Chapter 1

1. Introduction

1.1. Background

Mixed Model Just-in-Time (JIT) production system came into existence with the motto of reducing cost of diversified small lot instead large lot to minimize great inventories and to great shortage which produce only the necessary quantities at the necessary time of the competitive production centers have challenged to provide a variety of products at a very low cost by smoothing products and increasing computer applications.

Mixed Model Just-in-Time production system minimizes both the earliness and the tardiness penalties which respond to the customs demands for a variety of models provoking massive shortage of the products. This model has the aim to hold inventory and shortage cost as small as possible like as it also sequence the problems that consider other operational characteristic though limited to a small subset of them of the line also is to minimize sequence dependent work overload [2]. This system consists of a hierarchy of a finite and distinct level such as products, sub-assemblies, component parts, raw materials, etc.

The significant aim of the system is to obtain an optimum sequence of a number of products that minimizes deviation through the time between the actual and ideal production. This sequence always keeps the actual production one as close to each other as possible all the time. The sequence at the final level is crucial and affects all the supply chain as all other levels are also essentially fixed due to pull nature of the system. Balancing the schedule determines the Sequence of final Assembly which achieves the goals.

JIT has been firstly introduced by modern Toyota Production system as a Just in time scheduling. This concept has been used for the control of assembly system. The main aim of the JIT system is to satisfy the customers' demand for a variety of products without holding the large shortage also a JIT production system is to ensure that the quantity of each part used by the assembly process is kept as close to constant as possible per unit time. JIT system from conventional system is that subsequent process within the manufacturing system "pull" their parts require mends from the preceding processes. The pull process result in a Miltenburg and Sinnamon [20] and Miltenburg and Goldstrin [19] explain that the formulation of the multi-level system. The assembly line is used as a pull line and the system as a pull system. Kubiak

[17] gave a more specific distinction between these problems and the product refered single level problem as the Product Rate Variation (PRV) problem and the multi-level problem as the Output Rate Variation (ORV) problem. Kubiak also differentiated PRV problem as a total deviation PRV problem and maximum deviation PRV problem. The total deviation PRV problem goals to minimize the sum of total deviation and therefore looks on minimizing the total variation between the actual product and the ideal production in any shortage, such problem represents the problem considered by Miltenburg [21]. The maximum deviation PRV problem minimizes the maximum deviation of the actual product of a product from its ideal level of production. Such problem represents the problems considered by Steiner and Yeomnas [25]. The pull system where the final assembly line defines the scheduling and requests for demand down the level is represented by O.R.V. problem. The problems are considered by Miltenburg and Sinnamon [20] and Miltenburg and Goldstein [19] are categorized as ORV problem.

Kubiak [16] proves that optimal JIT sequences are cyclic. 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 It also has important consequences on the computational time complexity if all existing algorithms for PRV [8]. In 2004 Brauner and Crama [7] determine a set of algebraic necessary and sufficient condition for existence of a maximum deviation JIT schedule with a give objective value.

Miltenburg [21] and Miltenburg and Sinnamon [19] observe the existence of the cyclic sequences for the total PRV problems. Kubik [16] shows that the cyclic sequences are optimal and Dhamala and Kubiak [13] gave opinion that cyclic sequences in the ORV are optimal.

1.2. Objective of the Study

The main objective of this Dissertation is to study the Sequencing Approach for Mixed-Model Just in Time Production system. In this dissertation, Heuristic and Dynamic Programming Approach only focus.

The general objectives of this study were stated as:

- To study different methods in Just-in-time production system.

- To study different objective functions under Just-in time production environment.
- To find out possibility of sequencing approach in Just-in-time production system.

1.3. Organization of the Thesis

This is organized as follows:

The **Chapter 1** briefly describes the introduction to scheduling, significance of the problem, machine scheduling and scheduling problem and also introduces the scheduling environment.

The **Chapter 2** describes algorithms and computational complexity. The theoretical basis of computer science has been formulated. Computational resources and complexity classes are described.

The **Chapter 3** describes the formal description of scheduling theory, different types of scheduling problems and solution strategies.

The **Chapter 4** discusses the single machine scheduling problems and methods previously used to solve similar scheduling problems in the literature.

The **Chapter 5** describes the past work in the similar problem and some traditional approach to solve the Different types of sequencing Approaches.

In the **Chapter 6** computational experiments are performed on randomly generated real size data and the solutions obtained for both Heuristic Algorithm and Dynamic Algorithm are compared.

In **Chapter 7** Conclusion and directions of future research is given.

1.4. Methodology

A large number of papers related to the Scheduling, Optimization Problem, Heuristic Algorithm Problem, and Dynamic Algorithm Problem have been collected and studied that deal with the Mixed Model Just-in-time sequencing Production System. The necessary documents are collected from the internet and from my supervisor. Basically the papers are related with scheduling problem in JIT environment at single and Multi-Level production. Beyond papers, some books on scheduling algorithms are also followed.

Chapter 2

2. Computational Complexity

Computational complexity theory is a branch of the theory of computation in computer science that investigates the problems related to the resources required to run algorithms, and the inherent difficulty in providing algorithms that are efficient algorithms for both general and specific computational problems. Complexity theory attempts to describe how difficult it is for an algorithm to find a solution to a problem. This differs from computability theory, which describes whether a problem can be solved at all. Furthermore, much of complexity theory deals with decision problems. A decision problem is one where the answer is always "yes" or "no". Some problems are un-decidable, or at least seem so, so complexity theory can be used to distinguish problems where it is certain to get a correct "yes" or "no" (not necessarily both). A problem that reverses which can be relied upon is called a complement of that problem. Complexity theory analyzes the difficulty of computational problems in terms of many different computational resources. A problem can be described in terms of many requirements it makes on resources: time, space, randomness, alternation, and other less-intuitive measures (vague).

2.1. Turing Machine

Turing Machine is basic abstract symbol-manipulating device which, despite their simplicity, can be adapted to simulate the logic of any computer algorithm. It was described in 1936 by Alan Turing. Turing Machine is not intended as a practical computing technology, but a thought experiment about the limits of mechanical computing. Thus it was not actually constructed. Studying its abstract properties yields many insights into computer science and complexity theory.

A Turing Machine that is able to simulate any other Turing Machine is called Universal Turing Machine (UTM, or simply a universal machine). A more mathematical–oriented definition with a similar "universal" nature was introduced by Alonzo Church, whose work on lambda calculus intertwined with Turing's in a formal theory of computation known as the Church-Turing thesis. The thesis states that Turing machines indeed capture the informal notion of effective method in logic and mathematics, and provide a precise definition of an algorithm or 'mechanical procedure'. Some of the examples of Turing Machine are: Turing's very first machine, copy routine, 3-state busy beaver.

2.2. Functions:

A function associates one quantity, the argument of the function, also known as the input, with another quantity, the value of the function, also known as the output. A function assigns exactly one output to each input. The argument and the value may be real number, but they can also be elements from any given set. An example of a function is f (x) = 2x, a function which associates with every number the number twice as large. Thus, with the argument 5 the value 10 is associated, and this is written f (5) = 10.

One precise definition of a function is an ordered triple of sets, written (X, Y, F), where X is the domain, Y is the co-domain, and F is a set of ordered pairs (a, b). In each of the ordered pairs, the first element is from the domain, the second element b is from the co-domain, and a necessary condition is that every element in the domain is the first element in exactly one ordered pair. The set of all b is known as the image of the function, and need not be the whole of the co-domain. Many authors use the term "range" to mean the image, while some use "range" to mean the co-domain.

The notation f: X→Y indicates that f is a function with domain X and co-domain Y, and the function f is said to map or associate elements of X to elements of Y. A function f whose values are in the set of real number R is called a real-valued function and is non-negative if $f \geq 0$. Since we shall be mostly interested in teal 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) ≤ f(y) whenever$x \leq y$. Similarly, f is a called monotonically decreasing if f(x) ≥ f(y) whenever $x \leq y$.

The function f is said unimodal if for some value such that either (a) or (b) holds:

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

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

If the domain and co-domain are both the set of real numbers, using the ordered triple scheme we can, for example, write the function $y = x^2$ as

2.3. Graph Theoretical Denotations

A graph G= (V, E), the edge set E consist of unordered pairs of vertices, rather than ordered pairs. That is, and edge is a set {u, v}, where, u, v ∈ V and u ≠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'}∈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)∈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 other in V_2. In such a case G is called a bipartite graph and we denote 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 ∪ U, E) is a V-convex if there is an ordering on V such that [v_i, u_k] ∈ E and [v_j, u_k] ϵ E with v_i, v_j ∈ V, $v_i < v_j$ implies that [v_p, u_k] ∈ E for $v_i \leq v_p \leq v_j$ [34].

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

2.4. Algorithms

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 ∏: Z→Y, where Z is the set of all problems instances I and Y is the set of the solutions. An algorithm is a set of precise instructions for performing a computation or solving an optimization problem.

In other words, an algorithm is any well defined computation procedure that takes some value, or a 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 transforms the input into the output. To represent algorithms we use English language, however, for the simplicity we use pseudo code that can represent an algorithm in a 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 input 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.

Complexity of Algorithms

When an algorithm is defined, 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 measured 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. 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 an n_0 such that for all $n \leq n_0, 0 \leq f(n) \leq c \times g(n)$.

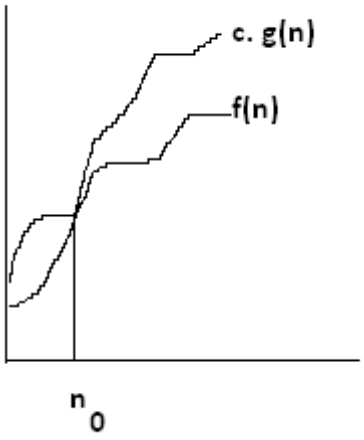

Figure 1: Graphical notation of f(n) = O (g(n)) [5]

A polynomial time (polynomial) algorithm is the one whose time complexity functions T(k) ∈ 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 $\prod$ is called polynomial solvable if there is a polynomial time algorithm solving it. A problem $\prod$ 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∈ $\prod$. Hence, the notion of pseudo-polynomial solvable depends on the magnitude of the largest input data involved.

Given any problem instance I∈ Z of an optimization problem to minimize a certain sum or bottleneck objective function γ with respect to constraint X, the optimal solution is given by $\gamma(x_0) = \min\{\gamma(x) | x \in X\}$, therefore $\prod(I)=\gamma(x_0)$. However the range Y must contain elements to represent "unbounded" and "infeasibility", too, in general. A problem $\prod$ 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 gamma. For example, given an additional threshold value y for the objective function γ we ask: does there exists a feasible solution x ∈ 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 complements of the problems in NP, i.e. for every 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 $\prod_2$ reduces to another decision problem $\prod_1$, denoted by $\prod_2 \propto \prod_1$, if there exists a polynomial time transformation function h: $Z_2 \rightarrow Z_1$ such that $\prod_2(I)$ = yes for I ∈ Z_2 if and only if $\prod_1(h(I))$ = yes for h(I) ∈ Z_1. A decision problem $\prod_1$ is called NP-complete if $\prod_1$∈ NP and for any other known decision problem $\prod_2$∈ NP we have $\prod_2$ $\prod_1$. Since, it follows from $\prod_2$ $\prod_1$ that the problem $\prod_1$ is at least as hard as the problem, $\prod_2$ it is sufficient to consider any known NP-complete problem $\prod_2$ 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 and theoretical computer science. An optimization problem is called NP-hard if the corresponding decision problem is NP-complete.

2.5. Heuristics Programming:

George Polya defines heuristic as "the study of the methods and rules of discovery and invention" (Polya 1945). This meaning can be traced to the term's Greek root, the verb *eurisco*, which means "I discover". When Archimedes emerged from his famous bath clutching the golden crown, he shouted "Eureka!' meaning "I have found it!'. In state space search, *heuristics* are formalized as rules for choosing those branches in a state space that are most likely to lead to an acceptable problem solution.

Al problem solvers employ heuristics in two basic situations:

1. A problem may not have an exact solution because of inherent ambiguities in the problem statement or available data. Medical diagnosis is an example of this. A given set of symptoms may have several possible causes; doctors use heuristics to choose the most likely diagnosis and formulate a plan of treatment. Vision is another example of an inherently inexact problem. Visual scenes are often ambiguous, allowing multiple interpretations of the connectedness, extent, and orientation of objects. Optical illusions exemplify these ambiguities. Vision systems use heuristics to select the most likely of several possible interpretations of a given scene.
2. A problem may have an exact solution, but the computational cost of finding it may be prohibitive. In many problems (such as chess), state space growth is combinatorially explosive, with the number of possible states increasing exponentially or factorially with the depth of the search. In these cases, exhaustive, *brute-force* search techniques such as depth-first or breadth-first search may fail to find a solution within any practical length of time. Heuristics attack this complexity by guiding the search along the most "promising" path through the space. By eliminating unpromising states and their descendants from consideration, a heuristic algorithm can (its designer hopes) defeat this combinatorial explosion and find an acceptable solution.

Unfortunately, like all rules of discovery and invention heuristics are fallible. A heuristic is only an informed guess of the next step to be taken in solving a problem. It is often based on experience or intuition. Because heuristics use limited information, such as the descriptions of the states currently on the open list, they are seldom able to predict the exact behavior of the state space farther along in the search. A heuristic can lead a search algorithm to a suboptimal solution or fail to find any solution at all. This is an inherent limitation of heuristic search. It cannot be eliminated by "better" heuristics or more efficient search algorithms (Garey and Johnson 1979).

Heuristics and the design of algorithms to implement heuristic search have long been a core concern of artificial intelligence research. Game playing and theorem proving are two of the oldest applications in artificial intelligence; both of these require heuristics to prune spaces of possible solutions. It is not feasible to examine every inference that can be made in a mathematics domain or every possible move that can be made on a chessboard. Heuristic search is often the only practical answer.

2.6. 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 sun-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-problem.

A dynamic programming algorithm solves every sub-problem just once and then saves its answer in a table, thereby 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 finish finding 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 optimal value.

The development of a dynamic programming algorithm can be broken in to 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 for computed information.

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

2.7. Complexity Classes:

In computational complexity theory, a complexity class is a set of problems of related complexity. A typical complexity class has a definition of the form: the set of problems that can be solved by abstract machine M using O(f(n)) of resource R (n is the size of the input) . A complexity class is the set of all the computational problems which can be solved using a certain amount of a certain computational resources. There are several complexity classes in the theory of computation. Some of the major classes are discuses

2.7.1. Class P

The complexity class P is the class of decision problems that can be solved by a deterministic machine in polynomial time. This class corresponds to an intuitive idea of the problems which can be effectively solved in the worst cases.

Example 2.1 The problem of sorting n numbers can be done in O(n2) time using the quick sort algorithm in worst case. Thus all sorting problems are in P.

2.7.2. Class NP

The complexity class NP is the set of decision problems that can be solved by a Non-deterministic Turing machine in polynomial time. This class contains many problems that people would like to be able to solve effectively, including the Boolean satisfiability problem,

the Hamiltonian path problem and the vertex cover problem. All the problems in this class have the property that their solutions can be checked efficiently.

Example 2.2 A vertex cover of an undirected graph G = (V, E) is a subset of $V' \subseteq V$ such that if (u, v) ϵ E, then uϵ V' and vϵ V' or both. That is, each edge touches at least one vertex V'. The vertex-cover problem is to find such a vertex cover of minimal cardinality. This problem is in NP.

2.7.3. NP-Complete

In computational complexity theory, the complexity class NP-complete (abbreviated NP-C or NPC, NP standing for Nondeterministic polynomial time) is a class of problems having two properties:

- Any given solution to the problem can be verified quickly (in polynomial time); the set of problems with this property is called NP.
- If the problem can be solved quickly (in polynomial time), then so can every problem in NP.

2.7.4. NP-Hard

NP-hard (nondeterministic polynomial-time hard), in computational complexity theory, is a class of problems informally “at least as hard as the hardest problems in NP”. A problem H is NP-hard if and only there is an NP-complete problem L that is polynomial time Turing-reducible to H. In otherworld’s, L can be solved in polynomial time by an oracle machine with an oracle for H. Informally we can think of an algorithm that can call such an oracle machine as subroutine for solving H, and solves L in polynomial time if the subroutine call takes only one step to compute.

2.7.5. P=NP Question

The question of whether NP=P (can problems that can be solved in nondeterministic polynomial time also always be solved in deterministic polynomial time ?) is one of the most important open question in theoretical computer science and ultra modern mathematics because of the wide implications of a solution. If the answer is yes, many important problems can be shown to have more efficient solutions that are now used with reluctance because of

unknown edge cases. These include various types of integer programming in operations research, many problems in logistics, protein structure prediction in biology, and the ability to find formal proofs of pure mathematics theorems. The P=NP problem is one of the Millennium prize problems proposed by the Clay Mathematics Institute the solution of which is a US$1,000,000 prize for the first person to provide a solution .

2.7.6. NP-Incomplete

Incomplete problems are those neither in NP that are neither NP-complete nor in P. In other words, incomplete problems can neither be solved in polynomial time nor are they hard problems. It has been shown that if P=NP is found false then there exist NP-incomplete problems.

2.7.7. Co-NP

Co-NP is the set containing the complement problems (i.e. problems with the yes/no answers reversed) of NP problems. It is believed that the two classes are not equal; however it has not yet been proven. It has been shown that if these two complexity classes are not equal, then it follows that no NP-Complete problem can be in co-NP and no co-NP-Complete problem can be in NP.

2.8. 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 bounds methods. Here, we give a brief sketch of these combinatorial optimization problems and also discuss some of the methods.

2.8.1. 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 $c_1x_1+c_2x_2+\ldots\ldots+c_nx_n$. (2.1)

Subjected to:

$$a_{11}x_1+a_{12}x_2+\ldots\ldots +a_{1n}x_n \leq b_1$$

$$a_{21}x_1+a_{22}x_2+\ldots\ldots +a_{2n}x_n \leq b_2$$

.

. +(2.2)

.

$$a_{m1}x_1+ a_{m2}x_2+ \ldots + a_{mn}x_n \leq b_m$$

$$x_j \geq 0 \text{ for } j=1,2,\ldots\ldots n$$

The objective is to minimize the costs. The c_1, $c_{2, \ldots\ldots,}c_n$ vector is referred to as the cost vector . The variable x_1, x_2 ,….., x_n have to be determined so that the objective function $c_1x_1+c_2x_2+\ldots\ldots+c_nx_n$ 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 non-linear program (NLP) is a generalization of a linear program that allows the objective function and / or the constraints to be non-linear 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.

2.8.2. 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 [24].

Let us consider a graph G = (V, E) together with a fixed matching M of G. Edges in M are called 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 Y U, E) within n= |V| ≤ |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] [14,24] gives necessary and sufficient conditions for the existence of a matching with cardinality n.

Theorem 2.8.2.1 ([14, 24]) Let G= (V Y U, E) be a bipartite graph within |V|≤|U| = m. Then there exists in G a matching with cardinality n if and only if | N(X) |≥ |X| for all X ⊆V.

A maximum matching M in a bipartite graph G= (V Y U, E) can be calculated in O(min(|V|,|U|, |E|) time.

Algorithm 2.8.2.1 The Bipartite Algorithm

Input: A bipartite graph B= (V_1 V_2, E),

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

begin

for all v ∈ V_1 V_2 do mate [v]=0; (comment: initialize)

State: begin

for all v ∈ V_1 do exposed[v]=0;

A= ; (comment: begin construction of the auxiliary graph (V, A))

for all [v,u] ∈ E do

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

if mate[u]≠v then A=A (V, mate[u]);

Q= ;

```
        for all ∈ V1 do if mate[v] then Q=Q    {v}, label[v]=0;
    while Q ≠    do
        begin
        let v be a node in Q;
        remove v from Q;
        if exposed[v] ≠ 0 then argument(v), go to state;
        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

2.8.3. Assignment Problem

Consider the complete bipartite graph, G= (V_1 V_2, $V_{1\times}$ V_2). Assume w.l.o.g that n=|V_1|≤|V_2|=m. Associated with each arc (i, j) there is a real number c_{ij}. An assignment is given by one-to-one mappingφ: V_1 V_2 . The assignment problem is to find an assignment φ 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:

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

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

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

We describe the Hungarian method [24], and use the following terminology and notations:

A label of vertices in a graph G=(V, E) is an array with |V| entries representing the predecessor vertex of all vertices. The label of a vertex v ∈ V is denoted by label[v]. To represent the current matching in the complete bipartite graph G=(VYU, E), we use the array mate having 2n entries where mate[w] for any vertex w ∈ V Y U denotes the vertex w' which is the mate of w. For any v∈ 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=1,2,….,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 2.8.3.1 [24] the Hungarian Method

Input: An n × n, matrix $[c_{ij}]$ of non-negative integers.

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

begin

for all v_i V do mate $[v_i]$:=0, α_i =0;

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

(comment: initialize)

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

begin

A:=∅;

for all v∈V do exposed[v]:=0;

for all u∈U do exposed[u]:= ;

for all v_i, u_j 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 { (v_i, mate $[u_j]$)};

(comment: construct the auxiliary graph)

Q:= ;

for all $v_i \in V$ do

if mate[v_i]=0 then

begin

if exposed[v_i]≠o then augment (v_i), go to endstage:

Q:=Q {v_i};

label[v_i] :=0;

for all u_k∈U do

if 0 < c_{jk} –α_j – β_k < slack[u_k] then slake[u_k]: = c_{jk} - α_j – β_k, nhbor[u_k]: =v_j;

end;

end

modify;

go to search

endstage: end

end

procedure modify

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

begin

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

for all v_i∈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] -2 θ_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 {mate[u]},A:=A 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

Theorem 2.8.3.1 The Algorithm 2.8.3.1 correctly solves the assignment problem for a complete bipartite graph with 2n nodes in $O(n^3)$ time.

Chapter 3

3. Scheduling Problems

In this chapter, the basic formulations of the scheduling problems are described. The classification of scheduling problems mentioned in this chapter follows the notation used in [8].

Scheduling problems are encountered in all types of systems, since it is necessary to organize AND/OR distribute the work between many entities. A definition of scheduling problem and its components are described in different literature in different way.

"Scheduling is to forecast the processing of a work by assigning resources to tasks and fixing their start time. The different components of scheduling problem are the tasks, the potential constraints, the resources and the objective function. The tasks must be programmed to optimize a specific objective function. Of course, often it will be more realistic in practice to consider several criteria", *Carlier and Chretienne*[9].

A schedule is an allocation of one or more time intervals to each job one or more machines. A scheduling is called optimal if it minimizes 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. 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 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.

3.1. Schedules and their representation

Let there be m number of machines, M_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 [8]. A schedule is often represented by Gantt chart. Gantt chart can be machine oriented or job oriented.

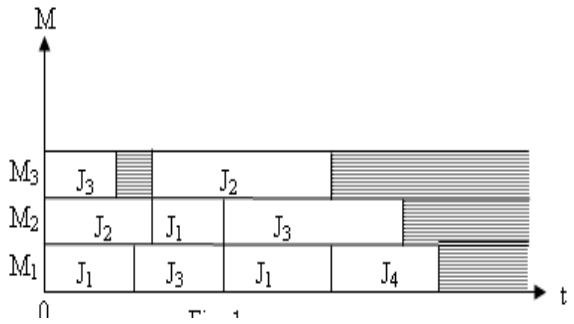

Fig 2: Machine Oriented Gantt Chart [8]

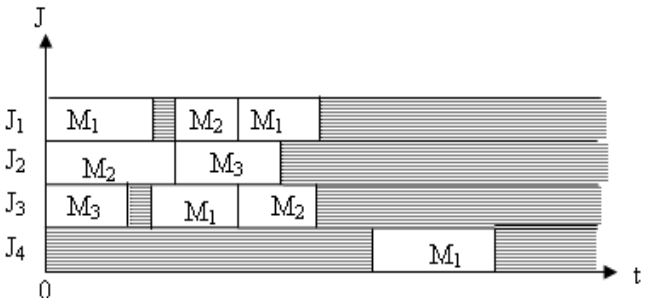

Fig 3: Job Oriented Gantt Chart [8]

3.2. Some Application Area of Scheduling

The application of scheduling is seen in diversified sectors of activity. Some application areas in computer science and engineering are described below.

3.2.1 Production Scheduling

Scheduling is an important tool for manufacturing and engineering, where it can have a major impact on the productivity of a process. In manufacturing, the purpose of scheduling is to minimize the production time and costs, by telling a production facility what to make, when, with which staff, and on which equipment. Production scheduling aims to maximize the efficiency of the operation and reduce costs.

Production scheduling tools greatly outperform older manual scheduling methods. These provides the production scheduler with powerful graphical interfaces which can be used to visually optimize real-time workloads in various stages of production, and pattern recognition allows the software to automatically create scheduling opportunities which might not be apparent without this view into the data. For example, an airline might wish to minimize the number of airport gates required for its aircraft, in order to reduce costs, and scheduling software can allow the planners to see how this can be done, by analyzing time tables, aircraft usage, or the flow of passengers.

3.2.2 Scheduling in operation system

Scheduling theory is excessively used in 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. Some basic algorithms used in OS for uni-processor computer are:

a. First Come First Server (FCFS):

At any instance when the 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.

b. Shorts Job First (SJF):

At any instance when the machine is idle, select the available job having shortest expected processing time. In this case of tie, the FCFS is used.

c. 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.

d. 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 another process. If the process has finished before the quantum has elapsed, the CPU switching is done when the process blocks, of course.

3.2.3 I/O Scheduling

I/O scheduling is the term used to describe the method computer operating systems decide the order that block I/O operations will be submitted to the disk subsystem. I/O scheduling is sometimes called 'disk scheduling'. I/O scheduling usually has to work with hard disks which share the property that there is long access time for requests which are far away from the current position of the disk head (this operation is called a seek).To minimize the effect this has on system performance, most I/O schedulers implement a variant of elevator algorithm which re-order the incoming randomly ordered requests into the order in which they will be found on the disk.

3.2.4 Timetable Scheduling

In timetable scheduling problems, examination subjects must be slotted to certain times that satisfy several of constraints. They are NP-completeness problems, which usually lead to satisfactory but suboptimal solutions. Along with this, timetable scheduling problems concern all educational establishments or universities, since they involve timetabling of courses assuring the availability of teachers, students and classrooms. These problems are just as much the object of studies.

2.3.5 Project Scheduling

Project Scheduling problems comprise a vast literature. We are interested more generally in problems of scheduling operations which use several resources simultaneously (money, personnel, equipment, raw material etc.), these resources being available in known amounts. In other words, we deal with the multi-resource scheduling problem with cumulative and non-renewable resources.

3.3.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 3.2.1: $1|r_i: pmtn||L_i|max$

Is the problem of finding a preemptive schedule on one machine for a set of n-job 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.

3.4.Benefit 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 manufacturing environment are listed below:

1. **Set up times are significantly reduced in the warehouse:**

Cutting down 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 area may need improvement.

2. **The flows of goods from warehouse to shelves are improved:**

Having employees focused on specific area 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 task 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 situation 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 hour:**

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.

3.5.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.

2. 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 form one format into another, for example byte code 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.

3. Just-in-Time Sensor Networks:

Many areas of research in sensor networks deal directly with the ability to adapt to changing conditions. This has resulted in the ability to dynamically change attributes such as routing paths, MAC protocols, program images, and duty cycling. Yet there are several sensor network optimizations and adaptations that cannot be accomplished through software changes alone. The lack of hardware capabilities or poor geographic layouts of nodes are characteristics that create upper bounds on the ability of software protocols to optimize communication and coverage capabilities. Specially, a sensor network is deployed (either randomly or placed in a specific location), sits statically for several months collecting data, and adapts itself through various protocols. Yet this often overlooks potential optimizations gained by adding motes to the network on-demand and within seconds. This introduces a shift in the traditional outdoor, static sensor network paradigm by considering the possibilities and limitations of a rapid, just-in-time deployment.

4. Just-in-Time to Enable Optical Networking For Grids

Many of today's computer and data intensive e-science applications are looking to Grid based technologies to meet their high demands. Until recently, the Grid community focused primarily on maximizing the availability, sharing, and utilization of resources such as CPU power and storage. Now, many in the Grid community are starting to regard the network as another vital Grid resource, to be used to provide large, fast data flows with minimal latency and jitter. MCNC Research and Development Institute and North Carolina State University (NCSU) have developed a Just-in-Time control plane, signaling scheme, and various software and hardware components that are synergistic with these needs. This includes an overview of the Just-In-Time control plane and Grid JIT service that has been developed for optical networks and describes several related projects.

Chapter 4

4. Mathematical Model Formulation

When production system consists of constant rate of usage of all parts, Just-in-time systems are suitable. However, the variability between the actual and the ideal production due to integral nature of production appears. This leads the sequencing problem to minimize the variation so that a balanced sequence of diversified products that minimizes the earliness and tardiness penalties could be obtained in a reasonable time. Before starting problem formulation, we assume that the systems have sufficient capacity, negligible switch-over cost and production in unit time. Kubiak [2] refers to single level problem as Product Rate Variation (PRV) problem and multi level problem as Output Rate Variation (ORV) problem.

4.1. The Output Rate Variation Problem (ORVP) Formulation

The production system consists of hierarchy of several distinct production levels such as products, sub-assemblies, component parts, raw material, etc. A mixed-model multi-level problem falls under ORV problem. Consideration of part demand rate reduces problem into the ORV problem.

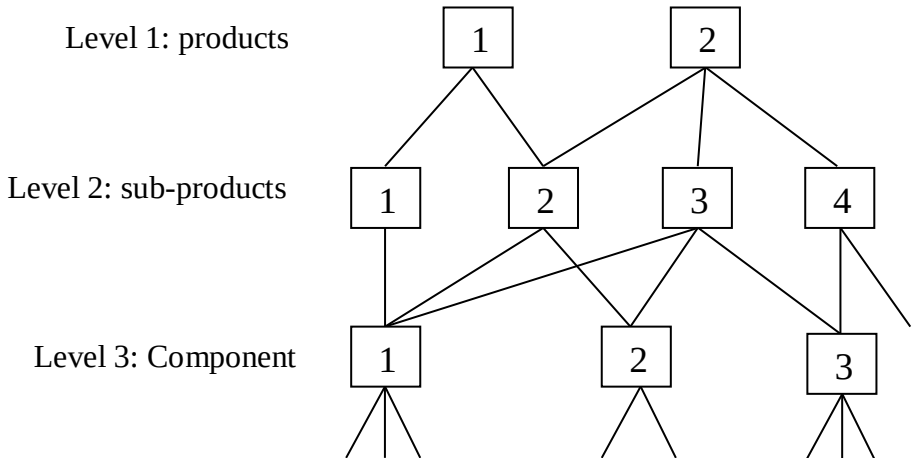

Figure 4: Mixed-Model Multi-Level Production System [20]

4.2.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, $d_2,\ldots,d_n$ 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 is produced in each time period. A schedule is called an ideal schedule if at each time period; 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 as product as balanced as possible by keeping the quantity of each product of each product used by assembly line as constant as possible. The objectives formulations of PRVP are given as follows.

For each i=1,2,…n; let f_i, be a unimodel convex function with minimum in $f_i(0)=0$ (i=1,2,………V). Then minmax and minsum PRVP can be formulated as follows:

$$Minimize\left[F = max_{i,k} f_i(x_{i,k} - kr_i)\right] \ldots\ldots\ldots\ldots\ldots\ldots \quad (4.1)$$

$$Minimize\left[F = \sum_{k=1}^{d}\sum_{i=1}^{n} f_i\left(x_{i,k} - kr_i\right)\right] \ldots\ldots\ldots\ldots\ldots\ldots \quad (4.2)$$

Subject to the contraints;

$$\sum_{i=1}^{n} x_{1,k} = k, \qquad k = 1,2,\ldots\ldots\ldots,D \qquad (4.3)$$

$$x_{i,D} = d_i, \qquad i = 1,2,\ldots\ldots\ldots,n \qquad (4.4)$$

$$x_{i,0} = 0, \qquad i = 1,2,\ldots\ldots\ldots,n \qquad (4.5)$$

$$x_{i,k} - x_{i,k-1} \geq 0, \qquad i = 1,2,\ldots\ldots\ldots,n;\ k = 1,2,\ldots\ldots.D, \qquad (4.6)$$

$$x_{i,k} \geq 0, integer \qquad i = 1, 2, \dots\dots\dots\dots\dots, n; k = 1,2, \dots\dots\dots D, \tag{4.7}$$

Here, equality (4.3) means that *k* parts (copies) have to be produced in first k time periods ; equality (4.4) indicates that all demands must be fulfilled within D time periods; inequality (4.6) 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 here maximum deviation, where as 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 $i = 1, 2, \dots, n$. Under this case,

(4.1) takes the form:

$$F^a_{max} = \max_{1 \leq i \leq n; 1 \leq k \leq D} |x_{i,k} - kr_i| \tag{4.8}$$

And (4.2) takes from:

$$F^a_{sum} = \sum_{k=1}^{D} \sum_{i=1}^{n} | x_{i,k} - kr_i | \tag{4.9}$$

Case 2: if $f_i(x) = x^2$ under which (4.6) takes the form:

$$F^s_{max} = \max_{1 \leq i \leq n; 1 \leq k \leq D} (x_{i,k} - kr_i)^2 \tag{4.10}$$

And (4.7) takes from:

$$F^s_{sum} = \sum_{k=1}^{D} \sum_{i=1}^{n} (x_{i,k} - kr_i)^2 \tag{4.11}$$

For simplicity, we introduce the abbreviation that problem F^a_{max} means the problem defined by (4.1) with objective function (4.8) under the constraints (4.3), (4.4), (4.5), (4.6) and (4.7); problem; F^a_{sum} means the problem defined by (4.2) with objective function (4.9) under the constraints (4.3), (4.4), (4.5), (4.6) and (4.7). We find the pseudo-polynomial algorithms separately for the problem $F^a_{max}, F^s_{max}, F_{sum}$.

The mixed-model JIT sequencing problems are repressively denoted by maximum deviation Just-in –Time (MDJIT) problems. Similarly the abbreviated from MMJIT refers to mixed-model Just-in-Time and MMJITSP refers to MMJIT sequencing problem.

4.3. Pegged ORV Problem:

Steiner and Yeomans [25] shows that the ORV problem under the puffing assumption can be reduced to weighted PRV problem. Under the pegging assumption, part of output *i* at production levels which fed the level *l* 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 *l* 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. Similarly with the same assumption the weighted max-abs ORV problem can be formulated.

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

$$= \max_{p,i,l,k}\{W_{il}\, t_{ilp}\left|x_{plk} - kr_{p1}\right|\}$$

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

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

Chapter 5

5. Solution Procedure for PRV Problem

5.1. Heuristic Approach

The ORVP is computationally more challenging. The problem $\tilde{G}$ is NP-hard in the ordinary sense as the NP-hard scheduling problem, around the shortest job reduces to the ORVP and the problem $\widetilde{F_a}$ with only two levels is NP-hard in the strong sense as the strongly NP-hard 3-partition problem transforms into the ORVP in pseudo- polynomial time. However, a number of heuristics gives rise to suboptimal solutions.

The goal chasing methods GCM I and GCM II used in Toyota, construct a sequence filling one position at a time from first slot to the last one. The variability is considered at the sub-assembly level whereas the variability at the product level is ignored. 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 in GCM II. GCM I and GCM II are my topic. A myopic polynomial heuristic, extended goal chasing method (EGCM) that considers more levels, adopts GCM I and GCM II as a special case. The myopia lies in the fact that it only takes one step. Taking two steps into account, the myopia can be reduced.

Three algorithms and two heuristics are formulated in [21]. The algorithm 1 and the algorithm 3 with heuristic 1 (MA3H1) consider the product rates, not the parts usage rates. It is a one-stage myopic heuristic with complexity $O(nD)$. The algorithms may not yield feasible sequence but if feasible it is optimal, too. The algorithm 3 with heuristic 2 (MA3H2) is the improved two-stage heuristic with complexity $O(n^2D)$. MA3H2 is of highest quality for feasible solutions among GCM I, GCM II, MA3H1 and MA3H2.

Time spread (TS) heuristic employs similar procedure as GCM1 with function in which time required to assemble products are applied. Comparison of different methods through simulation analysis show that TS and MA3H2 seem to be effective [25].

Inman and Bulfin's earliest due date (EDD) rule based on ideal time of production of each product [13], Ding and Cheng's two-stage algorithm that minimizes the variation of the two

stages and MA3H2 heuristic obtain good solutions. Modified forms of these, with appropriate weights, are useful alternatives for frequent updates of sequencing.

A local search heuristic that attempts to swap the order of assembly of a pair of products provides near-optimal sequence for realistic-size problems in a reasonable time. It may be extended considering release date and due date constraints.

The problem with a bicriterion objective of part usage and setup time has inversely correlated objective values. An efficient frontier, where simultaneously maximization of feasibility and minimization of setup is desired, is exploited. Such frontier is explored using heuristics such as tabu search, simulated annealing, genetic algorithm, ant colony optimization approach, beam search heuristic, artificial neural network etc.

Suboptimal solutions using heuristics, for example, tabu search and branch and bound to the problem with the objective for parts usage and work load [26], parts usage and line length, parts usage and line stoppage, [15] can be obtained.

5.2. Dynamic programming

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

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

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

Where w_{il} be a weighting factor which reflects the relative importance of balancing the sequence for part i at level l.

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

$$x_{ilk} - y_{lk} r_{il} = \sum_{p=1}^{n_l} t_{ilp} x_{plk} - r_{il} \sum_{p=1}^{n_l} \sum_{i=1}^{n_l} t_{ilp} x_{plk}$$

$$= \sum_{p=1}^{n_l} \left(t_{ilp} - r_{il} \sum_{p=1}^{n_l} t_{ilp} \right) x_{plk}$$

$$= \sum_{p=1}^{n_l} \delta_{ilp} x_{plk}$$

Where ; $\delta_{ilp} = t_{ilp} - r_{il} \sum_{p=1}^{n_l} t_{ilp}$

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

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

Where $\gamma_{ilp} = w_{il} \delta_{ilp}$ is the measure of the weighted deviation in the usage of part i at level l from the proportional usage per unit of product p. Let $\Gamma = (\gamma_{ilp})_{n \times n_1}$ 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 $\left(\sum_{m=1}^{l-1} n_m + 1\right)^{th}$ row and the p^{th} column of the matrix Γ. The maximum norm of a vector $\mathrm{a} = (\mathrm{a}_1, \ldots., \mathrm{a}_\mathrm{n})$ is defined to be $||\mathrm{a}||_1 = \max_{1<i<n} \{|\mathrm{a}_\mathrm{i}|\}$. Then the objective function G'^{w}_{max} can be written as $G'^{w}_{max} = \max_k ||\Gamma \mathrm{X}_k||_1$, where $\mathrm{X_k} = (\mathrm{x_{ilk}}, \ldots., \mathrm{x_{n_1 lk}})$ is the cumulative, level 1 production vector through the first k stage. Let the demands vector at level 1 be $d = d_{11}, \ldots\ldots, d_{n_1 1} = d_1, \ldots\ldots, d_{n_1}$ and the states in a sequence be $X = (x_1, \ldots\ldots, x_{n_1})$ with $|X| = \sum_{i=1}^{n_1} x_i$, $\mathrm{i} = 1, \ldots., \mathrm{n_1}$ where x_i is the cumulative production of product $i, x_i \leq d_i$. Let $e_i = (0, \ldots, 1, \ldots, 0)$ 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 stage X. The norm $||\Gamma \mathrm{X}||_1$ represents the maximum deviation of actual production from desired one over all products and parts in stage X at $k = |X|$.. The following DP recursion holds for $\phi(X)$.

$\phi(\emptyset) = \phi(X : X = 0) = 0$

$\phi(\emptyset) = \phi(x_1, \ldots\ldots, x_{n_1}) = \min\{\max\{\phi(X - e_i), ||\Gamma X_k||_1, \} : i = 1, \ldots\ldots, n_1; x_i \geq 1\}$

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

Theorem 5.2.1

The DP recursion solves the JIT scheduling problem in

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

5.3. Assignment Method

The problem can be solved pseudo-polynomials transforming the problem into an equivalent assignment problem. Calculation of the assignment costs is based on the level curves $f_i(j - kr_i), j = 0,1, \ldots, d_i;\ \ k = 0,1, \ldots D$ and the positions in which each copy j of product i, $j = 0,1, \ldots, d_i;\ \ i = 1, \ldots n$ is sequenced.

If all copies of product i are sequenced at their ideal positions $Z_{ij} = \left\lceil\frac{2j-1}{2r_i}\right\rceil$, the ceiling of the unique crossing point satisfying $f_i(j - kr_i) = f_i(j - 1 - kr_i),\ j = 0,1, \ldots, d_i$, the product i will contribute the cost to the total cost in $f_j f_i(j - kr_i)$ of the solution and an optimal sequence is obvious. Sequencing the products at their ideal positions minimizes the problems F and G, however, leads to infeasibility when more than one copy competes for the same ideal position in the sequence. Competition occurs in general case. Higher priority is given to j' over j whenever $j' < j$ to avoid competition and (i, j) is assigned to a position $k, k \neq \left\lceil\frac{2j-1}{2r_i}\right\rceil$.

The new assignment contributes additional cost $C_{ijk} \geq 0$ where

$$C_{ijk} = \begin{cases} \sum_{l=k}^{x_{ij}-1} \psi_{ijl} & if\ k < Z_{ij,} \\ 0 & if\ k = Z_{ij,} \\ \sum_{l=z_{ij}}^{k-1} \psi_{ijl} & if\ k > Z_{ij,} \end{cases}$$

With $\psi_{ijl} = \begin{cases} f(j - lr_i) - f_i(j - 1 - lr_i), & if\ l < z_{ij}, \\ f(j - 1 - lr_i) - f_i(j - lr_i), & if\ l \geq z_{ij} \end{cases}$

The assignment problem equivalent to the problem G is,

$$\min \sum_{k=1}^{D} \sum_{i=1}^{n} \sum_{j=1}^{d_i} C_{ijk}\, x_{ijk} \tag{3.1}$$

subject to,

$$\sum_{i=1}^{n} \sum_{j=1}^{d_i} x_{ijk} = 1,\ k = 1, \dots, D \tag{3.2}$$

$$\sum_{k=1}^{D} x_{ijk} = 1,\ i = 1, \dots, n; j = 1, \dots, d_i \tag{3.3}$$

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

Let $\chi = \{(i,j,k) | i = 1, \dots, n; j = 1, \dots, d_i; k = 1, \dots, D\}$, be the set of the assignment of (i,j) to k. A set $X \subseteq \chi$ is X-feasible if the following constraints hold.

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

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

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

Constraints c_1 and c_2 are related to the assignment problem. Constraint c_3 imposes an order on copies of a product.

Theorem 5.1

For any feasible $X \subset \chi,\ G = \sum_{(i,j,k) \in X} C_{ijk} + \sum_{i=1}^{n} \sum_{k=1}^{D} in\, f_j\, f_i(j - kr_i)$.

The result becomes an inequality without c_3. An optimal solution cannot be obtained by simply solving the assignment problem since c_3 is not the assignment type.

Theorem 5.2

If X satisfies c_1 and c_2 , then X^* satisfying c_1, c_2, c_3 with $c(X) \geq c(X^*)$ can be determined in $O(D)$ time. Moreover, each copy in the sequence s^* from X^* preserves the order that it has in the sequence from$X$.

Since there are D^2 values ψ_{ijl}, D^2 values C_{ijk} and each takes $O(D)$ time to calculate, the Hungarian method takes $O(D^3)$ time to solve the assignment problem with $2D$ nodes. The assignment can be made order preserving in $O(D)$ time. Hence, an optimal solution to the problem G can be obtained in $O(D^3)$ time. A number of algorithms solve the assignment problem of the problem G [24].

The approach for the problem G is applicable in every l_p- norm and particular to l_∞- norm, see [21].

The corresponding assignment problem equivalent to the problem *F* is

$$\min \sum_{k=1}^{D} \sum_{i=1}^{n} \sum_{j=1}^{d_i} B_{ijk}\, x_{ijk} \qquad (3.4)$$

subject to the constraints (3.2) and (3.3) where

$$B_{ijk} = \max\{f_i(j-1-(k-1)r_i)\,, f_i(j-kr_i)\},\ i = 1, \dots, n;\ \ j = 1, \dots, d_i;\ \ k = 1, \dots, D.$$

The assignment costs grow to the left and to the right from the ideal positions in the assignment matrix, [3]. One ideal position exists in each row of the matrix; however, there exist two ideal positions in the case of a competition.

The problem is solved by means either of specific bottleneck assignment algorithms or as a sequence of assignment problem with some modifications such as use of a binary matrix instead of the bottleneck assignment matrix and application of bisection search to find the optimal bottleneck value [3]. Optimal solution can be obtained in $O(D^3)$ time.

The bottleneck assignment costs B_{ijk} for which $f_i(x_{ik} - kr_i) < 1, i = 1, \dots, n; k = 1, \dots, D$, can be calculated in time $O(nD)$, but it remains open whether the problem can be solved in $O(nD)$. If it exists, it would be better than the existing solution procedures [17].

A cyclic sequence substantially reduces the time complexity. Such sequences exist in the problem G_s [47, 49]. The cyclic sequences are optimal, too. A concatenation s^m of m copies of an optimal sequence s for the instance $(d_1, \dots, d_n)$ of the problem G_a is optimal for $(md_1, \dots, md_n), m \geq 1$. It builds a sequence for a longer time horizon. Such a sequence can be found under the assumption $f_i = f \forall_i, i = 1, \dots, n$ where f is convex and symmetric with minimum 0 at 0.

5.4. Perfect Matching Method

The problem F_a is solved by reducing it to an order-preserving perfect matching problem via single machine scheduling release/due date decision problem. The perfect matching problem is constructed in a V_1-convex bipartite graph $G = (V_1 \cup V_2, \mathcal{E}$ with $V_2 = \{(i,j) | \ i = 1, \dots, n; j = 1, \dots d_i\}$, set of the j^{th} copy of product $i, V_1 = \{1, \dots, D\}$, the starting times and the edge set $\mathcal{E}$ with the earliest starting time $E(i,j)$ and the latest starting time $L(i,j)$ for (i,j) defined as $\mathcal{E} = \{\big(k, (i,j)\big) | k \in [E(i,j), L(i,j) \subseteq V_1$. For a given bound B and the level curves $|j - kr_i|$, $i = 1, \dots, n; j = 0,1, \dots, d_i; k = 0,1, \dots, D$, the values $E(i,j)$ and $L(i,j)$, $i = 1, \dots, n; j = 1, \dots, d_i$, are calculated in time $O(D)$ as the unique integers $E(i,j) = \left\lceil \frac{j-B}{r_i} \right\rceil$ and $L(i,j) = \left\lfloor \frac{j-1-B}{r_i} + 1 \right\rfloor$[7].

A modified version of earliest due date (EDD) rule with O (|E|) in V_1-convex bipartite graph $(V_1 \cup V_2, \mathcal{E})$ finds an order-preserving perfect matching for the upper bound $B \leq 1$.

A stronger upper bound has been obtained for the problem F_a. If B^* be any optimal value, then $\frac{1}{\Delta_i}\left\lfloor \frac{\Delta_i}{2} \right\rfloor \leq B^* \leq 1 - \frac{1}{D}$ where, $\Delta_i = \frac{D}{\gcd(d_i, D)}, i = 1, \dots, n$ [7], and $B^* \leq 1 - \frac{1}{2(n-1)}$. Therefore, it holds $\frac{1}{\Delta_i}\left\lfloor \frac{\Delta_i}{2} \right\rfloor \leq B^* \leq 1 - \max\left\{\frac{1}{D}, \frac{1}{2(n-1)}\right\}$ for $n \geq 2$. The optimal value B^* cannot be less than $\frac{1}{2}$ for even Δ_i since $\frac{1}{\Delta_i}\left\lfloor \frac{\Delta_i}{2} \right\rfloor = \frac{1}{2}$ and cannot be less than $\frac{1}{3}$ for odd Δ_i since $\frac{1}{3} \leq \frac{1}{\Delta_i}\left\lfloor \frac{\Delta_i}{2} \right\rfloor < \frac{1}{2}$. It is natural to seek instances with optimal value less than $\frac{1}{2}$.

It has been shown that only the standard instance i.e. the instance with $d_1 \leq \cdots \leq, d_n$, $\gcd(d_1, \ldots, d_n) = 1$ has optimal $B = \frac{2^{n-1}-1}{2^n-1} < \frac{1}{2}$ if and only if $d_i = 2^{i-1}, i = 1, \ldots, n$ [35, 14]. It came into existence as the small deviations conjecture [13]. If optimal $B = \frac{1}{2}$, all products must be sequenced in the ideal position optimal $B = \frac{2j-1}{2r_i}$ for each optimal j, which happens if optimal d_n is divisible by each optimal $d_i, i = 1, \ldots, n-1$. This geometric proof exploits a natural symmetry of regular polygons inscribed in a circle of circumference *D* such that each polygon corresponds to a different product having optimal d_i corners for product optimal i at optimal $\left\lceil \frac{2j-1}{2r_i} \right\rceil$ points on the perimeter of the circle. Consequently, *n* demands are the first non-negative powers of 2.

The small deviations conjecture is shown to be true as a consequence of the Fraenkel's conjecture for symmetric case using a fact that a solution to the problem F_a with $d_i = 2^{i-1}$ for $i = 1, \ldots, n, n > 2$, is periodic, symmetric and balanced word. The Fraenkel's conjecture for symmetric case states that a periodic, symmetric and balanced word with $r_1 < \cdots < r_n, n > 2$, exists if and only if $r_i = \frac{2^{i-1}}{2^n-1}$.

A δ-balanced word on a finite set $\{1, \ldots, n\}$ is an infinite sequence $s = s_1 s_2 \ldots$ with $s_i \in \{1, \ldots, n\}$ such that every two subsequences of equal length consist of only those letters whose numbers of occurrences in each subsequence differ by at most a positive integer δ. Note that 1-balanced word is a balanced word. Consider a finite word W on $1, \ldots, n$ of length D with d_i occurrences of a letter i and $r_i = \frac{d_i}{D}$, the rate of letter i with $r_1 \leq \cdots \leq r_n$. W is said to be symmetric if $W = W^R$, a mirror reflection of W. An infinite word w is periodic if $w = WW \ldots$ for some W.

For a sequence s with maximum deviation *B*, any infinite periodic word w, with period s is 1-balanced, 2-balanced and 3-balanced on each product i, if $B < \frac{1}{2}, B < \frac{3}{4}$, and $B < 1$, , respectively.

Unfortunately, the 1-balanced words are unlikely for most rates to exist. There exists an optimal sequence for the problem F_a in the set of all 3-balanced words. However, it remains unresolved whether there always exist 2-balanced word that is optimal for the problem F_a.

The challenging problem of balanced words in practice is to construct an infinite periodic sequence over a finite set of letters with given rates and distributed as evenly as possible.

Though, only the instance $d_i = 2^{i-1}, i = 1, \dots, n, n > 2$, has $B < \frac{1}{2}$ for the problem F_a, for $n = 2$, infinitely many instances with $B < \frac{1}{2}$ exist i.e. the optimal value of the problem F_a, is less than $\frac{1}{2}$ if and only if one of demands d_1 or d_2 is odd and the other even [14]. A sequence with distances $\left\lceil \frac{D}{d_1} \right\rceil$ and $\left\lceil \frac{D}{d_1} \right\rceil$ for product 1 with demand d_1 and $\left\lceil \frac{D}{d_2} \right\rceil$ and $\left\lceil \frac{D}{d_2} \right\rceil$ for product 2 with demand d_2 is optimal for two product case. This procedure solves both the problem F_a and the response time variability problem for $n = 2$, which is not true in general for $n > 2$. The response time variability problem minimizes the variability of time for which clients, events, jobs or products wait for the next turn in obtaining the resources necessary for their advance. This problem intends to utilize the resources so as to ensure a fair sharing of common resources between the products which requires to be evenly distributed such that the occurrences in any two consecutive items of the same product is to keep at constant distance as much as possible all the time. The general case of the problem is NP-hard. This result naturally motivates to look at other possible common solutions with respect to different objectives.

The EDD algorithm matches each ascending $k \in V_1$ to the unmatched (i, j) with the smallest $L(i, j)$. Since $E(i, j)$ and $L(i, j)$ are strictly monotonic increasing for consecutive copies of each product and $E(i, j + 1)$ cannot be less than $L(i, j)$ with $B < 1$ the algorithm ensures the perfect matching to be order-preserved.

The weighted problem can analogously be reduced to the order-preserved perfect matching problem. Heavy weightage for particular copies of a product restricts the time window $[E(i, j), L(i, j)$ and increases the separation of consecutive copies of that product in the sequence. $E(i, j)$ And $L(i, j)$ are calculated as the integers $E(i, j) = \left\lceil \frac{jw_i - B}{r_i w_i} \right\rceil$ and $L(i, j) = \left\lfloor \frac{(j+1)w_i + B}{r_i w_i} + 1 \right\rfloor$.

An order-preserved perfect matching gives rise to a feasible solution.

The necessary and sufficient condition for a feasible solution to the problem F_a is the following.

Theorem 5.3

The problem F_a *has a feasible solution if and only if for all* $k_1,\ k_2 \in \{1, \dots, D\}$ *with* $k_1 < k_2$ *and,* $\sum_{i=1}^{n} max(0, \lfloor (k_2 r_i + B \rfloor - \lceil (k_1 - 1) r_i - B \rceil) \geq k_2 - k_1 + 1$
and $\sum_{i=1}^{n} max(0, \lceil (k_2 r_i + B \rceil - \lfloor (k_1 - 1) r_i + B \rfloor) \leq k_2 - k_1 + 1.$

The theorem tests the feasibility of B in time $O(nD^2)$ though less efficient than $O(D)$ time and of a pair $(k_1,\ k_2),\ k_1,\ k_2 \in \{1, \dots, D\}$ in $O(n)$ time [38].

The perfect matching using a certain bound obtained through a bisection search in the interval $\left[1 - r_{max}, 1 - \frac{1}{D}\right]$ yields an optimal sequence in $O(D\ log\ D)$ time. The lower bound $1 - r_{max}$ is tight.

Since the deviations are multiples of $\frac{1}{D}$ and the upper bound is $1 - \frac{1}{D}$ the bound for the optimal value can be only $B = \frac{k}{D}$ with $k \in \{1, \dots, D-1\}$ [42]. This fact can be implemented to calculate possible optimal values for the problem F_a only for these values. The optimal sequences of an instance $d_1 = 2, d_2 = 3, d_3 = 5$ obtained at bound $B = \frac{1}{2}$ are 3-2-1-3-2-3-3-1-2-3 and 3-2-1-3-3-2-3-1-2-3, here the 5^{th} and the 6^{th} positions are swapped.

An optimal sequence for the weighted problem is obtained as follows:

Theorem 5.4

An optimal sequence for the weighted problem can be determined when a bisection search is performed in the interval $[min_i w_i(1 - r_i), max_i w_i]$ *in exact pseudo-polynomial time* $O(D \log(D\Phi max_i(w_i)))$, *where* Φ *is a positive integer constant that depends on the problem data.*

The exact complexity of the problem F_a still remains open. The problem F_a has been proved to be *Co-NP* but remains open whether it is *Co-NP*-complete or polynomially solvable [13]. Observation of the input size $O(\sum_{i=1}^{n} \log d_i) = O(n \log D)$ and the involvement of variables nD and $O(nD)$ constraints in the model indicate that an expectation of a polynomial algorithm for this problem seems far from trivial.

There exists cyclic optimal sequence for the problem F_a. Let u_i be a factor of D and d_i with $d_i = u_i\, v_i$ for product i. Each copy of product i is labeled as $(e-1)v_i + j$ where $e = 1, \dots, u_i$ and $j = 1, \dots, v_i$, the e^{th} period of copies of product i that consists of v_i copies of product i. There will be u_i such periods for each product. If all of one period's early (late) starting times are calculated, then the early and the late starting times for all copies in all periods can be calculated from these values. When $u_i = \gcd(d_i, D) = 1$, the time required to calculate the starting times can be reduced by a factor of 2.

Theorem 5.5

If $u = gcd(d_i, \dots, d_n)\,, i = 1, \dots, n$, then the problem F_a consists of u repetitions of the optimal sequence.

The problem G can be represented as a complete convex bipartite weighted graph on $V_1 = \{1, \dots, D\}$. Since each (i,j) can be produced at any instant k, it is clear that $E(i,j) = 1$ and $L(i,j) = D$. The cost C_{ijk} for (i,j) at k is taken as the weight for the edge $(k,(i,j))$. The problem is to find a perfect matching with minimum sum of the weights.

Theorem 5.6

A sequence s for the problem G is optimal if and only if there is a minimum weight perfect matching M with a weight function $w: V_1 \cup V_2 \to R$ such that $w_k + w_{i,j} \le C_{ijk} \forall \big(k,(i,j)\big) \in \mathcal{E}$ and $\sum_{k \in V_1} w_k + \sum_{(i,j) \in V_2} w_{i,j} = \sum_{k,(i,j) \in M} C_{ijk} = s$.

Let us say an incomplete convex bipartite graph on V_1 if weights are attributed to only those edges $(k,(i,j))$ of which $k \in [E(i,j), L(i,j)]$ with $B \le 1$. This substantially reduces the number of weights to be calculated. A 1-bounded optimal solution for the problem G, if exists, could be obtained in $O(nD^2 \log D)$ time, since $|E| \le (n+2)D$ holds for $B \le 1$ [56].

Theorem 5.7

The sequence optimal to the problem G with $f_i = f$, $\forall_i$ and $-1 \le x_{ik} - kr_i \le 1$ for the incomplete graph is also optimal to the problem G for the complete graph.

This result cannot be generalized for non-identical cost functions in $[-1, 1]$ As an example, the instance $(24, 24, 28, 28, 42, 42, 42, 48, 16)$ with the cost functions, $f_1(x) = f_2(x) =$

$\alpha_1|x|, f_3(x) = f_4(x) = \alpha_2|x|, f_5(x) = f_6(x) = f_7(x) = f_8(x) = \alpha_3|x|$, $f_9(x) = \alpha_4|x|$, $f_{10}(x) = \alpha_5|x|$, where $\alpha_1 = 168^2.8.7.\alpha_2$, $\alpha_2 = 168^2.6.6.\alpha_3, \alpha_3 = 168^2.4.2.\alpha_4, \alpha_4 = 168^2.7.\alpha_5, \alpha_5 = 1$ shows that $-1 \leq x_{ik} - kr_i \leq 1$ will not hold for some positions.

But the existence of such a solution is rarely possible. The question of determining minimum B such that the optimal solution to the problem G is B-bounded remains unanswered. It is shown that the upper bound on the optimal value of the problem G is nD though the bound is not tight. However, the lower bound for the problem G_s is $\sum_{i=1}^{n} \frac{D^2 - d_i^2}{12D}$. Note that a solution is said to be B-bounded or B-feasible if the deviation is less than a given bound B.

The perfect matching method can also be applied to the generalized pinwheel scheduling problem or the Liu-Layland periodic scheduling in hard real-time environments, see. The generalized pinwheel scheduling problem for n pairs of positive integers $(a_1, b_1), \dots, (a_n, b_n)$ is to find an infinite sequence $s = s_1 s_1 \dots$ on finite set $\{1, \dots, n\}$ such that $s_j \in (1, \dots, n\}, j \in N$ and any subsequence of s consisting of b_i consecutive elements of s contains i at least a_i times, $i \in \{1, \dots, n\}$. The solution procedure to the problem F_a with $B < 1$ and the rates $r_i = \frac{a_i + 1}{b_1}, i = 1, \dots, n$ yields a generalized pinwheel schedule for the instance $(a_1, b_1), \dots, (a_n, b_n)$ if $\sum_{i=1}^{n} \frac{a_i}{b_i} + \frac{1}{b_i} \leq 1$ [17]. The Liu-Layland periodic scheduling problem is to find an infinite sequence $s = s_1 s_1 \dots$ on a finite set $\{1, \dots, n\}$ such that $s_j \in \{1, \dots, n\}, j \in N$ and a preemptive and periodic job j occurs exactly C_i times on any subsequence of s consisting of T_i consecutive elements of s with $C_i \leq T_i, i \in \{1, \dots, n\}$, where C_i and T_i are the run-time and request period for job i. The solution to the problem F_a with $B < 1$ and rates $r_i = \frac{C_i}{T_i}, i = 1, \dots, n$, is a periodic schedule.

5.5. Simultaneous optimality

Study of finding solutions that minimize a number of objective functions simultaneously is useful. Such solutions not only reduce time complexity of the problem but also are more applicable in practice.

A Pareto algorithm that determines all Pareto optimal sequences for the bicriterion sequencing problem with the objectives F_a and G exist. The algorithm determines an order preserving perfect matching with $B \leq 1$. Then a minimum weight order-preserving perfect

matching with the weight C_{ijk} for the edge $\big(k,(i,j)\big), i = 1,\dots,n; j-1,\dots d_i; k = 1,\dots,D$ is determined. The corresponding production sequence is a Pareto optimal sequence. A Pareto optimal solution can be determined in $O(nD^2 \log D)$ time and all Pareto optimal solutions in $O(nd_{max}D^2 \log D)$ time.

Let S_1 be the set of all 1-feasible sequences. The two problems are S_1-equivalent if both have the same set of optimal sequences on S_1. The problems G_a and G_s on S_1 have the same cost $C_{ijk} \geq 0$ for $k \in [E(i,j), L(i,j)$ [18] and are S_1-equivalent [9]. The assumption in that the S_1-equivalence is due to symmetry and convexity of the objectives is not true. The instance $(23, 23, 1, 1, 1, 1)$ with the function

$$f(x_{ik} - kr_i) = \begin{cases} -\frac{1}{1-x^*}(x_{ik} - kr_i) - \frac{x^*}{1-x^*} & (x_{ik} - kr_i) \leq -x^* \\ 0 & -x^* \leq (x_{ik} - kr_i) \leq x^* \\ \frac{1}{1-x^*}(x_{ik} - kr_i) - \frac{x^*}{1-x^*} & x^* \leq (x_{ik} - kr_i) \end{cases}$$

is a counterexample [9], where x^* is optimal value to the problem F_a. An optimal sequence for the problem G_a in $S_1$is optimal for the problem G_s in S_1 too. With this, the problem G_s can be solved by means of solving the problem G_a in S_1. It is advantageous for the complexity since the conversion of the floating point numbers to integers of absolute penalties required is smaller in magnitude than that of the square penalties. An optimal solution in S_1 to the problem G_s may not be optimal to the problem G_a [9]. If the problem G_a has no optimal solution in S_1, the optimality is not guaranteed, however, it provides a lower and upper bounds for the optimal solution to the problem G_s. The problems G_a and G_s may not have optimal sequences in S_1[9].

Chapter 6

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".

6.1 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 grate 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. 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 [21].

Step 1: Set h=1

Step 2: Tentatively schedule product h to be produced in stage k. calculates the variation for stage k and calls it $V1_1$.

Step 3: 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 $V_h = V1_{h+} V2_h$.

Step 4: Put h=h+1, if h>n go to step 5, otherwise go to step 2, where n is the number of products.

Step 5: 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: 6.1.1 For the demand vector (2000, 3000, 5000, 1000) the corresponding data are presented in Table 1: Schedule generated for demand vector D = (2000, 3000, 5000, 1000) using heuristic, in which the sequence is the 1000 reparation of the cycle 3-2-1-3-4-3-2-3-1-2-3.

Stagge (k)	X[0]	X[1]	X[2]	X[3]	M[0]	M[1]	M[2]	M[3]	Product Schedule	sum((M[i][k]-X[i][k]^2)	Total Variation
1	0.181818182	0.272727273	0.454545455	0.090909091	0	0	1	0	+3	0.41322314	0.41322314
2	0.363636364	0.545454545	0.909090909	0.181818182	0	1	1	0	+2	0.380165289	0.79338843
3	0.545454545	0.818181818	1.363636364	0.272727273	1	1	1	0	+1	0.446280992	1.239669421
4	0.727272727	1.090909091	1.818181818	0.363636364	1	1	2	0	+3	0.247933884	1.487603306
5	0.909090909	1.363636364	2.272727273	0.454545455	1	1	2	1	+4	0.512396694	2.2
6	1.090909091	1.636363636	2.727272727	0.545454545	1	1	3	1	+3	0.694214876	2.694214876
7	1.272727273	1.909090909	3.181818182	0.636363636	1	2	3	1	+2	0.247933884	2.94214876
8	1.454545455	2.181818182	3.636363636	0.727272727	1	2	4	1	+3	0.446280992	3.388429752
9	1.636363636	2.454545455	4.090909091	0.818181818	2	2	4	1	+1	0.380165289	3.768595041
10	1.818181818	2.727272727	4.545454545	0.909090909	2	3	4	1	+2	0.41322314	4.181818182
11	2	3	5	1	2	3	5	1	+3	0.0	4.181818182
Schedule List: 3-2-1-3-4-3-2-3-1-2-3 No of Cycle: 1000											

Table 1: Schedule generated for demand vector D= (2000, 3000, 5000, 1000) using Heuristic Nearest integer point

6.2 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 stage 1, 2, 3, ……, k-1 as fixed.

The mathematical expression of this decision rule is:

Schedule the product I with the lowest;

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

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

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:

$$V_p = W_1 + \sum_{i=2}^{L}\sum_{i=1}^{n_l} W_l[\left(x_{il(k-1)} - t_{ilp}\right) - \left(y_{l(k-1)} + \alpha_{lp}\right) \times r_{il}]^2$$

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

Since the objective function is to be minimized. Let product p be scheduled rather than product $p'\ if\ V_p < V_{p'}$. Cancelling 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 is does not consider the effect of its current decision on the variation in future cycled. That is, it may achieve low variability at stage k at the expense of high variability at stage k+1.

Miltenburg and Sinnamon [20] introduce another scheduling heuristic of complexity $O(n_1{}^2(n_2 + \cdots + n_1)^2$ for each stage. This is heuristic attempts to remedy the myoic 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.

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. Compute $V_l = V_1 + V_2$.

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_1 in stage k.

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 4.

	Product	
Sub Assembly	1	2
1	1	2
2	0	4
3	1	0

Figure 5: 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 2: Assembly and demand data for example 6.2.1

t_{ilp} – Number of parts for one unit of product 1

Product	Sub-assembly, *l*=2				Demand	Ratio
l=1	*i*=1	2	3	Total	d_{il}	r_{il}
1	1	0	1	2	3	0.5455
2	2	4	0	6	5	0.4545
Demand						
d_{il}	16	20	6	42		
Ratios						
r_{il}	0.38	0.4762	0.1429			

Table 2: Assembly and Demand Data for Example 6.2.1

The calculations for heuristic of Miltenburg and Sinnamon for the first 13 stage are shown in table 4: Detail Schedule of Example 6.2.1. This procedure is repaired for 100 times and final schedule is 1-2-1-2-1-2-1-2-1-2-1 with a total variation of 163.5116484000000159 over 13 cycles.

6.3 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 foal 2. Indeed, goal 2 is a classical one.

Let there are *n* products to be produced with demands d_1, d_2, …… d_n, 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 $0 \leq x_{i,k} \leq d_i$ for all *i=1, 2, …. ,n* be *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 can be formulated as;

$$\text{Minimize} \sum_{k=1}^{D}(\alpha_U U_k - \alpha_L L_k)^2$$

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

Let f_n 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 takes the form:

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

Now we consider the DP procedure.

Let $d = (d_1, d_2, \ldots., d_n)$ be the product requirements vector. Define subsets in a schedule as $X = (x_1, x_2, \ldots., x_n)$; where x_i is a non negative integer representing the production of exactly x_i units of product i, $x_i \leq d_i$ for all i. Let e_i be the i^{th} unit vector; with n entries, having i^{th} entry 1 and remaining all zero. A subset X cab 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 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)*.

$$f(X) = f(x_1, x_2, \dots, x_n) = \min \{f(X - e_i) + g(X) | \, i = 1,2, \dots . n; x_i - 1 \geq 0\}$$
$$f(X) = f(X | x_1 = 0 \, ; i = 1,2, \dots . n) = f(0,0, \dots ,0) = 0.$$

Clearly $f(X) \geq 0$ and $g(X | x_1 = 0 \, ; i = 1,2, \dots . n) = 0$. The following theorem tells about the computational efficiency of the above procedure [22].

Example 6.3.1 Demand Vector (300, 600, 900) and Time T= (2, 5, 1) the schedule generated by DP is shown in Table 3: Schedule generated by dynamic programming, with number of cycle 300.

Stagge (k)	(x_1, x_2, x_3,...)	P-Index	Product Schedule	X-e	f(x-e)	g(x)	f(x)	Expand
1	1-0-1		1	000	0	5.806	5.806	E
1	0-1-0		2	000	0	11.472	11.472	E
1	0-0-1		3	000	0	3.139	3.139	E
2	1-1-1	3	2	100	5.806	5.555	11.362	E
2	1-0-1	4	3	100	5.806	12.889	18.695	
2	1-1-0	5	1	011	11.472	5.555	17.028	
2	0-2-0	6	2	011	11.472	45.889	57.36100	E
2	0-1-1	7	3	011	11.472	3.222	14.69399	
2	1-0-1	8	1	001	3.139	12.889	16.028	E
2	0-1-1	9	2	001	3.139	3.222	6.361	E
2	0-0-2	10	3	001	3.139	12.556	15.69499	E
3	1-2-0	11	2	110	11.362	28.25	39.612	E
3	1-1-1	12	3	110	11.362	1.25	12.612	
3	1-2-0	13	1	020	57.361	28.25	85.611	
3	0-2-1	14	3	020	57.361	28.25	83.611	
3	1-1-1	15	2	101	16.028	1.25	17.278	
3	1-0-2	16	3	101	16.028	26.25	42.278	
3	1-1-1	17	1	011	6.361	1.25	7.611	E
3	0-2-1	18	2	011	6.361	26.25	32.611	E
3	0-1-2	19	3	011	6.361	1.25	7.611	E
3	1-0-2	20	1	002	15.6945	26.25	41.945	E
3	0-1-2	21	2	002	15.6945	1.25	16.945	
3	0-0-2	22	3	002	15.6945	28.25	43.045	E
Schedule: 1-3-2-3-3-2-1 No. of Cycle: 300								

Table 3: Schedule generated by dynamic programming

Theorem 6.3.1 [22]

The DP recursion solves the JIT scheduling problem in

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

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 $O(n\prod_{i=1}^{n}(d_i+1))$

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 sondtructed at the end. Therefore, the space requirements are $O(n\prod_{i=1}^{n}(d_i+1))$.

Notice that the total number of feasible schedules is,

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

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

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

$$= \left(\frac{D+n}{n}\right)^n$$

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 [22].

Chapter 7

Conclusion and Recommendation

Mixed model JIT system requires producing only the necessary products in necessary quantities at necessary time. This problem minimizes both the earliness and the tardiness penalties that respond to the customer demands for a variety of models without holding large inventories or incurring shortage. PRVP is an important production problem that arises on mixed model JIT assembly line. This problem consists in sequencing copies of different products in such a way that actual productions are as close as possible to their ideal production.

In this dissertation, the mathematical models for MMJITSP and different sequencing approaches developed till date have been analyzed. The MMJITSP with the goal of keeping constant rate of usage of parts is focused. The study shows that the problems have real world exciting applications as well as interesting mathematical features of theoretical value. We explicitly explore, with justification of the ground for future research, the questions which still remain open and are challenging.

The problem, under the assumption that the products require approximately the same number and mix of parts or the pegging assumption (single-level) is solvable. A pseudo- polynomial algorithm of the assignment problem is applicable to the problem *G*. The approach can also be applied to the bottleneck PRVP with necessary modification.

The relation between different sequencing approaches will be foremost topic for the further investigation and to determine an algorithm which simultaneously optimizes both Dynamic and Heuristics objectives will be the most interesting topic for the research.

Different types of sequencing approaches for PRVP are optimal and existence of such sequences considerably reduces the computational effort. The question, whether sequencing approaches to ORVP are optimal, is still open.

REFERENCES:

[1]. Aigbod, H., *"Some Structural Properties for JIT Level schedule problem",* Production Planning and Control, 11, 4 (2000), 357-362.

[2]. Bautista J. and Cono J. *"Minimizing Work overload in mixed-model Assembly Lines."* International Journal Production Economics, 112 (2008), 177-191.

[3]. Bautista J., Companys R., and Corominas A., *"Modeling and solving the production rate variation problem",* TOP, 5, 2, 221-239, 1997a.

[4]. Blazewicz, J., Ecker, K.H., Pesch, E., Schmidt, C. and Weglarz, J., "*Scheduling Computer and manufacturing Processes",* Springer, Berlin (1996).

[5]. Blum, M, "A machine-independent theory of the Complexity of reserve functions", Journal of the ACM 14, 2(1967)322-336.

[6]. Boyseb, N., Fliedner, M., and Scholl, A., *"Scheduling mixed-model assembly lines,"* Survey, classification and model critique. European Journal of Operational Research, 192, 2 (2009), 349-373.

[7]. Brouner , N., and Crama , Y., *"The maximum deviation just-in-time scheduling problem*", Discrete Applied Mathematics 134 (2004) 25-50.

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

[9]. Corominas A., and Moreno N., *"On the relations between optimal solutions for different types of min-sum balanced JIT optimization problems*", INFOR, 41(4), 333-339, 2003.

[10]. Dhamala T. N. and Kubiak W., "*A brief survey of Just-in-Time Sequences for mixed model system",* International Journal of operational research, 2, 2 [2005], 38-47.

[11]. Dhamala T.N., Khadka S.R., "*A Review on sequencing approach for Mixed-Model Just-in-Time Production System",* Iranian Journal of Optimization, 2006.

[12]. Dhamala T.N., Khadka S.R., and Lee, M. H. "*On Sequencing approach for mixed model Just-in-Time production system",* Asia-Pacific journal of operational research, 37 [2008], 1-25.

[13]. Dhamala T. N. and Kubiak W., "*A brief survey of Just-in-Time Sequences for mixed model system",* International Journal of operational research, 2, 2[2005], 38-47.

[14]. Inman R. R., and Bulfin R. L., *"Sequencing JIT mixed-model assembly lines",* Management Science, 37(7), 901-904, 1991.

[15]. Kotani S., Ito T., and Ohno K., *"Sequencing problem for a mixed-model assembly line in the Toyota production system",* International Journal of Production Research, 42(23), 4955-4974, 2004.

[16]. Kubiak, W., *"Cyclic just-in-time sequence are optimal",* Journal of Global Optimization 27 (2003) 333-347.

[17]. Kubiak W., *"Proportional optimization and fairness",* International Series in Operations Research and Management Science, 127, Springer, 2009.

[18]. Lebacque V., Jost V., and Brauner N., Simultaneous optimization of classical objectives in JIT scheduling, *European Journal of Operations Research*, 182, 29-39, 2007.

[19]. Miltenburg J. and Goldstein T., "*Developing production schedules which balance part usage and smooth production loads for Just-in-Time production system*", Naval research logistics, 38(1991), 893-910.

[20]. Miltenburg, J. and Sinnamon, G., *"Scheduling Mixed-model multi-level Just-in-time production systems"*, International Journal of Production Research 27, 9 (1989) 1487-1509.

[21]. Miltenburg, J., "*Level Schedules for Mixed-Model Assembly Lines in Just-in-time production system"*, Management Science 35 2 (1989) 1992-207.

[22]. Miltenburg J., Steiner and Yeomans S., *"A Dynamic Programming Algorithm Fro Scheduling Mixed-Model,* Just-In-Time Production System, Mathematical and ComputerModeling, 13 (1990), 57-66.

[23]. Monden, Y., *"Toyota Production System",* Industrial Engineers and Management Press, Norcross, GA (1983).

[24]. Moreno N., and Corominas A., "*Solving the min-sum product rate variation problem as an assignment problem*", International Journal of Flexible Manufacturing Systems, 18(a), 269-284, 2006.

[25]. Steiner G. and Yeomand S., "*Optimal level schedules in Mixed-Model Multi-Level Just-in-Time assembly system with pegging*", European journal of Operational research, 95, (1996), 38-52.

[26]. Sumichrast R. T., Russell R. S., and Taylor B. W., *"A comparative analysis of sequencing procedures for mixed-model assembly lines in a just-in-time production system",* International Journal of Production Research, 30(1), 199-214, 1992.

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

[28]. www.assignmentproblem.com.

[29]. Wikipedia, the Free Encyclopedia (http://en/wikipedia.org 2009).

Appendix

Basic Mathematic Notations

Set theory, Sequence and Series

N	:	Set of Naturals
R	:	Set of real Numbers
R^+	:	Set of Positive Real Number
$\{a_1, a_2, \dots, a_n\}$	:	Set of Objects $a_1, a_2, \dots, a_n$
$(a_1, a_2, \dots, a_n\}$	:	A sequence of numbers $a_1, a_2, \dots, a_n$
S	:	A sequence

Data Problems

n	:	Number of jobs
m	:	Number of machine
l	:	Product Level
J_i	:	Job number i = 1, 2, … ,n.
n_i	:	Number of operations of job J_i
m^l	:	number of machines at stage l
M_j	:	Machine number j = 1, 2, … ,m.
$O_{i,j}$	:	Operation j of job J_i
r_i	:	Release time of job J_i
d_i	:	Due date of job J_i
s_i	:	desired start time of job J_i
$p_{i,j}$	:	Processing time of job $O_{i,j}$
W_i or w_i	:	Weight associated to job J_i
D	:	Total Demand