



**On the Solvability and Implementation  
of  
Mixed-Model Just-In-Time Production System**

**Dissertation**

**Submitted To**

**Central Department of Computer Science and Information Technology**

**Institute of Science and Technology**

**Tribuvan University**

**In Partial Fulfillment of the Requirements for the Degree of**

**Master of Science  
in  
Computer Science and Information Technology**

**By:  
Deepak Panday**

**September, 2008  
Kirtipur, Nepal**

## **Recommendation**

This is to certify that the thesis titled “**On the Solvability and Implementation of Mixed-Model Just-In-Time Production System**”, submitted by Deepak Panday in partial fulfillment of the requirement for the award of the degree of Masters of Science in Computer Science and Information Technology has been carried out under my supervision. The dissertation fulfills the requirement related to the nature and standard of the work for the award of Master of Science in Computer Science and Information Technology and no part of this thesis has been published or submitted for the award of any degree elsewhere in the past.

**Dr. Tanka Nath Dhamala,**  
Head,  
Center Department of Computer Science  
and Information Technology,  
Tribhuvan University, Nepal

( Supervisor )

## Evaluation Committee

We certify that we have read this dissertation and in our opinion, it is satisfactory in the scope and quality as a dissertation in the partial fulfillment for the requirement of Master's Degree in Computer Science and Information Technology from Tribhuvan University, Nepal.

.....

.....

**Dr. Tanka Nath Dhamala,**  
Head,  
Center Department of Computer Science  
and Information Technology,  
Tribhuvan University, Nepal

**Dr. Tanka Nath Dhamala,**  
Head,  
Center Department of Computer Science  
and Information Technology,  
Tribhuvan University, Nepal

( **Supervisor** )

.....

( **External Examiner** )

.....

( **Internal External** )

**Date :-----**

## **Acknowledgements**

It has been a matter of great pride for me to study at Central Department of Computer Science and Information Technology at Tribhuvan University (CDCSIT-TU) where I got an opportunity to enhance my knowledge level in splendid environment provided by the department.

I am especially obliged to my supervisor Dr. T.N. Dhamala, Head, Central Department of Computer Science and Information Technology, for his constructive criticism, valuable suggestions to make my work fruitful.

I am always eager to express my thanks to Prof. Dr. D.D. Paudyal, former Head, Central Department of Computer Science and Information Technology, for his constant motivation in our study. I really owe special thanks to Prof. Dr. S.R. Joshi, Mr. M. Pokheral, Dr. S. Shakya, Mr. M.B. Khatai, Mr. A. Timilsina for providing valuable education.

Finally but not the least, my sincere gratitude goes to my friends, colleagues, family members for their kind cooperation in the completion of this thesis.

September, 2008

**-Deepak Panday**

## **Abstract**

Scheduling problems are most primitive problems in Computer Science and Industries. Obtaining an optimal sequence in mixed-model production system under the just-in-time philosophy is one of such a challenging problem. The problem in a multilevel facility are strongly NP-hard, however, the single-level problems are pseudo-polynomial solvable. In this dissertation, developments of mixed-model just-in-time production problems are studied thoroughly. Different purposed algorithms are tested for their solvability and implementation purpose. Lastly, more practical mixed-model just-in-time sequencing problem is considered with the given set of sequences as precedence constraints. An efficient algorithm, which obtains an optimal solution for the maximum deviation objective in single level is studied and is extended as a solution for overlapping sequences.

# Table of Contents

<b>1</b>	<b>Introduction.....</b>	<b>1</b>
1.1	Background.....	1
1.2	Literature review.....	2
1.3	Organization of the thesis.....	4
1.4	Methodology.....	5
<b>2</b>	<b>Computational complexity.....</b>	<b>7</b>
2.1	Asymptotic notation.....	8
2.2	Complexity classes.....	9
<b>3</b>	<b>Scheduling algorithms.....</b>	<b>11</b>
3.1	Classification of scheduling problems.....	11
3.1.1	Scheduling problems.....	11
3.1.2	The three-field notation $r s x$ .....	12
3.1.3	Examples.....	15
3.2	Combinatorial optimization.....	17
3.2.1	Linear and integer programming.....	17
3.2.2	The assignment problem.....	18
3.3	Single machine scheduling problems.....	19
3.4	Maximum lateness and related criteria.....	19
3.5	Some simple scheduling algorithm.....	21
3.5.1	Minimizing maximum lateness.....	21
3.5.2	Minimizing total delay.....	22
3.6	Realistic scheduling problem.....	22
3.6.1	Online scheduling.....	23
3.6.2	Real-time scheduling.....	23
3.7	Scheduling in operating system.....	24
<b>4</b>	<b>Just-in-time production system.....</b>	<b>26</b>

4.1	Some key elements of JIT.....	26
4.2	JIT system examples.....	27
4.2.1	JIT-style learning and training.....	27
4.2.2	Kanban – a communication tool in JIT production system.....	28
4.2.3	Toyota production system.....	29
4.3	Mixed-model production.....	29
4.4	Push versus pull production system.....	30
<b>5</b>	<b>Mixed-model just-in-time production system.....</b>	<b>31</b>
5.1	The PRV problem formulation.....	31
5.2	The ORV problem formulation.....	33
5.3	The total PRV problem.....	36
5.3.1	Assignment costs.....	36
5.3.2	Assignment method.....	37
5.4	The pegged ORV problem.....	38
5.5	Cyclic sequence.....	39
<b>6</b>	<b>Mixed-model JIT problems.....</b>	<b>40</b>
6.1	Nearest integer point problem.....	40
6.2	Dynamic programming algorithm.....	43
6.2.1	Problem statement.....	43
6.2.2	Dynamic programming (DP) procedure.....	44
6.3	Min-max-absolute-chain algorithm.....	45
6.3.1	Min-max-absolute-chain-algorithm.....	46
6.3.2	Min-max-absolute-chain-algorithm for overlapping sequence.....	46
<b>7</b>	<b>Implementation.....</b>	<b>48</b>
7.1	Architecture view of project.....	48
7.1.1	UTIL package.....	50
7.1.2	GUI package.....	51
7.2	Nearest integer point implementation issue.....	52

7.2.1	Comparative study of Algorithm 1 and Heuristic 1 .....	55
7.3	Earliest due date implementation issue .....	57
7.4	Dynamic programming implementation issue .....	59
7.5	Cost assignment problem .....	62
7.5.1	Comparative study of min-sum PRV problem .....	64
7.6	Min-max-absolute-chain sequencing problem .....	66
7.6.1	Non-overlapping sequences .....	66
7.6.2	Overlapping-sequences .....	68
<b>8</b>	<b>Conclusion and Recommendation .....</b>	<b>70</b>
<b>9</b>	<b>References .....</b>	<b>71</b>



## List of Tables

Table 1	Schedule generated by Algorithm 1.....	53
Table 2	Schedule generated by Heuristic 1.....	54
Table 3	Analytical view of Algorithm 1 vs. Heuristic 1.....	56
Table 4	Schedule generated by earliest due date .....	58
Table 5	Schedule generated by dynamic programming.....	61
Table 6	Excess inventory or shortage costs calculated.....	63
Table 7	Schedule generated by cost assignment problem .....	63
Table 8	Comparative studies of min-sum PRV problems .....	65
Table 9	Calculation of window value for non-overlapped sequence.....	67
Table 10	Output: min-max-absolute-chain (non-overlapping sequence) .....	68
Table 11	Calculation of window value for overlapped sequence .....	69
Table 12	Output: min-max-absolute-chain algorithm (overlapping sequences).....	69

## List of Figures

Figure 1 Gantt Charts.	12
Figure 2 $P \mid \text{prec}; p_i = 1 \mid C_{\max}$	16
Figure 3 $1 \mid \text{batch} \mid \sum w_i C_i$	17
Figure 4 Architectural view of whole project	49
Figure 5 Interface of main GUI	52
Figure 6 Architectural view of nearest-integer-point package	52
Figure 7 Date input frame for nearest-integer-point package.	53
Figure 8 Architectural view of earliest due date package	57
Figure 9 Date input frame for earliest due date package	58
Figure 10 Architectural view of dynamic programming	59
Figure 11 Date input frame for dynamic programming testing	60
Figure 12 Architectural view of cost assignment problem	62
Figure 13 Date input frame for cost assignment problem	63
Figure 14 Architectural view of min-max-absolute-chain sequencing problem	66
Figure 15 Input frame: min-max-absolute-chain (non-overlapping sequence)	67
Figure 16 Input frame: min-max-absolute-chain (overlapping sequence)	69

# **1 Introduction**

## **1.1 Background**

Just-in-time manufacturing means producing the necessary items in necessary quantities at the necessary time. It is a philosophy of continuous improvement in which non-value-adding activities (or wastes) are identified and removed.

JIT applies primarily to *repetitive manufacturing* processes in which the same products and components are produced over and over again. The general idea is to establish flow processes (even when the facility uses a jobbing or batch process layout) by linking work centers so that there is an even, balanced flow of materials throughout the entire production process, similar to that found in an assembly line. To accomplish this, an attempt is made to reach the goals of driving all inventory buffers toward zero and achieving the ideal lot size of one unit.

The basic elements of JIT were developed by Toyota in the 1950's, and became known as the Toyota Production System (TPS). JIT was well established in many Japanese factories by the early 1970's. JIT began to be adopted in the U.S. in the 1980's (General Electric was an early adopter), and the JIT/lean concepts are now widely accepted and used.

## 1.2 Literature review

The JIT production systems have been used in mixed-model assembly lines in order to respond to the customer demands for a variety of products without holding large inventories or incurring large shortages. Such mixed-model must have negligible changeover costs between the products and the production must be small lot fashion.

Under the assumption that the products required approximately the same number and mix of parts, Miltenburg [24], purposed an optimal formulation that aims to minimize the total deviation or sum of all deviations of the real production from the ideal production, and defined the problem as a nonlinear integer programming. This assumption reduced the formulation into a single-level and focused study on the final assembly line only. The final assembly line controls the JIT system by keeping a constant rate of usage of every part used by the line. Miltenburg and Sinnamon [26] and Miltenburg and Goldstein [25] extended the formulation to multi-level system. Here the assembly line is used as a pull line that defines the requirements down the level. Such assembly line is called the final assembly line and the system as a pull system.

Kubiak [17] gave a more specific distinction between these problems and referred single-level problem as the Product Rate Variation (PRV) problem and the multi-level problem as the Output Rate Variation (ORV) problem. Kubiak distinguished PRV problem as total deviation PRV problem and maximum deviation PRV problem. The total deviation PRV problem aims to minimize the sum of total deviations and hence looks on minimizing the total variation between the actual product and the ideal production at any stage. Such problem represents the problem considered by Miltenburg [24]. The maximum deviation PRV problem minimizes the maximum deviation of the actual product of a product from its ideal level of production. Such problem represents the problem considered by Steiner and Yeomnas [35]. The pull system where the final

assembly line defines the scheduling and requests for demand down the level is represented by ORV problem. The problems consider by Miltenburg and Sinnamon [26] and Miltenburg and Goldstein [25] are categorized as ORV problem.

Kubiak [17] and Kubiak et al. [21] proved that even special cases of the ORV problem are NP-hard. Miltenburg et al. [27] and Kubiak et al. [21] present the dynamic programming approach as a solution procedure for the ORV problem. Steiner and Yeomans [34] showed that the ORV problem could be reduced to weighted PRV problem using concept of pegging (i.e. the parts dedicated to be assembled into the different products are distinguished). In [19,20] Kubiak and Sethi showed that the nonlinear integer programming problem used for sequencing flexible transfer lines could be reduced the problem as an assignment problem (cost assignment problem). The earliest due date (EDD) algorithm developed by Inman and Bulfin [14] provided a robust solution to the min-sum PRV problem. By using the graph-theoretic approach, Steiner and Yeomans [35], proved that the maximum deviation PRV problem (the Bottleneck PRV problem) could be solved with a pseudo-polynomial algorithm by reducing it to release date / due date decision problem.

Brauner and Crama [2] showed that the problem is Co-NP and polynomially solvable when the number of products are fixed. Miltenburg and Simmamon [26] studied that theoretical base for ORV problem. They presented a mathematical model and two heuristics for finding a good scheduler of ORV problem. Steiner and Yeomans [34] investigated that if outputs at production levels, which feed the final assembly level, are dedicated or pegged to the final product into which they will be assembled, and then there could exist an efficient scheduling algorithm to determine the optimal level schedule. They showed that the ORV problem under pegging assumption is equivalent to a weighted single-level PRV problem and by modifying PRV version of the problem, pegged ORV problem could be solved to optimal which is polynomial in the total product

demand and the weight factors. Miltenburg [24] (for PRV) and Miltenburg and Sinnamon [26] (for ORV) observed the existence of the cyclic sequences for scheduling larger products. Kubiak [18] proved that for any PRV problem if  $S$  is an optimal sequence with input demands  $d_1, \dots, d_i, \dots, d_n$ , then concatenation  $S^m$  of  $m$  copies of  $S$  is an optimal sequence with input demands  $md_1, \dots, md_i, \dots, md_n$ . Dhamala and Kubiak [10] conjectured that cyclic sequences in the ORV case were optimal, too.

Dhamala [8] had developed a min-max-absolute chain algorithm for combining chain sequences with objective of minimizing the maximum deviation. In [16], Kovalyov, Kubiak, and Yeomans, had done comparative study of balanced mixed-model JIT optimization algorithm. Similar, comparative study had been done by Lebacque, Jost and Brauner [22]. Dhamala and Khadka [9] had review different mixed-model JIT sequences problems.

### **1.3 Organization of the thesis**

This dissertation has been divided in 8 chapters. The brief descriptions of these following chapters are :

Chapter 1 gives a brief introduction and background of the problem raised in this dissertation. Chapter 2 deals with computational complexity theory. In this chapter nature of good and bad algorithms are studied thoroughly. A synopsis of asymptotic notation and different complexity class are given. In Chapter 3, the formal description of scheduling theory is summarized. Section 3.1 deals with classification of scheduling problem. Combination optimization is studied in Section 3.2. Single machine scheduling problem is studied in Section 3.3. Similarly some simple scheduling algorithms have been mentioned in Section 3.5.

Section 3.6 deals with some realistic scheduling problems and likely a survey on scheduling problems in Operation System are done in Section 3.7.

In Chapter 4, JIT production system is studied in brief. Section 4.1 deals with different JIT key elements. Some of JIT production systems have been mentioned in Section 4.2. Section 4.3 includes definition of mixed-model production system and Section 4.4 differentiates between push and pull System. In Chapter 5, different mathematical models for mixed-model JIT problem are studied thoroughly. Section 5.1 details with the PRV problem formulation. In Section 5.2, ORV problem is studied. Section 5.3 analyzes the total PRV problem. Section 5.4 describes the pegged ORV problem. And, possibility of existence of cyclic sequence is studied in Section 5.5.

In Chapter 6, we review some of the solution for the mixed-model JIT problems. Section 6.1 details the nearest integer point problem and its different algorithms and heuristics. Section 6.2 describes the dynamic programming algorithm. And lastly min-max-absolute-chain algorithm is studied in Section 6.3. Moreover, this Section also contains the possible extension of min-max-absolute-chain algorithm for overlapping sequence. In Chapter 7, implementations issues are covered. This chapter also contains the conclusion, further suggestions and recommendations.

## **1.4 Methodology**

The initial phase of the research is devoted on the detail study on the relevant documents on different sequencing algorithms. The major portion of the thesis is devoted on the analysis of mixed-model just-in-time production problems. After the study of

those problems, different mixed-model JIT problems are tried to implement for testing their solvability and efficiency.

An application is developed to eliminate the constraints of non-overlapping sequence in min-max-absolute-chain-algorithm proposed by Dhamala [8].

The implementation of this thesis is made in JAVA. Each mixed-model just-in-time production problem has its own package. The secondary data are taken from related papers for testing the correctness of the algorithms.



## 2 Computational complexity

Computational complexity deals with analysis of algorithms related with computer science and applied mathematics. It analyzes the nature of problem solvable by algorithms and classifies them into several classes regarding their difficulty and resources required for execution of the algorithm (eg. computation time). A typical question of this research is, “As the size of the input to an algorithm increases, how do the running time and memory requirements of the algorithm change and what are the implications and ramifications of that change?” The major factors considered during computational complexity are the time complexity and space complexity. The time complexity of a problem is the number of steps that it takes to solve an instance of the problem as a function of the size of input. The space complexity of a problem is a related concept that measures the amount of space, or memory required by the algorithm [4].

### Decision problems

The primitive step during the computational complexity is to identify either the problem is solvable or not and either any algorithm can solve a problem or not. Such a problem where the answer is always yes or not is called decision problem [6].

### Example 2.1

A well-known decision problem IS-PRIME returns a yes answer when a given input is a prime and no otherwise, while a problem IS-COMPOSITE determines whether a given integer is not a prime number. Decision problems are often considered because an arbitrary problem can always be reduced to some decision problem [6].

## **Turing machine**

Turing machines are extremely basic abstract symbol-manipulating devices, which, despite their simplicity, can be adapted to simulate the logic of any computer that could possibly be constructed. Alan Turing described them in 1936. It is purely a theoretical state machine, with infinite memory and computation time as its construction base on assumption that “Every solvable problem must be solved by Turing machines”[12].

### **2.1 Asymptotic notation**

Space and time complexity of any problem is highly influenced by the real machine used for computation. If a bad machine is used, even a better algorithm may appear inefficient compared to a bad algorithm in a better machine. Hence, machine difference must be minimized, and the goal can be achieved by considering with instances whose input size is very large. To describe the behavior of algorithm for large input, the concept of asymptotic order is used. Three asymptotic orders are frequently used, big O (upper-bound), big-Omega (lower-bound) and big-theta (tight-bound) [6, 30, 31].

#### **Definition 2.1.**

The function  $f(n) = O(g(n))$  (read as “f of n is big oh of g of n”) if and only if there exist positive constants  $c$  and  $n_0$  such that  $f(n) \leq c \times g(n)$  for all  $n, n \geq n_0$ .

## 2.2 Complexity classes

### Class P

Class P is one of the most fundamental complexity classes. It contains all decision problems, which can be solved by deterministic Turing machine using a polynomial amount of computational time [30, 12].

### Example 2.2

The problem of sorting  $n$  numbers can be done in  $O(n^2)$  time using insertion sorting in even worse case, thus all sorting problems fall under P class.

### Class NP

The class NP consists of those decision problems whose positive solution can be verified in polynomial time given the right information, i.e. those problems whose solution can be found in polynomial time on a non-deterministic machine [6,12,39].

### Polynomial Time Reduction

A function  $f:\{0,1\}^* \rightarrow \{0, 1\}^*$  is said to be polynomial-time computable if there exist a polynomial-time algorithm  $A$  such that, for all input  $x \in \{0, 1\}^*$ , produces the output  $f(x)$  in polynomial time. Again, let  $X$  and  $Y$  be two problems. We say that  $X$  is polynomially reducible to  $Y$  i.e.  $X \leq_p Y$ , if there exist a polynomial-time computable function  $f:\{0, 1\}^* \rightarrow \{0, 1\}^*$  such that for all  $x \in \{0, 1\}^*$ ,  $x \in X$  if and only if  $f(x) \in Y$ . The notation  $X \leq_p Y$  is often states as “ $Y$  is as harder as  $X$ ” [6,30].

**Lemma 2.1** Let  $X$  and  $Y$  be two decision problems such that  $X \leq_p Y$ , then  $Y \in P \Rightarrow X \in P$ .

### **NP-Complete class**

A decision problem  $X$  is NP-Complete if

1.  $X \in \text{NP}$
2. Every problem in NP is reducible to  $X$ .

i.e, NP-complete are the hardest problems among the NP-class.

### **Example 2.3**

One example of an NP-complete problem is the subset sum problem which is: given a finite set of integers, determine whether any non-empty subset of them sum to zero. A supposed answer is very easy to verify for correctness, but there is no known efficient algorithm to find an answer; that is, all known algorithms are impractically slow for larger sets of integers.

### **NP-hard**

It is a class of all problems  $X$  such that for all  $Y \in \text{NP}$ ,  $Y \leq_p X$  i.e. there may be a problem  $X$  which is as hard as any problem in NP, but one may not be able to prove its NP-completeness. NP-hard can contain problems other than decision problems [6, 30].

**Theorem 2.1.**  $P \subseteq \text{NP}$

**Theorem 2.2.** If any NP-complete problem is polynomial time solvable, then  $P = \text{NP}$ .

**Theorem 2.3.** If  $P \neq \text{NP}$ , then no NP-hard problem can be solved in polynomial time.

### **3 Scheduling algorithms**

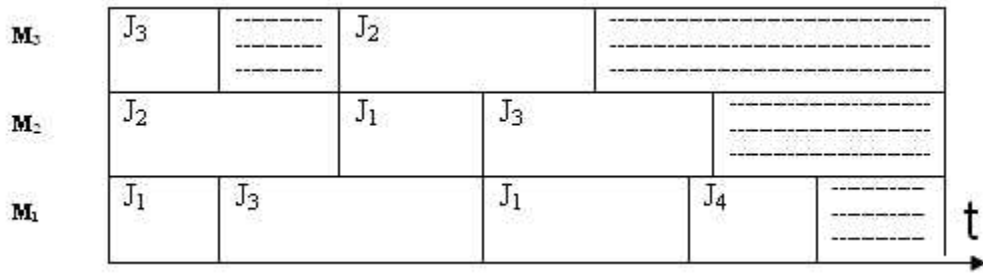
In a scheduling problem one has to find time slots in which activities (or jobs) should be processed under given constraints. The main constraints are resource constraints and precedence constraints between activities. In this chapter, the basic formulation of the scheduling theory is studied. The definition and classification of scheduling problems and examples mentioned in this chapter follow the notation used in [13]. As in this work single machine case is considered, only single machine scheduling problem are only briefed below the chapter.

#### **3.1 Classification of scheduling problems**

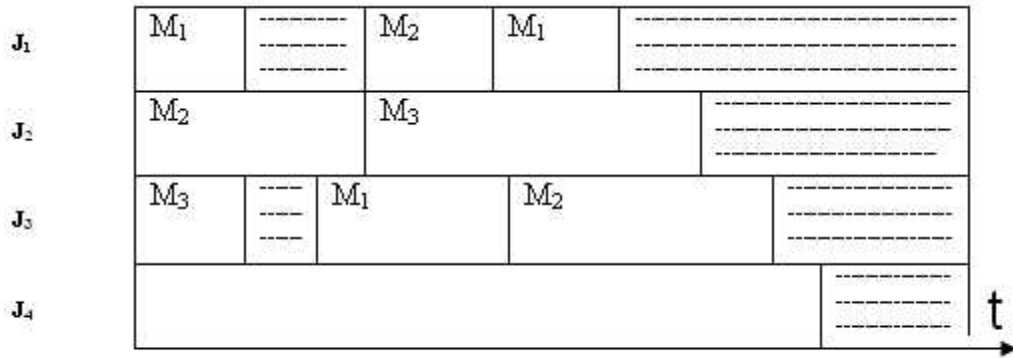
The theory of scheduling is characterized by a virtually unlimited number of problem types. Here basic classifications of the scheduling problems are covered.

##### **3.1.1 Scheduling problems**

Suppose that  $m$  machines  $M_j$  ( $j = 1, 2, 3, \dots, m$ ) have to process  $n$  jobs  $J_i$  ( $i = 1, 2, \dots, n$ ). A schedule is for each job an allocation of one or more time intervals on one or more machines. Schedules may be represented by Gantt charts as shown in Figure 1 *Gantt chats* may be machine oriented (Figure 1(a)) or job oriented (Figure 1(b)).



(a)



(b)

Figure 1 Gantt Charts.

### 3.1.2 The three-field notation $r|s|x$

Every scheduling problem can be denoted by the three-field notation  $r|s|x$ , where,  $r$  denotes machine environment,  $s$  denotes the jobs and  $x$  denotes objective function used in scheduling [4].

### **Machine environment**

The  $\Gamma$  notation denoting the machine environment consists of concatenation of  $\Gamma_1$  and  $\Gamma_2$  where  $\Gamma_1$  denotes the machine characteristic and  $\Gamma_2$  denotes number of machines used. Further more  $\Gamma_1 \in \{W, P, Q, R, O, F, J\}$  where

W : single machine

P : identical parallel machine

Q : uniform parallel machine

R : unrelated parallel machine

O : open shop

F : flow shop

J : job shop

Similarly,  $\Gamma_2 \in \{w, k\}$  where,

w : number of machines is assumed to be variable

k : k number of machines.

### **Job characteristics**

The  $S$  - field denotes the jobs and their interrelations is detailed by concatenation of six variables as  $S = S_1 \cdot S_2 \cdot S_3 \cdot S_4 \cdot S_5 \cdot S_6$  where,

$$S_1 = \begin{cases} w, & \text{if preemption is not allowed,} \\ \text{pmtm,} & \text{if preemption is allowed} \end{cases}$$

Preemption allow a job being processed to be paused arbitrarily, and start some another available job.

$$S_2 = \begin{cases} w, & \text{if no resource constraints,} \\ \text{resc}, & \text{if resource constraints are given.} \end{cases}$$

$$S_3 = \begin{cases} w, & \text{if no precedence constraints,} \\ \text{prec}, & \text{if precedence in the form of an arbitrary DAG given} \\ \text{tree}, & \text{if precedence in the form of tree given} \\ \text{intree}, & \text{if precedence in the form of intree given} \\ \text{outtree}, & \text{if precedence in the form of outtree given.} \end{cases}$$

$$S_4 = \begin{cases} w, & \text{if release date is zero for all job,} \\ r_j, & \text{if release date is given for each job.} \end{cases}$$

$$S_5 = \begin{cases} w, & \text{if jobs have arbitrary processing time,} \\ p, & \text{if all jobs have processing time equal to p.} \end{cases}$$

$$S_6 = \begin{cases} w, & \text{if no deadlines given,} \\ d, & \text{if jobs have deadlines.} \end{cases}$$

### Objective function

Let us denote each job by  $J_i$ , their finish times or the completion times by  $C_j$ , similarly release times by  $r_j$ , priority or weight assigned to each job by  $w_j$ . Then following parameter can be calculated.

$$\text{Flow time } F_j = C_j - r_j$$

$$\text{Lateness } L_j = C_j - d_j$$

$$\text{Tardiness } T_j = \max\{C_j - d_j, 0\}$$

$$\text{Earliness } E_j = \max\{d_j - C_j, 0\}, \quad \text{and on the basis of above parameter we can have}$$

following objective functions :

$$\text{Schedule length (makespan) } C_{\max} = \max\{C_j\}$$

$$\text{Weighted completion time } \sum w_j C_j$$

$$\text{Total completion time } \sum C_j$$



$$\text{Mean flow time } F_{\text{mean}} = (1/n) \sum F_j$$

$$\text{Flow time variance } F_{\text{var}} = (1/n) \sum (F_j - F_{\text{mean}})^2,$$

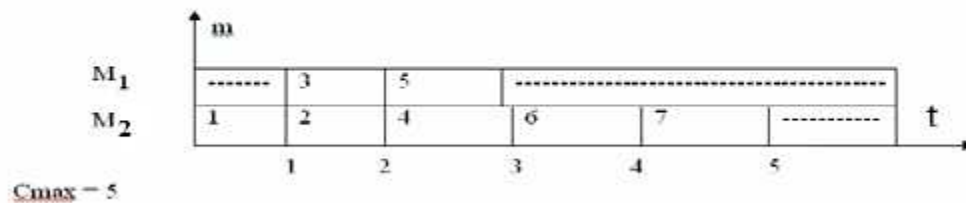
In scheduling problems, one of these objective function has to be minimized.

### 3.1.3 Examples

To illustrate the three field notation  $r | S | x$  we present some examples. In each case we will give the description of the problem. Furthermore, we will specify an instance and present a feasible schedule for the instance in form of Gantt chars.

#### Example 3.1

$P | \text{prec}; p_i = 1 | C_{\text{max}}$  is the problem of scheduling jobs with unit processing times and arbitrary precedence constraints on  $m$  identical machines such that makespan is minimized. An instance is given by a directed graph with  $n$  vertices and the number of machines. Figure 2 show an instance of the problem and a corresponding feasible schedule.



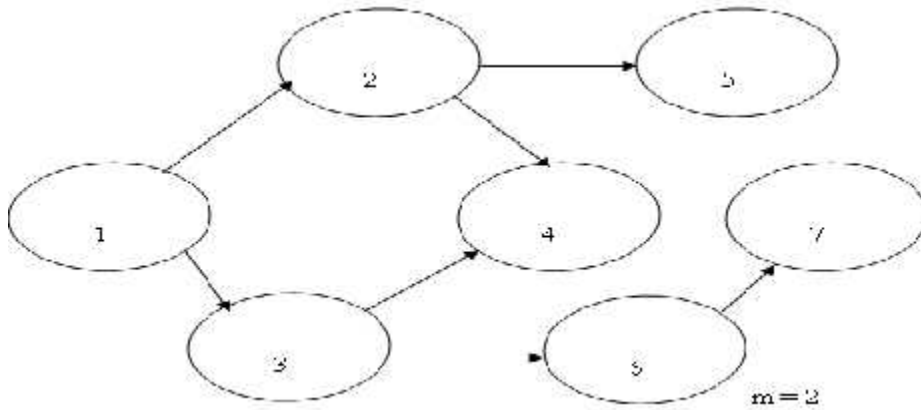


Figure 2  $P | \text{prec}; p_i = 1 | C_{\max}$

**Example 3.2**

$1 | \text{batch} | \sum w_i C_i$  is the problem of splitting a set of jobs into batches and scheduling these batches on one machine such that weighted flow time is minimized these batches on one machine such that weighted flow time is minimized.

Figure 3 shows an instance of this problem and a corresponding schedule with three batches.

i	1	2	3	4	5	6	
$p_i$	3	2	2	3	1	1	
$w_i$	1	2	1	1	4	4	s = 1

The objective value for the schedule is

$$\begin{aligned} \sum w_i C_i &= 2 \times 3 + (1 + 1 + 4) \times 10 + (1 + 4) 15. \\ &= 141 \end{aligned}$$

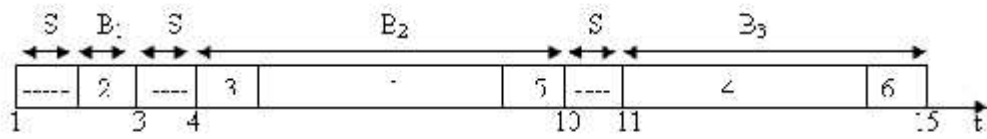


Figure 3  $1 | \text{batch} | \sum w_i C_i$

### 3.2 Combinatorial optimization

Scheduling problem can be solved by converting them into some feasible programming like linear programs, maximum flow problems or transportation problems. Such problems are stated as combinatorial optimization problems. Beyond these there are some other standard techniques like dynamic programming and branch and bound methods to solve the scheduling problems.

#### 3.2.1 Linear and integer programming

A linear program is an optimization problem of the form

$$\text{minimizes } z(x) = c_1 x_1 + \dots + c_n x_n \tag{2.1}$$

subjected to:

$$\left. \begin{array}{l} a_{11} x_1 + \dots + a_{1n} x_n \geq b_1 \\ \cdot \\ \cdot \\ a_{m1} x_1 + \dots + a_{mn} x_n \geq b_m \end{array} \right\} \tag{2.2}$$

$$x_i \geq 0 \text{ for } i=1, \dots, n.$$

A vector  $x = (x_1, \dots, x_n)$  satisfying (2.2) is called feasible solution. The problem is to find a feasible solution, which minimizes (2.1). A linear program that has a feasible solution is called feasible.

An integer linear program is a linear program in which all variables  $x_i$  are restricted to be integer. If the variables  $x_i$  can take only the values 0 or 1 then the corresponding integer linear program is called a binary linear program. If in a linear program only some variables are restricted to be integer then we have a mixed integer linear program.

### 3.2.2 The assignment problem

Consider the complete bipartite graph,  $G = (V_1 \cup V_2, V_1 \times V_2)$ . Assume w.l.o.g that  $n = |V_1| \leq |V_2| = m$ . Associated with each arc  $(i, j)$  there is a real number  $c_{ij}$ . An assignment is given by a one-to-one mapping  $\{ : V_1 \rightarrow V_2$ . The assignment problem is to find an assignment  $\{$  such that

$$\left\{ \sum_{i \in V_1} c_{i\{i)} \text{ is minimized.} \right.$$

Assume that  $V_1 = \{i, \dots, n\}$  and  $V_2 = \{i, \dots, m\}$ . Then the assignment problem has the following linear programming formulation with 0-1- variable  $x_{ij}$ :

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^n \sum_{j=1}^m c_{ij}x_{ij} \quad \text{s.t.} \\ & \sum_{j=1}^m x_{ij} = 1 \quad i= 1, \dots, n \end{aligned}$$

$$\sum_{i=1}^n x_{ij} \leq 1 \quad j=1, \dots, m$$

$$x_{ij} \in \{0, 1\} \quad i = 1, \dots, n; j = 1, \dots, m$$

### 3.3 Single machine scheduling problems

There are many algorithms published for single machine scheduling problems, which are polynomial or pseudo-polynomial (e.g.  $1 \mid r_j \mid L_{\max}$  is polynomially solvable under certain constraints). It is useful to note the following general result which holds for single machine problems: if all  $r_j = 0$  and if the objective function is a monotone function of the finish times of the jobs, then only schedules without preemption and without idle times need to be considered. This follows from the fact that the optimal objective value does not improve if preemption is allowed.

### 3.4 Maximum lateness and related criteria

Let us consider the problem  $1 \mid r_j \mid L_{\max}$ . This is NP-hard problem. However, under the following cases the problem can be polynomially solvable [4].

- a) take  $r_j = r \quad \forall j = 1, \dots, n$

We have an optimal schedule by applying earliest due date rule i.e. schedule jobs in order of non decreasing due dates.

b) take  $d_j = d \ \forall j=1, \dots, n$ .

We have an optimal schedule by scheduling jobs in order of non-decreasing release dates.

c)  $p_j = 1 \ \forall j=1, \dots, n$

At any time schedule an available job with smallest due date.

All of these rules take  $O(n \log n)$  steps. Cases *a*, *b* and *c* may be extended to the corresponding problems with precedence relations between jobs. Case *b* required modification of the release dates while in case *a* and *c* similar modification is to be done in due date using the following rule.

If  $i \rightarrow j$  and  $d'_i = d_j - p_j < d_i$ , then replace  $d_i$  by the modified due date  $d'_i$ .

#### **Algorithm Modify $d_j$**

```
1.   for i = 1 to n do  $n_i = 0$ 
2.   for i = 1 to n do
3.       for all  $j \in IP(i)$  do  $n_j = n_{j+1}$ 
4.    $F = W$ 
5.   for i = 1 to n do
6.       if  $n_i = 0$  then  $F = F \cup \{i\}$ 
7.   while  $F \neq W$  do
8.   begin
9.       choose  $j \in F$ 
10.      for all  $i \in IP(j)$  do
11.          begin
12.               $d_i = \min \{d_i, d_j - p_j\}$ 
```

```

12              $n_i = n_i - 1$ 
13             if  $n_i = 0$  then  $F = F \cup \{i\}$ 
                end
14.          $F = F \setminus \{j\}$ 
            end

```

### 3.5 Some simple scheduling algorithm

W.A. Horn [13] presented some simple scheduling algorithm where jobs require only one operation on a single machine. Some of the algorithms are stated as below.

1. Minimizing maximum lateness
2. Minimizing total delay.

#### 3.5.1 Minimizing maximum lateness

Assuming all jobs are available at time 0 and for each job  $i$  due date  $d_i$  have been given, we can get minimize the maximum lateness scheduling the jobs in the order of increasing  $d_i$ . For such case W.A. Horn presents the following algorithm.

**Algorithm 3.1** Let  $E$  be the subset of all jobs with earliest time of availability, which we designate  $A_1$ . Let  $A_2$  be the next earliest time of availability. Let job  $i$  be a job in  $E$  which has earliest due date  $d_i$  of all jobs in  $E$ . Let  $L_1 = \min\{d_i, A_2 - A_1\}$ . Assign job  $i$  to the interval  $[A_1, A_1 + L_1]$ . If job  $i$  is not completed, reduce  $d_i$  by  $L_1$ , otherwise drop job  $i$  from

the list. Repeat this operation for all remaining jobs, stipulating a minimum availability time of  $A_1 + L_1$ , for all remaining jobs. Continue until all jobs are completely assigned.

### 3.5.2 Minimizing total delay

When we have no due dates are given and our objective is to minimize the total delay i.e.  $\sum_i (f_i - a_i)$ , where  $f_i$  is the finished time and  $a_i$  is the available time for each job  $i$ , W.A. Horn states the following algorithms.

**Case 1** If all jobs are available at the same time then schedule jobs in the order of increasing  $d_i$ , where  $d_i = f_i - a_i$ .

**Case 2.** If jobs are available at different time then apply the following algorithm.

### 3.6 Realistic scheduling problem

All the scheduling problems so far we have discussed are the theoretical model of scheduling problems. The model of scheduling problem that have practical signification and could be carried and tested in real life falls under realistic scheduling problem. We have different models of realistic scheduling problems like online scheduling problems, real-time scheduling problems and just-in-time scheduling problems. The nature of realistic scheduling problems varies on how they address the real time difficulties like unavailability of job information until it arrives, resource constraints and so on [33,37].



### 3.6.1 Online scheduling

In online version of scheduling, the scheduler receives jobs that arrive over time, and generally must schedule the jobs without any knowledge of the future.  $1 | r_j | \sum C_j$  is an online scheduling problem. In such a problem goal of scheduling problem remains to reduce the average response time and efficient utilization of resources [15].

### 3.6.2 Real-time scheduling

Real-time scheduling problems are the typical computer related online version of scheduling problem [33, 37]. Generally, real-time system is an operating system embedded in some electronic devices and the correct functioning of the system depends on the time when jobs are completed. Some of the real-time systems are:

#### 3.6.2.1 Just-in-time debugging

The Windows operating system has the built-in capability to perform "just-in-time" debugging. Just-in-time, or JIT, debugging is where an application crashes while not running under a debugger, and the operating system arranges to start up an available debugger and attach it to the crashed process in order to obtain a back trace. The system registry contains an entry for the debugger that should be invoked when this happens. The Functional Developer Professional and Enterprise Editions are capable of acting as a JIT debugger; during the installation process you have the opportunity to install Functional Developer as your machine's default debugger. For more detail see [11].

### 3.6.2.2 JIT compiler

In computing, just-in-time compilation (JIT), also known as dynamic translation, is a technique for improving the runtime performance of a computer program. JIT builds upon two earlier ideas in run-time environments: bytecode compilation and dynamic compilation. It converts code at runtime prior to executing it natively, for example byte code into native machine code. The performance improvement over interpreters originates from caching the results of translating blocks of code, and not simply reevaluating each line or operand each time it is met interpreted language. It also has advantages over statically compiling the code at development time, as it can recompile the code if this is found to be advantageous, and may be able to enforce security guarantees. Thus JIT can combine some of the advantages of interpretation and static compilation.

Several modern runtime environments, such as Microsoft's .NET Framework and most implementations of Java and most recently Action script 3, rely on JIT compilation for high-speed code execution. For more detail see [36].

## 3.7 Scheduling in operating system

Scheduling issues have a great impact in computer science. Almost all operation system (OS) needs their job scheduled in a efficient way. In an OS, a machine is a processor and jobs are processes i.e. job is a program ready for execution [28, 33, 37]. Machine environment varies on no of processor used, either preemption of program allowed or not and on the resource constraints. Objective function may be one of the following:

**Process utilization:** Average fraction of time during which processor is busy

**Throughput:** Number of processes executed per unit time.

**Average waiting time:** Time that a process spends waiting resources.

**Average response time:** Time taken by a process to response after it is released.

## **4 Just-in-time production system**

### **4.1 Some key elements of JIT**

This section contains a brief description of key elements used in mixed-model just-in-time system.

#### **1. Reduce or eliminate setup times:**

Aim for single digit setup times (less than 10 minutes) or "one-touch" setup -- this can be done through better planning, process redesign, and product redesign. A good example of the potential for improved setup times can be found in auto racing, where a NASCAR pit crew can change all four tires and put gas in the tank in under 20 seconds [5].

#### **2. Reduce lot sizes:**

Reducing setup times allows economical production of smaller lots; close cooperation with suppliers is necessary to achieve reductions in order lot sizes for purchased items, since this will require more frequent deliveries.

#### **3. Reduce lead times:**

Production lead times can be reduced by moving work stations closer together, applying group technology and cellular manufacturing concepts, reducing queue length and improving the coordination and cooperation between successive processes; delivery lead times can be reduced through close

cooperation with suppliers, possibly by inducing suppliers to locate closer to the factory.

4. **Flexible work force:**

Workers should be trained to operate several machines, to perform maintenance tasks, and to perform quality inspections. In general, JIT requires teams of competent, empowered employees who have more responsibility for their own work. The Toyota Production System concept of “respect for people” contributes to a good relationship between workers and management.

5. **Small-lot conveyance:**

Use a control system such as a *kanban* (card) system (or other signaling system) to convey parts between workstations in small quantities (ideally, one unit at a time). In its largest sense, JIT is not the same thing as a kanban system, and a kanban system is not required to implement JIT (some companies have instituted a JIT program along with a MRP system), although JIT is required to implement a kanban system and the two concepts are frequently equated with one another.

## **4.2 JIT system examples**

### **4.2.1 JIT-style learning and training**

The best kind of quality-oriented learning (and training) is just-in-time-style learning, i.e., learning that happens on the job and knowledge is applied immediately as

needed. The sooner you can apply the material you learned, the better you will understand it and the longer it will be retained. Instead of training masses of employees for long periods, in JIT-style training, education is implemented as an ongoing series of short sessions (just a few hours a week) during which employees are taught only what they can apply soon, without suffering information overload.

Innovative e-learning services create new opportunities for such on the job JIT-style learning and training. In particular, this first-ever Ten3 online Business e-Coach provides very effective JIT-style e-learning opportunity which is available free anytime to anybody [32].

#### **4.2.2 Kanban – a communication tool in JIT production system**

Being a very important tool for just-in-time production, kanban has become synonymous with the JIT production system.

Kanban, meaning label or signboard, is used as a communication tool in JIT system. A kanban is attached to each box of parts as they go to the assembly line. A worker from the following process goes to collect parts from the previous process leaving a kanban signifying the delivery of a given quantity of specific parts. Having all the parts funneled to the line and used as required, the same kanban is returned back to serve as both a record of work done and an order for new parts. Thus kanban coordinates the inflow of parts and components to the assembly line, minimizing the processes [5].

### **4.2.3 Toyota production system**

Toyota Motor Corporation's vehicle production system is a way of "making things" that is sometimes referred to as a "lean manufacturing system" or a "just-in-time (JIT) system," and has come to be well known and studied worldwide. This production control system has been established based on many years of continuous improvements, with the objective of "making the vehicles ordered by customers in the quickest and most efficient way, in order to deliver the vehicles as quickly as possible" [38].

The Toyota Production System (TPS) was established based on two concepts: The first is called "jidoka"(which can be loosely translated as "automation with a human touch") which means that when a problem occurs, the equipment stops immediately, preventing defective products from being produced; The second is the concept of "just-in-time," in which each process produces only what is needed by the next process in a continuous flow.

Based on the basic philosophies of jidoka and just-in-time, the TPS can efficiently and quickly produce vehicles of sound quality, one at a time, that fully satisfy customer requirements.

### **4.3 Mixed-model production**

Mixed-model production is the practice of assembling several distinct models of a product on the same assembly line without changeovers and then sequencing those models in a way that smoothes the demands for upstream components [5].

The objective is to smooth demand on upstream work centers, manufacturing cells or suppliers and thereby reduce inventory, eliminate changeovers, improve kanban

operation. It also eliminates difficult assembly line changeovers.

Toyota developed the concept in the 1960's in response to the problems created by line changeovers. It was originally applied to long assembly lines such as those used in automotive.

#### 4.4 **Push versus pull production system**

**Push system:** Total demand is forecast, and the producer allocates ("pushes") items to users based on the expected needs of all users. Finished goods accumulate in inventory. - Produce for Forecast [28].

**Pull system:** Each user requests ("pulls") items from the producer only as they are required. Units are only produced if there is demand for them. - Produce For Demand

- production is pulled through the supply chain in response to actual demand
- first seen in just-in-time systems in Japan



## 5 Mixed-model just-in-time production system

Just-in-time systems are formulated under the assumption that the system have negligible switching over cost from one product to another and that each products are produced in a unit time [9, 22]. The system has a constant rate of usage of all parts and the sequencing problem aims to minimize the variation so that earliness and tardiness penalties are minimized.

### 5.1 The PRV problem formulation

Under the assumption that the product requires approximately the same number and mix of parts, Miltenburg [24, 17] reduces the sequencing problem into the product rate variation (PRV) problem, a single-level case [24, 17].

Suppose  $D$  units of  $n$  products are to be produced with respective demands  $d_i$ ,  $i = 1, \dots, n$  with  $D = \sum_{i=1}^n d_i$  during a specified time horizon. Then the objective is to maintain the cumulative production  $x_{ik}$ , a non-negative integer,  $i = 1, \dots, n$ ;  $k=1, \dots, D$  of product  $i$  during the time periods 1 through  $k$  as close to the ideal production  $kr_i$ , a non-negative rational number,  $i=1, \dots, n$ ;  $k = 1, \dots, D$  with  $r_i = \frac{d_i}{D}$  and  $\sum_{i=1}^n r_i = 1$  as possible. The total production time horizon is partitioned into  $D$  equal time units of which 1 time unit is required for a unit of a product to be produced. In [24] Miltenburg formulates a non-linear integer programming with nonnegative, convex and symmetric function having minimum 0 at 0 as the sum of the square and the absolute deviations between the actual and the ideal production. Kubiak and Sethi [19] generalize this

problem as the unimodal convex function  $f_i(x)$  satisfying  $f_i(0) = 0$ ,  $f_i(y) > 0$  for  $y \neq 0$ ,  $i = 1, \dots, n$ . The mathematical model of the PRV problem P1 [19, 24, 35] is as follows:

$$\text{minimize } \left[ F = \max_{i,k} f_i(x_{ik} - kr_i) \right] \quad (1)$$

and

$$\text{minimize } \left[ G = \sum_{k=1}^D \sum_{i=1}^n f_i(x_{ik} - kr_i) \right] \quad (2)$$

subject to

$$\sum_{i=1}^n x_{ik} = k, \quad k = 1, \dots, D \quad (3)$$

$$x_{i(k-1)} \leq x_{ik}, \quad i = 1, \dots, n; k = 2, \dots, D \quad (4)$$

$$x_{iD} = d_i \quad x_{i0} = 0, \quad i = 1, \dots, n \quad (5)$$

$$x_{ik} \geq 0, \text{ integer} \quad (6)$$

The constraint (3) shows that exactly  $k$  units of products are produced in the periods 1 through  $k$ . (4) states that the total production is a non-decreasing function of  $k$ . (5) guarantees the demands are met exactly. (3), (4), and (6) ensure that exactly one unit of a product is sequenced during a time unit. In Particular we can extend the above formulation as:

$$F_a = \max_{i,k} |x_{ik} - kr_i|$$

$$F_s = \max_{i,k} (x_{ik} - kr_i)^2$$

$$G_a = \sum_{k=1}^D \sum_{i=1}^n |x_{ik} - kr_i|$$

$$G_s = \sum_{k=1}^D \sum_{i=1}^n (x_{ik} - kr_i)^2$$

in P1 as the particular objectives. Here we denote, for example, problem  $F_a$  for the problem P1 with the objective function  $F_a$  and the constraints (3) – (6). The sequence (let say  $s = s_1s_2\dots\dots s_D$ ) generated by P1 always keeps the actual production level  $x_{ik}$  as close to the ideal production level  $kr_i$  as possible all the times.

An alternative objective, i.e. the minimization of the deviations between the times at which a unit of a product be actually produced and the time at which the unit of the product is needed to be produced, has been introduced by Inman and Bulfin [14]. This objective appropriates with the objective established by Miltenburg [24].

## 5.2 The ORV problem formulation

In the more practical approach, we have a production system that consists of a hierarchy of several distinct production levels such as products, sub-assemblies, component parts, raw materials, etc. In such system, the part demand rate at upper level defines the part demand rate down the level. Such system is known as Output rate variation (ORV) problem, a mixed-model multi-level JIT sequencing problem [17, 25, 26]. Hence, the mathematical model of the ORV problem must incorporate the method so that parts fit together to form products.

Let the systems consist of  $L$  different production levels  $l$ ,  $l = 1, \dots, L$  with product level 1.  $d_{il}$  be the demand for part type  $i$  of level  $l$ ,  $i = 1, \dots, n_l$ ,  $n_l$  the number of different part types of level  $l$ .  $t_{ilp}$  represents the number of total units of part type  $i$  at level  $l$  required to produce one unit of product  $p$ ,  $p = 1, \dots, n_1$  and  $d_{il} = \sum_{p=1}^{n_1} t_{ilp}d_{p1}$ , the

dependent demand for part  $i$  of level  $l$  determined by  $d_{p1}$ ,  $p = 1, \dots, n_1$ . Note that  $t_{ilp} = 1$  for  $i = p$ , and 0 otherwise.  $D_1 = \sum_{i=1}^{n_1} d_{i1}$  stands for total part demands of level 1 with demand rate  $r_{i1} = \frac{d_{i1}}{D_1}$  and  $\sum_{i=1}^{n_1} r_{i1} = 1$  for  $l = 1, \dots, L$ .

This is a non-preemptive model. The total time horizon in the product level is partitioned into  $D_1$  equal time units such that there will be  $k$  complete units of various products  $p$  at level 1 during the first  $k$  time units. As the demands of part type requirement at the lower level are pulled forward according to the need of the product level, the system is also referred as pull system.

Let  $x_{ilk}$  denotes the quantity of part  $i$  produced at level  $l$  in the time units 1 through  $k$  and  $y_{lk} = \sum_{i=1}^{n_l} x_{ilk}$  be the total quantity produced at level  $l$  during the time units 1 through  $k$ . Clearly, at level 1,  $y_{1k} = \sum_{i=1}^{n_1} x_{i1k} = k$ . The required cumulative production for part  $i$  of level  $l$ ,  $l = 2$  through  $k$  time units will be  $x_{ilk} = \sum_{p=1}^{n_l} t_{ilp} x_{p1k}$ . Consider  $f_i$  unimodal convex function with minimum 0 at 0,  $i = 1, \dots, n_1$ . The mathematical model for the ORV problem P2 [19, 26] is as follows:

$$\text{minimize } [ F = \max_{i,l,k} f_i(x_{ilk} - y_{lk}r_{il}) ] \quad (7)$$

and

$$\text{minimize } [ G = \sum_{k=1}^{D_1} \sum_{l=1}^L \sum_{i=1}^{n_l} f_i(x_{ilk} - y_{lk}r_{il}) ] \quad (8)$$

subject to

$$x_{ilk} = \sum_{p=1}^{n_l} t_{ilp} x_{p1k}, \quad i = 1, \dots, n_l; \quad l = 1, \dots, L; \quad k = 1, \dots, D_1 \quad (9)$$

$$y_{lk} = \sum_{i=1}^{n_l} x_{ilk}, \quad l = 2, \dots, L; \quad k = 1, \dots, D_1 \quad (10)$$

$$y_{1k} = \sum_{p=1}^{n_1} x_{p1k} = k, \quad k = 1, \dots, D_1 \quad (11)$$

$$x_{p1k} = x_{p1(k-1)}, \quad p = 1, \dots, n_1; k = 1, \dots, D_1 \quad (12)$$

$$x_{p1D} = d_1, \quad x_{p10} = 0, \quad p = 1, \dots, n_1 \quad (13)$$

$$x_{i1k} = 0, \text{ integer}, \quad i = 1, \dots, n_1; l = 1, \dots, L; k = 1, \dots, D_1 \quad (14)$$

Constraint (9) ensures that the necessary cumulative production of part  $i$  of level  $l$  by the end of time unit  $k$  is determined explicitly by the quantity of products produced at level  $l$ . Constraints (10) and (11) show the total cumulative production of level  $l$  and level 1, respectively, during the time units 1 through  $k$ . Constraint (12) ensures that the total production of every product over  $k$  time units is a non-decreasing function of  $k$ . Constraint (13) guarantees that the demands for each product are met exactly. Constraints (11), (12), (14) ensure that exactly one unit of a product is scheduled during one time unit in the product level. In particular, denote

$$F_a = \max_{i, l, K} |x_{i1k} - y_{1k}r_{il}|$$

$$F_s = \max_{i, l, K} (x_{i1k} - y_{1k}r_{il})^2$$

$$G_a = \sum_{k=1}^{D_1} \sum_{l=1}^L \sum_{i=1}^{n_{l1}} |x_{i1k} - y_{1k}r_{il}|$$

$$G_s = \sum_{k=1}^{D_1} \sum_{l=1}^L \sum_{i=1}^{n_{l1}} (x_{i1k} - y_{1k}r_{il})^2$$

in P2 as the particular objectives. We denote, for example, problem  $F_a$  for the problem P2 with objective function  $F_a$  and the constraints (9)-(14).

## 5.3 The total PRV problem

### 5.3.1 Assignment costs

Kubiak and Sethi [19] reduces the total PRV problem G with unimodal, convex, symmetric and non negative function  $f_i(0) = 0$ ,  $f_i(y) > 0$  for  $y \neq 0$  to an assignment problem that can efficiently be solved pseudo-polynomially with time complexity  $O(D^3)$ .

The ideal position for the production of  $(i, j)$ , the  $j^{\text{th}}$  copy of product  $i$  is,  $Z_{ij} = \left\lceil \frac{2j-1}{2r_i} \right\rceil$ , the point of intersection between  $f_i$  for  $(i, j)$  and  $f_i$  for  $(i, j-1)$ ,  $i = 1, \dots, n$ ;  $j = 1, \dots, d_i$  i.e. the unique crossing point satisfying  $f_i(j - k_{ij}r_i) = f_i(j - 1 - k_{ij}r_i)$ . If all copies of product  $i$  are scheduled at their ideal positions, the product  $i$  will contribute the cost  $\inf f_i(j - kr_i)$  to the total cost of the solution. The ideal position  $\left\lceil \frac{2j-1}{2r_i} \right\rceil$  minimizes both the problems F and G, however, leads to infeasibility whenever more than one copy competes for the same ideal position in the sequence. Higher priority is given to  $j$  over  $j'$  whenever  $j < j'$ .

Let  $X = \{(i, j, k) | i = 1, \dots, n; j = 1, \dots, d_i; k = 1, \dots, D\}$ . The cost  $C_{ijk} = 0$  for  $(i, j, k) \in X$  with respect to the ideal position  $Z_{ij}$  of assigning  $(i, j)$  to the time unit  $k$  is defined as follows.

$$C_{ijk} = \begin{cases} \sum_{i=k}^{Z_{ij}^*} \{^i_{jl}\}, & \text{if } k < Z_{ij}, \\ 0 & \text{if } k = Z_{ij}, \\ \sum_{i=Z_{ij}}^{k-1} \{^i_{jl}\} & \text{if } k > Z_{ij}, \end{cases}$$

where,

$$\{ x_{jl}^i = \begin{cases} f_i(j-lr_i) - f_i(j-1-lr_i) & \text{if } l < Z_{ij}, \\ f_i(j-1-lr_i) - f_i(j-lr_i) & \text{if } l \geq Z_{ij} \end{cases}$$

### 5.3.2 Assignment method

The total PRV problem in the form of assignment problem P4 [19] is

$$\min \left[ H = \sum_{k=1}^D \sum_{i=1}^n \sum_{j=1}^{d_i} C_{ijk} x_{ijk} \right] \quad (18)$$

subject to

$$\sum_{i=1}^n \sum_{j=1}^{d_i} x_{ijk} = 1, k = 1, \dots, D \quad (19)$$

$$\sum_{k=1}^D x_{ijk} = 1, i = 1, \dots, n; j = 1, \dots, d_i. \quad (20)$$

where,

$$x_{ijk} = \begin{cases} 1, & \text{if } (i, j) \text{ is assigned to time unit } k \\ 0, & \text{otherwise} \end{cases}$$

Kubiak and Sethi [20] show that if a sequence  $\{x_{ijk}\}$ ,  $i = 1, \dots, n$ ;  $j = 1, \dots, d_i$  and  $k = 1, \dots, D$  is optimal to problem H then the sequence  $\left\{ \sum_{l=1}^k \sum_{j=1}^{d_i} x_{jl}^i \right\}$ ,  $i = 1, \dots, n$ ;  $k = 1, \dots, D$  is optimal to problem G. Hence, any optimal solution to problem G can be constructed from any optimal solution of problem H in  $O(D)$  time. It is, by induction, shown that there is at least a sequence that preserves order i.e.  $(j+1)^{\text{st}}$  copy of product  $i$  will not appear before the  $j^{\text{th}}$  copy in the sequence. The assignment problem with 2D nodes can be solved in  $O(D^3)$  time [17].

A set  $X \subseteq X$  is  $X$  feasible if the following constraints hold:

$c_1$  : For each  $k, k = 1, \dots, D$ , there is exactly one  $(i, j), i = 1, \dots, n; j = 1, \dots, d_i$  such that  $(i, j, k) \in X$ , i.e., exactly one copy is produced at one time unit.

$c_2$  : For each  $(i, j), i = 1, \dots, n; j = 1, \dots, d_i$ , there is exactly one  $k, k = 1, \dots, D$  such that  $(i, j, k) \in X$ , i.e., each copy is produced exactly once.

$c_3$ : If  $(i, j, k), (i, j, k') \in X$  and  $k < k'$  then  $j < j'$ , i.e., lower indices copies are produced earlier.

$c_1$  and  $c_2$  are the assignment problem constraints whereas  $c_3$  is different that imposes an order on copies of a product. The sequence  $s = s_1s_2 \dots s_D$  with  $s_k = j, k = 1, \dots, D$ , if  $(i, j, k) \in X$ , for some  $j$ , is feasible for any instance  $(d_1, \dots, d_n)$ . Kubiak and Sethi [15] obtain the result.

## 5.4 The pegged ORV problem

Goldstain and Yeoman [34] show that the ORV problem under the pegging assumption can be reduced to the weighted PRV problem [34]. Pegged parts of output  $i$  at level  $l, l = 2, \dots, L$ , are dedicated to be assembled into the particular product at level  $l$  such that the parts are dedicated to be assembled into the different products are distinct. They give the first mathematical formulation for pegging in a JIT environment. They also formulated several heuristic solution techniques for the pegged, multi-level min-sum model (ORV problem). The pegged ORV problem  $F_e$  with absolute deviation function can be written as



$$\text{Min } \underset{h,i,j,k}{\text{Max}} \{ W_{h1} |x_{h1k} - kr_{h1}|, W_{ij} |x_{h1k} t_{ijh} - kt_{ijh} r_h| \},$$

$h = 1, \dots, n_1; i = 1, \dots, n_1; k = 1, \dots, D_1; l = 2, \dots, L$  with constraints (9) - (14).

Letting  $l = 1, \dots, L$  and transforming the weighting actors [25] the pegged ORV problem has been reduced in [34] to the following weighted PRV problem  $\min \max_{x_{i,k}} w_i^* |x_{ik} - kr_i|, i = 1, \dots, n; k = 1, \dots, D$ , where  $w_i^* = \max_{i,l} \{ w_{il}(t_{iln}) \}$  with constraints(3) - (6).

## 5.5 Cyclic sequence

Miltenburg [24] and Miltenburg and Sinnamon [26] study the existence of the cyclic sequence in problem  $G_s$ . They introduce the concept of cyclic sequencing and show that the time complexity of the existing algorithms (pseudo-polynomially solvable) can be substantially reduced. Still the big question whether a concatenation sum of  $m$  copies of an optimal sequence  $s$  for the Instance  $(d_1, \dots, d_n)$  is optimal for  $(m d_1, \dots, m d_n)$ ,  $m \geq 1$  to build a sequence for a longer time horizon remains open.

Kubiak in [18] reviews the PRV problem and shows that the cyclic JIT sequences are optimal. He takes a PRV problem with a given  $n$  products  $1, \dots, i, \dots, n$  and  $n$  positive integer demands  $d_1, \dots, d_i, \dots, d_n$ . He supposes a sequence  $\Gamma = \Gamma_1, \dots, \Gamma_T, T = \sum_{i=1}^n d_i$ , of the products, where product  $i$  occurs exactly  $d_i$  times that always keeps the actual product level, equal the number of product  $i$  occurrences in the prefix  $\Gamma_1, \dots, \Gamma_t, t = 1, \dots, T$ , and the desired product level, equal  $r_i t$ , where  $r_i = d_i/T$ , of each product  $i$  as close to each other as possible. And proves if  $S$  is an optimal sequence for  $d_1, \dots, d_i, \dots, d_n$ , then concatenation  $S^m$  of  $m$  copies of  $S$  is an optimal sequence for  $m d_1, \dots, m d_i, \dots, m d_n$ .

## 6 Mixed-model JIT problems

### 6.1 Nearest integer point problem

Aiming to minimize the total deviation or sum of all deviation of the real production from the ideal but rational production, Miltenburg in [26] has purposed some algorithms and heuristics.

#### Problem statement

Define the point  $X_k = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$  where  $x_{i,k} = kr_i$ ,  $\sum_{i=1}^n x_{i,k} = k$ , and  $\mathbb{R}$  is the set of real number. Problem is to find the “nearest” integer point  $M_k = (m_{1,k}, m_{2,k}, \dots, m_{n,k}) \in \mathbb{Z}^n$  to the point  $M_k$  where  $\sum_{i=1}^n m_{i,k} = k$ ,  $\mathbb{Z}$  is the set of nonnegative integers and “nearest” means minimize  $\sum_{i=1}^n (m_{i,k} - x_{i,k})^2$

**Algorithm 1** The following algorithm finds the nearest integer point  $M = (m_{1,k}, m_{2,k}, \dots, m_{n,k}) \in \mathbb{Z}^n$  to a point  $X = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$ . where  $\sum_{i=1}^n m_i = \sum_{i=1}^n x_i = k$ .

1. Calculate  $k = \sum_{i=1}^n x_i$
2. Find the nearest nonnegative integer  $m_i$  to each coordinate  $x_i$ . That is, find  $m_i$  so that  $|m_i - x_i| \leq 0.5$ ,  $i = 1, 2, \dots, n$ .

3. Calculate  $k_m = \sum_{i=1}^n m_i$ 
  - a. if  $k - k_m = 0$  stop. The nearest integer point is  $M = (m_1, m_2, \dots, m_n)$
  - b. if  $k - k_m > 0$  go to step 5.
  - c. if  $k - k_m < 0$  go to step 6.
  
4. Find the coordinate  $x_i$ , with the smallest  $m_i - x_i$ . Increment the value of this  $m_i$ ;  $m_i \rightarrow m_{i+1}$ . Go to step 3.
  
5. Find the coordinate  $x_i$ , with the largest  $m_i - x_i$ . Decrease the value of this  $m_i$ ;  $m_i \rightarrow m_{i-1}$

### Problem with algorithm 1

For  $X = (30/13, 30/13, 5/13)$  the integer point is  $(2, 2, 1)$ . Then for  $X = (36/13, 36/13, 6/13)$  the integer point is  $(3, 3, 0)$ . Production schedule is 1, 2, -3 which is impossible as production cannot be destroyed. Hence the schedule is not feasible.

### Conclusion

Algorithm-1 may lead to infeasible solution.

### Algorithm 2

1. Solve the problem P1 (using Algorithm 1), and determine whether the schedule is feasible. (It is feasible if  $m_{i,k} - m_{i,k-1} \geq 0$  for all  $i, k$ .) If the schedule is feasible, stop. Otherwise, to go step 2.
2. For the infeasible schedule determined in step 1, find the first (or next) stage  $l$  where  $m_{i,l} - m_{i,l-1} < 0$ . Set  $i$  = number of product  $i$ , for which  $m_{i,j} - m_{i,l-1} < 0$ . Reschedule stages  $l - 1, l - 1 + 1, \dots, l+1$  by considering all

possible sequences that begin with the schedule for stage  $l - 1$  and end with the schedule for stage  $l + 1$ .

- 3 Repeat step 2 for other stages where  $m_{i,k} - m_{i,k-1} < 0$ . Then stop.

### Problem with algorithm 2

In general there are  $n! / (n - 2)!$  possible sequences, each of length  $+ 2$ , to consider for each infeasibility. While total enumeration works for small problems of this type (products where similar part requirements) it does not work well for larger problems, nor for problems where products have differing part requirements.

### Algorithm 3

1. Solve problem P1 (using Algorithm-1), and determine whether the schedule is feasible. (It is feasible if  $m_{i,k} - m_{i,k-1} \geq 0$  for all  $i, k$ .) If the schedule is feasible, stop.
2. For the infeasible schedule determine in step 1, find the first (or next) stage  $l$  where  $m_{i,l} - m_{i,l-1} < 0$ . set  $n_i$  = number of products  $i$ , for which  $m_{i,l} - m_{i,l-1} < 0$ , and beginning at stage  $l - 1$  use Heuristic 1 or Heuristic 2 to schedule stages  $l - 1, l - 1 + 1, \dots, l + W$ , where  $W \geq 0$ .  $l + W$  is the first stage where the schedule determined by heuristic matches the schedule determined in step 1.
3. Repeat step 2 for other schedule determined in step 1.

### Heuristic 1.

For a stage  $k$ , schedule the product  $i$  with the lowest  $X_{i,k} - kr_i$ .

### Heuristic 2.

For each stage  $k$ :

1. set  $h=1$

2. Tentatively schedule product  $h$  to be produced in stage  $k$ . Calculate the variation for stage  $k$  and call it  $V_{1h}$
3. Schedule the product  $I$  with the lowest  $x_{i,k} - (k+1)r_i$ ,
4. Increment  $h$ ;  $h \rightarrow h + 1$ . If  $h > n$  go to step 5, otherwise go to step 2
5. Schedule the product  $h$  with the lowest  $V_h$ .

## 6.2 Dynamic programming algorithm

### 6.2.1 Problem statement

In [27] Miltenburg, Steiner and Yeomans consider both the usage goal and loading goal as a prime factor in min sum PRV problem and aims to

$$\text{minimize } \sum_{k=1}^D (r_u U_k + r_L L_k) \quad (D1)$$

s.t.

$$\sum_{l=1}^n x_{ik} = k, \quad k = 1, 2, \dots, D$$

$$x_{iD} = d_i, \quad i = 1, 2, \dots, n$$

$$x_{i0} = 0 \quad i = 1, 2, \dots, n$$

$$x_{i(k-1)} \leq x_{ik}, \quad \forall i, k.$$

where,

$r_u$  is relative weight for usage goal

$r_L$  is relative weight for loading goal.

$$U_k = \sum_{l=1}^n (x_{ik} - kr_i)^2$$

$$L_k = \sum_{l=1}^n t_i^2 (x_{ik} - kr_i)^2$$

Redefining  $T_i^2 = r_u + r_L t_i^2$ , the objective function in D1 can be re-written as

$$\text{minimize } \sum_{k=1}^D \sum_{l=1}^n T_i^2 (x_{ik} - kr_i)^2$$

## 6.2.2 Dynamic programming (DP) procedure

The objective function in problem (D1) requires the minimization of a quadratic integer function. The large number of integer variables, along with the first and last groups of constraints, makes it impossible to solve by general integer programming techniques. Here is a special purposed DP procedure instead, which enables us to find the optimal JIT schedule for practical-sized problems.

Let  $d = (d_1, d_2, \dots, d_n)$  be the product requirements vector. Define subsets in a schedule as  $X = (x_1, x_2, \dots, x_n)$ , where  $x_i$  is a non-negative integer representing the production of exactly  $x_i$  units of product  $i$ ,  $x_i \leq d_i$ , for all  $i$ . Let  $e_i$  be the usual  $i^{\text{th}}$  unit vector; with  $n$  entries, all of which are zero except a single 1 in the  $i^{\text{th}}$  place. A subset  $X$  can be schedule in the first  $k$  stages if  $K = |X| = \sum_{i=1}^n x_i$ . Let  $f(X)$  be the minimal total variation of any schedule where the products in  $X$  are produced during the first  $k$  stages.

$$\text{Let } g(X) = \sum_{i=1}^n T_j^2 (x_{jk} - kr_j)^2$$

The following DP recursion holds for  $f(X)$ :

$$f(X) = f(x_1, x_2, \dots, x_n)$$

$$\begin{aligned}
&= \min\{f(\mathbf{X} - \mathbf{e}_i) + g(\mathbf{X}) \mid i = 1, 2, \dots, n; x_i - 1 \geq 0\}, \\
f(\mathbf{X}) &= f(\mathbf{X} \mid \forall x_n = 0) \\
&= f(0, 0, \dots, 0) \\
&= 0
\end{aligned}$$

it is clear that  $f(\mathbf{X}) \geq 0$ , and it follows easily from the definition of the  $r_i$ 's that  $g(\mathbf{X} \mid \forall x_i = d_i) = 0$ .

### Theorem

The DP recursion solves the JIT scheduling problem in

$$O\left(n \prod_{i=1}^n (d_i + 1)\right) \quad \text{time}$$

and

$$O\left(\prod_{i=1}^n (d_i + 1)\right) \quad \text{space.}$$

## 6.3 Min-max-absolute-chain algorithm

In [8] Dhamala, extended the formulation of single-level JIT sequencing problem under a number of chain constraints. He proposed the following min-max-absolute-chain-algorithm.

### 6.3.1 Min-max-absolute-chain-algorithm

Given:  $d_i^t$  for  $i = 1, 2, \dots, n_t$  and  $t=1, 2, \dots, m$ ;

an upper bound  $B$  for min-max-absolute-chain-problem;

$chain_1, chain_2, \dots, chain_t, \dots, chain_m$ ;

Update: number of demands  $n = n_t$ ;

demand rates  $d_i$  for  $i = 1, 2, \dots, n$ ;

total demand  $D = \sum d_i$ .

Step 1: Calculate windows  $[E(i, j), L(i, j)]$  for  $j = 1, 2, \dots, d_i$  and

$i = 1, 2, \dots, n$  by STEINER / YEOMANS [35]

Step 2: Modify the due date  $L(i, j)$ .

if  $(i, j) \rightarrow (i', j')$  then  $L(i, j) := \min \{L(i, j), L(i', j')\}$ .

Step 3: Schedule the jobs by EDD -Algorithm of HORN [13].

Output :  $B$  feasible for  $(n, D)$  if  $L_{\max} \leq 0$ .

### 6.3.2 Min-max-absolute-chain-algorithm for overlapping sequence

- Let assign the chain Id with each job



- Hence let say if job  $a$  falls in chain<sub>1</sub> and chain<sub>2</sub>, the job  $a$  will be converted into job  $a_0$  and job  $a_1$  where 0 is the id of chain<sub>1</sub> and 1 is the id of chain<sub>2</sub>.
- Now the input become pseudo-Non-Overlapping sequence and hence can be solved by the algorithm purposed at 4.3.3.

## **7 Implementation**

After the study of different types of problems and algorithms, the application software to illustrate these problems and algorithm is implemented in Java. For each problem a package is defined. The Java Swing is used to develop GUI.

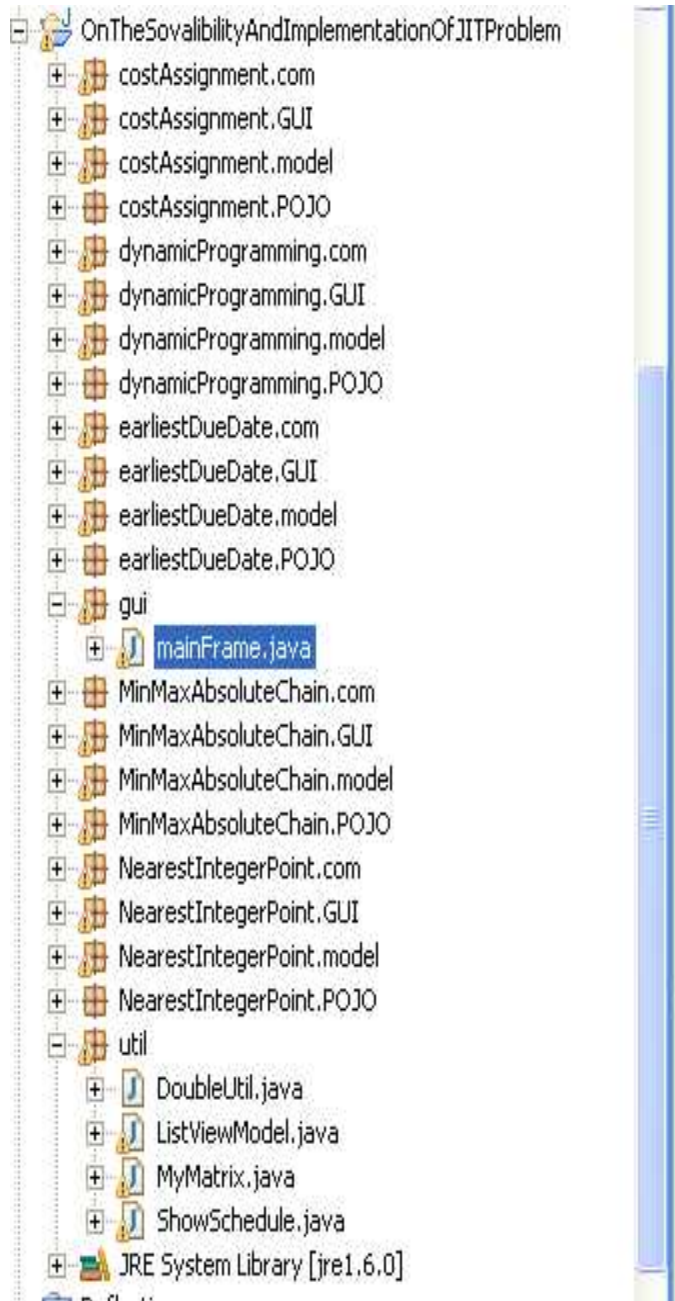
### **Machine Specification**

Pentium 4 CPU 3.06 GHz.

496 MB RAM.

### **7.1 Architecture view of project**

The architecture view of whole implementation is shown by the following tree view.



**Figure 4** Architectural view of whole project

The implementation project consists of implementation of five problems:

- 1) Nearest integer point problem

- 2) Earliest due date problem
- 3) Cost assignment problem
- 4) Dynamic programming problem
- 5) Min-max-absolute-chain problem with constraints

For each package we have the same architecture. Each consists of 4 sub-packages:

- a) GUI
- b) Com
- c) POJO
- d) Model

GUI package consists of the class files that define the graphic interface to input data. The com package consists of the class files that hold the main algorithm. The POJO consists of class files that define the structure of the entities required for the algorithm. Finally the model sub-package consists of the class files that contain definition of data and tabular view.

Beside these we have two sub-packages

- a) UTIL
- b) GUI

These two packages consist of class files that are used by all of the five projects.

### **7.1.1 UTIL package**

The UTIL package consists of four class files:

DoubleUtil.java

ListViewMode.java

MyMatrix.java

ShowSchedule.java

DoubleUtil.java consists the logic required to operate with double values like :

```
Double getRoundDouble(double value,  
                        int no_of_decimal_point_to_round);  
  
int upperFloor(double d);
```

Class ListViewModel consists the structure of the table that holds the list of data to view the results.

Class MyMatrix consists the definition for the matrix that is used to hold records

Finally, class ShowSchedule is used to view the result list defined by class listViewModel.

### **7.1.2 GUI package**

The GUI package consists of

- i) mainframe.java

Mainframe.java is the main controller for the implementation. It consists of menu for switching between the projects. The interface of mainframe looks like:



Figure 5 Interface of main GUI

## 7.2 Nearest integer point implementation issue

Nearest integer point falls under the min-sum PRV problem. The main objective of the program is to minimize the total variation. Architecture view of Nearest Integer Point Problem implementation is shown by the following tree view.

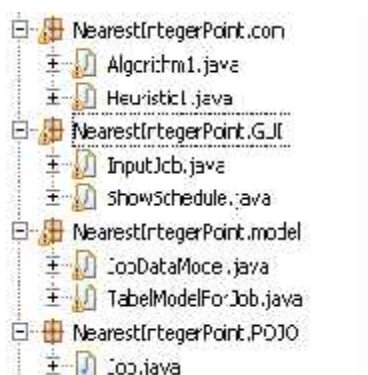


Figure 6 Architectural view of nearest-integer-point package

This Package implements the nearest Integer Point algorithm developed by Meltenburg and one of his heuristic.

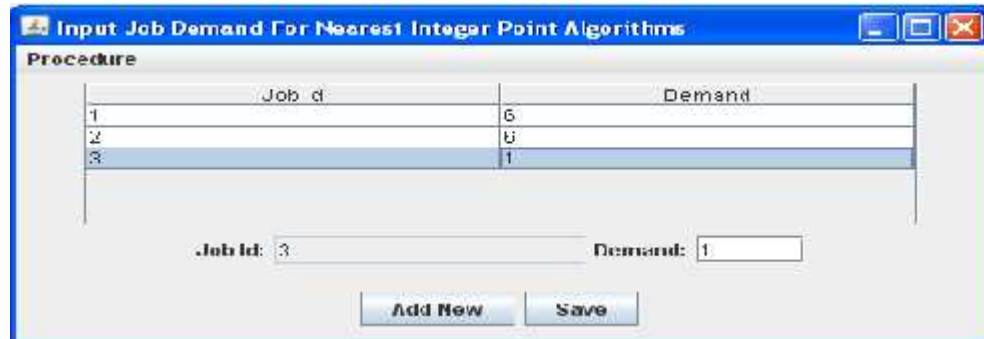


Figure 7 Date input frame for nearest-integer-point package.

The figure 7 show a sample date used for testing the algorithm. There are  $n = 3$  products with demands  $D = (6, 6, 1)$  to be assembled on a mixed-model assembly line. Hence the vector of demand ratios, is  $r = (6/13, 6/13, 1/13)$ .The schedule generated by algorithm 1 is demonstrated by the figure below.

Stage (k)	X[0]	X[1]	X[2]	M[0]	M[1]	M[2]	Product Re.	sum(int[0][k]->[	Total variation
1	0.47154	0.47154	0.77092	1	0	0	+1	0.50858	0.50858
2	0.92303	0.92303	1.5336	1	1	0	+2	0.0365	0.54008
3	1.37452	1.37452	2.2077	2	1	0	+1	0.57910	0.72968
4	1.81615	1.81615	3.0739	2	2	0	+2	0.17201	0.23627
5	2.26763	2.26763	4.0432	2	2	1	+0	0.56035	0.03402
6	2.73923	2.73923	5.1154	3	3	0	+1 +2 -3	0.31953	0.5385
7	3.23077	3.23077	6.3846	3	3	1	+3	0.31953	2.47337
8	3.73231	3.73231	7.8538	4	4	0	+1 +2 -3	0.56876	3.04142
9	4.23385	4.23385	9.5231	4	4	1	+3	0.17201	3.3343
10	4.73537	4.73537	11.3823	5	5	0	+1 +2 -3	0.70757	4.07101
11	5.23692	5.23692	13.4315	5	5	1	+3	0.0365	4.30651
12	5.73843	5.73843	15.6707	6	6	0	+1 +2 -3	1.27011	5.0462
13	6.0	6.0	17	6	6	1	+3	0	5.32607

Table 1 Schedule generated by Algorithm 1

The optimal production over 5 stages is (2, 2, 1) while the optimal production over 6 stages is (3, 3, 0). During the sixth stage one unit of product 1 and one unit of product 2 must be produced while one unit of product 3 must be destroyed. Of course this is impossible. Only one product can be assembled during a stage and products assembled earlier cannot be destroyed. However, based on these results, we can develop a number of “feasible” schedules.

The figure below demonstrates the schedule generated by the heuristic 1 as purposed by Meltenbure to as a “feasible” schedule solution.

Stage (k)	X[1]	X[2]	M[1]	M[2]	W[2]	Product Schedule	sum(m[k]-x[k])	Total Variation
1	0.46154	0.46154	1	1	0	+	0.50888	0.50888
2	0.92303	0.92303	1	1	0	+2	0.1365	0.64438
3	1.38462	1.38462	2	1	0	+	0.57988	1.22426
4	1.84615	1.84615	2	2	0	+2	0.14201	1.36627
5	2.30769	2.30769	2	2	1	+3	0.56805	1.93432
6	2.76923	2.76923	3	2	1	+	0.53491	2.46923
7	3.23077	3.23077	3	3	1	+2	0.50953	3.03876
8	3.69231	3.69231	4	3	1	+	0.72189	3.81065
9	4.15385	4.15385	4	4	1	+2	0.14201	3.95266
10	4.61538	4.61538	5	4	1	+	0.50888	4.46154
11	5.07692	5.07692	5	5	1	+2	0.1365	4.59805
12	5.53846	5.53846	5	5	1	+	0.50888	5.07692
13	6.0	6.0	5	5	1	+2	0.1365	5.07692

Table 2 Schedule generated by Heuristic 1

The variable k counts the stage of the production. That is, k runs from 1 to total demand and at each stage one of the items is produced. Array X [] consist the rational value of demand of the product to the total demand. Array M [] consists the nearest integer to the array X [] of respective product. At any stage k, M [] consists of the integer value denoting the number of items to be produced at the stage and product schedule is identified according to  $M[k] - M[k-1]$ .



Under this topic we have implemented Algorithm 1 and Heuristic 1 purposed by Miltenburg [24] and mentions on 6.1. Algorithm 1 found to be easier then Heuristic 1 while coding and understanding, however, Heuristic 1 removes the infeasibility issue found in Algorithm 1. The analytical study of Algorithm 1 and Heuristic 1 under different cases (considering the number of infeasible case encounter and total variation) is tabulated as below:

### 7.2.1 Comparative study of Algorithm 1 and Heuristic 1

<b>CASE VS ALGORITHM</b>	<b>TOTAL VARIATION</b>	<b>INFEASIBILITY</b>
CASE : (6,6,1)		
ALGORITHM 1	5.38462	4
HEURISTIC 1	5.07692	0
CASE : (5,2,4,7)		
ALGORITHM 1	7.09259	1
HEURISTIC 1	6.64815	0
CASE : (2,3,4)		

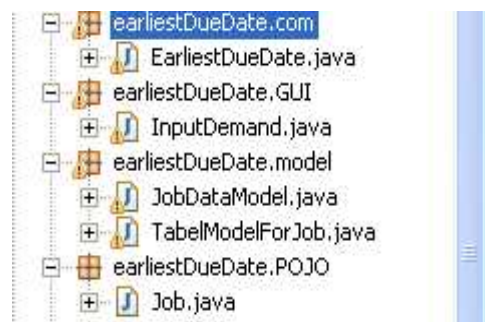
ALGORITHM 1	2.37037	0
HEURISTIC 1	2.37037	0
CASE : (1,5)		
ALGORITHM 1	1.05556	0
HEURISTIC 1	1.05556	0
CASE: (1,5,7,3)		
ALGORITHM 1	6.625	2
HEURISTIC 1	6.125	0
CASE: (1,5,7,3,9)		
ALGORITHM 1	13.12	3
HEURISTIC 1	12.32	0

**Table 3** Analytical view of Algorithm 1 vs. Heuristic 1

From study we can say that heuristic 1 removes the case of infeasibility seen in Algorithm 1 and also minimize the total variation in the case when infeasibility occurs.

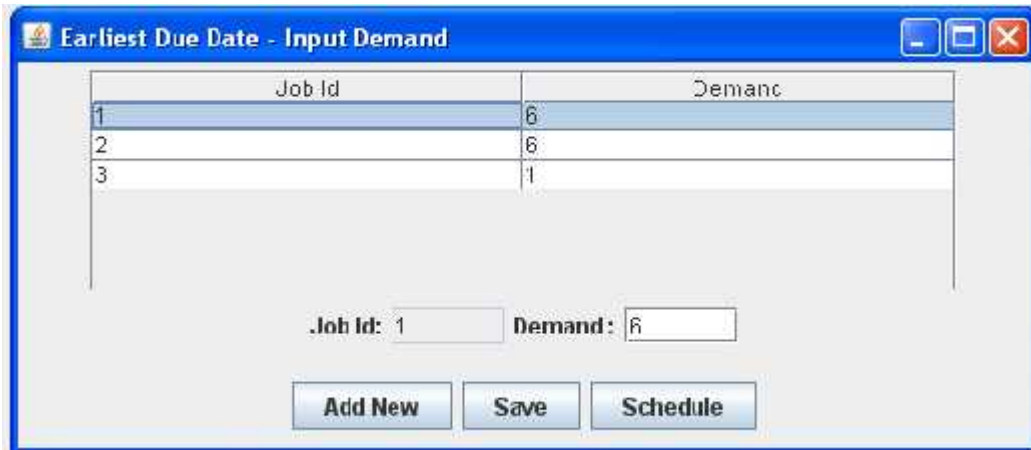
### 7.3 Earliest due date implementation issue

Earliest Due Date is a min-sum PRV problem and is an easier solution of Nearest Integer Point Algorithm [24]. It is simple and doesn't lead any infeasible case as given by Algorithm 1 of Meltenburg, Hence, doesn't need any heuristics and is more faster than Algorithm 2 or Algorithm 3 of Meltenburg. However the total variation seems to be little deviating than those given by Algorithm 3. Still, being simple to understand and fast execution time it has advantage over Meltenburg's algorithms. The architecture view of Earliest Due Date Problem implementation is shown by the following tree view.



**Figure 8** Architectural view of earliest due date package

To demonstrate the EDD approach consider Example 2 of Miltenburg (1989) with  $n = 3$ ,  $D_1 = 6$ ,  $D_2 = 6$ ,  $D_3 = 1$  and  $T = 13$ . There are a total of 13 jobs to be assigned to 13 positions.



**Figure 9** Date input frame for earliest due date package

The following table list the calculated due date value according to the algorithm considered.

Product	Unit	Due Date
1	1	1.083
	2	3.25
	3	5.417
	4	7.583
	5	9.75
	6	11.917
2	1	1.083
	2	3.25
	3	5.417
	4	7.583
	5	9.75
	6	11.917
3	1	6.5

Schedule List :  
1-2-1-2-1-2-3-1-2-1-2-1-2-

**Table 4** Schedule generated by earliest due date

The variable unit run from 1 to the number of items to be product for each product  
i. and due date for product of each unit of each product is calculated by the formula  $t_{ik} =$

$\lceil [(k-1/2)T] / D_i \rceil$ ,  $i=1, \dots, n$ ;  $k = 1, \dots, D_i$  as mention by Inman and Bulfin[14]. At each stage, we choose a unit of the product whose due date is minimum, as production unit.

## 7.4 Dynamic programming implementation issue

Dynamic Programming is of the solution for ORV problems. The architecture view of Dynamic Programming Problem implementation is shown by the following tree view.

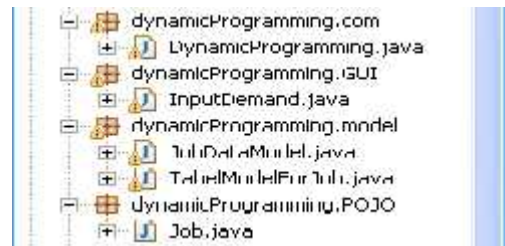
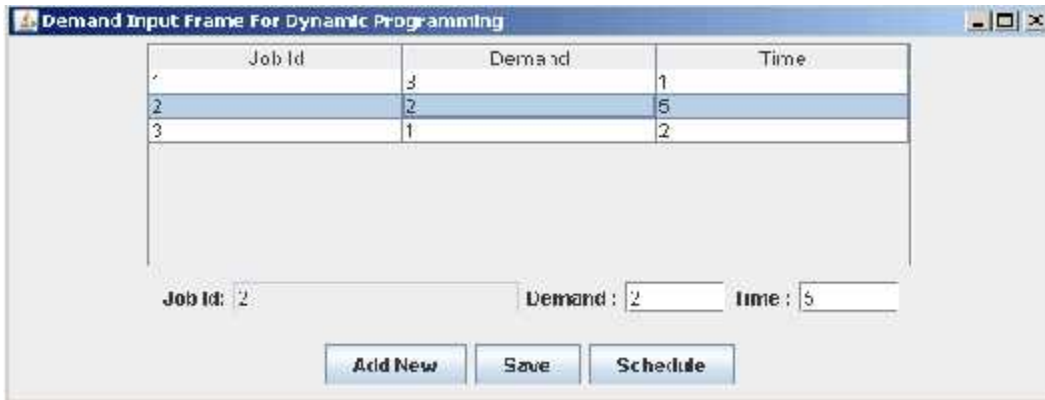


Figure 10 Architectural view of dynamic programming

The variable stage runs from 1 to the total number of demand. The array  $X[]$  at any stage consists of possible combination of the production items. The value of  $f(X-e)$ ,  $g(x)$  is calculated as mention in session 6.2.2. The value of  $f(x)$  at any stage is the sum of  $f(X-e)$  and current  $g(x)$ . There may be many possible combination production at any stage and are denoted by the Expand field whose values is  $E$ . One of the combination at a stage whose Expand field is  $E$  is chosen according to the minimum  $f(x)$ .

Consider a three-product example with the date shown in fig 5.7. Three products with demands 3, 2 and 1 are to be produced. The DP procedure is used to determine the

optimal production schedule and this is done in table below



**Figure 11** Date input frame for dynamic programming testing

At each stage  $k$ , all subsets  $X$  which are feasible ( $|X| = k$  and  $x_i \leq d_i \forall i$ ) are generated. For each subset, all possible  $X - e_i$  are then generated. The value of  $f(X - e_i)$  is available from the computations done at stage  $k - 1$ ; the value of  $g(X)$  is computed; and the two are added together. The minimum of these values is  $f(X)$ .

Consider, for example, the subset  $X = (1, 2, 0)$  at  $k = 3$   $e_i$  is either  $(1, 0, 0)$  or  $(0, 1, 0)$  but not  $(0, 0, 1)$  since  $x_3 = 0$ ; so  $X - e_i$  takes the values  $(0, 2, 0)$  and  $(1, 1, 0)$ . From the computation done at  $k = 2$ ,  $f(0, 2, 0) = 57.35$  and  $f(1, 1, 0) = 6.36$ .

$$\begin{aligned}
 g(X) &= g(1, 2, 0) = 12(1-3(.5))^2 + 52(2-3(.333))^2 + 22(0 - 3(0.167))^2 \\
 &= .0250 + 25(1) + 4(0.250) \\
 &= 26.25
 \end{aligned}$$

Therefore,  $f(1, 2, 0) = \min(57.35 + 26.25, 6.36 + 26.25) = 32.61$ . The total minimum variability for the problem is 13.97 and by working backwards through table 5 we obtain the optimal sequence, 1-2-3-1-2-1.

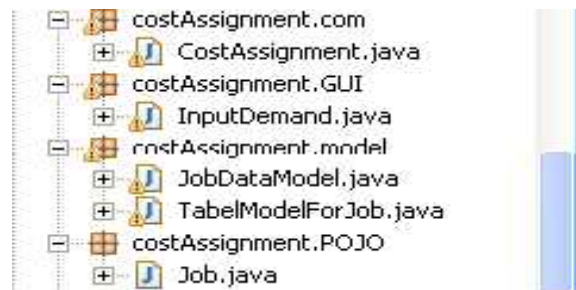
Dynamic Programming Schedule									
Stage	(i1,i2,...,in)	Index	P-Index	Product B...	x = e	f(x-e)	g(x)	f(x)	Expand
1	1-1-0		-1	1	-0-0-0	0	5.139	3.139	E
1	0-1-0		-1	2	-0-0-0	0	11.472	11.472	E
1	0-0-1		-1	3	-0-0-0	0	6.806	6.806	E
2	2-0-0	3	C	1	1-0-0	3.139	12.556	15.69499...	E
2	1-1-0	4	L	2	1-0-0	3.139	5.222	6.361	E
2	1-0-1	5	C	3	1-0-0	3.139	12.389	16.028	E
2	1-1-0	6	1	1	0-1-0	11.472	5.222	11.69388...	
2	0-2-0	7	1	2	0-1-0	11.472	40.389	57.36100...	E
2	0-1-1	8	1	3	0-1-0	11.472	6.556	11.028	
2	1-0-1	9	2	1	0-0-1	5.306	12.389	18.695	
2	0-1-1	10	2	2	0-0-1	5.306	6.556	11.362	E
3	3-0-0	11	C	1	2-0-0	15.69499...	20.25	43.945	E
3	2-1-0	12	C	2	2-0-0	15.69499...	1.25	16.945	
3	2-0-1	13	C	3	2-0-0	15.69499...	20.25	41.945	E
3	2-1-0	14	4	1	1-1-0	6.361	1.25	1.611	E
3	1-2-0	15	4	2	1-1-0	6.361	20.25	32.611	E
3	1-1-1	16	4	3	1-1-0	6.361	1.25	1.611	E
3	2-0-1	17	C	1	1-0-1	16.028	20.25	42.278	
3	1-1-1	18	6	2	1-0-1	16.028	1.25	17.278	
3	1-2-0	19	7	1	0-2-0	57.36100...	20.25	83.611	
3	0-2-1	20	7	3	0-2-0	57.36100...	20.25	85.611	
3	1-1-1	21	10	1	0-1-1	11.362	1.25	12.612	
3	0-2-1	22	10	2	0-1-1	11.362	20.25	39.612	E
4	3-1-0	23	11	2	3-0-0	43.945	5.556	49.501	
4	3-0-1	24	11	3	3-0-0	43.945	40.389	89.634	
4	3-0-1	25	13	1	2-0-1	41.945	40.389	87.634	E
4	2-1-1	26	13	2	2-0-1	41.945	5.222	45.167	
4	3-1-0	27	14	1	2-1-0	7.311	5.556	13.167	E
4	2-2-0	28	14	2	2-1-0	7.311	12.389	20.6	E
4	2-1-1	29	14	3	2-1-0	7.311	5.222	10.633	
4	2-2-0	30	15	1	1-2-0	32.611	12.389	45.6	
4	1-2-1	31	15	3	1-2-0	32.611	12.556	45.16699...	
4	2-1-1	32	16	1	1-1-1	7.311	5.222	10.633	E
4	1-2-1	33	16	2	1-1-1	7.311	12.556	20.16699...	F
4	1-2-1	34	22	1	0-2-1	39.612	12.556	52.168	
5	3-1-1	35	25	2	3-0-1	87.634	11.472	93.306	
5	3-2-0	36	27	2	3-1-0	13.167	6.806	13.973	E
5	3-1-1	37	27	3	3-1-0	13.167	11.472	24.639	
5	3-2-0	38	23	1	2-2-0	20.6	6.806	23.306	
5	2-2-1	39	23	3	2-2-0	20.6	3.139	23.639	
5	3-1-1	40	32	1	2-1-1	10.633	11.472	22.305	E
5	2-2-1	41	32	2	2-1-1	10.633	3.139	13.972	E
5	2-2-1	42	33	1	1-2-1	20.16699...	3.139	23.30599...	
6	3-2-1	43	35	3	3-2-0	18.973	0.0	18.973	
6	3-2-1	44	43	2	3-1-1	22.305	0.0	22.305	
6	3-2-1	45	41	1	2-2-1	13.972	0.0	13.972	E

Schedule :  
1-2-3-1-2-1

Table 5 Schedule generated by dynamic programming

## 7.5 Cost assignment problem

Cost assignment problem is min-sum PRV problem. Differ to Meltenburg's Algorithms or the EDD rule, the cost assignment problem calculate the cost of scheduling each product and chose the one with minimum cost. As Meltenburg's Algorithm 1, Cost assignment problem does not product any infeasible hence doesn't need any heuristic solution. So, is reliable then Meltenburg's Algorithms. Calculating the cost through excess inventory or shortage calculation add extra work hence it is slower then EDD rules or Meltenburg's algorithm but is more perfect and have minimum total variation then both Meltenburg's Algorithm or EDD. The architecture view of Cost Assignment Problem implementation is shown by the following tree view.



**Figure 12** Architectural view of cost assignment problem

Consider three part types  $A_1$ ,  $A_2$  and  $A_3$  with demands  $d_1 = 2$ ,  $d_2 = 3$  and  $d_3 = 5$ , respectively, as shown in Figure below.



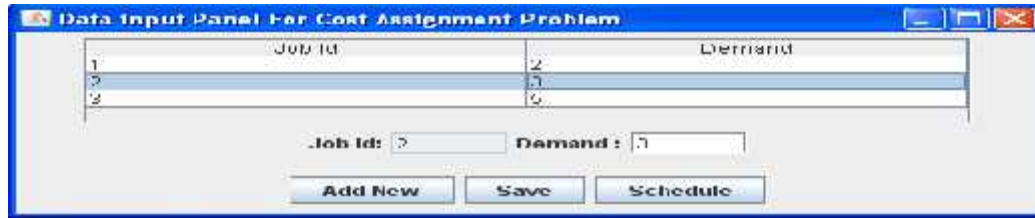


Figure 13 Data input frame for cost assignment problem

Thus,  $T = 10$ ,  $r_1 = 0.2$ ,  $r_2 = 0.3$  and  $r_3 = 0.5$ . The ideal position computed by using the formula is shown in the table below.

Product	S.No	1	2	3	4	5	6	7	8	9	10
1	1	0.8	0.2	0.2	0.6	1.0	1.0	1.0	1.0	1.0	1.0
	2	1.0	1.0	1.0	1.0	1.0	0.8	0.2	0.2	0.8	1.0
2	1	0.4	0.2	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	2	1.0	1.0	1.0	0.6	1.0	0.6	1.0	1.0	1.0	1.0
3	1	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.2	0.4	1.0
	2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	4	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0
5	1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0
	2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0

Table 6 Excess inventory or shortage costs calculated

Product	S.No	1	2	3	4	5	6	7	8	9	10
1	1	0.8	0.2	0.2	0.2	0.8	1.8	2.8	3.8	4.8	5.8
	2	5.0	4.0	3.000...	2.0000...	1.0	0.0	0.2	0.3	0.2	0.0
2	1	0.4	0.2	0.2	1.0	2.0	3.0	4.0	5.0	6.0	7.0
	2	3.6	2.6	1.3	3.6	0.0	0.0	0.6	1.3	2.6	3.6
3	1	7.0	6.0	5.0	4.0	3.0	2.0	1.0	0.2	0.0	0.4
	2	0.0	0.0	1.3	2.0	3.0	4.0	5.0	6.0	7.0	0.0
3	1	2.0	1.0	0.2	3.0	1.0	2.0	3.0	4.0	5.0	6.0
	2	4.0	3.0	2.0	1.0	0.0	0.0	1.0	2.0	3.0	4.0
5	1	6.0	5.0	4.0	3.0	2.0	1.0	0.0	0.0	1.0	2.0
	2	0.0	7.0	6.0	5.0	4.0	3.0	2.0	1.0	0.0	0.0

Schedule:  
3-2-1-3-2-3-3-1-2-3

Table 7 Schedule generated by cost assignment problem

### 7.5.1 Comparative study of min-sum PRV problem

The following table lists the nature of Algorithm1, Heuristic 1, EDD and Cost Assignment problem over different input cases (run time calculated in millisecond):

CASE/Algorithm	Total Run Time		
<b>Case:(6,6,1)</b>			
Algorithm 1	16	1, 2, 1, 2, 3, (1, 2, -3), 3, (1, 2, -3), 3, (1, 2, - 3), 3, (1, 2, -3), 3	Not Feasible
Heuristic 1	31	1, 2, 1, 2, 3, 1, 2, 1, 2, 1, 2, 1, 2	Feasible
EDD	15	1, 2, 1, 2, 1, 2, 3, 1, 2, 1, 2, 1, 2	Feasible
Cost Assignment	141	1, 2, 1, 2, 3, 1, 1, 2, 2, 1, 2, 1, 2	Feasible
<b>Case: (3, 5, 7, 2)</b>			
Algorithm 1	31	3, 2, 1, 3, 4, 2, 3, 1, 3, 2, 3, 2, 4, 3, 1, 2, 3	Feasible
Heuristic 1	31	3, 2, 1, 3, 4, 2, 3, 1, 3, 2, 3, 2, 4, 3, 1, 2, 3	Feasible
EDD	18	3, 2, 1, 3, 4, 2, 3, 1, 2, 3, 3, 2, 4, 3, 1, 2, 3	Feasible
Cost Assignment	156	3, 2, 1, 3, 4, 2, 3, 1, 3, 2, 3, 2, 4, 3, 1, 2, 3	Feasible

<b>Case: (7, 3, 1)</b>			
Algorithm 1	15	1, 2, 1, 1, 3, (1, 2, -3), 1, 1, 2, 1	Not Feasible
Heuristic 1	31	1, 2, 1, 1, 3, 1, 2, 1, 1, 2, 1	Feasible
EDD	14	1, 2, 1, 1, 3, 1, 1, 1, 1, 2, 2	Feasible
Cost Assignment	140	1, 2, 1, 1, 1, 2, 3, 1, 1, 2, 1	Feasible
<b>Case : (5, 3, 7)</b>			
Algorithm 1	25	3, 1, 2, 3, 1, 3, 2, 3, 1, 3, 1, 3, 2, 1, 3	Feasible
Heuristic 1	25	3, 1, 2, 3, 1, 3, 2, 3, 1, 3, 1, 3, 2, 1, 3	Feasible
EDD	12	3, 1, 2, 3, 1, 3, 1, 2, 3, 3, 1, 3, 2, 1, 3	Feasible
Cost Assignment	125	3, 1, 2, 3, 1, 3, 2, 1, 3, 3, 1, 3, 2, 1, 3	Feasible

Table 8 Comparative studies of min-sum PRV problems

## 7.6 Min-max-absolute-chain sequencing problem

The min-max-absolute-chain sequencing problem is min-max PRV problem. The architecture view of the implementation of modified Min-Max-Absolute-Chain algorithm for both overlapping and non-overlapping sequences as chain constraints is presented below in tree view.



**Figure 14** Architectural view of min-max-absolute-chain sequencing problem

### 7.6.1 Non-overlapping sequences

Testing for the modified algorithm for non overlapping sequence is done in the implementation as follows.

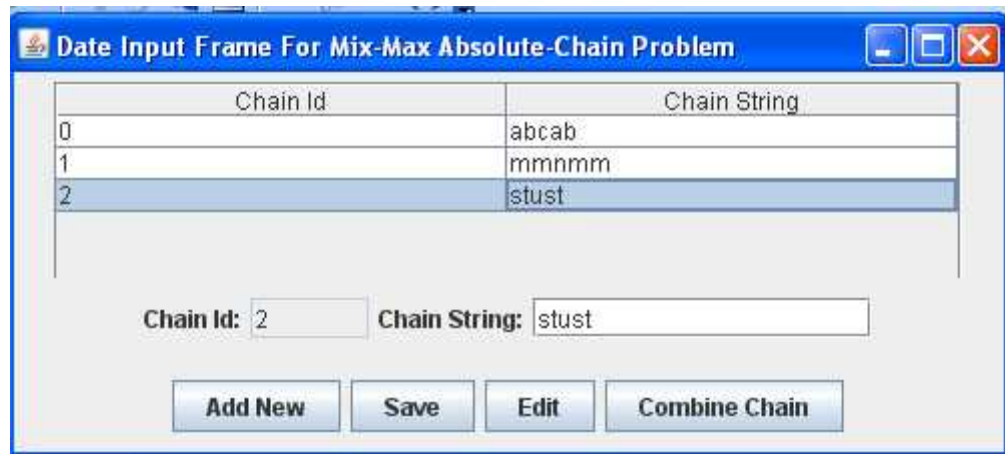


Figure 15 Input frame: min-max-absolute-chain (non-overlapping sequence)

Chain Id	Job Name	Earliest Due Date	Late Due Date
C	a	4.0	6.0
C	b	4.0	6.0
C	c	8.0	9.0
C	a	12.0	13.0
C	b	12.0	13.0
1	m	3.0	6.0
1	n	3.0	6.0
1	m	8.0	9.0
1	n	8.0	9.0
1	m	14.0	14.0
1	n	14.0	14.0
2	s	4.0	6.0
2	t	4.0	6.0
2	u	8.0	9.0
2	s	12.0	13.0
2	t	12.0	13.0

Table 9 Calculation of window value for non-overlapped sequence

Chain Id	Job Name	Earliest Due Date	Late Due Date
0	a	4.0	5.0
0	b	4.0	5.0
1	c	0.0	3.0
1	a	12.0	13.0
1	b	12.0	13.0
1	m	7.0	7.0
1	n	7.0	7.0
1	m	8.0	8.0
1	n	8.0	8.0
1	m	14.0	14.0
1	n	14.0	14.0
2	s	4.0	5.0
2	t	4.0	5.0
2	u	8.0	8.0
2	s	12.0	13.0
2	t	12.0	13.0

Schedule :  
mnbstcmnabstmn

Table 10 Output: min-max-absolute-chain (non-overlapping sequence)

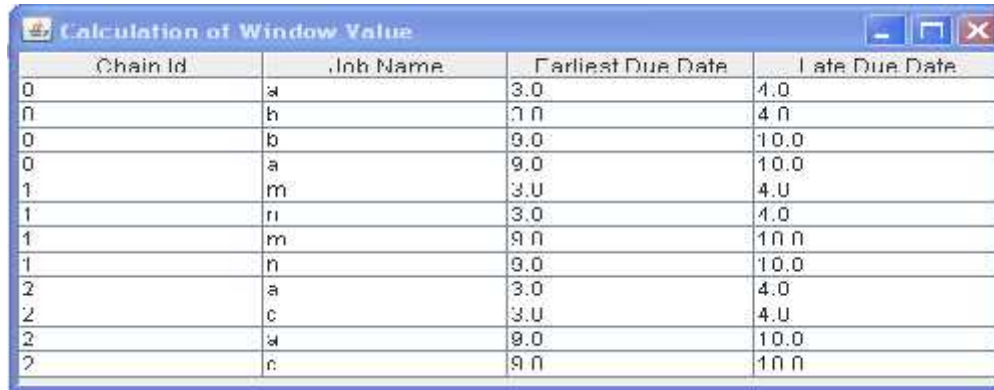
## 7.6.2 Overlapping-sequences

Testing for the modified algorithm for overlapping sequence is done in the implementation as follows.

Chain Id	Chain String
0	abba
1	mnmn
2	acac

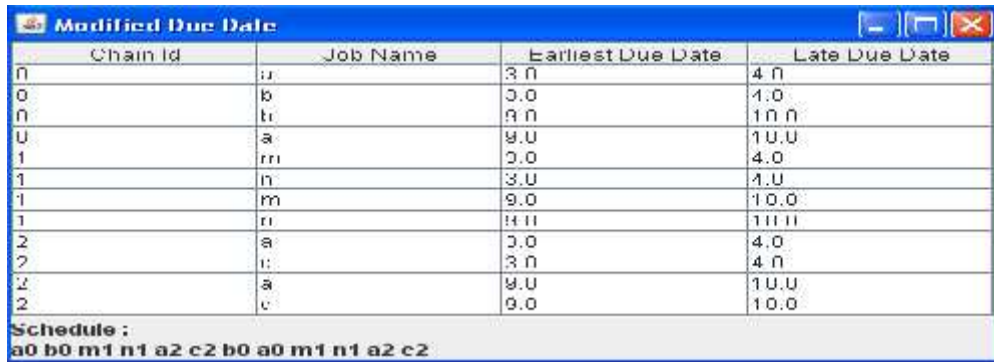
Chain Id:  Chain String:

**Figure 16** Input frame: min-max-absolute-chain (overlapping sequence)



Chain Id	Job Name	Earliest Due Date	Late Due Date
0	a	3.0	4.0
0	b	3.0	4.0
0	b	9.0	10.0
0	a	9.0	10.0
1	m	3.0	4.0
1	n	3.0	4.0
1	m	9.0	10.0
1	n	9.0	10.0
2	a	3.0	4.0
2	c	3.0	4.0
2	a	9.0	10.0
2	c	9.0	10.0

Table 11 Calculation of window value for overlapped sequence



Chain Id	Job Name	Earliest Due Date	Late Due Date
0	b	3.0	4.0
0	b	9.0	10.0
0	a	9.0	10.0
1	m	3.0	4.0
1	n	3.0	4.0
1	m	9.0	10.0
1	n	9.0	10.0
2	a	3.0	4.0
2	c	3.0	4.0
2	a	9.0	10.0
2	c	9.0	10.0

**Schedule :**  
a0 b0 m1 n1 a2 c2 b0 a0 m1 n1 a2 c2

Table 12 Output: min-max-absolute-chain algorithm (overlapping sequences)

## **8 Conclusion and Recommendation**

In this dissertation, different algorithms and heuristics of mixed-model-JIT sequencing problems have been studied. Most of these algorithms and some heuristics are implemented. Finally, a new algorithm is purposed which modifies the min-max-absolute-chain algorithm to adopt overlapping sequence.

It is let to identify either the algorithm can be implemented for cyclic sequence or it can be extended for priority chain system and for multi-processor system. Likewise, much more work is left to convert the pseudo solution raised in this dissertation to the real version. It is also remains to study these problems for sum deviation objectives. These works are left open for the further study.



## 9 References

1. J. Bautista, R. Companys , A. Corominas, A note on the relation between the product rate variation (PRV) problem and the apportionment problem, The Journal of Operational Research Society, 47, 1410-1414.
2. N. Brauner, Y. Crama, The maximum deviation just-in-time scheduling problem, Discrete Applied Mathematics, 134 (2004) 25-50.
3. N. Brauner, V. Jost, W. Kubiak, On symmetric fraenkel's and small deviations conjecture, Les Cahiers du Laboratoire Laibniz- IMAG, 54, Grenoble, France, 2004.
4. P.Brucker : Scheduling Algorithms: 2nd edition, Springer, Heidelberg, 1995.
5. C.H. Chew, Intelligence system for business, The Pennsylvania State University, [www.net1.ist.psu.edu](http://www.net1.ist.psu.edu).
6. T.H. Corman, C.E. Leiser, R.L. Rivest, C. Stein: Introduction to algorithm, 2nd edition, Prentice-Hall of India Pvt. Ltd., 2004.
7. A. Corominas, N. Moreno, On the relation between optimal solutions for different type of min-sum balanced JIT optimization problems, INFOR, 41, 4 (2003) 333-339.
8. T. N. Dhamala, "Just-in-time sequencing algorithms for mixed-model production system", Nepal Math Sciences.
9. T .N. Dhamala, S. R. Khadka, Just-in-time sequencing for mixed-model production system revisited, Discrete Optimization Submitted, 2007.
10. T. N. Dhamala, W. Kubiak, A brief survey of just-in-time sequencing for mixed-model systems, International Journal of Operations Research, 2, 2 (2005) 38-47.
11. Debugging and Interactive Development, [www.opendylan.org](http://www.opendylan.org) .

12. J. E. Hopcraft, R. Motwani, J. D. Vilman: Introduction to Automata Theory, Language and Computation, 2nd edition, Pearson Education Pvt. Ltd., 2004.
13. W. A. Horn, Some simple scheduling algorithms, *Naval Research Logistics Quarterly* 21 (1974) 177-185.
14. R. R. Inman, R. L. Bulfin, Sequencing JIT mixed-model assembly lines, *Management Science*, 37, 7 (1991) 901-904.
15. W. Jawor, Three dozen papers on online algorithm, *ACM SIGACT News*, 36, 1 (2005).
16. M. Y. Kovalyov, W. Kubiak, J. S. Yeomans, A computational analysis of balanced JIT optimization algorithm, *Information Processing and Operational Research*, 39, 3 (2004) 4955-4974.
17. W. Kubiak, Minimizing variation of production rates in just-in-time systems: A survey, *European Journal of Operational Research*, 66 (1993), 259-271.
18. W. Kubiak, Cyclic just-in-time sequence are optimal, *Journal of Global Optimization*, 27 (2003), 333-347.
19. W. Kubiak, S. Sethi, A note on “level schedules for mixed-model assembly lines in just-in-time production system”, *Management Science*, 37, 1 (1991) 121-122.
20. W. Kubiak, S. Sethi, Optimal just-in-time schedules for fixable transfer lines, *The International Journal of Flexible Manufacturing System*, 6 (1994) 137-154.
21. W. Kubiak, G. Steiner, J. S. Yoemans, Optimal level schedules for mixed-model multi-level just-in-time assembly system, *Annals of Operations Research*, 69 (1997) 241-259.
22. V. Lebacque, V. Jost, N. Brauner, Simultaneous optimization of classical objectives in JIT scheduling, Submitted to Elsevier Science, 2005.
23. Q. Lee, Lean manufacturing strategy, [www.strategosinc.com](http://www.strategosinc.com)

24. J. Miltenburg, Level schedules for mixed-model assembly lines in just-in-time production systems, *Management Science*, 35,2 (1989) 192-207.
25. J. Miltenburg, T. Goldstein, Developing production schedules which balance part usage and smooth production loads for just-in-time production systems, *Naval Research Logistics*, 38 (1991) 893-910.
26. J. Miltenburg, G. Sinnamon, Scheduling mixed-model multi-level just-in-time production systems, *International Journal of Production Research*, 27, 9 (1989) 1487-1509.
27. J. Miltenburg, G. Steiner, J.S. Yeomans, A dynamic programming algorithm for scheduling mixed-model just-in-time production systems, *Mathematics Computer Modeling*, 13, 3 (1990) 57-66.
28. M. Milenkovic: *Operating System*, 2nd edition, Tata McGraw Hill, 2002.
29. Mixed model production system, University of Guelph, [www.uoguelph.ca](http://www.uoguelph.ca).
30. K. H. Rosen: *Discrete Mathematics and Its Application*, 2nd edition. Tata McGraw Hill, 2003.
31. S. Sahni, E. Morowitz, S. Rajasekaran, *Computer Algorithm C++*, reprint-2002, Galgotia Publication Pvt. Ltd.
32. Stage-by-sage business development, [www.1000ventures.com](http://www.1000ventures.com).
33. W. Stalling: *Operating System*, 2nd edition, Prentice-Hall of India Pvt. Ltd., 2002.
34. G. Steiner, J.S. Yeomans, Optimal level schedules in mixed-model multi-level JIT assembly systems with pegging, *European Journal of Operational Research*, 95 (1996) 38-52.
35. G. Steiner, J. S. Yeomans, Level schedules for mixed-model, just-in-time processes, *Management Science*, 36, 6 (1993) 728-735.
36. T. Sukanuma, T. Ogasawara, Overview of the IBM Java Just-in-Time Compiler, *IBM System Journal*, 39, 1 ( 2000).

37. A. Tanenbaum: Modern Operating System, 2nd Edition, Prentice Hall of India Pvt. Ltd., 2005.
38. Toyota Motor Corporation Global Site, [www.toyota.co.jp](http://www.toyota.co.jp).
39. M.A. Weiss: Data Structures and Algorithm Analysis in C, 2nd edition, Florida International University, 2002.