1 INTRODUCTION

Mixed model just-in-time production system has been developed with a goal of reducing cost of diversified small-lot instead large-lot that minimizes large inventories and large shortages. The MMJIT production system minimizes both the earliness and the tardiness penalties that respond to the customer demand for a variety of products without incurring large inventory and shortages. This requires designing and controlling the system in such a way that the only required products are produced in the necessary quantity when needed. The main aim of the system is to obtain a sequence of a number of products that minimizes deviation throughout the time, between the actual and the ideal (desired) production. This maintains the final assembly line keeping rate of parts usage as constant as possible. The sequence at the final level is crucial and affects the entire supply chain as all other levels are also inherently fixed due to pull nature of the system. The problem that deals with the final level only is called the single level problem whereas dealing with more than one level is multilevel problem. This is called leveling or balancing the schedule.

Mixed model assembly lines with negligible changeover costs between the products allow manufacturing of different models of a common base product in evenly distributed sequences on the same line; Boysen et al. [3]. Monden [28] creates the first interest in JIT sequencing problem of single level mixed model assembly line. He describes a local search heuristic "Goal Chasing Method" to deal with the problem. Miltenburg [24] considers the problem of determining the sequence for producing different products on the line that keeps a constant rate of usage of every part used by the line. In other words, the quantity of each part used by the mixed model assembly line per unit of time should be kept as constant as possible. This allows very little variability in the usage of each part from one time horizon to the other. Miltenburg and Sinnamon [25] and Miltenburg and Goldstein [26] extend the formulation to multi-level system. Kubiak [18] 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. He districted PRV problem as total deviation PRV problem and maximum deviation PRV problem. Such problem represent the problem considered by Steiner and Yeomans [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 [25] and Miltenburg and Goldstein [26] are categorized as ORV problem.

Miltenburg [24] and Miltenburg and Sinnamon [25] observe the existence of the cyclic sequences for the total PRV problem. Kubiak [21] proves that optimal JIT sequences are *cyclic*. Dhamala and Kubiak [11] conjecture that cyclic sequences in the ORV are optimal, too. This provided an important theoretical support to the usual for JIT systems practice of repeating relatively short sequence to build a sequence for a longer time horizon [24][28]. It also has important consequences on the computational time complexity of all existing algorithms for PRV [17]. Steiner and Yeoman [35], following the optimization algorithm for the total deviation given in Kubiak and Sethi [18] give a graph theoretic optimization algorithm for minimizing maximum deviation JIT single-level sequencing problem. They also give an algorithm for minimizing multi-level maximum deviation JIT assembly systems under the pegging assumption, Steiner and Yeoman [36]. If outputs at production levels which feed the final assembly level are dedicated to the final product into which they will be assembled, then the problem with pegging is equivalent to a weighted single level problem which can then be minimized by modified algorithm for un-weighted single-level problem. Kovalov et al [16] perform a large–scale computational study to examine various optimization algorithms formulating several open questions of previous time as conjectures and answer them by means of extensive computational testing.

The upper and lower bounds on the threshold value of the un-weighted max-abs problem are established in Steiner and Yeoman [35], for the first time. Consequently they established the bounds for the same problem with the weighted case [36]. Dhamala et al. [12] established the lower and upper bounds on the threshold value both for weighted and un-weighted max-sqr problems. Dhamala and Khadka [10] and Dhamala et al. [13]

established that there exists no feasible solution for any instance of the max-absolute problem when the deviation is less than 1/3 and no feasible solutions for any instance of the max-sqr problem when the deviation is less than 1/9 (cf. [12]).

This dissertation has been divided in 7 chapters. The brief descriptions of these are as follows: Chapter 2 reviews some fundamental definitions relating to the subsequent study of this work. In Chapter 3 a synopsis introduction of the scheduling problem is given. In Chapter 4, the mathematical formulation of JIT production system is given with multilevel and single level formulation. Finally the pegged ORVP problem which can be reduced to weighted single level problem is formulated in Section 4.3.

Chapter 5 is the main focus of this thesis in which the solution procedure both for min-sum and min-max PRVP are explored. Since the cyclic solutions are optimal for both problems with corresponding program, the optimal cyclic sequences for both problems are presented for some instances. The implementation issue of EDD methods for optimal solutions for min-max problem is the contents of Section 5.2. The nearest integer point method to find a solution for min-sum problem with an example is presented in Section under different algorithms and heuristics established in Miltenburg [24]. In Section cost assignment approaches to the solution for min-sum PRVP is given. As the cost assignment is fruitful only for the practical sized problem, the dynamic programming approach to the min-sum is the content of Section 5.5. In Chapter 6, the Toyota's Goal Chasing method for sum-deviation ORVP is explored under parts usage goal. As the min-sum ORVP is NP-hard, the heuristic approaches purposed by Miltenburg and Sinnamon [25] is analyzed in detail with implementation and an example is presented in which cyclic solution is obtained. Chapter 7 contains the conclusion, further suggestions and recommendations.

The main work of this dissertation is to study the different cyclic sequencing procedures to PRVP with their implementation issues. The heuristic to solve ORVP approaches are explored. Moreover examples are presented for some solution approaches.

2 FUNDAMENTAL BACKGROUND

Functions

Given two sets A and B, a function *f* is a binary relation on A × B such that for all a ∈A, there exists precisely one b ∈B such that (a, b) ∈*f*. the set A is called the domain of *f*, and the set B is called the co-domain of *f*. We sometimes write $f : A \to B$; and if (a, b) ∈f, we write $b = f(a)$, since b is uniquely determined by the choice of a.

A function f whose values are in the set of real numbers R is called a real-valued function and is non-negative if $f \geq 0$. Since we shall be mostly interested in real valued function of real variable throughout this thesis, we write only "function" to mean the real-valued function of real variable unless otherwise specified. The function f is said to be monotonically increasing if $f(x) \leq f(y)$ whenever $x \leq y$. Similarly, f is called monotonically decreasing if $f(x) \geq f(y)$ whenever $x \leq y$.

The function f defined over a set $A \subseteq \mathrm{R}$ is said to take on its maximum and minimum over A at the points x^* and x' respectively if $f(x') \leq f(x) \leq f(x^*)$ *for all* $x \in A$.

The function f is said to be unimodal if for some value a (the mode) such that either (i) or (ii) holds:

(i) f is monotonically increasing for $x \leq a$ and monotonically decreasing for $x \geq a$. In that case, the maximum value of f is $f(a)$ and there are no other local maxima.

(ii) f is monotonically decreasing for $x \leq a$ and monotonically increasing for $x \geq a$. In that case, the minimum value of f is $f(a)$ and there are no other local minima.

A function f is said to be convex over a convex set $A \subseteq \mathrm{R}$ if for any two points $x, y \in A$ and for all α, $0 \leq \alpha \leq 1$, $f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y)$.

The floor function (it is often also called greatest integer function) denoted by

$\lfloor x \rfloor$ assigns to the real number x the largest integer that is less than or equal to x.

The ceiling function denoted by $\lceil x \rceil$ assigns to the real number x the smallest integer that is greater than or equal to x

Graph Theoretical Denotations

A graph as a mathematical structure is a pair $G = (V, E)$ where $V = \{v_1, ..., v_n\}$ is a non-empty finite set of *vertices*, and E has as elements subsets of V of cardinality two called *edges*. An edge between two vertices v_i and v_j for $i \neq j$ is denoted by $[v_i, v_j]$. A directed graph (or diagraph) G is a pair, (V, E) where V is called a finite set of vertices and E is a is a set of ordered pairs of vertices called arcs; that is, $A \subseteq V \times V$ .. In an undirected graph $G = (V, E)$, the edge set E consists of unordered pairs of vertices, rather than ordered pairs. That is, an edge is a set $\{u, v\}$, where $u, v \in V$ and $u \neq v$. The unqualified term graph usually means undirected graph.

A path of length k from u to a vertex u' in a graph $G = (V, E)$, where $\{u, u'\} \in E$, is a sequence $\{v_0, v_1, ..., v_k\}$ of vertices such that $u = v_0$, $u' = v_k$ and $(v_{i-1}, v_i) \in E$ for $i = 1,2,...k$. The length of the path is the number of edges in the path.

Let $G = (V, E)$ be a graph in which the vertex set V can be partitioned into two disjoint sets, V_1 and V_2, and each edge in E has one vertex in V_1 and another in V_2. In such a case G is called a bipartite graph and we denote by $G = (V_1 \cup V_2, E)$. If a graph has no such a partition, we say it non-bipartite. A bipartite graph $G = (V_1 \cup V_2, E)$ is said to be complete if each vertex of V_1 is connected to each vertex of V_2. The bipartite graph $G = (V \cup U, E)$ is *V-convex* if there is an ordering on V such that $[v_i, u_k] \in E$ and $[v_j, u_k] \in E$ with $v_i, v_j \in V$, $v_i < v_j$ implies that $[v_p, u_k] \in E$ for $v_i \leq v_p \leq v_j$ (see [34]).

The graph $G = (V, E)$ together with a function $W : E \to R^+$ is called the edge weighted graph and together with a function $W : V \to R^+$ is called vertex weighted graph, where R^+ the set of all nonnegative real numbers is.

Algorithms and Heuristics

A *computational problem* is a mathematical object representing a general question that might want to solve and is independent of its specific input. A problem with a specific set of inputs is called an *instance*. Hence, a computational problem is a function $\Pi : \mathsf{Z} \to \mathrm{Y}$, where Z is the set of all problems instances I and Y is the set of solutions. An algorithm is a set of precise instructions for performing a computation or solving an optimization problem [31].

In other words, an algorithm is any well-defined computation procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output. To represent algorithms we use English language however for the simplicity we use pseudo code that can represent an algorithms in clear manner like in English language and gives the implementation view as in the programming languages.

There are several properties that algorithms generally share. They are useful to keep in mind when algorithms are described. These properties are:

1. **Input/output:** An algorithm has input or set of inputs values from the set that has possible input values and for each inputs an algorithm produces the solution of the problem that are in the set of output values.
2. **Definiteness:** Each step must be clear and unambiguous.
3. **Correctness:** An algorithm produced output must be correct for each set of input values.
4. **Finiteness:** An algorithm must terminate after finite amount of time for every possible set of values.

5. **Effectiveness:** Each step must be executable in finite time.
6. **Generality:** The devised algorithm must be capable of solving the problem of similar kind for all possible inputs.

Heuristic is the art and science of discovery and invention. The word comes from the Greek root as “eureka”, means “to find”. A heuristic for a given problem is a way of direction towards a solution (see [32]). It is different from an algorithm in that it merely serves as a rule of thumb or guideline, as opposed to an invariant procedure. Heuristics may not achieve the desired outcome, but can be extremely valuable to problem-solving (see [43]). Good heuristics can dramatically reduce the time required to solve a problem by eliminating the need to consider unlikely possibilities or irrelevant states.

The mathematician Gege Polya popularized heuristics in the twentieth century in his book *How to solve it* [43]. He was motivated by his experiences in mathematics education where students are taught mathematical proofs, without learning techniques to formulate proofs themselves. *How to solve it* is a collection of ideas about heuristics that he taught to math students: ways of looking at problem and casting about for solutions that often give results very quickly (see [43]).

Complexity of Algorithms

When an algorithm is designed it must be analyzed for its efficiency. The efficiency of an algorithm is measured in terms of complexity. The complexity of algorithms is mentioned in terms of resource needed by the algorithm. We generally consider two kinds of resources used by an algorithm time and space. The measure of time required by an algorithm to run is given by *time complexity* and the measure of space (compute memory) required by an algorithm is given by *space complexity*. Here in this dissertation we generally discuss time complexity of an algorithm that is given by the number of operations needed by an algorithm for given set of inputs. Since actual time required may vary from computers to computers we use number of operations required to measure the time complexity.

Let f and g be functions from the set of real numbers to the set of real numbers. A function $f(x) = O(g(x))$ if and only if there exists two constants c and n_0 such that for all $n \geq n_0$, $0 \leq f(n) \leq c \times g(n)$

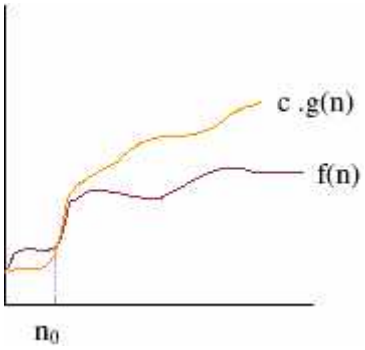


Figure 1: Graphical notation of *f (n) = O (g (n))*

A polynomial time (polynomial) algorithm is the one whose time complexity functions $T(k) \in O(h(k))$, where h is some polynomial and k is the input length of an instance I. If time complexity function cannot be bounded by the polynomial function, it is called exponential time algorithm. A computational problem Π is called polynomial solvable if there is a polynomial time algorithm solving it. A problem Π is called pseudo-polynomial solvable if the time complexity function $T(k)$ is polynomial with respect to $|I|$ and $\max(I)$, where $|I|$ and $\max(I)$ respectively denotes the input length and the largest number appearing in the instance $I \in \Pi$. Hence, the notion of pseudo-polynomial solvable depends on the magnitude of the largest input data involved.

Given any problem instance $I \in \mathbf{Z}$ of an optimization problem to minimize a certain sum or bottleneck objective function with respect to constraint set X, the optimal solution is given by $\gamma(x_0) = \min\{\gamma(x) \mid x \in X\}$, therefore $\Pi(I) = \gamma(x_0)$. However, the range Y must contain elements to represent "unbounded" and "infeasibility", too, in general. A problem Π is called decision problem if $Y = \{yes, no\}$. Each optimization problem has its decision counterpart which is associated by defining an additional threshold value y for the

corresponding objective function . For example, given an additional threshold value y for the objective function we ask: does there is a feasible solution $x \in X$ such that $\gamma(x) \leq y$?

In complexity classes, the set of all decision problems which are polynomial solvable is denoted by P. The class of all decision problems whose all yes instances can be checked for validity in polynomial time, given some additional information called certificate, is denoted by NP (Non-deterministic Polynomial Time).

Similarly, the class of all problems that are the complements of the problems in NP, i. e. for every no instance I there exists a concise certificate for I, which can be checked for validity in polynomial time, is denoted by Co-NP.

We say that a decision problem Π_2 reduces to another decision problem Π_1, denoted by $\Pi_2 \propto \Pi_1$, if there exists a polynomial time transformation function $h: \mathsf{Z}_2 \rightarrow \mathsf{Z}_1$ such that $\Pi_2(I) = yes$ for $I \in \mathsf{Z}_2$ if and only if $\Pi_1(h(I)) = yes$ for $h(I) \in \mathsf{Z}_1$. A decision problem Π_1 is called NP-complete if $\Pi_1 \in NP$ and for any other known decision problem $\Pi_2 \in NP$ we have $\Pi_2 \propto \Pi_1$. Since it follows from $\Pi_2 \propto \Pi_1$ that the problem Π_1 is at least as hard as the problem, Π_2 it is sufficient to consider any known NP-complete problem Π_2 in the complexity hierarchy. The "problem reducibility" relation is a transitive relation on the class of decision problems. A decision problem in NP is called NP-complete in strong sense if it can be solved pseudo-polynomial only if P = NP, which is one of the major open problems in modern mathematics. An optimization problem is called NP-hard if the corresponding decision problem is NP-complete.

Dynamic Programming

Dynamic programming solves problems by combining the solutions to sub-problems. This is a modification of the divide-and-conquers approach. Divide-and-conquer algorithms partition the problem into independent sub-problems, solve the sub-

problem recursively, and then combine their solutions to solve the original problem. In contrast, dynamic programming is application when the sub-problems are not independent, that is, when sub-problems share sub-sub-problems.

A dynamic programming algorithm solves every sub-problem just once and then saves its answer in a table, there by avoiding the work of re-computing the answer every time the sub-problem is encountered.

Dynamic programming is typically applied to optimization problems. In such problems there can be many possible solutions. Each solution has a value, and we wish to find a solution with the optimal (minimum or maximum) value. We call such a solution an optimal solution to the problem, as opposed to the optimal solution, since there may be several solutions that achieve the optimal value.

The development of a dynamic programming algorithm can be broken into a sequence of four steps.

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution in a bottom-up fashion.
4. Construct an optimal solution from computed information.

Steps 1-3 form the basis of a dynamic programming solution to a problem. Step 4 can be omitted if only the value of an optimal solution is required. When we do perform Step 4, we sometimes maintain additional information during the computation in Step 3 to ease the construction of an optimal solution.

Combinatorial Optimization

Some scheduling problem can be solved efficiently by reducing them to well known combinatorial optimization problems like linear programs, maximum flow problem

or transportation problem. Others can be solved by using standard techniques like dynamic programming and branch and bound methods. Here we give a brief sketch of these combinatorial optimization problem and also discuss some of the methods.

Integer Programming

A linear programming refers to an optimization problem in which the objective and the constraints are linear in the variables to be determined. An LP can be expressed as follows:

Minimize $c_1 x_1 + c_2 x_2 + ... + c_n x_n$. (2.1)

Subjected to:

$$\begin{aligned} a_{11} x_1 + a_{12} x_2 + ... + a_{1n} x_n &\le b_1 \\ a_{21} x_1 + a_{22} x_2 + ... + a_{2n} x_n &\le b_2 \\ &\vdots \\ a_{m1} x_1 + a_{m2} x_2 + ... + a_{mn} x_n &\le b_m \end{aligned} \qquad (2.2)$$

$$x_j \ge 0 \quad \text{for} \quad j = 1,2,...n$$

The objective is to minimize the costs. The $c_1, c_2, ..., c_n$ vector is referred to as the cost vector. The variables $x_1, x_2, ...x_n$ have to be determined so that the objective function $c_1 x_1 + c_2 x_2 + ... + c_n x_n$ is minimized. The quantities $a_{1j}, a_{2j}, ...a_{mj}$ defines the activity vector *j*. The $b_1, b_2, ...b_m$ are referred to as the resource.

A nonlinear program (NLP) is a generalization of a linear program that allows the objective function and/or the constraints to be nonlinear in $x_1, x_2, ...x_n$. An integer

program (IP) is a linear program with the additional requirements that the variables $x_1, x_2, ...x_n$ have to be integers.

The linear program (LP) is solvable problem and integer program is NP-hard problem.

Bipartite Matching Problem

A *matching* M of a graph $G = (V, E)$ is a subset of the edges with the property that no two edges of M share the same node. Given a graph $G = (V, E)$, the matching problem is to find a *maximum matching* M of G (see [29]). When the cardinality of a matching is $\left\lfloor \frac{|V|}{2} \right\rfloor$, the largest possible in a graph with $|V|$ nodes, we say that the matching is *complete, or perfect* and the problem of finding a perfect matching M of G is called the *perfect matching problem* (see [29]).

Let us consider a graph $G = (V, E)$ together with a fixed matching M of G. Edges in M are called a *matched* edges; the other edges are *free.* If $[u, v]$ is a matched edge, then u and v are *mate* to each other. Nodes that are not incident upon any matched edges are called *exposed*; the remaining nodes are matched.

Now, consider a bipartite graph $G = (V \cup U, E)$ with $n = |V| \le |U| = m$. For any subset X of vertices, denote by $N(X)$ the neighborhood of X, i.e. the set of all vertices adjacent to at least one vertex in X. Clearly, n is an upper bound for the perfect matching in G. The following theorem due to *Hall* [1935] (see [6, 7]) gives necessary and sufficient conditions for the existence of a matching with cardinality n.

Theorem 2.6.2.1 ([6, 7]) *Let* $G = (V \cup U, E)$ *be a bipartite graph with* $n = |V| \le |U| = m$. *Then there exists in G a matching with cardinality n if and only if*

$$|N(X)| \geq |X| \quad \textit{for all } X \subseteq V .$$

A maximum matching M in a bipartite graph $G = (V \cup U, E)$ can be calculated in $O(\min(|V|, |U|).|E|)$ time (see [29]).

Algorithm 2.6.2.1 The Bipartite Algorithm

Input: *A bipartite graph* $B = (V_1 \cup V_2, E)$,

Output: *The maximum matching of B, represented by the array mate.*

begin

for all $v \in V_1 \cup V_2$ ***do*** *mate[v]=0; (**comment:** initialize)*

stage: ***begin***

for all $v \in V_1$ ***do*** *exposed[v]=0;*

$A = \phi$ *; (**comment:** begin construction of the auxiliary graph* (V, A)*)*

for all $[v,u] \in E$ ***do***

if *mate[u]=0* ***then*** *exposed[v]=u* ***else***

if *mate[u]* $\neq v$ ***then*** $A = A \cup (v,$ *mate[u]);*

$Q = \phi$ *;*

for all $v \in V_1$ ***do if*** *mate[v]* ***then*** $Q = Q \cup \{v\}$, *label[v]=0;*

while $Q \neq \phi$ ***do***

begin

let v be a node in Q;

remove v from Q;

```
        if exposed[v] ≠ 0 then argument(v), go to stage;
        else
            for all unlabeled v' such that (v,v') ∈ A do
            label[v']=v, Q = Q ∪{v'};
        end
    end
end
procedure augment(v)
    if label[v]=0 then mate[v]=exposed[v],
                        mate[exposed[v]]=v;
    else begin
        exposed[label[v]]=mate[v];
        mate[v]=exposed[v];
        mate[exposed[v]]=v;
        augment(label[v])
        end
```

Figure 2: The Bipartite Matching Algorithm

Assignment Problem

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

$\sum_{i \in V_1} c_{i\varphi(i)}$ is minimized.

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

minimize $\sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij}x_{ij}$

Such that:

$\sum_{j=1}^{m} x_{ij} = 1$ i= 1, …., n

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

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

We describe the Hungarian method [29], and use the following terminology and notations:

A label of vertices in a graph $G = (V, A)$ is an array with $|V|$ entries representing the predecessor vertex of all vertices. The label of a vertex $v \in V$ is denoted by *label*[v]. To represent the current matching in the complete bipartite graph $G = (V \cup U, E)$ we use the array mate having 2n entries where *mate[w]* for any vertex $w \in V \cup U$ denotes the vertex w' which is the mate of w. For any $v \in V$ *exposed [v]* is a node of U that is exposed and is adjacent to v; if no such node exists, *exposed*[v] = 0. Now, for $j = 1,...,n$, *slack* [u_j] is the minimum of $(c_{ij} - \alpha_i - \beta_j)$ over all labeled vertices v_i of V and *nhbor*[u_j] is the particular labeled vertex v_i with which slack[v_j] is achieved.

Algorithm 2.6.3.1 [29] The Hungarian method

Input: An $n \times n$, matrix $[c_{ij}]$ of nonnegative integers.

Output: An optimal complete matching (given in terms of the array mate) of the complete bipartite graph $G = (V \cup U, E)$ with $|V| = |U| = n$ under the cost c_{ij} .

begin

for all $v_i \in V$ do $mate[v_i] := 0, \alpha_i = 0$;

for all $u_j \in U$ do mate $[u_j] := 0$, $\beta_j := \min_i \{c_{ij}\}$;

*(**comment** : initialize)*

*for i := 1,…,n **do** (**comment** : repeat for n stages)*

begin

A := Ø;

for all *$v \in V$* ***do*** *exposed[v] := 0;*

for all *$u \in U$* ***do*** *exposed[u] := ∞ ;*

for all *v_i, u_i with $v_i \in V$, $u_j \in U$, and $\alpha_i + \beta_j = c_{ij}$ do*

if *$mate[u_j] = 0$* ***then*** *$exposed[v_i] := u_j$*

else $A := A \cup \{(v_i, mate[u_j])\}$;

*(**comment** : construct the auxiliary graph)*

Q := Ø;

for all *$v_i \in V$ do*

if $mate[v_i] = 0$ then

begin

if *exposed*[v_i] $\neq$ *0* **then** *augment* (v_i), **go to** *endstage;*

$Q := Q \cup \{v_i\}$;

label[v_i] := *0;*

for all $u_k \in U$ *do*

if $0 < c_{ik} - \alpha_i - \beta_k < slack[u_k]$ **then** $slake[u_k] := c_{ik} - \alpha_i - \beta_k$, $nhbor[u_k] := v_i$;

end

search: **while** $Q \neq \emptyset$ *do*

begin

let v_i *be any node in Q;*

remove v_i *from Q;*

for all *unlabeled* $v_j \in V$ *with* $(v_i, v) \in A$ **do**

begin

label[v_j] := v_i;

$Q := Q \cup \{v_j\}$;

if *exposed*[v_j] $\neq$ *0* **then** *augment* (v_j), **go to** *endstage;*

for all $u_k \in U$ **do**

if $0 < c_{jk} - \alpha_j - \beta_k < slack[u_k]$ **then** $slake[u_k] := c_{jk} - \alpha_j - \beta_k$, $nhbor[u_k] := v_j$;

end;

end

modify;

go to *search*

endstage: ***end***

end

procedure *modify*

*(**comment** : it calculate* θ_1*, updates the* α'*s and* β'*s, and activates new nodes to continue the search)*

begin

$\theta_1 := \frac{1}{2} \min_{u \varepsilon U} \{slake[u]>0\};$

for all $v_i \in V$ ***do***

if v_i *is labeled then* $\alpha_i := \alpha_i + \theta_1$ *else* $\alpha_i := \alpha_i - \theta_1$*;*

for all $u_j \in U$ ***do***

if *slack* $[u_j] = 0$ ***then*** $\beta_j := \beta_j - \theta_1$ ***else*** $\beta_j := \beta_j + \theta_1$*;*

for all $u \in U$ *with slack[u]>0 do*

begin

slack[u]:=slack[u]-2θ_1*;*

if *slack[u]=0* ***then*** *(**comment**: new admissible edge)*

if *mate[u]=0* ***then*** *exposed[nhbor[u]]:=, augment(nhbor[u]),* ***go*** *to endstage;*

else *(**comment**: mate[u]* $\neq$ *0)*

label[mate[u]]:=nhbor[u], Q:=Q$\cup${*mate[u]*}*, A:=A*$\cup${*(nhbor[u], mate[u])*}*;*

end

end

procedure *argument (v)*

if *label [v] =0* ***then*** *mate[v]:= exposed [v],*

mate [exposed[v]]:=v;

else begin

exposed[label [v]] := mate[v];

mate[v]:= exposed [v];

mate[exposed[v]]:= v;

augment(label[v])

end

Figure 3: The Hungarian method

Theorem 2.6.3.1 [29] *The* ***Algorithm 2.6.3.1*** *correctly solves the assignment problem for a complete bipartite graph with 2n nodes in* $O\left(n^3\right)$ *time.*

3 SCHEDULING PROBLEMS

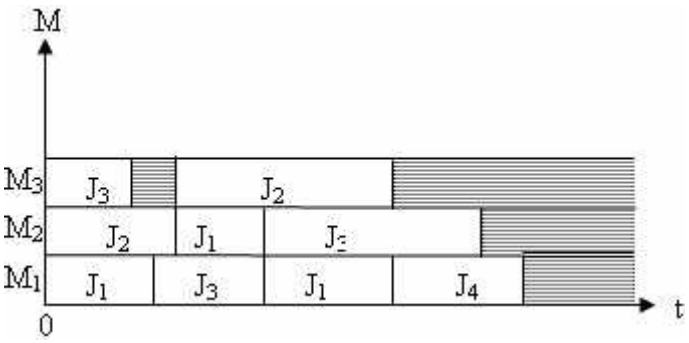
In this chapter, the basic formulations of the scheduling problem are described. The classification of scheduling problems mentioned in this chapter follows the notation used in [7].

A schedule is an allocation of one or more time intervals to each job on one or more machines. A scheduling is called optimal if it minimized a given objective function means to establish an assignment of resources to consumers for a certain period of time in a way that a certain objective is optimized [9]. The policy used to determine this assignment is called scheduling algorithm.

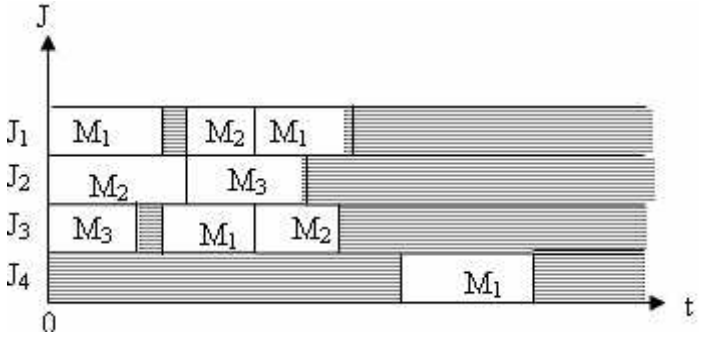
Scheduling theory is excessively used in the computer manufacturing to schedule the jobs. The multiprogramming characteristic of computer is due to the good scheduling of jobs in the CPU because the CPU can only process one job at a time. In this case the objective function is to maximize the CPU utilization.

Schedules and their Representations

Let there be m number of machines, M_i, i = 1, 2….m, which have to process n jobs, J_j, j = 1, 2…n. Besides, there is an objective function which gives the cost of scheduling. The problem is to assign the jobs an allocation of one or more time intervals on one or more machines; such an assignment is called a schedule (see [7]). A schedule is often represented by a Gantt chart. Gantt chart can be machine oriented or Job oriented.



Machine oriented Gantt chart



Job oriented Gantt chart

Figure 4: Gantt chart

H. Brasel introduced a new approach of modeling scheduling problems called block-matrix model in 1990 (see [4, 5]). It is easy, comprehensible and can be applied to simplify the algorithm in this field. In the block-matrices model, all graph theoretical structures of scheduling problems are basically described by means of special kind of matrix called latin rectangles with sequence property (see also [9])

The disjunctive graph is proposed by B. Roy and B. Sussman in 1964 (see [7, 38, 39]) and is used in the literature of scheduling theory. It is based upon the mathematical discipline graph theory. Disjunctive graphs are widely used to represent certain feasible for scheduling problems. The set of feasible schedules, which are represented in this way,

always contains an optimal solution for the problem if the objective is regular (see also [9]).

Three Field Notation

For specifying scheduling problems, three-field notation is popularly used. This notation is due to Graham et al. [14] (also see [1, 7]). In this scheme, a problem is denoted as $\alpha|\beta|\gamma$, where the first field $\alpha = \alpha_1\alpha_2$ describes the machine environment, where α_1 denotes the machine characteristic and α_2 denotes the number of machines used. The single machine environment is described by $\alpha_1 = \circ$, and $\alpha_2 = 1$, i.e. $\alpha = 1$, where $\circ$ denotes the empty symbol. The second field β describes the job characteristic. If we denote preemption, precedence relation, release date (r_i), processing time (p_i) and due date

(d_i), respectively, by $\beta_1, \beta_2, \beta_3, \beta_4$, *and* β_5, then $\beta \in \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5\}$. The third field, , denotes the sum and max objective function. Generally, the objective function will be the completion time C_i of job i. For the fixed parameter d_i as the due dates of job i, lateness, tardiness and earliness of job i are defined, respectively, by

$$L_i = C_i - d_i, \quad T_i = \max\{C_i - d_i, 0\}, \quad E_i = \max\{0, d_i - C_i\},$$

which are usual objective functions to be measured under the JIT environments.

Earliest Due Date (EDD) Algorithm

Whenever a machine is freed, the job with the earliest due date is selected to be processed next. This rule is to minimize the maximum lateness among the jobs waiting for processing. Actually, in a single machine setting, with n- jobs available at time 0, the EDD rule does minimize the maximum lateness.

Example 3.3.1: $1|r_i; d_i; pmtn||L_i|\max$

Is the problem of finding a preemptive schedule on one machine for a set of n-jobs with given release times $r_i \neq 0$ and due dates $d_i \neq 0$ such that the objective function $|L_i|\max$ is minimized.

Benefits of Just-in-Time Production Systems

JIT makes production operation more efficient, cost effective and customer responsive. JIT allows manufacturers to purchase and receive components just before they are needed on the assembly line, thus relieving manufacturers of the cost and burden of housing and managing idle parts.

The main benefits of the JIT manufacturing environment are listed below (see [33, 41]):

1. **Set up times are significantly reduced in the warehouse:** Cutting down the set up time to be more productive will allow the company to improve their bottom line to look more efficient and focus time spent on other areas may need improvement.
2. **The flows of goods from warehouse to shelves are improved:** Having employees focused on specific areas of the system will allow them to process goods faster instead of having them vulnerable to fatigue from doing too many jobs at once and simplifies the tasks at hand.
3. **Employees who possess multiple skills are utilized more efficiently:** Having employees trained to work on different parts of the inventory cycle system will allow companies to use workers in situations where they are needed when there is a shortage of workers and a high demand for a particular product.
4. **Better consistency of scheduling and consistency of employee work hours:** if there is no demand for a product at the time, workers don't have to be working. This can save the company money by not having to pay workers for a job not completed or

could have them focus on other jobs around the warehouse that would not necessarily be done on a normal day.

5. **Increased emphasis on supplier relationships:** No company wants a break in their inventory system that would create a shortage of supplies while not having inventory sit on shelves. Having a trusting supplier relationship means that we can rely on goods being there when we need them in order to satisfy the company and keep the company name in good standing with the public.
6. **Supplies continue around the clock keeping workers productive and business focused on turnover:** Having management focused on meeting deadline will make employees work hard to meet the company goals to see benefits in terms of job satisfaction, promotion or even higher pay.

Applications of Just-in-Time Production System

The following are the applications of JIT:

1. **In real time scheduling:** Real time scheduling problems are principally online versions of Just-in-Time scheduling problems, but popularly, the nomenclature “real time” refers to computer related problems. These types of scheduling problems occur in real-time system. Generally a real-time system is an operating system embedded in some electrical device. In a real-time system, the correct functioning of the system depends on the time when jobs are completed. In a soft-real-time system, early and tardy jobs degrade the quality of the output, while in a hard-real-time system; such jobs make the output invalid. The book of Tanenbaum [40] provides an introduction for real-time scheduling problem in computer system.
2. **Scheduling in operating system:** Scheduling theory is excessively used in computer manufacturing to schedule the jobs in CPU, memory, printing buffer and other devices for processing jobs. The multiprogramming characteristic of computer is due to the good scheduling of jobs in the CPU because the CPU can only process the job at a time. In this case the objective function is to maximize the CPU

utilization (see [40]). Some basic algorithm used in OS (see [23, 40])for uniprocessor computer are:

i. First Come First Serve (FCFS): At any instance when machine is idle, select the available jobs in the order they request. When the first job enters in the system it is started immediately and allowed to run as long as it wants.

ii. Shortest Job First (SJF): At any instance when the machine is idle, select the available job having shortest expected processing time. In the case of tie the FCFS is used.

iii. Shortest Remaining Time Next (SRTN): At any instance schedule the job whose remaining time is the shortest. When a new job arrives, its time is compared with the current process' remaining time. If new job needs less time to finish than the current process, the current process is suspended and new job started. It is applicable to preemptive system.

iv. Round-Robin: Each process is assigned a time interval, called quantum, which it is allowed to run. If the process is still running at the end of the quantum, the CPU is preempted and given to another process. If the process has finished before the quantum has elapsed, the CPU switching is done when the process blocks, of course.

3. **Just-in-Time compilation:** In computing, Just-in-Time, also known as dynamic translation for improving the runtime performance of a computer program. It converts, at runtime, code from one format into another, for example bytecode into native machine code. The performance improvement originates from caching the results of translating blocks of code, not simply evaluating each line or operand separately, or compiling the code at development time. JIT builds upon two earlier ideas in run-time environments: *bytecode compilation and dynamic compilation* (see [37]).

4 MATHEMATICAL MODEL OF JIT PRODUCTION

Output Rate Variation Problem

Consider L different production levels $l; l = 1,2,...,L;$ where level 1 is the final assembly line. For each $l = 1,2,...,L$; let there be n_l different part types with demands $d_{il}; i = 1,2,...,n_l$. Let t_{ilp} denote the total number of units of output i at level l required to produce one unit of product p; $p = 1,2,...,n_1$ so that the dependent demand for part i of level l determined by the final product demands d_{p1} is $d_{il} = \sum_{p=1}^{n_1} t_{ilp} d_{p1}$. We see that $t_{i1p} = 1$ for $i = 1$ and 0 otherwise. For each $l = 1,2,...,L$; let $D_l = \sum_{i=1}^{n_l} d_{il}$ be the total output demand of level l. The demand ratio for part i at level l is $r_{il} = \frac{d_{il}}{D_l}$ for each $i = 1,2,...,n_l$ and we have $\sum_{i=1}^{n_l} r_{il} = 1$ for each $i = 1,2,...,n_l$.

A copy of a product (model) is said to be in stage $k; k = 1,2,...,D_1$ if k units of products have been produced at level1. The product level (level1) has a time horizon of D_1 units and there will be k units of various products p , completely produced, at level1 during the first k stages. Let the cumulative production of part i at level l during the first k stages be denoted by x_{ilk} so that the total quantity of various parts produced at level l during the first k stages is $y_{lk} = \sum_{i=1}^{n_l} x_{ilk}$ units. We have $y_{1k} = \sum_{i=1}^{n_1} x_{i1k} = k$ at level1. In fact, $x_{ilk} = \sum_{p=1}^{n_1} t_{ilp} x_{p1k}$ must hold for $l \geq 2$.

With these notations, the constraints and various objectives for mixed model multi-level JIT assembly systems are formulated as the following [20, 24].

For each $i = 1,2,...,n_l$; let f_{il} be a unimodal symmetric convex function with $f_{il}(0) = 0$, minimum. Then the mixed model multi-level JIT scheduling problem defined by (4.1) is to minimize one of the objectives:

$$G_{\max} = \max_{i,l,k} f_{il}(x_{ilk} - y_{lk} r_{il}), \tag{4.1}$$

and

$$G_{sum} = \sum_{k=1}^{D_1} \sum_{l=1}^{L} \sum_{i=1}^{n_l} f_{il}(x_{ilk} - y_{lk} r_{il}) \tag{4.2}$$

Subject to the constraints

$$x_{ilk} = \sum_{p=1}^{n_1} t_{ilp} x_{p1k}, \qquad i = 1,2,...,n_l;\ l = 1,2,...,L;\ k = 1,2,...,D_1 \tag{4.3}$$

$$y_{lk} = \sum_{i=1}^{n_l} x_{ilk}, \qquad l = 2,...,L;\ k = 1,2,...,D_1 \tag{4.4}$$

$$y_{1k} = \sum_{i=1}^{n_1} x_{i1k} = k, \qquad k = 1,2,...,D_1 \tag{4.5}$$

$$x_{p1k} \geq x_{p1(k-1)}, \qquad p = 1,2,...,n_1;\ k = 1,2,...,D_1 \tag{4.6}$$

$$x_{p1D_1} = d_{p1},\ x_{p10} = 0,\ p = 1,2,...,n_1 \tag{4.7}$$

$$x_{ilk} \geq 0, \text{ integer}, \qquad i = 1,2,...,n_l;\ l = 1,2,...,L;\ k = 1,2,...,D_1. \tag{4.8}$$

Constraint (4.3) indicates that the necessary cumulative production of part i of level l by the end of stage k is determined explicitly by the quantity of products produced at product level. Constraints (4.4) and (4.5) compute the total cumulative production at level l and level1, respectively, during the first k stages. Constraint (4.6) shows that the total production of every product over k stages is a non-decreasing function of k. Constraint (4.7) ensures that the production requirements for each product are met

exactly. Constraints (4.5), (4.6) and (4.8) indicate that exactly one unit of a product is to be produced in the product level during each stage.

The deviation between actual and ideal production can be visualized as in Figure 5.

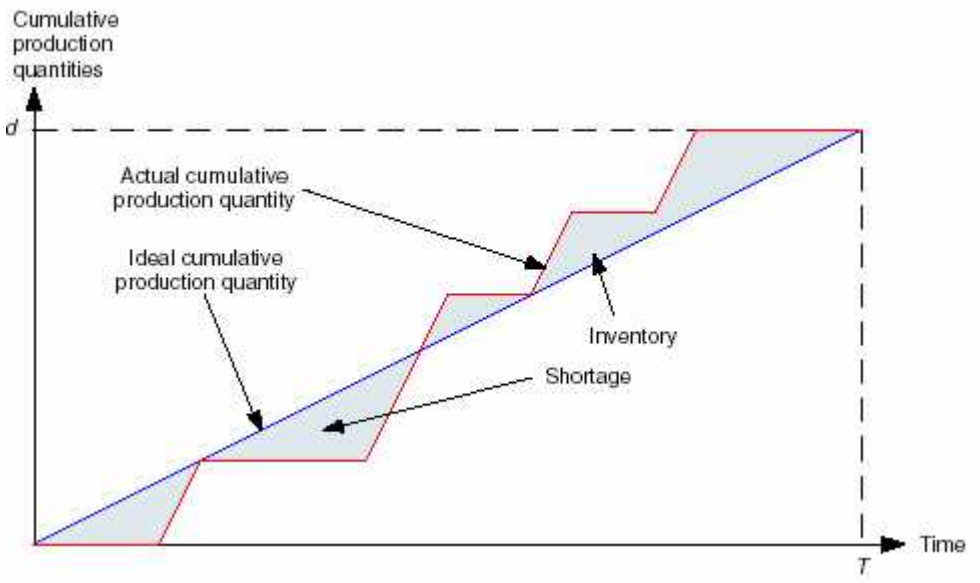


Figure 5: The ideal and actual commutative production quantities

In particular, taking $f_{il}(x) = |x|$ (3.1) and (3.2) take the forms:

$$G'_{\max} = \max_{i,l,k} |x_{ilk} - y_{lk}r_{il}| \tag{4.9}$$

and $$G'_{sum} = \sum_{k=1}^{D_1} \sum_{l=1}^{L} \sum_{i=1}^{n_l} |x_{ilk} - y_{lk}r_{il}|, \text{ respectively.} \tag{4.10}$$

Also, taking $f_{il}(x) = x^2$ (3.7) and (3.8) take the forms:

$$G''_{\max} = \max_{i,l,k} (x_{ilk} - y_{lk}r_{il})^2 \tag{4.11}$$

and $G''_{sum} = \sum_{k=1}^{D_1} \sum_{l=1}^{L} \sum_{i=1}^{n_l} (x_{ilk} - y_{lk} r_{il})^2$, respectively. (4.12)

The multi-level problem is a difficult optimization problem; however, various heuristic solution procedures of finding good solution with reasonable computational effort are desired, in literature, to solve real-life instances of the problem. Here, we focus the following particular case.

Product Rate Variation Problem

Mixed model single-level JIT assembly system, a particular case of multi-level system with only one level, the product level; assumes that different products (or models) require the same number and mix of components and that the processes have negligible switch over costs from one product to another and so allow for diversified small–lot production. In this section the mathematical formulation of the single-level system are discussed. Assume that there are *n* products (or models) to be produced during a specified planning time horizon with demands $d_1, d_2, ..., d_n$ for all $i = 1,2,...,n$. Put $D = \sum_{i=1}^{n} d_i$ and the time horizon be divided into D time units (i.e. an implied time horizon of D time units can be inferred), where one copy of a product will be produced in each time period. A schedule is called *an ideal schedule* if at each time period k; $k = 1,2,...,D$ the line has been assembled $k \frac{d_i}{D}$ parts of product i; $i = 1,2,...,n$. The aim of JIT sequence is to keep the real production of a product i in each time unit k as close as possible to the ideal production rate $r_i = \frac{d_i}{D}$. Let $x_{i,k}$ denote the real cumulative production of product i in time periods 1 to k, inclusive.

The most important goal of JIT production system is to keep the schedule as balanced as possible. Thus, our objective is to schedule the assembly line so that the proportion of each product i produced over a time period to the total production is as close to r_i as

possible. In other words, this model aims to hold inventory and shortage costs as smooth as possible by keeping the production rate of each product as balanced as possible by keeping the quantity of each product used by assembly line as constant as possible. The objectives formulated in [24] and generalized in [20] are given as follows.

For each $i = 1,2,...,n$; let f_i be a unimodal symmetric convex function with $f_i(x) > 0$ for $x \neq 0$; $f_i(0) = 0$, minimum. The problem defined by (4.2) is to minimize one of the objectives:

$$F_{\max} = \max_{1 \leq i \leq n} f_i(x_{i,k} - kr_i), \tag{4.13}$$

$$F_{sum} = \sum_{k=1}^{D} \sum_{i=1}^{n} f_i(x_{i,k} - kr_i) \tag{4.14}$$

Subject to the constraints:

$$\sum_{i=1}^{n} x_{i,k} = k, \qquad k = 1,2,...,D \tag{4.15}$$

$$x_{i,D} = d_i, \qquad i = 1,2,...,n \tag{4.16}$$

$$x_{i,0} = 0, \qquad i = 1,2,...,n \tag{4.17}$$

$$x_{i,k} - x_{i,k-1} \geq 0, \qquad i = 1,2,...,n \;\; ; \;\; k = 1,2,...,D \tag{4.18}$$

$$x_{i,k} \geq 0, \text{ integer}, \qquad i = 1,2,...,n \;\; ; \;\; k = 1,2,...,D. \tag{4.19}$$

Here, equality (4.15) means that k parts (copies) have to be produced in first k time periods ; equality (4.16) indicates that all demands must be fulfilled within D time periods; inequality (4.18) shows that a produced copy cannot be destroyed (i.e. for each i, the number of produced copies of i cannot decrease with time).

Here, $F_{\max}$ seeks to minimize the deviations for each product and hence the maximum deviation, whereas F_{sum} objective is to find the lowest possible total deviation. More specifically we consider the following cases:

Case (1): If $f_i(x) = |x|$ for all $i = 1,2,\ldots,n$. Under this case, (4.13) takes the form:

$$F^a_{\max} = \max_{1\le i\le n;1\le k\le D} |\, x_{i,k} - kr_i \,| \tag{4.20}$$

and (4.14) takes the form:

$$F^a_{sum} = \sum_{k=1}^{D}\sum_{i=1}^{n} |\, x_{i,k} - kr_i \,| \tag{4.21}$$

Case (2): If $f_i(x) = x^2$ under which (4.18) takes the form:

$$F^s_{\max} = \max_{1\le i\le n;1\le k\le D} (x_{i,k} - kr_i)^2 \tag{4.22}$$

and (4.19) takes the form:

$$F^s_{sum} = \sum_{k=1}^{D}\sum_{i=1}^{n} (x_{i,k} - kr_i)^2 \tag{4.23}$$

For simplicity, we introduce the abbreviation that $problem\, F^a_{\max}$ means the problem defined by (4.2) with objective function (4.20) under the constraints (4.15), (4.16), (4.17), (4.18) and (4.19); $problem\, F^a_{sum}$ means the problem defined by (4.2) with objective function (4.21) under the constraints (4.15), (4.16), (4.17), (4.18) and (4.19); etc. We find the pseudo-polynomial algorithms separately for the problem $F^a_{\max}, F^s_{\max}, F_{sum}$ in [35, 6]

The mixed-model maximum deviation and sum deviation JIT sequencing problems are, respectively, denoted by maximum deviation just-in-time (MDJIT) and sum deviation just-in-time (SDJIT) problems (see [6, 11]). Similarly the abbreviated form

MMJIT refers to mixed-model Just-in-Time and MMJITSP refers to MMJIT sequencing problem. The bottleneck objective functions seek to find smooth sequences at each stage and as a result it precludes the possibility of relatively large deviations in every time period. In contrast the min-sum objective functions are concerned for finding the smooth sequences on the average which may result in relatively large deviations in certain time periods

Pegged ORV Problem

Steiner and Yeomans [36] shows that the ORV problem under the pegging assumption can be reduced to weighted PRV problem. Under the pegging assumption, parts of output i at production levels which fed the level 1 are dedicated or pegged to the specific final product into which they will be assembled. This assumption decomposes the lower level parts that will be assembled into different level 1 products into disjoint sets. As a result, a distinction is made between t_{ilh} and t_{ilp}, $h \neq p$ for each part i at level l. With this assumption the multi level min-sum JIT sequencing problem can be reduced to a weighted single level problem (cf. [44], also see [13, 11]). Similarly with the same assumption the weighted max-abs ORV problem can be formulated as (cf. [36], also see [11]).

$$G_{\max}^{peg} = \max_{p,i,l,k} \left\{ W_{p1} \left| x_{p1k} - kr_{p1} \right|, W_{il} \left| x_{p1k}\, t_{ilp} - kt_{ilp}\, r_{p1} \right| \right\}$$
$$= \max_{p,i,l,k} \left\{ W_{il}\, t_{ilp} \left| x_{plk} - kr_{p1} \right| \right\}$$

$p = 1,...,n_1;\ i = 1,...,n_l;\ k = 1,...,D_1;\ l = 1,...,L$. Now letting $w_{p1} = \max_{i,l} \left\{ w_{il}\, t_{ilp} \right\}$ the objective function reduced to $G_{\max}^{peg} = \max_{p,i,l,k} \left\{ w_{p1} \left| x_{p1k} - kr_{p1} \right| \right\}$. Now dropping out the superfluous subscript 1 the problem is reduced to the weighted PRV problem.

$$\min \left[G_{\max}^{peg} = \max_{i,k} \left\{ w_i \left| x_{ik} - kr_i \right| \right\} \right], i = 1,...,n;\ k = 1,...,D$$

5 SOLUTION PROCEDURE FOR PRV PROBLEM

Cyclic Sequences

An instance of PRV-JIT problem is said to be standard if $0 < d_1 \le d_2 \le ... \le d_n$, $n > 1$and the greatest common divisor of $d_1,...,d_n, D$ is 1, i.e., $\gcd(d_1,...,d_n, D) = 1$.

Let β be the optimal solution to the mixed model Just-in-Time production system for the standard demand vector $(d_1, d_2,..., d_n)$. Now if the concatenation of m copies of β for any $m \ge 1$is an optimal solution to the mixed model Just-in-Time production system with demand vector $(md_1, md_2,..., md_n)$, then the solution β_m is called the cyclic solution with a cycle β and number of cycle m. The existence of cyclic sequences significantly reduces the computational complexity. In fact the optimal sequence to original problem can be obtained by first calculating the gcd (Greatest Common Divisor) m of $d_1, d_2,..., d_n$ then by obtaining the optimal sequence for the demand vector $\frac{d_1}{m}, \frac{d_2}{m},..., \frac{d_n}{m}$ and finally by concatenating the sequence m times to construct an optimal sequence for the original demand vector $(d_1, d_2,..., d_n)$. The existence of optimal cyclic sequence for the problem F^s_{sum} with $n = 3$, and for demand vector *(600, 600, 100)* is presented in [24] for the first time. Consequently Kubiak [17] proves this concept analytically. We sketch the proof here.

Theorem 5.1.1: [17] *Let* $\sigma = \sigma_1, ..., \sigma_D, \sigma_{D+1}, ..., \sigma_{2D}$, *be a feasible sequence for* $2d_1, ..., 2d_n$. *Then, a sequence* $\mu = \mu_1, ..., \mu_D, \mu_{D+1}, ..., \mu_{2D}$, *where i occurs* d_i *times in the second half* $\mu_{D+1}, ..., \mu_{2D}$ *can be constructed such that* $F_{sum}(\mu) \le F_{sum}(\sigma)$.

Theorem 5.1.2: [17] *Let* β *be an optimal sequence for the problem* F_{sum} *with the demands* $d_1, d_2, \ldots, d_n$. *Then* $\beta^m, m \geq 1$, *is optimal for the demands* $md_1, md_2, \ldots, md_n$

Proof: The theorem obviously holds for $m = 1$. Suppose, by the induction, that the theorem holds for any m, $1 \leq m \leq k$. We prove that it also holds for $m = k+1$. Consider an optimal sequence $\sigma_1, \ldots, \sigma_{mD}$ for $md_1, md_2, \ldots, md_n$. If m is even, then by Theorem , this sequence can be transformed without cost increasing into a sequence $\mu_1, \ldots, \mu_{m\frac{D}{2}}, \mu_{1+m\frac{D}{2}}, \ldots, \mu_{mD}$, where i occurs $\frac{md_i}{2}$ times in each of the two halves of . Thus, each half must be optimal for $\frac{md_1}{2}, \ldots, \frac{md_n}{2}$. Therefore, by the inductive assumption, each half is the concatenation of $\frac{m}{2}$ copies of β and the theorem holds for even $m = k+1$. If m is odd, then consider sequence $\beta\sigma$ for $(m+1)d_1, \ldots, (m+1)d_n$. We have $F_{sum}(\beta\sigma) = F_{sum}(\beta) + F_{sum}(\sigma)$. By Theorem 5.1.1, $\beta\sigma$ can be transformed without cost increasing into a sequence $\mu_1, \ldots, \mu_{(m+1)\frac{D}{2}}, \mu_{1+(m+1)\frac{D}{2}}, \ldots, \mu_{(m+1)D}$ where i occurs $(m+1)\frac{d_i}{2}$ times in each of the two halves of . Thus, each half must be optimal for $(m+1)\frac{d_1}{2}, \ldots, (m+1)\frac{d_n}{2}$. Therefore, by the inductive assumption, each half is the concatenation of $\frac{(m+1)}{2}$ copies of β, and

$F_{sum}(\beta\sigma) = F_{sum}(\beta) + F_{sum}(\sigma) \geq (m+1)F_{sum}(\beta)$. Consequently, $F_{sum}(\sigma) \geq mF_{sum}(\beta)$ this proves the theorem for odd $m = k+1$. This completes the proof.

Clearly this idea reduces computational requirements considerably when the greatest common divisor $m > 1$.

The existence of cyclic solutions to the max-abs PRVP is established by Steiner and Yeomans [36] for the first time both for weighted and un-weighted PRVPs. This

concept is generalized by Dhamala et al. [12] to established that the optimal sequences to the max-sqr PRVP are cyclic both for weighted and un-weighted cases. Now as $m = \gcd(d_1,...,d_n)$, then the demand requirement vector becomes $d = (d_1,...,d_n) = (mm_1,...,mm_n)$ with $d_i = mm_i, \forall\, i = 1,...,n$.

Letting $A = \sum_{i=1}^{n} m_i$, then $D = \sum_{i=1}^{n} d_i = mA$ and $r_i = \frac{d_i}{D} = \frac{mm_i}{mA} = \frac{m_i}{A}$.

Let each copy of the product be labeled as $(k-1)m_i + c$, where $k = 1,...,m$ *and* $c = 1,...,m_i$. Then for each fixed value of k, there will be a group of m_i copies of product i in the range

$$[(k-1)m_i + 1...(k-1)m_i + m_i].$$

This range will be referred to as the k^{th} tier of copies for product i (cf. [36]).

Lemma 5.1.1 [12, 36] *For a threshold value* $B < 1$, *we have the linear relations*

$$E(i, km_i + c) = E(i,c) + kA \text{ and } L(i, km_i + c) = L(i,c) + kA.$$

Proof: We give the proof only for the case $F^{a}_{\max}$ of *Steiner and Yeomans* [36] and the proof for the case $F^{s}_{\max}$ can be obtained similarly from [10].

$$E(i, km_i + c) = \left\lceil \frac{km_i + c - B}{r_i} - 1 \right\rceil$$

$$= \left\lceil \frac{(k-1)m_i + c - B + m_i}{r_i} - 1 \right\rceil$$

$$= \left\lceil \frac{(k-1)m_i + c - B}{r_i} - 1 + A \right\rceil$$

$$= \left\lceil \frac{(k-1)m_i + c - B}{r_i} - 1 \right\rceil + A$$

$$= E(i,(k-1)m_i + c) + A$$

$$= E(i,c) + kA.$$

Similarly, it can be proved that $L(i, km_i + c) = L(i,c) + kA$

The Lemma 5.1.1 implies that only the early and late producing times for copies $c = 1,...,m_i$ in the first tier need to be calculated, as the produce times for all copies in the remaining tiers are linear function of those in the first tier.

Lemma 5.1.2 [36] *For bottleneck* $B < 1$, *then for all* $i = 1,...,n$ *and* $k = 1,...,m$, *we have*

(a) $L(i, km_i) \le kA$

(b) $(k-1)A < E(i,(k-1)m_i + 1)$

(c) $[E(i,(k-1)m_i + 1)...L(i, km_i)] \cap [E(i, km_i + 1)...L(i,(k+1)m_i)] = \emptyset,$

where $[a...b]$ denotes the set of all integers between *a* and *b* including both.

Now, we state the main result from [36]

Theorem 5.1.3 *Let* $m = \gcd(d_1,...,d_n)$. *Then the problem*

(a) $F^{a}_{\max}$ *has an optimal sequence which consists of m repetitions of the optimal sequence to the sub-problem where* $d = (m_1,...,m_n)$.

(b) $F^{s}_{\max}$ *has an optimal sequence which consists of m repetitions of the optimal sequence to the sub-problem where* $d = (m_1,...,m_n)$.

Proof:

(a) Consider any optimal sequence $s = (s_1,...,s_D)$ to the problem $F^{a}_{\max}$ with objective value $B^* < 1$. Such a solution always exists as the optimal value for max-abs is strictly less than one. If *s* itself is an *m* repetitions of the optimal sequence to the sub-problem where $d = (m_1,...,m_n)$, then there is nothing to prove further. Else, a copy (i, j) of product *i* occupies a position in the interval $[E(i, j)...L(i, j)]$. Moreover, Lemma 5. 1.1 implies that each interval $[(k-1)A...kA-1]$ consists of *m* units of products and hence by Lemma 5.1.2(c) we can rearrange the products sub-sequence on each of the interval $[(k-1)A...kA-1]$, for $k = 2,...,m$ as in the first interval $[0..A-1]$ without destroying the B^* feasibility of the sequence.

(b) The proof directly follows from Theorem 5.1.3 and (a).

It is also noted that the cyclic sequence analogously exists for the weighted problem with appropriate weights (see [10, 36])

Earliest Due Date Algorithms

In this section we describe a graphic approach for solving the max-abs problem [35]. In this procedure, decision version of the problem with certain target value for objective as a threshold value, is reduced to a perfect matching problem in a bipartite graph. Then Glover's modified EDD rule is used for the matching problem to decide whether the

decision problem has 'yes' answer. Then an optimal is obtained by using the matching problem and bisection search within the bounds for target value after determination of the bounds. The procedure in detail is presented in [35]

Release Date/Due Date Decision Problem

For n products with demands d_i ; $i = 1,2,...,n$; d_i being positive integers ; consider the problem $F_{\max}^{a}$

Let B be a target value for the objective function; (i, j) denote the j^{th} copy of product i. Then the earliest starting time $E(i, j)$ for (i, j) must be the unique integer satisfying $\frac{j-B}{r_i} - 1 \le E(i, j) < \frac{j-B}{r_i}$ and latest starting time $L(i, j)$ of (i, j) must be the unique integer satisfying

$\frac{j-1+B}{r_i} - 1 < L(i, j) \le \frac{j-1+B}{r_i}$. This provides the formulae:

$$E(i, j) = \left\lceil \frac{j-B}{r_i} - 1 \right\rceil \tag{5.1}$$

$$\text{and } L(i, j) = \left\lfloor \frac{j-1+B}{r_i} \right\rfloor \tag{5.2}$$

For a given B, we can determine $E(i, j)$ and $L(i, j)$ for all i and for all j in $O(D)$ time.

Consider the decision problem defined by (5.1): "Does there exist a $n \times D$ matrix $(x_{i,k})$ with $\max_{1 \le i \le n; 1 \le k \le D} | x_{i,k} - kr_i | \le B$ satisfying all constraints of (4.15)-(4.19)?" This problem can be viewed as the problem of determining whether there is a feasible schedule of D unit-time jobs on a single machine with release dates and due dates for each job. So the

decision problem can be represented as a matching problem in a bipartite graph as in the next subsection.

Perfect Matching Problem and EDD Rule

For a given target value B as threshold value for decision problem, determine $E(i, j)$ and $L(i, j)$ for all i and for all j according to (5.1) and (5.2). Define the bipartite graph $G = (V_1 \cup V_2, E)$;

Where, $V_1 = \{0,1,2,..., D-1\}$, $V_2 = \{(i, j) \mid i = 1,2,...,n; j = 1,2,..., d_i\}$ and $(k,(i, j)) \in E$ if and only if $k \in [E(i, j), L(i, j)]$ i.e. if and only if (i, j) may start at time k. Then the bipartite graph G is $V_1 - convex$. Here finding a feasible sequence for problem (P5.1) is analogous to finding a perfect matching in G such that lower numbered copies of a product are matched to earlier starting times than higher numbered copies. Such a matching is called *order preserving*.

Example: 5.2.1 For Demand vector (300, 600, 900), the schedule so constructed by the EDD algorithm with considering the upper bound $1 - \frac{1}{D}$ is presented in Table 1. This consists of 300 cycles of the sequence 3-2-3-1-2-3.

Product	Unit	Due Date
1	1	5.0
2	1	3.0
--	2	6.0
3	1	2.0
--	2	4.0
--	3	6.0

Schedule List :
3 - 2 - 3 - 1 - 2 - 3 -
No of Cycle :300

Table 1: Schedule generated by EDD for max-abs

5.2.3 EDD for min-sum-sqr

Inman and Bulfin [15] define the ideal position for copy (i, j) as

$$k_{ij} = \frac{2j-1}{2r_i} = \left\lceil \frac{\left(j - \frac{1}{2}\right)D}{d_i} \right\rceil$$

Let Z_{ij} denotes the time at which copy (i, j) actually produced. Then Inman and Bulfin [15] consider the following problem:

$$\text{Minimize} \quad \sum_{i=1}^{n}\sum_{j=1}^{d_i}\left(Z_{ij} - k_{ij}\right)^2 \tag{5.3}$$

Subject to

$$Z_{ij} \le Z_{i(j+1)}, \qquad i = 1,...,n; \quad j = 1,...,d_i - 1 \tag{5.4}$$

$$1 \le Z_{ij} \le D, \qquad i = 1,...,n; \quad j = 1,...,d_i \tag{5.5}$$

$$Z_{ij} \ne Z_{i'j'}, \qquad (i, j) \ne (i', j') \tag{5.6}$$

$$Z_{ij} \in \mathrm{W}, \qquad i = 1,...,d_i \tag{5.7}$$

Constraint (5.4) ensures that the production time of each copy of a product type *i* is a strictly increasing function of each copy j. Constraint (5.5) guarantees that the production time of any copy of any product lies in the internal [1...D]. Constraint (5.6) is the only linking constraint and is not in the standard integer programming format and it specifies that only one copy of any product type can be produced in each period. By defining k_{ij} as the due-date of copy (i, j) where each copy of product is treated as a

separate job, *Inman and Bulfin* [15] observe that problem defined by (5.3) may be interpreted as a single machine scheduling problem

$$1\left| p_{(i,j)} = 1 \right| \sum_{(i,j)\in I} (E_{(i,j)} + T_{(i,j)}), \qquad (5.8)$$

where $p_{(i,j)}$, $E_{(i,j)}$ and $T_{(i,j)}$ respectively represents the processing time, earliness and tardiness of copy (i, j) and $I = \{(i, j) | \, i = 1,...,n; \; j = 1,...,d_i\}$. And in conclusion they suggest the following.

Theorem 5.2.1 *The optimal sequence for problem defined by (5.3) is to order the copies following the EDD rule for the problem (5.8).*

The EDD procedure can run in $O(nD)$ time and the EDD rule also gives an optimal sequence for the sum of absolute deviations as well (see [15, 18]).

Example: 5.2.2 For Demand vector (300, 600, 900), the schedule so constructed by the EDD algorithm with considering the due date $\left\lceil \frac{\left(j - \frac{1}{2}\right)D}{d_i} \right\rceil$ for the j[th] copy of product *i* of min-sum-sqr is presented in Table 2: Schedule generated by EDD for min-sum This consists of 300 cycles of the sequence 3-2-1-3-2-3.

Product	Unit	Due Date
1	1	3.0
2	1	1.5
--	2	4.5
3	1	1.0
--	2	3.0
--	3	5.0

Schedule List :
3 - 2 - 1 - 3 - 2 - 3 -
No of Cycle :300

Table 2: Schedule generated by EDD for min-sum

Nearest Integer Point Problem

This algorithm obtains the optimal solution for problem defined by (4.23) with finding the 'nearest' integer point $M = (m_1,..., m_n) \in \mathrm{W}^n$ to a point $X = (x_1,....x_n) \in \mathrm{R}^n$ where

$$\sum_{i=1}^{n} m_i = \sum_{i=1}^{n} x_i = k .$$

Step1: Calculate $\sum_{i=1}^{n} x_i = k$.

Step2: Find the nearest nonnegative integer m_i to each co-ordinate xi. That is, find m_i, such that $|m_i - x_i| \leq \frac{1}{2}, \qquad i = 1,2,..., n$.

Step3: Calculate $k_m = \sum_{i=1}^{n} m_i$

Step4: i) If $k - k_m = 0$ stop. The nearest integer point is $M = (m_1, m_2..., m_n)$.

ii) If $k - k_m > 0$ go to Step 5.

iii) If $k - k_m < 0$ go to Step 6.

Step5: Find x_i with the smallest $m_i - x_i$. Put $m_i = m_i + 1$, Go to Step 3.

Step6: Find x_i with the largest $m_i - x_i$.

Put $m_i = m_i - 1$. Go to Step 3.

Miltenburg [24] has proven that this algorithm provides an optimal solution to the problem defined by (4.23) with neglecting the Constraints (4.16), (4.17), and (4.18) and clearly this optimal solution also satisfy the Constraint (4.16) and (4.17) of problem defined by (4.23). But unfortunately, solution given by this algorithm generally may- not satisfy the order Constraint (4.18). Hence, this algorithm will not give, in general, the feasible solution for product rate variation Just-in-Time problems with sum-square deviation objective function.

Example 5.3.1 Suppose there are n=4 products with the demands vector D = (2, 3, 5, 7) to be produced on a mixed model JIT production system under sum-sqr objective. The vector of demand ratio is r = (2/17, 3/17, 5/17.7/17). Now the schedule given by nearest integer point is shown in Table 3. In this table there is no possibility of destroying any products at any stage and the so obtained schedule is optimal to the problem defined by (4.23).

Stage (s	X[0]	X[1]	X[2]	X[3]	M[0]	M[1]	M[2]	M[3]	Product Schedule	sum((m[j][k]-s[j][k])^2)	Total Variation
1	0.17647	0.176...	0.29412	0.41176	0	0	0	1	+4	0.47751	0.47751
2	0.23529	0.352...	0.50024	0.02353	0	0	1	1	+3	0.30032	0.05013
3	0.35294	0.529...	0.88235	1.23529	0	1	1	1	+2	0.41522	1.27536
4	0.47059	0.705...	1.17647	1.64706	0	1	1	2	+4	0.46337	1.73702
5	0.58824	0.882...	1.47059	2.05882	1	1	1	2	+1	0.4083	2.14533
6	0.70588	1.058...	1.76471	2.47059	1	1	2	2	+3	0.36678	2.51211
7	0.82353	1.235...	2.05882	2.88235	1	1	2	3	+4	0.10331	2.61592
8	0.94118	1.411...	2.35294	3.29412	1	2	2	3	+2	0.56055	3.17647
9	1.05882	1.588...	2.64706	3.70588	1	2	2	4	+4	0.6782	3.85467
10	1.17647	1.764...	2.94118	4.11765	1	2	3	4	+3	0.10331	3.95798
11	1.29412	1.941...	3.23529	4.52941	1	2	3	5	+4	0.36678	4.32026
12	1.41176	2.117...	3.52941	4.94118	1	2	4	5	+3	0.4083	4.73356
13	1.52941	2.294...	3.82353	5.35294	2	2	4	5	+1	0.46337	5.19693
14	1.64706	2.470...	4.11765	5.76471	2	2	4	6	+4	0.41522	5.61216
15	1.76471	2.647...	4.41176	6.17647	2	3	4	6	+2	0.38032	5.99248
16	1.88235	2.823...	4.70588	6.58824	2	3	5	6	+3	0.47751	6.47059
17	2.0	3.0	5.0	7.0	2	3	5	7	+4	0.0	6.47059

Table 3**:** Schedule generated for demand vector D = (2, 3, 5, 7) by nearest integer point

Example 5.3.2 The schedule constructed by the nearest integer point corresponding to the JIT scheduling problem with demand vector D = (2, 3, 5, 1) is presented in Table 4 which does not produce the optimal schedule as the product 4 must be destroyed at stage 6.

Stage...	X[0]	X[1]	X[2]	X[3]	M[0]	M[1]	M[2]	M[3]	Product Schedule	sum((m[j][k]-x[j][k])...	Total Variation
1	0.18...	0.27...	0.454...	0.090...	0	0	1	0	-3	0.47322	0.41322
2	0.36...	0.54...	0.909...	0.181...	0	1	1	0	-2	0.38017	0.79339
3	0.54...	0.81...	1.363...	0.272...	1	1	1	0	-1	0.44628	1.23967
4	0.72...	1.09...	1.818...	0.363...	1	1	2	0	-3	0.24793	1.4876
5	0.90...	1.36...	2.272...	0.454...	1	1	2	1	4	0.5124	2.0
6	1.09...	1.63...	2.727...	0.545...	1	2	3	0	-2 +3 -4	0.5124	2.5124
7	1.27...	1.90...	3.181...	0.636...	1	2	3	1	-4	0.24793	2.76033
8	1.45...	2.18...	3.636...	0.727...	1	2	4	1	-3	0.44628	3.20661
9	1.63...	2.45...	4.090...	0.818...	2	2	4	1	-1	0.38017	3.58678

Table 4**:** Schedule generated for demand vector D = (2, 3, 5, 1) by nearest integer point

The solution to problem defined by (4.23) neglecting the Constraints (4.16), (4.17), (4.18) obtained by algorithm1 is a level or balanced schedule for the mixed-model single-level JIT assembly systems. However, the schedule may not be feasible. Therefore, the following algorithm is proposed by Miltenburg [24] which ensures that a feasible

schedule is found.

Step1: Solve problem defined by (4.23) without Constraints (4.16), (4.17), (4.18) using nearest integer point, and determine whether the schedule is feasible. The schedule will be feasible if $m_{i,k} - m_{i,k-1} \geq 0$ for all i and k. If the schedule is feasible, stop. This is the optimal schedule. Otherwise go to Step 2.

Step2: For the infeasible schedule determined in Step 3 find the first (or next) stage l where $m_{i,l} - m_{i,l-1} < 0$ .Set ∂ = number of products i, for which $m_{i,l} - m_{i,l-1} < 0$. Reschedule stage l-∂, l-∂+1, ...,l +1 by considering all possible sequences that began with schedule for stage l -∂-1, and end with the schedule for stage l +1.

Step3: Repeat Step 2 for other stages where $m_{i,l} - m_{i,l-1} < 0$. Then stop.

In general there are $\frac{n!}{(n-\partial-2)!}$ possible sequences each of length ∂ +2 to consider, for each infeasibility. And so to determine the best one, i.e. the minimum variation among all possible sequences the total enumeration is needed and consequently the time complexity of this algorithm will not be polynomial in the input size of the instance. Therefore, practically this algorithm works for those instances with small number of input size (products with similar part requirements) and it will not perform efficiently for those problems with large number of input size not for problems where products have different part requirements. As a result to perform large sized problems efficiently, Miltenburg [24] proposed other scheduling algorithms. Computationally, the following algorithm is faster, and can be used for large problems obtaining a feasible schedule for the mixed model JIT assembly systems.

Step1: Solve problem defined by (4.23) with out Constraints (4.16), (4.17), (4.18), using nearest integer point, and determine whether the schedule is feasible. It is feasible if $m_{i,k}$-$m_{i,l-1} \geq 0$ for all i and k. If the schedule is feasible, stop. This is optimal

schedule. Otherwise go to step 2.

Step2: 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 ∂ = number of product i, for which $m_{i,l}$-$m_{i,l-1}$< 0, and beginning at stage $l - \partial$ use heuristic nearest integer point to schedule stages $l - \partial$, $l-\partial+1,\ldots, l+$; where ≥ 0. $l+$ is the first stage where the schedule determined by the heuristic matches the schedule determined in Step 1.

Step3: Repeat Step 2 for other stages where $m_{i,k}$-$m_{i,k-1}$ < 0. Then stop.

Heuristic nearest integer point

For a stage *k*, schedule the product *i* with the lowest $x_{i,k-1}$-kr_i.

This is a myopic heuristic in that it does not consider the effect on future stages of its current decision. Its great advantage is that it is one-pass algorithm. It does one calculation for each product and then makes a selection. For each stage the computational complexity is $O(n)$ since n comparison should be made in each stage (period). This is found to be satisfactory algorithm. Because of the myopic nature of this heuristic the following two pass heuristic of complexity $O(n^2)$ for each stage was developed by Miltenburg [24]

Step1: Set h=1

Step2: Tentatively schedule product h to be produced in stage k. calculates the variation for stage k and calls it $V1_h$.

Step3: Schedule the product i with lowest $x_{i,x}$-(k+1) r_i for stage k+1. Notice that this is the decision rule of heuristic. Calculate the variation for stage k+1 & call it $V2_h$. Calculate $V_h = V1_h+V2_h$.

Step4: Put h = h+1. If h>n go to Step 5, otherwise go to Step 2, where n is the number of products.

Step5: Schedule the product h with the lowest V_h.

It is observed that this heuristic bases its scheduling decision on two stages- the current stage and the next stage. It approximates the variability over these two stages & schedules so that this variability is as small as possible.

Example 5.3.3 For the demand vector (2000, 3000, 5000, 1000) the corresponding data are presented in Table 5**:** Schedule generated for demand vector D = (2000, 3000, 5000, 1000) using heuristic, in which the sequence is the 1000 repetition of the cycle 3-2-1-3-4-3-2-3-1-2-3.

Stage (k)	X[0]	X[1]	X[2]	X[3]	M[0]	M[1]	M[2]	M[3]	Product Schedule	sum((mj[k]-xj[k])^2)	Total Variation
1	0.18182	0.27273	0.45455	0.09091	0	0	1	0	-3	0.41322	0.41322
2	0.36364	0.54545	0.90909	0.18182	0	1	1	0	-2	0.38017	0.79339
3	0.54545	0.81818	1.36364	0.27273	1	1	1	0	-1	0.44628	1.23967
4	0.72727	1.09091	1.81818	0.36364	1	1	2	0	-3	0.24793	1.4876
5	0.90909	1.36364	2.27273	0.45455	1	1	2	1	-4	0.5124	2.0
6	1.09091	1.63636	2.72727	0.54545	1	1	3	1	-3	0.69421	2.69421
7	1.27273	1.90909	3.18182	0.63636	1	2	3	1	-2	0.24793	2.94215
8	1.45455	2.18182	3.63636	0.72727	1	2	4	1	-3	0.44628	3.38843
9	1.63636	2.45455	4.09091	0.81818	2	2	4	1	-1	0.38017	3.76860
10	1.81818	2.72727	4.54545	0.90909	2	3	4	1	-2	0.41322	4.18182
11	2.0	3.0	5.0	1.0	2	3	5	1	-3	0.0	4.18182

Schedule List : 3-2-1-3-4-3-2-3-1-2-3

No of Cycle :1000

Table 5**:** Schedule generated for demand vector D = (2000, 3000, 5000, 1000) using heuristic nearest integer point

Dynamic Programming Algorithm

In this section we discuss a dynamic programming (DP) algorithm to deal with JIT production schedule for a mixed model facility. The procedure has considered the joint problem with the two typical goals.

1. USAGE GOAL: maintaining a constant rate of usage of all items in the facility.

2. LOADING GOAL: smoothing the work load on the final assembly process to reduce the chance of production delays and stoppages.

It is to be noted that goal 1 is mainly focused in this dissertation and is more important than goal 2. Indeed, goal 2 is a classical one.

Let there are n products to be produced with demands $d_1, d_2, ..., d_n$ in a certain time horizon. The time to produce one unit of product i be denoted by t_i; $i = 1,2,...,n$ and put $D = \sum_{i=1}^{n} d_i$, $r_i = \frac{d_i}{D}$.

The specified time horizon be inferred into D time units and during each time period k ; $k = 1,2,...,D$; exactly one unit of a product should be produced. Let $x_{i,k}$ denote the total production of product i over the first k periods; where $0 \le x_{i,k} \le d_i$ for all $k = 1,2,...,D$.

Then, $\sum_{i=1}^{n} x_{i,k} = k$; $k = 1,2,...,D$ and $x_{i,k}$ is non negative integer for all $i = 1,2,...,n$; $k = 1,2,...,D$.

Suppose that the schedule for the first k stages be determined i.e. $x_{i,k}$ for $i = 1,2,...,n$ be known. Then the usage variability at stage k is $U_k = \sum_{i=1}^{n} (x_{i,k} - kr_i)^2$ and the loading variability at stage k is $L_k = \sum_{i=1}^{n} t_i^{\,2} (x_{i,k} - kr_i)^2$.

Therefore the problem defined by (5.2) can be formulated as

Minimize $\sum_{k=1}^{D} (\alpha_U U_k + \alpha_L L_k)$

Subjected to the Constraints (4.15) - (4.19)

Where α_U, α_L are relative weights for the USAGE goal and LOADING goal respectively. So the problem defined by (5.2) is a joint problem.

Let f_k denote the joint variability at stage k. Then

$$f_k = \alpha_U \sum_{i=1}^{n} (x_{i,k} - kr_i)^2 + \alpha_L \sum_{i=1}^{n} t_i^2 (x_{i,k} - kr_i)^2$$

$$= \sum_{i=1}^{n} (\alpha_U + \alpha_L t_i^2)(x_{i,k} - kr_i)^2$$

$$= \sum_{i=1}^{n} T_i^2 (x_{i,k} - kr_i)^2 \text{ ; Where } T_i^2 = \alpha_U + \alpha_L t_i^2 .$$

Therefore the objective function of the problem defined by (5.2) takes the form:

Minimize $\sum_{k=1}^{D} \sum_{i=1}^{n} T_i^2 (x_{i,k} - kr_i)^2$; where call T_i, the implied production time for period i.

Now we consider the DP procedure presented by Miltenburg et al. [27]

Let $d = (d_1, d_2, ..., d_n)$ be the product requirements vector. Define subsets in a schedule as $X = (x_1, x_2, ..., x_n)$; where x_i is a non negative integer representing the production of exactly x_i units of product i, $x_i \le d_i$ for all i. Let e_i be the i^{th} unit vector; with n entries, having i^{th} entry 1 and remaining all zero. A subset X can be scheduled 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 scheduled (produced) during the first k stages. Let $g(X) = \sum_{j=1}^{n} T_j^2 (x_j - kr_j)^2$. The

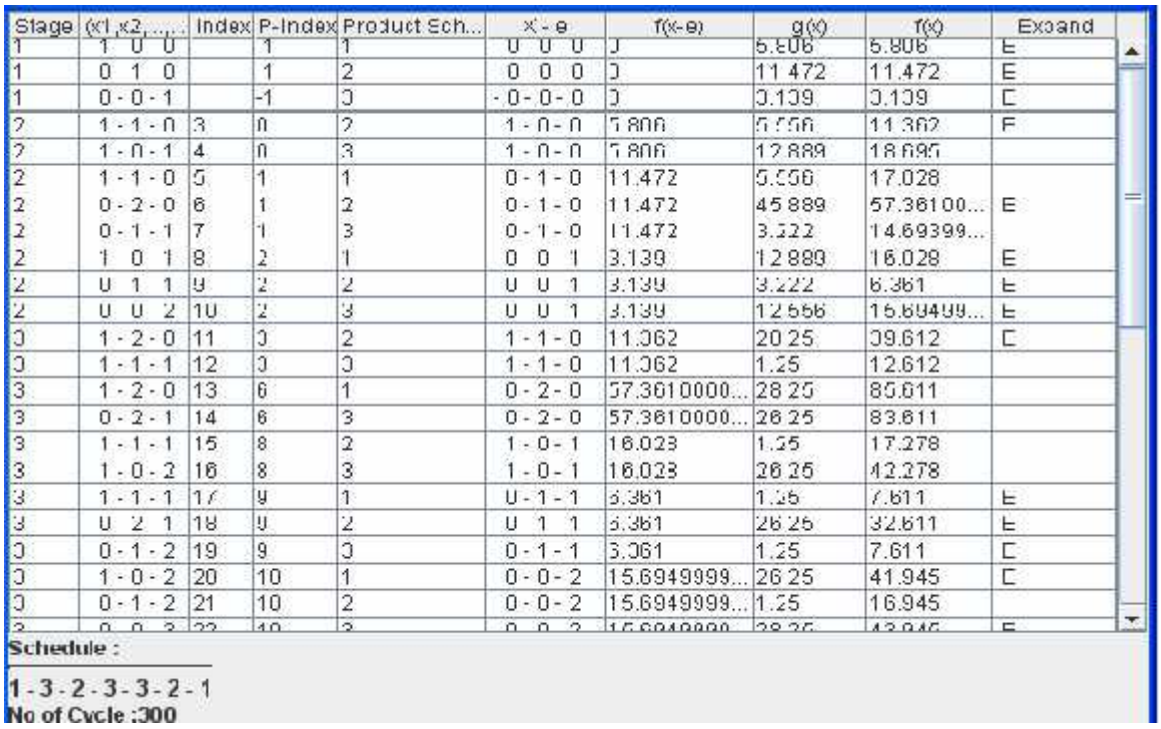
following (DP) recursion (R1) holds for $f(X)$:

$$f(X) = f(x_1, x_2, ..., x_n) = \min\{f(X - e_i) + g(X) \mid i = 1,...,n; x_i - 1 \geq 0\}$$

$$f(X) = f(X \mid x_i = 0; i = 1,...,n) = f(0,0,...,0) = 0.$$

Clearly $f(X) \geq 0$ and $g(X \mid x_i = 0; i = 1,...,n) = 0$. The following theorem tells about the computational efficiency of the above procedure [27]

Example 5.4.1 Demand vector (300, 600, 900) and Time T = (2, 5, 1) the schedule generated by DP for min-sum is shown in Table 6: Schedule generated by dynamic programming, with number of cycle 300.



Stage	(x1 ,x2,...,.	Index	P-Index	Product Sch...	x' - e	f(x-e)	g(x)	f(x)	Expand
1	1 0 0		1	1	0 0 0	0	5.806	5.806	E
1	0 1 0		1	2	0 0 0	0	11.472	11.472	E
1	0 - 0 - 1		-1	0	- 0 - 0 - 0	0	0.139	0.139	C
2	1 - 1 - 0	3	0	2	1 - 0 - 0	5.806	5.556	11.362	F
2	1 - 0 - 1	4	0	3	1 - 0 - 0	5.806	12.889	18.695	
2	1 - 1 - 0	5	1	1	0 - 1 - 0	11.472	5.556	17.028	
2	0 - 2 - 0	6	1	2	0 - 1 - 0	11.472	45.889	57.36100...	E
2	0 - 1 - 1	7	1	3	0 - 1 - 0	11.472	3.222	14.69399...	
2	1 0 1	8	2	1	0 0 1	3.139	12.889	16.028	E
2	0 1 1	9	2	2	0 0 1	3.139	3.222	6.361	E
2	0 0 2	10	2	3	0 0 1	3.139	12.556	15.69499...	E
0	1 - 2 - 0	11	0	2	1 - 1 - 0	11.362	20.25	39.612	C
0	1 - 1 - 1	12	0	0	1 - 1 - 0	11.362	1.25	12.612	
3	1 - 2 - 0	13	6	1	0 - 2 - 0	57.3610000...	28.25	85.611	
3	0 - 2 - 1	14	6	3	0 - 2 - 0	57.3610000...	26.25	83.611	
3	1 - 1 - 1	15	8	2	1 - 0 - 1	16.023	1.25	17.278	
3	1 - 0 - 2	16	8	3	1 - 0 - 1	16.023	26.25	42.278	
3	1 - 1 - 1	17	9	1	0 - 1 - 1	6.361	1.25	7.611	E
3	0 2 1	18	9	2	0 1 1	6.361	26.25	32.611	E
0	0 - 1 - 2	19	9	0	0 - 1 - 1	6.361	1.25	7.611	C
0	1 - 0 - 2	20	10	1	0 - 0 - 2	15.6949999...	26.25	41.945	C
0	0 - 1 - 2	21	10	2	0 - 0 - 2	15.6949999...	1.25	16.945	
3	0 0 3	22	10	3	0 0 2	15.6949999...	28.25	43.945	E

Schedule :

1 - 3 - 2 - 3 - 3 - 2 - 1

No of Cycle :300

Table 6: Schedule generated by dynamic programming

Theorem 5.4.1 [27]

The DP recursion solves the JIT scheduling problem in

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

Proof:

Suppose *g(X)* represents the contribution of each product to the objective function at stage *k*. The minimization in recursion (R1) is done over all possible choices of the product to be in this last position. Since x_i can assume the values 0, 1, 2, .. d_i, the number of sets, or states, in the DP recursion is $\prod_{i=1}^{n}(d_i+1)$.

For each set X there are at most n $f(X-e_i)$ values, to each of which must be added $g(X)$, whose calculation required $O(n)$ times. Therefore, the computational time is $O(n)$ for each set, and $O\left(n\prod_{i=1}^{n}(d_i+1)\right)$

for the entire problem. The value *f(X)* and the produce i, where the minimum occurs in recursion (R1), must be saved for each set *X*, so that the optimal solution can be constructed at the end. Therefore, the space requirements are $O\left(\prod_{i=1}^{n}(d_i+1)\right)$

Notice that the total number of feasible schedules is

$$\frac{D!}{d_1!d_2!.....d_l!}$$

This is considerably larger than the number of states in the DP recursion. Furthermore

$$\prod_{i=1}^{n}(d_i+1) \leq \left(\frac{d_1+d_2+.....+d_n+n}{n}\right)^n = \left(\frac{D+n}{n}\right)^n$$

Therefore the growth rate of the number of sets is polynomial in D although it is exponential with n. this clearly shows that the procedure is effective for small n even with large D. we see that the DP algorithms is efficient only for practical sized problems with the analysis are proposed in [27].

Min-max Absolute-chain Algorithm

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

Given: d_i^t for i = 1, 2, ……, n_t and t=1, 2, ... , m;

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

$chain_1$, $chain_2$, ……., $chain_t$, ..., $chain_m$;

Update: number of demands n = n_t;

demand rates di for i = 1, 2, ……, n;

total demand D = d_i.

Step 1: Calculate windows [E(i , j), L(i, j)] for j = 1, 2, ……, d_i and i = 1, 2, ……, 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 .

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

Example: 5.5.1 The schedule for min-max absolute chain algorithm is shown Table 8, for inputs of Figure 6: Input data for min-max absolute-chain algorithm.

Figure 6: Input data for min-max absolute-chain algorithm

Chain Id	Job Name	Earliest Due Date	Late Due Date
0	x	3.0	3.0
0	y	3.0	3.0
0	x	8.0	8.0
0	y	8.0	8.0
0	x	13.0	13.0
0	y	13.0	13.0
1	r	3.0	3.0
1	s	3.0	3.0
1	r	8.0	8.0
1	s	8.0	8.0
1	r	13.0	13.0
1	s	13.0	13.0
2	c	3.0	3.0
2	c	8.0	8.0
2	c	13.0	13.0

Table 7: Calculation of window value

Chain Id	Job Name	Earliest Due Date	Late Due Date
0	x	3.0	3.0
0	y	3.0	3.0
0	x	8.0	8.0
0	y	8.0	8.0
0	x	13.0	13.0
0	y	13.0	13.0
1	r	3.0	3.0
1	s	3.0	3.0
1	r	8.0	8.0
1	s	8.0	8.0
1	r	13.0	13.0
1	s	13.0	13.0
2	c	3.0	3.0
2	c	8.0	8.0
2	c	13.0	13.0
Schedule :			
x y r s c x y r s c x y r s c			

Table 8: Output of min-max absolute chain algorithm

Cost Assignment Problem

Let Z_{ij} denotes the period in which the copy (i, j) is produced. Then the problem defined by (4.14) can be restated as

$$\text{minimize } F_{sum} = \sum_{i=1}^{n} \left\lceil \sum_{k=0}^{Z_{i1}-1} f_i(0 - kr_i) + \sum_{k=Z_{i1}}^{Z_{i2}-1} f_i(1 - kr_i) + \ldots + \sum_{k=Z_{id_i}}^{D} f_i(d_i - kr_i) \right\rceil \quad (5.9)$$

such that

$$Z_{i,j+1} \geq Z_{ij} + 1,\ j = 1, \ldots, d_i;\ i = 1, \ldots, n \quad (5.10)$$

$$1 \leq Z_{ij} \leq D,\ j = 1, \ldots, d_i;\ i = 1, \ldots, n \quad (5.11)$$

$$Z_{ij} \neq Z_{i'j'} \text{ for } (i, j) \neq (i', j'),\ Z_{ij} \geq 0 \quad (5.12)$$

Note that Constraint (5.12) is the only linking constraint in problem defined by

(5.9), which aims specify that only copy of each product can be produced in each period.

The min-sum PRVP can be reduced to an assignment problem and hence can be solved by Hungarian method which is presented in Figure 3: The Hungarian method. For the corresponding assignment problem we consider the vertex sets $V_1 = \{(i, j) : i = 1, \ldots, n,\ j = 1, \ldots, d_i\}$ and $V_2 = \{1, \ldots, D\}$. We now have to calculate the appropriate costs to specify its objective function. More specifically, these costs must be such that the assignment problem with these costs has an optimal solution, which is both optimal and feasible for problem (5.9).

Let C_{ijk} denotes the cost of assigning (i, j) to the k^{th} period and let

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

Then the assignment problem is

$$\text{minimize } C = \sum_{i=1}^{n} \sum_{j=1}^{d_i} \sum_{k=1}^{D} C_{ijk} x_{ijk} \tag{5.13}$$

$$\text{such that} \sum_{i=1}^{n} \sum_{j=1}^{d_i} x_{ijk} = 1,\ k = 1, \ldots, D \tag{5.14}$$

$$\sum_{k=1}^{D} x_{ijk} = 1,\ i = 1, \ldots, n,\ j = 1, \ldots, d_i \tag{5.15}$$

Constraints on the assignment problem require that

a) For each (i, j) in V_1 there is exactly one k in V_2, i.e. each copy is produced exactly once.

b) For each k in V_2, there is exactly one (i, j) in V_1, i.e. exactly one copy is produced at a time.

But Constraints (5.10) on problem defined by (5.9) requires an additional property that

c) For any two copies (i, j) and (i, j') of a product i, with $j < j'$, if (i, j) is produced at k and (i, j') is produced at k' then $k < k'$.

Example 5.6.1 For demand vector D = (300, 600, 900) the schedule generated by cost assignment method is shown in Table 10**:** Schedule generated by cost assignment problem, with the repetition of 300 cycles.

Product	S.No	1	2	3	4	5	6
1	1	0.667	0.333	0.0	0.333	0.667	1.0
2	1	0.333	0.333	1.0	1.0	1.0	1.0
	2	1.0	1.0	1.0	0.333	0.333	1.0
3	1	0.0	1.0	1.0	1.0	1.0	1.0
	2	1.0	1.0	0.0	1.0	1.0	1.0
	3	1.0	1.0	1.0	1.0	0.0	1.0

Table 9**:** Excess inventory or shortage costs calculated

Product	S.No	1	2	3	4	5	6
1	1	1.0	0.333	0.0	0.0	0.333	1.0
2	1	0.333	0.0	0.333	1.333	2.333	3.333
	2	3.333	2.333	1.333	0.333	0.0	0.333
3	1	0.0	0.0	1.0	2.0	3.0	4.0
	2	2.0	1.0	0.0	0.0	1.0	2.0
	3	4.0	3.0	2.0	1.0	0.0	0.0

Schedule:
- 3 - 2 - 1 - 3 - 2 - 3
No of cycle :300

Table 10**:** Schedule generated by cost assignment problem

6 SOLUTION PROCEDURE FOR ORV PROBLEM

Here, we introduce the solution procedure for ORVP. The ORVP has been shown to be strongly NP-hard by reducing the known NP-hard scheduling problem “Around the shortest job to ORVP.

Toyota used the goal chasing methods GCM I and GCM II, Monden [28]. The heuristics GCM I and GCM II construct a sequence filling one position at a time from first slot to the last one. They, designed with product level and sub-assembly level, consider the variability at the sub-assembly level. GCM II compared to GCM I represents a decrease in computational time because the sum is formed only on the components of a given product.

Miltenburg and Sinnamon [25] extend GCM to extended goal chasing method with more levels. GCM is a special case of their myopic polynomial heuristic. They also introduce another polynomial heuristic to remedy the myopic problems of the previous heuristic.

In Section 5.1 it is shown that the optimal sequence both for min-max and min-sum PRVPs are cyclic. But Dhamala and Kubiak [11] conjecture that whether the optimal sequence to ORVP are cyclic.

Toyota’s Goal Chasing Method (GCM)

Among the major car manufactures, Toyota has always been an innovator in the areas of manufacturing and assembly. Toyota operates according to the JIT principle. Toyota’s most important goal in the operation of its mixed-model production system is to keep the rates of consumption of all parts constant. For sequencing mixed model multi level JIT production system, Toyota developed and used an algorithm known as the GCM to schedule automobile final assembly lines, (cf. [28] and also see [25]).

Goal chasing method (GCM) for a stage k $(1 \le k \le D)$ schedule the product p among n_1 products at with the lowest

$$GCM_{pk} = \sum_{i=1}^{n_2} [(x_{i,2,k-1} + t_{i,,2,p}) - k \times d_{i,2}) / D_i]^2 \qquad (6.1)$$

To minimize this objective the GCM algorithm is described as:

Algorithm 6.1 [30] Goal Chasing Method

Step1.Set $x_{i20} = 0, S_0 = \{1,2,...,n_1\}$ *and* $k = 1$

Step 2.Select for k^{th} *position in the sequence model p* that minimizes the measure*

$$GCM_{p^*k} = \min_{p \in s_{k-1}} \left\{ \sum_{i=1}^{n_2} [(x_{i,2,k-1} + t_{i,,2,p}) - k \times d_{i,2}) / D_i]^2 \right\}$$

Step3.If more copies of model p remain to be sequenced set* $S_k = S_{k-1}$. *If all*

Copies of model p now already have been sequenced set*

$S_k = S_{k-1} - \{p^*\}$.

Step 4. If $S_k = \phi$, *then stop.*

If $S_k \neq \phi$, *set* $x_{i2k} = x_{i,2,k-1} + t_{i2p^*}$, $i = 1,...,n_2$

Set k = k+1 and go to step 2

Figure 7: The Goal Chasing Algorithm

But in practice it is difficult to apply the Goal Chasing Method to all parts as the total number of parts required for a car is in around 20,000. Therefore, the parts are

represented only by their respective subassemblies. The number of subassemblies is around 20 and Toyota gives the important subassemblies additional weights. The sub assemblies include the following items (see [30]):

Engines,	Bumpers,
Transmissions,	steering assemblies,
Frames,	Wheels,
Front axles,	Doors,
Rear axles,	Air conditioners.

Since the GCM is developed only for two levels, it considers only the variability at the subassembly and the variability at final assembly is ignored. The GCM can also be extended for all levels and is known as Extended Goal Chasing Method (EGCM) (see [25]) in which, for example with L levels and for the objective $\sum_{k=1}^{D_1}\sum_{l=1}^{L}\sum_{i=1}^{n_l} w_l \left(x_{ijk} - d_{il}k / D_l\right)^2$ schedule the product p at stage k with the lowest

$$EGCM_{pk} = \sum_{l=1}^{L} \beta_{plk} \tag{6.2}$$

Where $\beta_{plk} = \sum \left[\left(x_{il(k-1)} + t_{ilp}\right) - k \times d_{il} / D_l\right]^2$ (6.3)

And to minimize this objective, a similar algorithm as that of 6.1 can be developed

Miltenburg and Sinnamon Heuristic Approach

Suppose that each product has significantly different sub-assembly, component and raw material requirements. Then the variation at all levels in the system must be considered when selecting a product schedule. Beginning with stage *1*, compose a schedule stage by stage using the following decision rule at each stage *k*, taking the schedule already determined for stages 1,2,3,...$k-1$as fixed.

The mathematical expression of this decision rule is:

Schedule the product i with the lowest

$$H_{pk} = W_l (x_{p1(k-1)} - kr_{pl}) + 0.5 \times \sum_{l=2}^{L} \beta_{plk}$$

Where $\beta_{plk} = \sum_{i=1}^{n_l} W_l (x_{pl(k-1)} + t_{ilp}) - (y_{l(k-1)} + \alpha_{lp}) \times r_{il}]^2$

And

$$\alpha_{lp} = \sum_{i=1}^{n_l} t_{ilp}$$

To see this consider a stage k, if product p is scheduled, the affected terms in the objective function of P4.1 for stage k are:

$$V_p = W_l + \sum_{l=2}^{L} \sum_{i=1}^{n_l} W_l [(x_{il(k-1)} + t_{ilp}) - (y_{l(k-1)} + \alpha_{lp}) \times r_{ip}]^2$$

where $\alpha_{lp} = \sum_{i=1}^{n_l} t_{ilp}$. Since the objective function is to be minimized. Let product p be scheduled rather than product p' if $V_p < V_{p'}$. Canceling the identical terms and simplifying the expression will show that this is equivalent to saying that $H_{pk} < H_{p'k}$.

This is a myopic heuristic in that it does not consider the effect of its current decision on the variation in future cycles. That is, it may achieve low variability at stage *k* at the expense of high variability at stage $k+1$.

Miltenburg and Sinnamon [25] introduce another scheduling heuristic of complexity

$O(n_1^2(n_2+...+n)^2$ for each stage. This is heuristic attempts to remedy the myopic problem of previous heuristic.

For each cycle k :

Step 1: Set $l=1$.

Step 2: Tentatively schedule product l *to be produced in stage* k *. Calculate the variation for stage* k *and call it* V_{1_l}.

Step 3: Find the product p *with the lowest* $H_{p(k-1)}$ *for stage* $k+1$. *Calculate the variation for stage* $k+1$ *and call it* V_{2_l} *. Compute* $V_l = V_{1_l} + V_{2_l}$

Step 4: Increment $l(l=l+1)$. *If* $l > n_1$ *go to Step 5, otherwise go to Step 2. Where* n_1 *is number of product.*

Step 5: Schedule the product p *with the lowest* V_l *in stage* k *.*

Figure 8: Miltenburg and Sinnamon heuristic approach

Example 6.2.1: we consider only two levels-product and sub-assembly. Suppose $n_1 = 2$ products with demands 600, 500 units. The product consists of $n_2 = 3$ different sub-assemblies. The bills of material are shown in Figure 9.

Sub-assembly	Product 1	2
1	1	2
2	0	4
3	1	0

Figure 9: Input demand for ORVP

To develop a production schedule t_{ilp}, d_{il} and r_{il} are calculated from these data and shown in Table 11: Assembly and demand data for Example 6.2.1.

t_{ilp} – Number of parts for one unit of product l

Product	Sub-assembly, $l = 2$				Demand	Ratio
$l = 1$	$i = 1$	2	3	Total	d_{l1}	r_{i1}
1	1	0	1	2	6	.5455
2	2	4	0	6	5	.4545
Demand d_{il}	16	20	6	42		
Ratios r_{il}	.38	.4762	.1429			

Table 11: Assembly and demand data for Example 6.2.1

The calculations for heuristic of Miltenburg and Sinnamon [25] for the first 13 stages are shown in Table 12: Detail Schedule of Example 6.2.1. This procedure is repeated for 100 times and final schedule is 1-2-1-2-1-2-1-2-1-2-1 with a total variation of 163.5116484000000159over 13 cycles.

Stage k	Product p	ψ_{p1k}	β_{p1k}	β_{p2k}	H_{pk}	Product Schedule	Variation	Total Variation
1	1	-0.5455	0.4131405	1.4737914	0.19139570000000006	1	1.8869319	1.8869319
	2	-0.4545	0.5951405	2.1229225999999994	0.6069612999999997			
2	1	-0.09099999999999997	1.652562	5.8951656	2.8565828	2	3.364252400000001	5.251184300000001
	2	-0.909	0.016561999999999	3.347690400000001	0.7648452000000006			
3	1	-1.6364999999999998	0.5372644999999999	0.059062399999999996	-1.6069687999999998	1	0.596326899999999	5.847511200000009
	2	-0.3635000000000016	3.0832644999999994	21.973161599999997	10.623080799999999			
4	1	-1.182	3.338248	5.8951656	1.7655828000000002	2	5.413938400000001	11.26144960000019
	2	-1.818	2.066248	3.347690400000001	-0.14415479999999947			
5	1	-2.7275	4.6035125	0.059062399999999996	-2.6979688	1	4.6625749	15.924024500000019
	2	-1.2725	7.513512500000001	21.973161599999997	9.714080799999998			
6	1	-2.2729999999999997	9.057058000000001	5.8951656	0.6745828000000005	2	11.49674840000001	27.42077290000029
	2	-2.7270000000000003	8.149058	3.347690400000001	-1.05315479			
7	1	-3.8185	12.7028845	0.059062399999999996	-3.7889687999999997	1	12.7619469	40.18271980000029
	2	-2.1815	15.976884499999999	21.973161599999997	8.805080799999999			

8	1	-3.364	18.808992	5.8951656	-0.4164171999999997	2	21.6126824	61.79540220000000029
	2	-3.636	18.264992	3.347690400000001	-1.9621547999999995			
9	1	-4.90949999999995	24.8353805	0.05906239999999996	-4.879968799999999	1	24.89444289999998	86.68984510000000009
	2	-3.090500000000005	28.473380499999998	21.973161599999997	7.896080799999998			
10	1	-4.455	32.59405	5.8951656	-1.5074172	2	35.76174040000001	122.4515855000000109
	2	-4.545	32.41405	3.347690400000001	-2.8711547999999993			
11	1	-6.0005	41.0010005	0.05906239999999996	-5.9709688	1	41.06006290000005	163.5116484000000159
	2	-3.999500000000003	45.0030005	21.973161599999997	6.987080799999998			

Table 12: Detail Schedule of Example 6.2.1

Dynamic Programming Algorithm

The weighted case of the objective $G'_{\max}$ and G''_{sum} can be formulated as:

$$G'^{w}_{\max} = \max_{i,l,k} w_{il} \left| x_{ilk} - y_{lk} r_{il} \right| \tag{6.1}$$

$$G''^{w}_{sum} = \sum_{k=1}^{D_1} \sum_{l=1}^{L} \sum_{i=1}^{n_l} w_{il} \, (x_{ilk} - y_{lk} r_{il})^2 \tag{6.2}$$

Subjected to the Constraints (4.3) – (4.8) where w_{il} be a weighting factor which reflect the relative importance of balancing the sequence for part i at level l.

Now, in this section we summarized an implicit enumeration dynamic programming (DP) procedure which can optimize the problem $G^{'w}_{\max}$. By definition, we have

$$\begin{aligned} x_{ilk} - y_{lk} r_{il} &= \sum_{p=1}^{n_1} t_{ilp} x_{p1k} - r_{il} \sum_{p=1}^{n_1} \sum_{i=1}^{n_l} t_{ilp} x_{p1k} \\ &= \sum_{p=1}^{n_1} \left(t_{ilp} - r_{il} \sum_{i=1}^{n_l} t_{ilp} \right) x_{p1k} \\ &= \sum_{p=1}^{n_1} \delta_{ilp} x_{p1k}, \end{aligned}$$

where

$$\delta_{ilp} = t_{ilp} - r_{il} \sum_{i=1}^{n_l} t_{ilp.}$$

Since $w_{il} \geq 0$, $x_{ilk} \geq 0$ *and* $r_{il} > 0$, then the deviation for part i of level l at stage k for $G^{'w}_{\max}$ would be

$$w_{il} \left| x_{ilk} - y_{lk} r_{il} \right| = \left| w_{il} \left(\sum_{p=1}^{n_1} \delta_{ilp} x_{p1k} \right) \right| = \left| \sum_{p=1}^{n_1} \gamma_{ilp} x_{p1k} \right|,$$

where $\gamma_{ilp} = w_{il} \delta_{ilp}$ is the measure of the weighted deviation in the usage of part i in level l from the proportional usage per unit of product p. Let $\Gamma = \left(\gamma_{ilp} \right)_{n \times n_1}$ be the matrix where $n = \sum_{l=1}^{L} n_l$ is the total number of different parts and products. Each row of Γ corresponds to either a product or a part at the corresponding levels. The value γ_{ilp} will be the element appearing in the $\left(\sum_{m=1}^{l-1} n_m + i \right) th$ row and the p$^{\text{th}}$ column of the matrix Γ. The maximum norm of a vector $a = (a_1, ..., a_n)$ is defined to be

$\|a\|_1 = \max_{1\le i\le n}\{|a_i|\}$. Then the objective function $G^{'w}_{\max}$ can be written as $G^{'w}_{\max} = \max_k \|\Gamma X_k\|_1$, where $X_k (= x_{11k},...,x_{n_1 1k})$ is the cumulative, level 1 production vector through the first k stages. Let the demand vector at level 1 be $d = (d_{11},...,d_{n_1 1}) = (d_1,...,d_{n_1})$ and the states in a sequence be $X = (x_1,...,x_{n_1})$ with $|X| = \sum_{i=1}^{n_1} x_i$, $i = 1,...,n_1$ where x_i is the cumulative production of product i, $x_i \le d_i$. Let $e_i = (0,...,1,...,0)$ be the unit vector with n_1 entries all of which are zero except for a single 1 in the i^{th} row. Let $\phi(X)$ be the minimum value of the maximum deviation for all parts and products over all partial sequences which lead to state X. The norm $\|\Gamma X\|_1$ represents the maximum deviation of actual production from desired one over all products and parts in state X at stage $k = |X|$. The following DP recursion holds for $\phi(X)$ (cf. [22], see also [13]):

$$\phi(\emptyset) = \phi(X : X = 0) = 0$$
$$\phi(X) = \phi(x_1,...,x_{n_1}) = \min_i \{\max\{\phi(X - e_i), \|\Gamma X\|_1\} : i = 1,...,n_1;\quad x_i \ge 1\}. \quad (6.3)$$

It can be observed that $\phi(X) \ge 0$ and $\|\Gamma(X : X = d)\|_1 = 0$ for any state X.

Theorem 6.3.1 [22] The DP Recursion (6.3) solves the MMJITSP $G^{'w}_{\max}$ in

$O\left(n_1 n \prod_{i=1}^{n_1}(d_i + 1)\right)$ Time and

$O\left(n_1 n \prod_{i=1}^{n_1}(d_i + 1)\right)$ Space.

7 CONCLUSION

In this dissertation, the MMJITSP with respect to different mathematical models and sequencing approaches developed till date have been analyzed. The mathematical formulations of the MMJITSP in scheduling theory are in the more challenging non-integer programming form. It is because the linear case of the integer programming has already been NP-hard. We implement some of the algorithms and heuristics both for single-level and multi-level JIT sequencing problem to obtain the cyclic sequences by which the computational complexity is significantly reduced. Our study shows that the MMJITSP has real world like existing application as well as in operating system.

When selecting a schedule for a mixed-model multi-level production system, is to keep a constant rate of usage for every part used by the system. In this dissertation we studied a theoretical basis of doing this. A mathematical model is explored and analyzed. Two scheduling heuristics of Miltenburg and Sinnamon are implemented and we give an example of it (cf. Example 6.2.1). Another heuristic called GCM to sum-deviation ORVP and dynamic programming approach both for sum-deviation and max-deviation ORVP are studied and explored.

The different solution approaches such as nearest integer point, dynamic programming, cost assignment and earliest due date algorithms for sum deviation PRVP are implemented with optimal cyclic solutions for some particular instances. Likewise, min-max absolute-chain and earliest due date algorithms are implemented to sequence products under the min-max objective for a pre-specified threshold value. The modified proof of the existence of cyclic solutions to min-max absolute problem is presented (cf. Theorem 5.1.3)

Most of the PRVP Just-in-Time problem had been efficiently solved by pseudo-polynomial algorithms depending on the input size of the demands; their complexity status is not yet clear. Even the basic min-max absolute deviation problem is Co-NP but it

is still open whether the problem is Co-NP-Complete or polynomial solvable.

ORVPs even with two levels are strongly NP-hard; however the developments of approximation procedures like GCM and dynamic programming procedure provide an interest to the researchers for the further improvements. But the problem, under the assumption that the products require approximately the same number and mix of parts or the pegging assumptions (single-level) is solvable.

The relation between optimal sequences of the min-sum and min-max problem will be the foremost topic for the further investigation and to determine an algorithm which simultaneously optimizes both min-sum and min-max objectives will be the most interesting topic for the research.

Cyclic JIT sequences for PRVP are optimal and existence of such sequences considerably reduces the computational effort. The question, whether cyclic sequences to ORVP are optimal, is still open.

REFERENCES

[1] H. Aigbdo, *Some Structural Properties for the JIT Level Schedule Problem*, Production Planning and Control, 11, 4 (2000), 357-362.

[2] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt and J. Weglarz, Scheduling Computer and Manufacturing Processes (Springer- Verlag, Berlin, 1996).

[3] N. Boysen, M. Fliedner and A. Scholl, *Sequencing mixed-model assembly lines*, Survey, classification and model critique. European Journal of Operational Research, to appear, 2007.

[4] H. Brasel., *Latin Rectangles in Scheduling Theory*, Working paper, Otto-von-Guericke-University, Magdeburg, Germany (1990).

[5] H. Brasel., *Matrices in Shop Scheduling Problems*, Working paper, Otto-von-Guericke-University, Magdeburg, Germany (2005).

[6] N. Brauner, and Y. Crama, *The Maximum Deviation Just-In - Time Scheduling Problem*, Discrete Applied Mathematics, 134 (2004) , 25-50.

[7] P. Brucker, Scheduling Algorithms, 2nd edition (Springer- Verlag, 1995).

[8] T. N. Dhamala, *Just-in-Time Sequencing Algorithms for Mixed-Model Production Systems*, The Nepali Mathematical Sciences Report, 24, 1 (2005), 25-34.

[9] T. N. Dhamala, Shop Scheduling Solution-Spaces with Algebraic Characterizations (Ph. D. Thesis Otto-von-Guericke University, Magdeburg, Germany, 2002).

[10] T. N. Dhamala and S. R. Khadka, *Bottleneck Product Rate Variation Problem with Absolute-Deviation Objective*, Submitted to The Nepali Mathematical Sciences Report, 27, 1-2 (2008).

[11] T. N. Dhamala and W. Kubiak, *A Brief Survey of Just-In-Time Sequencing for Mixed-Model Systems*, International Journal of Operations Research, 2(2005), 38-47.

[12] T. N. Dhamala, S. R. Khadka, M. H. Lee, Bottleneck Product Rate Variation Problem for Mixed-Model Just-In-Time Production System (Working Paper 2008).

[13] T. N. Dhamala, S. R. Khadka, M. H. Lee, *On Sequencing Approaches for Mixed-Model Just-In-Time Production Systems*, Asia-Pacific Journal of Operational Research, 37(2008), 1-25.

[14] R. E. Graham, E. L. Lawer, J. K. Lenstra and A. H. G. Rinnoy Kan, *Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey*, Annals of Discrete Mathematics, 5 (1979), 287-326.

[15] R. R. Inman, and R. L. Bulfin, *Sequencing JIT Mixed- Model Assembly Lines*, Management Science, 37(1991), 901-904.

[16] M. Y. Kovalyov, W. Kubik and J.S. Yeomans, *A Computational Analysis of Balanced JIT Optiminization Algorithms*, Journal of Information Systems and Operational Research- INFOR, 39 (Aug 2001), 299- 315.

[17] W. Kubiak, *Cyclic Just- in- Time Seqauence are Optima*l, Journal of Globle Optimization, 27 (2003a), 335- 347.

[18] W. Kubiak, *Minimizing Variation of Product Rates in Just - In - Time Systems: A Survey*, European Journal of Operational Research, 66 (1993), 259-271.

[19] W. Kubiak, *Solution of the Liu-Layland Problem via Bottleneck JIT Sequencing,* Journal of Scheduling, 8 (2005), 295-302.

[20] W. Kubiak and S. P. Sethi, *A Note on Level Schedules for Mixed- Model Assembly Lines in Just-In-Time Production Systems*, Management Science, 37(1991),121-122.

[21] W. Kubiak and S. P. Sethi, *Optinal Just - In- Time Schedules for flexible transfer lines*, The International Journal of Flexible Manufacturing Systems, 6 (1994), 137-154.

[22] W. Kubiak, G. Steiner and J.S. Yeomans, *Optimal Level Schedules for Mixed-Model, Multi-Level Just- In - Time Assembly Systems*, Annals of Operations Research, 69 (1997) , 241- 259.

[23] M. Milenkovic, Operating Systems. Tata McGraw-Hill (1997)

[24] J. Miltenburg, *Level Schedules for Mixed - Model Assembly Lines in Just - In – Time Production Systems*, Management Science, 35 (1989), 192- 207.

[25] J. Miltenburg and G. Sinnamon, *Scheduling Mixed - Model Multi-Level Just-In – Time Production Systems*, International Journal of Production Research, 27 (1989) , 1487- 1509.

[26] J. Miltenburg and 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.

[27] J. Miltenburg, G. Steiner and S. Yeomans, *A Dynamic Programming Algorithm For Screduling Mixed - Model, Just - In- Time Production Systems*, Mathematical and ComputerModeling,13(1990), 57-66.

[28] Y. Monden, Toyota Production Systems (Industrial Engineering and Management Press, Norcross, GA 1983).

[29] C. H. Papadimitriou and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity (Prentice-Hall of India Pvt. Ltd. 2003).

[30] M. Pinedo and X. Chao, Operations Scheduling with Application in Manufacturing and Services (Irwin, McGraw-Hill, 1999).

[31] K.H. Rosen, Discrete Mathematics and its Applications (TATA MCGRAW Hill Edition, (2003), New Delhi).

[32] S. J. Russel and P. Norving, Artificial Intelligence: A Modern Approach (Prentice Hall, 1995).

[33] Stage-by-sage business development, www.1000ventures.com.

[34] G. Steiner, and S. Yeomans, *A inear Time Algorithm fo Maximum Matching in Convex, Bipartite Graphs*, Computers Math. Applic., 31(1996), 91-96.

[35] G. Steiner and S. Yeomans, *Level Schedules for Mixed - Model Just - In - Time Processes*, Manage. Science., 39 (1993), 728-735.

[36] G. Steiner and J. S. Yeomans, *Optimal Level Schedules in Mixed - Model , Multi-Level JIT Assembly Systems with Pegging,* Europen Journal of Operational Research, 95 (1996) , 38-52.

[37] T. Suganuma and T. Ogasawara, *Overview of the IBM Java Just-in-Time compiler*, IBM System Journal, 39, 1 (2000).

[38] B. Sussman., *Scheduling Problems with Interval Disjunctions*, Operational Research, 16, (1972), 165-178.

[39] V.S. Tanaev., Y.N.,Sotskov and V.A. Strusevich, *Scheduling Theory Multi-Stage System* (Kluwer Academic Publishers, Dordrecht, Boston, London, 1994)

[40] A. Tanenbaum, Modern Operating Systems. Prentice-Hall of India Pvt. Ltd. (2004).

[41] Toyota Motor Corporation Global Site, www.toyota.co.jp.

[42] www.assignmentproblem.com.

[43] Wikipedia, the Free Encyclopedia (http://en.wikipedia.org 2008).

[44] S. Yeomans, Optimal Level Schedules for Mixed-Model Just-in-Time Assembly Systems (Ph. D. Thesis, McMaster University, Hamilton, Ontario, 1997).