.3	15.0	23.8	13.0
	<b>3</b>	17.9	9.6	20.8	21.7	8.6	18.5	15.6	16.4	18.8	12.7
	<b>4</b>	14.3	13.5	13.0	6.2	15.3	14.0	17.2	14.0	21.2	19.7
	<b>5</b>	12.2	25.0	17.8	15.8	19.8	22.7	24.0	9.9	9.9	9.6
	<b>6</b>	22.0	11.8	11.2	8.2	12.0	10.5	17.5	21.8	16.3	5.4
	<b>7</b>	21.4	19.1	6.6	11.8	20.5	20.2	10.1	13.5	8.2	10.2
		<b>112.2</b>	<b>100.2</b>	<b>104.5</b>	<b>92.1</b>	<b>100.7</b>	<b>121.6</b>	<b>116.4</b>	<b>99.8</b>	<b>114.5</b>	<b>82.7</b>

**Time Matrix no. 2**

		<b>Machines</b>									
		1	2	3	4	5	6	7	8	9	10
<b>Job Type</b>	<b>1</b>	5.4	10.5	7.0	18.9	7.8	23.8	18.5	19.9	6.3	13.0
	<b>2</b>	7.9	22.3	9.6	16.1	14.1	10.3	15.3	22.4	7.4	21.9
	<b>3</b>	16.5	22.6	21.6	15.6	15.1	13.1	12.8	20.6	19.7	15.4
	<b>4</b>	15.3	20.7	22.3	23.5	10.1	23.6	15.3	24.3	8.9	17.2
	<b>5</b>	12.7	13.6	19.3	7.0	14.3	14.5	24.6	21.9	17.2	14.3
	<b>6</b>	9.7	17.8	24.1	13.0	15.3	15.3	21.9	24.8	7.0	23.6
	<b>7</b>	14.5	22.9	18.3	8.0	12.3	19.8	13.3	24.4	16.1	15.0
		<b>82.1</b>	<b>130.4</b>	<b>122.1</b>	<b>102.1</b>	<b>89.0</b>	<b>120.5</b>	<b>121.8</b>	<b>158.2</b>	<b>82.6</b>	<b>120.5</b>

**Time Matrix no. 3**

		<b>Machines</b>									
		1	2	3	4	5	6	7	8	9	10
<b>Job Type</b>	<b>1</b>	5.1	12.5	17.6	20.8	23.4	16.6	18.8	10.6	9.6	6.2
	<b>2</b>	19.3	5.4	24.0	6.9	14.7	21.8	8.1	8.3	19.3	6.8
	<b>3</b>	17.8	17.1	23.9	7.8	20.8	23.5	18.2	8.6	19.3	12.0
	<b>4</b>	23.1	23.7	10.2	24.1	7.6	18.8	19.2	9.7	16.3	6.5
	<b>5</b>	5.8	24.6	12.0	23.1	17.2	19.5	23.6	23.1	8.2	10.4
	<b>6</b>	18.9	8.7	20.4	11.5	12.1	23.3	12.8	8.3	14.5	5.9
	<b>7</b>	19.9	6.2	7.2	11.2	24.7	7.1	5.3	17.8	24.5	7.5
		<b>110.0</b>	<b>98.1</b>	<b>115.1</b>	<b>105.4</b>	<b>120.5</b>	<b>130.6</b>	<b>106.0</b>	<b>86.2</b>	<b>111.6</b>	<b>55.3</b>

Time Matrix no. 4

		Machines									
		1	2	3	4	5	6	7	8	9	10
Job Type	1	17.7	11.6	13.0	5.8	12.8	18.0	8.2	22.5	7.1	6.1
	2	10.9	18.8	18.7	19.8	10.6	18.9	15.7	22.0	21.2	14.0
	3	17.4	9.0	9.1	9.2	19.0	15.5	17.3	14.1	19.7	10.3
	4	16.4	22.0	7.2	23.8	12.4	17.5	5.7	9.7	13.7	14.5
	5	8.8	8.7	18.6	21.3	23.2	19.8	24.5	21.6	16.8	19.8
	6	12.5	15.3	24.2	8.7	22.9	5.0	6.1	8.4	15.5	12.3
	7	13.1	18.9	23.4	6.0	9.4	18.6	11.7	22.5	12.8	12.2
		<b>96.8</b>	<b>104.2</b>	<b>114.3</b>	<b>94.5</b>	<b>110.2</b>	<b>113.3</b>	<b>89.3</b>	<b>120.9</b>	<b>106.7</b>	<b>89.3</b>

Time Matrix no. 5

		Machines									
		1	2	3	4	5	6	7	8	9	10
Job Type	1	16.5	24.5	23.5	7.5	9.2	16.9	22.3	5.9	6.7	17.8
	2	10.7	24.6	11.4	11.5	24.5	19.7	24.3	7.9	14.6	22.4
	3	15.9	20.2	19.3	21.6	10.7	19.3	6.4	23.4	16.4	20.5
	4	7.6	18.7	9.2	6.1	5.8	6.9	17.2	15.3	12.9	8.3
	5	8.2	14.0	20.0	12.7	9.1	21.9	5.3	21.9	21.1	22.5
	6	16.3	15.8	15.1	24.4	10.5	24.9	7.6	19.5	6.5	12.1
	7	12.1	16.4	12.2	11.8	10.4	14.8	23.9	20.4	20.8	12.4
		<b>87.3</b>	<b>134.2</b>	<b>110.6</b>	<b>95.6</b>	<b>80.2</b>	<b>124.4</b>	<b>106.9</b>	<b>114.3</b>	<b>99.0</b>	<b>116.0</b>

Time Matrix no. 6

		Machines									
		1	2	3	4	5	6	7	8	9	10
Job Type	1	9.2	16.7	20.6	16.1	15.8	12.6	7.5	22.4	5.5	8.3
	2	10.0	10.0	20.2	24.5	18.7	13.7	23.9	9.7	6.8	22.0
	3	17.4	8.2	20.6	8.2	14.7	19.8	22.5	10.5	11.3	5.3
	4	7.2	15.2	23.8	23.0	6.4	18.2	11.0	21.8	24.8	24.9
	5	23.0	12.8	7.4	20.6	12.6	5.2	16.6	23.2	12.8	24.4
	6	14.2	24.1	7.1	10.7	5.4	21.2	21.2	10.5	22.8	21.7
	7	19.3	10.7	17.5	17.5	7.5	18.3	17.8	11.0	8.3	18.9
		<b>100.1</b>	<b>97.6</b>	<b>117.2</b>	<b>120.6</b>	<b>81.0</b>	<b>109.1</b>	<b>120.5</b>	<b>109.1</b>	<b>92.2</b>	<b>125.5</b>

**Time Matrix no. 7**

		<b>Machines</b>									
		1	2	3	4	5	6	7	8	9	10
<b>Job Type</b>	1	17.9	5.6	8.3	17.7	12.4	12.7	7.5	10.8	17.8	10.9
	2	8.0	5.6	14.5	15.7	20.4	15.4	14.1	17.1	20.0	23.0
	3	5.5	15.0	16.3	12.4	13.3	17.5	15.6	24.9	12.0	18.9
	4	18.4	20.1	15.4	17.4	7.5	17.9	19.1	16.2	12.6	17.0
	5	10.6	6.6	17.7	13.2	13.3	22.1	15.8	8.2	21.5	22.9
	6	20.2	13.4	7.4	7.5	8.6	16.4	8.9	22.4	20.4	16.8
	7	6.7	12.7	15.8	12.1	9.4	23.6	6.7	7.5	16.3	17.1
		<b>87.3</b>	<b>79.0</b>	<b>95.4</b>	<b>96.1</b>	<b>84.9</b>	<b>125.6</b>	<b>87.8</b>	<b>107.0</b>	<b>120.5</b>	<b>126.6</b>

**Time Matrix no. 8**

		<b>Machines</b>									
		1	2	3	4	5	6	7	8	9	10
<b>Job Type</b>	1	24.7	9.5	16.2	14.6	12.6	16.0	6.5	10.0	20.4	22.1
	2	16.8	6.3	7.1	6.8	18.5	15.9	11.4	14.3	22.4	22.6
	3	24.1	5.5	14.0	18.2	14.4	19.7	14.2	13.9	6.2	24.2
	4	15.7	12.7	22.1	22.8	8.9	14.1	22.2	6.7	18.3	22.5
	5	5.4	25.0	17.3	6.9	22.3	23.3	18.5	7.3	19.8	15.2
	6	20.9	21.2	12.6	17.9	5.8	14.5	22.7	10.1	21.2	13.5
	7	8.4	14.2	17.0	5.2	23.3	18.8	20.7	11.4	14.4	20.1
		<b>116.0</b>	<b>94.4</b>	<b>106.4</b>	<b>92.3</b>	<b>105.6</b>	<b>122.2</b>	<b>116.3</b>	<b>73.6</b>	<b>122.7</b>	<b>140.3</b>

**Time Matrix no. 9**

		<b>Machines</b>									
		1	2	3	4	5	6	7	8	9	10
<b>Job Type</b>	1	14.0	19.6	15.8	5.6	6.4	16.1	20.1	5.5	17.7	15.4
	2	23.6	10.1	12.8	5.3	7.2	15.3	22.3	19.6	23.0	9.9
	3	7.1	23.4	8.0	20.3	19.0	11.0	17.1	9.4	17.4	11.1
	4	12.3	23.8	18.8	16.4	16.5	7.9	12.9	16.3	9.8	19.0
	5	12.0	14.2	23.2	12.7	11.1	10.2	8.4	23.4	6.5	18.8
	6	18.2	15.6	18.9	18.4	15.3	14.7	9.7	7.0	23.0	24.6
	7	17.8	13.0	17.1	23.6	24.9	18.6	22.6	19.3	14.9	17.9
		<b>105.1</b>	<b>119.7</b>	<b>114.6</b>	<b>102.2</b>	<b>100.4</b>	<b>93.8</b>	<b>113.1</b>	<b>100.5</b>	<b>112.1</b>	<b>116.7</b>

**Time Matrix no. 10**

		<b>Machines</b>									
		1	2	3	4	5	6	7	8	9	10
<b>Job Type</b>	<b>1</b>	5.1	15.5	15.5	13.2	24.0	19.8	16.8	6.1	5.6	13.7
	<b>2</b>	10.7	22.0	20.3	16.3	16.4	6.6	23.3	5.5	9.4	14.4
	<b>3</b>	8.3	8.7	23.0	15.5	15.4	22.0	22.6	20.8	12.6	13.2
	<b>4</b>	11.3	8.3	21.1	17.1	20.9	15.9	18.9	24.3	14.3	11.0
	<b>5</b>	21.3	9.6	21.2	12.7	14.7	18.9	19.1	23.3	19.1	16.7
	<b>6</b>	6.5	15.2	8.1	9.4	14.6	23.1	7.0	20.3	23.6	23.5
	<b>7</b>	15.6	14.3	6.4	14.5	22.4	9.9	21.7	20.0	22.7	13.4
		<b>78.7</b>	<b>93.5</b>	<b>115.5</b>	<b>98.7</b>	<b>128.3</b>	<b>116.3</b>	<b>129.3</b>	<b>120.4</b>	<b>107.5</b>	<b>105.9</b>

## Appendix C: C ++ Program to determine the “Best” sequence using FIH and MMVA

*The following is the code to the basic methodology. Variations in FIH algorithm, as a part of experimentation to improve the quality of the solution, were carried out. The code for these variations is not included*

```
/////////////////////////////////////////////////////////////////
// C++ PROGRAM FOR DETERMINING THE "BEST" SEQUENCE USING MODIFIED FIH HEURISTIC & //
// FLOYDS ALGORITHM //
// //
// Author: NIPUN P.PALEKAR //
// Date: JUNE 10, 2000 //
// Notes: 1. The sequence generated is not optimal but of a good quality. //
// 2. The FIH heuristic developed by Nawaz et al is used as a means of enumeration. //
// 3. The Floyds algorithm is modified to determine the critical path, which is the Flow time for the //
// enumeration. //
// 4. The values of # of Machines and # of Jobs need to entered in the code where they are defined //
/////////////////////////////////////////////////////////////////
```

```
#include<iostream>
#include<fstream>
#include<stdio.h>
#include<math.h>
#include<valarray>
#include"Matrix.h"
#include"Vec.h"
```

```
using namespace std;
```

```
ofstream outFile;
```

```
void main()
```

```
{
```

```
    // Variable Declaration
```

```
    int i; // Job Index
    int j; // Machine Index
    int k; // Used in Step II (temperory)
    const int machines=3; // No of Machines
    const int jobs=4; // No of Jobs
    int M; // CONWIP level
    int jobcycled;
    double mcjob[machines+2][jobs]; // The Job Vs Machine array
    double temp=0; // A Temperory variable
    double partialjobset[100][100]; // This array holds the partial job array
    double partialjobsettemp[100][100]; // This array holds the partial job array
    // temporarily when jobs are moved one
    // position below
```

```
    double partialjobsettempused[100][1000]; // This array holds the Partialjobtemp with columns
```



copied to include the CONWIP-Cyclic condition

```
double floyd(const int, const int, double[100][1000], const int);

ofstream output;

// Prompts the user to input the processing time values and the CONWIP level (determined
separately earlier)

cout<<"WELCOME TO NIPUN'S C++ PROGRAM TO GENERATE THE OPTIMAL
SEQUENCE" <<endl<<endl;
cout<<"Please enter the values of processing times & CONWIP level M." <<endl;
cout<<"The software will prompt you to enter the processing times of all jobs, machine by
machine" <<endl;
output.open("OutputSequence.txt");
output<<"OUTPUT FILE SHOWING THE GENERATION OF BEST SEQUENCING IN A
STEP BY STEP FASHION" <<endl<<endl;

// Input the first row which is the job number

for(i=0;i<jobs;++i)
{
    mcjob[0][i]=double(i+1);
}

// Input the time matrix values

for (i=0;i<machines;++i)
{
    for(j=0;j<jobs;++j)
    {
        cout<<"Enter element["<<i<<"]["<<j<<"] ";
        cin>>mcjob[i+1][j];
    }
}

// Input the CONWIP level

cout<<"Enter the optimal CONWIP level: M ";
cin>>M;

// Print the mcjob matrix

for(i=0; i<jobs; ++i)
{
    cout<<"J"<<mcjob[0][i]<<"\t";
}
cout<<endl;

for(i=0;i<machines;++i)
{
```

```

        for(j=0;j<jobs;++j)
        {
            cout<<mcjob[i+1][j]<<"\t";

        }
        cout<<endl;
    }
    cout<<endl<<endl;

```

// Calculate the column totals and add it as the last row

```

for(i=0;i<jobs;++i)
{
    temp = 0;
    for(j=0;j<machines;++j)
    {
        temp = temp + mcjob[j+1][i];
    }

    mcjob[machines+1][i]=temp;    // Add the column totals as the last row in the time matrix
}

```

// Print the above matrix with the sum row as the last row

```

for(i=0; i<jobs; ++i)
{
    cout<<"\t"<<"J"<<mcjob[0][i];
}
cout<<endl;

for(i=1;i<machines+2;++i)
{
    cout<<"\t";

    for(j=0;j<jobs;++j)
    {
        cout<<mcjob[i][j]<<"\t";
    }
    cout<<endl;
}
cout<<endl<<endl;

```

// Sort the mcjob matrix according to the last row values (which are the sums of columns)

```

for (i=0;i<jobs;++i)
{
    for(j=i+1;j<jobs;++j)
    {
        if (mcjob[machines+1][j]>mcjob[machines+1][i])
        {

```

```

        for (k=0;k<machines+2;++k)
        {
            // This will sort the columns
            temp = 0;
            temp = mcjob[k][i];
            mcjob[k][i] = mcjob[k][j];
            mcjob[k][j] = temp;
        }
    }
}

// Print the above matrix with the sum row as the last row

for(i=0; i<jobs; ++i)
{
    cout<<"\t"<<"J"<<mcjob[0][i];
}
cout<<endl;
for(i=1;i<machines+2;++i)
{
    cout<<"\t";

    for(j=0;j<jobs;++j)
    {
        cout<<mcjob[i][j]<<"\t";
    }

    cout<<endl;
}

cout<<endl<<endl;

// First partial matrix is only the first job (Copy full column except the last element which is the
total of the column)

// The partial matrix has all the rows except the last row which is the totals row.
for (i=0; i<machines+1;++i)
{
    partialjobset[i][0]= mcjob[i][0];
}

// Print the partial matrix

cout<<"partialjobsettemp: "<<endl;
for (i=0;i<machines+1;++i)
{
    cout<<partialjobset[i][0]<<endl;
}

```

```

// Form the partial time matrix which can be used to generate the Floyd matrix

outFile.open("floydmatrix.txt");

for (i=1; i<=jobs-1; ++i)          // i = number of jobs in the partial job array before inserting the chosen
job in any iteration
{
    int minj;
    double minfloyd = -10;          // Start value of critical path length (negative value is
                                    chosen as the first value)

    for (j=0; j<=i; ++j)           // j = Slot number where the new job is going to be
                                    inserted
    {

        // Copy all jobs from partialjobset to partialjobsettemp; Do not copy the first row
        // and last row
        int n,m;                    // n = Iteration number in the FIH heuristic
        for (n=0; n<i; ++n)
        {
            cout<<endl;
            for (m=1; m<machines+1; ++m)
            {
                partialjobsettemp[m-1][n]=partialjobset[m][n];
            }
        }
        // Copying ends here

        // Now shift all jobs from slot j, one slot right

        int l,p;
        for (k=i-1; k>=j; --k)
        {
            for (l=0; l<machines; ++l)
            {
                partialjobsettemp[l][k+1] = partialjobsettemp[l][k];
            }
        }

        // The following four lines insert the (i+1)th job in the vacant slot (column)

        for(p=0; p<machines; ++p)
        {
            partialjobsettemp[p][j]=mcjob[p+1][i];
        }

        // Form the new Partialjobtemp matrix with the columns copied to include the
        // CONWIP-Cyclic condition
        // Copy the cloumns M-1 times so that there are a total of (i+1)*M jobs which
        // make one cycle

        int y, x;

```

```

if (i>M)
{
    for (k=0;k<machines;++k)
    {
        for (y=0;y<M;++y)
        {
            for(x=0;x<i+1;++x)
            {
                partialjobsettempused[k][((i+1)*y)+x]=partialjobsettemp[k][x];
            }
        }
    }
}
else
{
    for (k=0;k<machines;++k)
    {
        for(x=0;x<i+1;++x)
        {
            partialjobsettempused[k][x]=partialjobsettemp[k][x];
        }
    }
}

```

// Returns the values of the jobcycled

```

if (i>M)
{
    jobcycled = (i+1)*M;
}
else
{
    jobcycled = i+1;
}

```

// Print the partialjobsettemp matrix

```

cout<<endl;
cout<<"i= "<<i<<"j= "<<j<<"\t";

outFile<<endl<<endl<<"The "<<i<<"th iteration; slot = "<<j<<endl;
outFile<<"Partialjobsettemp matrix is as shown below"<<endl;

for(p=0; p<machines; ++p)
{
    cout<<endl;
    outFile<<endl;
    for(l=0; l<i+1; ++l)
    {
        cout<<"\t"<<partialjobsettemp[p][l];
        outFile<<"\t"<<partialjobsettemp[p][l];
    }
}

```

```

    }
    outFile<<endl<<endl;

    // Call the Floyd heuristic to calculate the min for all values of j

    double floydtemp = floyd(machines, jobcycled,
                             partialjobsettempused,M);
    if(minfloyd<0)
        minfloyd = floydtemp;
    else
    {
        minfloyd = (minfloyd<floydtemp) ? minfloyd : floydtemp;
        minj = j-1; // The job number of all the jobs tested in the
                   // iteration which has the
                   // least critical path length. (This
                   // job is chosen & inserted in slot
                   // minj)
    }
}

// Update partialjobset array for the next iteration (Following two paragraphs)

// This stanza makes space for the new column

for (k=i-1;k>=minj;--k)
{
    for (int l=0;l<machines+1;++l)
    {
        partialjobset[l][k+1] = partialjobset[l][k];
    }
}

// This stanza insert the new job column
for(int p=0;p<=machines;++p)
{
    partialjobset[p][minj]=mcjob[p][i];
}
output<<"The optimal sequence for the partial job set when the # of jobs in the partial set
is "<<i+1<<" is as follows:"<<endl<<endl;

//Print the sequence after the smallest one is chosen
for(int l=0; l<i+1; ++l)
{
    output<<"\tJ"<<partialjobset[0][l];
}

for(p=1; p<machines+1; ++p)
{
    output<<endl;
    for(l=0; l<i+1; ++l)

```

```

        {
            output<<"\t"<<partialjobset[p][l];
        }
    }
    output<<endl<<endl;

}

outFile.close();
output.close();

}

//===== Main ends here =====//

// Floyd uses the partialjobsettempused array and determines the critical path

double floyd(const int machs, const int jbs, double partialjobsettempused[100][1000], const int M)
{
    // Form the matrix : floydmatrix[i][j]
    double result;

    int i,j,k;
    const int nodes = (machs+1)*jbs;
    const double INF = -10; //Infinity in floydmatrix are -10
    const double NO = -1; //Dashes in floydmatrix are -1
    Matrix<double> floydmatrix(nodes+1, nodes+1); //The job v.s. machine array

// The Floyd matrix is built before it is solved. The following 2 loops build it

// Fill in the infinities and zeros in the Floyd Matrix
for (i = 1;i<=nodes;i++)
{
    for (j = 1;j<=nodes;j++)
    {
        if(i!=j)
        {
            floydmatrix[i][j]=INF;
            if ((i%jbs)!=0)
            {
                if ((i-j)==(jbs-1))
                {
                    floydmatrix[i][j]=0;
                }
            }
        }
        }else
        {
            floydmatrix[i][j] = NO;
        }
    }
}
}

```

```

    }
}

// Add zeros for CONWIP level condition

if(M<machs)
{
    for(i=1;i<=(jbs-M);++i)
    {
        floydmatrix[jbs*machs+i][M+i]=0;
    }
}

// Fill in the processing time elements in the Floydmatrix
for(i = 0; i<machs+1; i++)
{
    for(j = 1; j<=jbs; j++)
    {
        if( (i!=0) && (j!=jbs) )
        {
            floydmatrix[i*jbs+j][(i*jbs)+j+1]=partialjobsettempused[i-1][j];
// for horizontal arcs in the network
        }
        if(i!=machs)
        {
            floydmatrix[i*jbs+j][(i*jbs)+j+jbs]=partialjobsettempused[i][j-1];
// For vertical arcs in the network
        }
    }
}

// Print the matrix Floyd

outfile<<"The corresponding Floyd Matrix is as follows:"<<endl<<endl;
for (i = 1;i<=nodes;i++)
{
    for (j = 1;j<=nodes;j++)
    {
        outfile<<floydmatrix[i][j]<<"\t";

        if(j==nodes)
            outfile<<endl<<endl;
    }
}

// Perform the matrix manipulations on the floyd matrix

for (k = 1;k<=nodes;k++)
{

```



```

for (i = 1;i<=nodes;i++)
{
    for (j = 1;j<=nodes;j++)
    {
        if ( (i!=j)&& (k!=i)&& (k!=j) )
        {
            if (floydmatrix[i][k]!=INF && floydmatrix[k][j]!=INF)
            {
                if (floydmatrix[i][j]==INF || floydmatrix[i][j]!=INF\
&&
(floydmatrix[i][j]<floydmatrix[i][k]+floydmatrix[k][j]))
                {
                    floydmatrix[i][j]=floydmatrix[i][k]+floydmatrix[k][j];
                }
            }
        }
    }
}

// The following stanza gives the max of the last column elements

for (j=0;j<nodes;++j)
{
    if(floydmatrix[j+1][nodes]>floydmatrix[j][nodes])
        result = floydmatrix[j+1][nodes];
}

return result;
}

// ===== Floyd ends here =====//

```

## **Appendix E: Typical Output Files from the Heuristic coded in C++**

### **OUTPUT FILE #1:**

OUTPUT FILE SHOWING THE GENERATION OF BEST SEQUENCING IN A STEP BY STEP FASHION

The optimal sequence for the partial job set when the # of jobs in the partial set is 1 is as follows:

J3  
9.89  
10.27  
11.41  
13.43

The optimal sequence for the partial job set when the # of jobs in the partial set is 2 is as follows:

J1	J3
8.31	9.89
7.72	10.27
9.12	11.41
6.48	13.43

The critical path length (FT) = 19.91

The optimal sequence for the partial job set when the # of jobs in the partial set is 3 is as follows:

J1	J5	J3
8.31	10.56	9.89
7.72	6.93	10.27
9.12	8.97	11.41
6.48	11.83	13.43

The critical path length (FT) = 38.29

The optimal sequence for the partial job set when the # of jobs in the partial set is 4 is as follows:

J1	J5	J6	J3
8.31	10.56	11.15	9.89
7.72	6.93	6.21	10.27

9.12	8.97	6.45	11.41
6.48	11.83	8.26	13.43

The critical path length (FT) = 129.89

The optimal sequence for the partial job set when the # of jobs in the partial set is 5 is as follows:

J1	J5	J6	J4	J3
8.31	10.56	11.15	6.02	9.89
7.72	6.93	6.21	6.03	10.27
9.12	8.97	6.45	9.88	11.41
6.48	11.83	8.26	12.98	13.43

The critical path length (FT) = 166.08

The optimal sequence for the partial job set when the # of jobs in the partial set is 6 is as follows:

J1	J5	J6	J4	J2	J3
8.31	10.56	11.15	6.02	9.05	9.89
7.72	6.93	6.21	6.03	8.01	10.27
9.12	8.97	6.45	9.88	7.73	11.41
6.48	11.83	8.26	12.98	9.74	13.43

The critical path length (FT) = 195.51

## OUTPUT FILE #2: Floyd Matrix

The 1th iteration; slot = 0  
 Partialjobsettemp matrix is as shown below

7.6	9.4
7.2	8.7
6.6	8.7

The corresponding Floyd Matrix is as follows:

-1	-10	7.6	-10	-10	-10	-10	-10
-10	-1	-10	9.4	-10	-10	-10	-10
-10	0	-1	9.4	7.2	-10	-10	-10
-10	-10	-10	-1	-10	8.7	-10	-10
-10	-10	-10	0	-1	8.7	6.6	-10
-10	-10	-10	-10	-10	-1	-10	8.7
-10	-10	-10	-10	-10	0	-1	8.7
-10	-10	-10	-10	-10	-10	-10	-1

The 1th iteration; slot = 1

Partialjobsettemp matrix is as shown below

9.4	7.6
8.7	7.2
8.7	6.6

The corresponding Floyd Matrix is as follows:

-1	-10	9.4	-10	-10	-10	-10	-10
-10	-1	-10	7.6	-10	-10	-10	-10
-10	0	-1	7.6	8.7	-10	-10	-10
-10	-10	-10	-1	-10	7.2	-10	-10
-10	-10	-10	0	-1	7.2	8.7	-10
-10	-10	-10	-10	-10	-1	-10	6.6
-10	-10	-10	-10	-10	0	-1	6.6
-10	-10	-10	-10	-10	-10	-10	-1

The 2th iteration; slot = 0  
 Partialjobsettemp matrix is as shown below

9.6	7.6	9.4
9.4	7.2	8.7
6.2	6.6	8.7

The corresponding Floyd Matrix is as follows:

-1	-10	-10	9.6	-10	-10	-10	-10	-10	-10	-10	-10
-10	-1	-10	-10	7.6	-10	-10	-10	-10	-10	-10	-10
-10	-10	-1	-10	-10	9.4	-10	-10	-10	-10	-10	-10
-10	0	-10	-1	7.6	-10	9.4	-10	-10	-10	-10	-10
-10	-10	0	-10	-1	9.4	-10	7.2	-10	-10	-10	-10
-10	-10	-10	-10	-10	-1	-10	-10	8.7	-10	-10	-10
-10	-10	-10	-10	0	-10	-1	7.2	-10	6.2	-10	-10
-10	-10	-10	-10	-10	0	-10	-1	8.7	-10	6.6	-10
-10	-10	-10	-10	-10	-10	-10	-10	-1	-10	-10	8.7
-10	-10	-10	-10	-10	-10	-10	0	-10	-1	6.6	-10
-10	-10	-10	-10	-10	-10	-10	-10	0	-10	-1	8.7
-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	-1

The 2th iteration; slot = 1  
 Partialjobsettemp matrix is as shown below

7.6	9.6	9.4
7.2	9.4	8.7
6.6	6.2	8.7

The corresponding Floyd Matrix is as follows:

-1	-10	-10	7.6	-10	-10	-10	-10	-10	-10	-10	-10
-10	-1	-10	-10	9.6	-10	-10	-10	-10	-10	-10	-10
-10	-10	-1	-10	-10	9.4	-10	-10	-10	-10	-10	-10
-10	0	-10	-1	9.6	-10	7.2	-10	-10	-10	-10	-10
-10	-10	0	-10	-1	9.4	-10	9.4	-10	-10	-10	-10
-10	-10	-10	-10	-10	-1	-10	-10	8.7	-10	-10	-10
-10	-10	-10	-10	0	-10	-1	9.4	-10	6.6	-10	-10
-10	-10	-10	-10	-10	0	-10	-1	8.7	-10	6.2	-10
-10	-10	-10	-10	-10	-10	-10	-10	-1	-10	-10	8.7
-10	-10	-10	-10	-10	-10	-10	0	-10	-1	6.2	-10
-10	-10	-10	-10	-10	-10	-10	-10	0	-10	-1	8.7
-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	-1

The 2th iteration; slot = 2  
 Partialjobsettemp matrix is as shown below

7.6	9.4	9.6
7.2	8.7	9.4
6.6	8.7	6.2

The corresponding Floyd Matrix is as follows:

-1	-10	-10	7.6	-10	-10	-10	-10	-10	-10	-10	-10
-10	-1	-10	-10	9.4	-10	-10	-10	-10	-10	-10	-10
-10	-10	-1	-10	-10	9.6	-10	-10	-10	-10	-10	-10
-10	0	-10	-1	9.4	-10	7.2	-10	-10	-10	-10	-10
-10	-10	0	-10	-1	9.6	-10	8.7	-10	-10	-10	-10
-10	-10	-10	-10	-10	-1	-10	-10	9.4	-10	-10	-10
-10	-10	-10	-10	0	-10	-1	8.7	-10	6.6	-10	-10
-10	-10	-10	-10	-10	0	-10	-1	9.4	-10	8.7	-10
-10	-10	-10	-10	-10	-10	-10	-10	-1	-10	-10	6.2
-10	-10	-10	-10	-10	-10	-10	0	-10	-1	8.7	-10
-10	-10	-10	-10	-10	-10	-10	-10	0	-10	-1	6.2
-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	-10	-1

### Appendix D: Test Excel Sheets for 3, 4, 5 job cases

Number of Jobs: 3

Number of Machines: 4

*The Optimal FT values are in bold italics*

	Matrix 1				Matrix 2			
SEQ	<i>M = 2</i>	<i>M = 3</i>	<i>M = 4</i>	<i>M = 5</i>	<i>M = 2</i>	<i>M = 3</i>	<i>M = 4</i>	<i>M = 5</i>
ABC	<b>125.3</b>	<b>141.1</b>	139.5	164.1	<b>140.6</b>	142.1	<b>146.9</b>	188
ACB	<b>125.3</b>	141.6	<b>138.5</b>	<b>163.1</b>	<b>140.6</b>	<b>141</b>	147.5	<b>186.6</b>
Ends	<i>BAC</i>	BAC	<i>BAC</i>	<i>BAC</i>	<i>BCA</i>	BCA	<i>BCA</i>	BCA
Ends_r	<i>CAB</i>	<i>CAB</i>	CAB	CAB	<i>ACB</i>	ACB	ACB	<i>ACB</i>
Percentage error from optimal	0	0	0	0	0	0	0	0

	Matrix 3				Matrix 4			
SEQ	<i>M = 2</i>	<i>M = 3</i>	<i>M = 4</i>	<i>M = 5</i>	<i>M = 2</i>	<i>M = 3</i>	<i>M = 4</i>	<i>M = 5</i>
ABC	<b>115.1</b>	133.2	128.6	133.2	<b>103.9</b>	106.5	<b>119.1</b>	<b>150.5</b>
ACB	<b>115.1</b>	<b>127</b>	<b>125.4</b>	<b>127</b>	<b>103.9</b>	<b>105.4</b>	120.2	152.5
Ends	<i>BCA</i>	BCA	BCA	BCA	<i>CAB</i>	BAC	<i>CAB</i>	<i>CAB</i>
Ends_r	<i>ACB</i>	<i>ACB</i>	<i>ACB</i>	<i>ACB</i>	<i>BAC</i>	<i>BAC</i>	BAC	BAC
Percentage error from optimal	0	0	0	0	0	0	0	0

	Matrix 5				Matrix 6			
SEQ	<i>M = 2</i>	<i>M = 3</i>	<i>M = 4</i>	<i>M = 5</i>	<i>M = 2</i>	<i>M = 3</i>	<i>M = 4</i>	<i>M = 5</i>
ABC	<b>107.9</b>	120.9	<b>119.5</b>	<b>144.6</b>	<b>115.4</b>	120.18	124.9	152.97
ACB	<b>107.9</b>	<b>116.5</b>	121	148.6	<b>115.4</b>	<b>115.6</b>	<b>123.98</b>	<b>149.81</b>
Ends	<i>CAB</i>	CAB	<i>CAB</i>	<i>CAB</i>	<i>CAB</i>	CAB	CAB	CAB
Ends_r	<i>BCA</i>	<i>BCA</i>	BCA	BCA	<i>BAC</i>	<i>BAC</i>	BAC	BAC
Percentage error from optimal	0	0	0	0	0	0	0	0

Number of Jobs: 3



Number of Machines: 4

	<b>Matrix 7</b>				<b>Matrix 8</b>			
<b>SEQ</b>	<i>M = 2</i>	<i>M = 3</i>	<i>M = 4</i>	<i>M = 5</i>	<i>M = 2</i>	<i>M = 3</i>	<i>M = 4</i>	<i>M = 5</i>
ABC	<b>113.59</b>	129.32	125.91	<b>152.74</b>	<b>107.47</b>	<b>114.96</b>	<b>114.84</b>	<b>114.96</b>
ACB	<b>113.59</b>	<b>125.96</b>	<b>125.09</b>	155.85	<b>107.47</b>	121.31	120.43	121.31
Ends	<i>BAC</i>	BAC	<i>BAC</i>	BAC	<i>CBA</i>	CBA	CBA	CBA
Ends_r	<i>CAB</i>	<i>CAB</i>	CAB	<i>CAB</i>	<i>ABC</i>	<i>ABC</i>	<i>ABC</i>	<i>ABC</i>
Percentage error from optimal	0	0	0	0	0	0	0	0

	<b>Matrix 9</b>				<b>Matrix 10</b>			
<b>SEQ</b>	<i>M = 2</i>	<i>M = 3</i>	<i>M = 4</i>	<i>M = 5</i>	<i>M = 2</i>	<i>M = 3</i>	<i>M = 4</i>	<i>M = 5</i>
ABC	<b>100.15</b>	<b>108.46</b>	<b>109.33</b>	122.47	<b>129.38</b>	139.27	137.31	<b>164.95</b>
ACB	<b>100.15</b>	108.59	109.37	<b>121.07</b>	<b>129.38</b>	<b>134.91</b>	<b>134.41</b>	172.91
Ends	<i>ABC</i>	<i>ABC</i>	<i>ABC</i>	ABC	<i>CAB</i>	<i>CAB</i>	CAB	<i>CAB</i>
Ends_r	<i>CBA</i>	CBA	CBA	<i>CBA</i>	<i>BAC</i>	BAC	<i>BAC</i>	BAC
Percentage error from optimal	0	0	0	0	0	0	0	0

Number of Jobs: 4  
 Number of Machines: 4

	Matrix 1			Matrix 2			Matrix 3		
SEQ	M=3	M=4	M=6	M=3	M=4	M=6	M=3	M=4	M=6
ABCD	176.5	198.8	298.2	<b>172</b>	182.4	182.4	147	157	157
BACD	175.5	<b>195.2</b>	298.2	174	186.5	186.5	<b>146</b>	<b>154.4</b>	<b>154.4</b>
ACBD	177	195.7	298.2	<b>172</b>	183.2	183.2	147	158.7	158.7
ADBC	<b>175.5</b>	198.8	<b>295.5</b>	174	184.3	184.3	148	155.1	155.1
CBAD	177	198.8	298.2	174	183.1	183.1	147	156	156
BDCA	177	198.8	295.5	<b>172</b>	<b>182.1</b>	<b>182.1</b>	147	154.7	154.7
Incr	DBAC	DABC	DABC	DABC	DABC	DABC	CBDA	CBDA	CBDA
Decr	BDCA	ADCB	ADCB	ACDB	ADCB	ADCB	BCAD	BCAD	BCAD
Ends	CBDA	<i>CDBA</i>	<i>CADB</i>	<i>DACB</i>	<i>CADB</i>	<i>CADB</i>	ABCD	ABCD	ABCD
Ends_r	<i>ADBC</i>	BDAC	BDAC	BDAC	BDAC	<i>BDCA</i>	DCBA	DCBA	<i>CDBA</i>
Percentage error from optimal	0	0	0	0	0	0	0.068	1.025641	0

SEQ	Matrix 4			Matrix 5			Matrix 6		
	M=3	M=4	M=6	M=3	M=4	M=6	M=3	M=4	M=6
ABCD									
BACD	151	<b>174.1</b>	264.5	150	159.4	169.5	160	184.1	247.5
ACBD	152	174.6	<b>262.7</b>	146	156.8	<b>158.9</b>	162	182.27	252.26
ADBC	147	175.9	263.4	147	162.9	172.1	<b>158</b>	183.24	<b>245.72</b>
CBAD	<b>144</b>	174.8	264.5	152	159.2	161.3	158	183.6	247
BDCA	154	175.9	262.7	<b>144</b>	<b>156.1</b>	161.7	163	<b>182.09</b>	251.39
	155	174.1	263.4	148	161	168.5	161	183.92	246.4
Incr	ACDB	ACDB	ACDB	ABDC	ABDC	ABDC	CABD	CABD	CABD
Decr	CABD	CABD	BCAD	BACD	BACD	BACD	ACDB	ACDB	ACDB
Ends	<i>BCAD</i>	<i>ABCD</i>	<i>BACD</i>	<i>CBAD</i>	<i>CBAD</i>	<i>DBAC</i>	<i>DACB</i>	<i>ADCB</i>	<i>DACB</i>
Ends_r	DACB	DACB	DACB	DABC	DABC	DABC	BCAD	BCAD	BCAD
Percentage error from optimal	0	0	0	0	0	0	0	0	0

Number of Jobs: 4  
 Number of Machines: 4

SEQ	Matrix 7			Matrix 8			Matrix 9		
ABCD	<i>M=3</i>	<i>M = 4</i>	<i>M = 6</i>	<i>M=3</i>	<i>M = 4</i>	<i>M = 6</i>	<i>M=3</i>	<i>M = 4</i>	<i>M = 6</i>
BACD	142	158.21	<b>214.19</b>	162	184.51	183.51	162	<b>152.45</b>	<b>226.68</b>
ACBD	142	163.59	225.19	160	182.77	182.77	165	155.68	226.91
ADBC	144	<b>156.45</b>	219.67	<b>158</b>	<b>179.62</b>	<b>177.62</b>	<b>158</b>	156.47	228.07
CBAD	144	170.13	224.81	163	184.98	184.98	163	154.99	226.68
BDCA	<b>141</b>	161.06	217.63	159	180.09	179.09	159	158.99	228.25
	143	157.41	222.34	160	181.83	179.83	160	153.36	228.83
Incr	CBDA	CBDA	CBDA	DABC	DABC	DABC	CBAD	BCAD	BCAD
Decr	BCAD	BCAD	BCAD	ACDB	ADCB	ADCB	BCDA	CBDA	CBDA
Ends	ABCD	ABCD	<i>ABCD</i>	DACB	CADB	CADB	DBCA	DCBA	DCBA
Ends_r	<i>DCBA</i>	DCBA	DCBA	<i>BDAC</i>	<i>BDAC</i>	<i>BDAC</i>	<i>ACBD</i>	<i>ABCD</i>	<i>ABCD</i>
Percentage error from optimal	0	1.112	0	0	0	0	0	0	0

SEQ	Matrix 10		
ABCD	<i>M=3</i>	<i>M = 4</i>	<i>M = 6</i>
BACD	168	185.42	185.42
ACBD	171	192.07	192.07
ADBC	167	190.65	190.65
CBAD	170	183.88	183.88
BDCA	169	186.84	186.84
	<b>165</b>	<b>183.49</b>	<b>183.49</b>
Incr	CDAB	CDAB	CDAB
Decr	DCBA	DCBA	DCBA
Ends	<i>BDCA</i>	<i>BDCA</i>	<i>BDCA</i>
Ends_r	ACDB	ACDB	ACDB
Percentage error from optimal	0	0	0

Number of jobs: 5

Number of Machines: 4

SEQ	Matrix 1		Matrix 2		Matrix 3		Matrix 4		Matrix 7		Matrix 8		Matrix 9	
	M = 4	M = 5	M = 4	M = 5	M = 4	M = 5	M = 4	M = 5	M = 4	M = 5	M = 4	M = 5	M = 4	M = 5
ABCDE	230.4	284.3	222.7	222.7	<b>181.59</b>	201	215.4	268.3	206.96	255.89	<b>191.37</b>	<b>194.35</b>	180.13	196.35
ABCED	235.4	285.8	219.9	219.9	210.2	217	215.1	269	229.34	259.01	198.59	206.92	183.09	194.88
ABECD	234.4	285.4	219.2	219.2	215.3	222.1	217.6	266.1	229.48	256.54	203.99	219.48	182.21	194.4
ABEDC	232.6	285.4	<b>218.8</b>	<b>218.8</b>	213.2	218.4	214.9	269.4	216.57	256.49	205	228.00	<b>176.54</b>	<b>179.5</b>
ABDCE	232.1	286.4	222.6	222.6	204.7	204.7	214.7	269.2	213.28	258.23	201.8	209.93	183.3	206.85
ABDEC	230.2	286.5	219.4	219.4	204.4	209.6	213.5	266.2	<b>204.59</b>	254.33	195.85	202.52	182.79	197.03
AEBCD	234.7	289	220.7	220.7	218.9	225.7	213.6	<b>263.9</b>	229.61	259.25	193.55	211.17	179.65	199.56
AEBDC	235.1	289	219.8	219.8	216.8	222	214.4	267.2	216.4	258.47	199.81	213.66	180.11	192.61
AEDBC	240.1	289	219.9	219.9	211.6	216.8	215.4	267	210.6	254.84	199.83	226.22	182.22	184.25
AEDCB	239.4	289	223.7	223.7	214.1	215.3	214.2	267.5	219.41	259.18	204.64	231.19	180.66	182.15
AECBD	233.7	287.4	221.3	221.3	217.4	224.2	214.4	264.4	231.88	258.33	199.17	217.84	177.02	194.75
AECDB	231.2	289	224.9	224.9	217.8	219	220.4	266.5	222.17	254.72	200.37	225.98	177.06	191.41
ADBCE	237.4	285.8	221.5	221.5	199.5	199.5	225.2	267.4	208.13	253.38	194.96	223.54	183.84	207.39
ADBEC	237.4	288	<i>218.8</i>	<i>218.8</i>	199.2	204.4	223.3	264.4	206.09	<b>249.67</b>	202.52	221.95	178.15	190.3
ADEBC	236.5	287.8	<i>218.8</i>	<i>218.8</i>	202.8	208	216.6	264.3	205.91	252.94	192.85	203.53	187.67	204.34
ADECB	237	287.8	222.4	222.4	205.3	206.5	215.4	265.8	224.1	255.82	199.37	210.28	179.08	200.98
ADCBE	237.9	286	220.8	220.8	198	198	214.4	267.3	208.71	256.4	204.96	229.95	184.18	196.7
ADCEB	237.9	286	220.7	220.7	197.9	<b>197.9</b>	214.4	267.5	210.02	259.18	215.67	229.86	181.84	195.19
ACBDE	<b>229.9</b>	<b>281.4</b>	222.6	222.6	206.9	206.9	212.6	267.5	215.93	254.12	194.83	212.34	176.7	192.64
ACBED	234.6	282.9	220	220	208.7	215.5	215.8	268.7	235	256.28	200.78	221.42	179.16	184.19
ACEBD	231.5	281.6	220.6	220.6	212.3	219.1	216	267.8	235.17	258.23	195.9	211.33	184.81	207.79
ACEDB	231.2	282.9	223.6	223.6	212.7	213.9	217	269.2	224.88	254.69	196.46	218.36	181.08	193.23
ACDBE	231.9	283.7	222	222	201.7	201.7	213.5	266.6	215.14	249.76	202.13	224.88	177.49	185.91
ACDEB	231.4	285.9	223.6	223.6	204.5	205.7	216	267	221.64	253.03	200.63	211.27	181.59	201
Ends	<i>CBDEA</i>	CBDEA	<i>AECDB</i>	AECDB	<i>BCDEA</i>	BEDCA	<i>ACBDE</i>	CADEB	<i>BCDEA</i>	DBECA	<i>CDEAB</i>	CDEAB	<i>DCABE</i>	DCABE
Ends_r	DBACE	DBACE	BCADE	BCADE	ADECB	ADECB	BDAEC	BCDAE	ACDBE	ACDBE	BEDAC	BEDAC	EBACD	EBACD
% error from optimal	0	0	0	0	0	4.016	0	0	1.145	0	0	0	0	0

## VITA

Nipun P. Palekar was born on April 14, 1976 in Bombay (Mumbai), India. He did his schooling and BS in Production Engineering from the University of Bombay, India. During his undergraduate program he was an intern at Larsen & Toubro at Bombay, for six months. After completion of his BS, he worked with Siemens in Bombay at their switchgear plant in the capacity of a Production Engineer and Shop Scheduler. It was at this job that he realized the importance of Optimization and properly planned Shop Scheduling and he decided to pursue his Master's degree in Industrial Engineering at Virginia Tech. During his MS, he maintained the strong academic achievements he had shown during his undergraduate studies. He worked for a year as a Research Assistant at Ericsson, Lynchburg with Dr. Sarin as the Principle Investigator. During his second year as a Master's student, he held the post of 'President of SME', Virginia Tech Chapter. After graduation, he will be working as an Industrial Engineer incharge of Factory Performance at the semiconductor fab at Motorola Inc. His research interests are in Sequencing & Scheduling, Production Management and Control, and Operations Management.