

Chapter 1

Introduction

1.1 Background

JIT (Just-in-Time) is the name used to describe a manufacturing system where the parts which are needed to complete the finished products are produced or arrive at the assembly site as they are needed. Just-in-Time is a Japanese manufacturing management method developed in 1970s. It was first adopted in Toyota manufacturing plants by *Taiichi Ohno*. The main concern at that time was to meet consumer demands.

The concept of penalizing jobs both for being tardy and for being early has proven one of the most important and fertile research topic in operations research. Sequencing different products with even distribution under Just-in-Time production system for minimization of earliness and tardiness penalties is a challenging non-linear integer programming problem. The purpose of Just-in-Time production is to reduce cost by eliminating waste. In sales the Just-in-Time concept is realized by producing only salable products or part in salable quantities. Mixed model assembly lines are used to produce many different products without carrying large inventories. An important and obvious optimization problem associated with these lines is that of determining the sequence for manufacturing different type of parts on the line. The sequence will depend upon the objective of the company.

Miltenburg [28] considers the quantity of each part used by mixed model assembly line per unit of time should be kept as constant as possible. Monden [30] states this as most important goal of Just-in-Time production system implemented by Toyota. Toyota's Goal Chasing Method, a local search heuristic, has been most popular for solving the problem. The sequence referred to as level, balanced or fair sequence always keeps actual production level and desired production level as close to each other as possible all the times.

The goal of Just-in-Time approach is to sequence small batches of variety of part types in order to satisfy customers demand for them with out holding excessive inventories or incurring large shortage. A sequence with this goal is termed as a balanced schedule, Miltenburg and Sinnamon [29]. In some special cases, this sequence will be optimal for level schedule problem for mixed model assembly line, Monden [30]. The level schedule problem is concerned with keeping as constant as possible the rate of usage of component parts going into part type being assembled.

Different cases arise here. One is component required for different part types are distinct. Other is different part types require the same number and mix of components. In this case, Monden [30] provides a simple heuristic method Goal Chasing Method II (GSM II) used at Toyota for sequencing its mixed model assembly line. Miltenburg has developed several heuristic methods and conducts computational experiments to compare their performance. Monden [30] and Miltenburg and Sinnamon [29] provide several heuristic methods for obtaining level schedule in general case. Kubiak and Sethi [25] reduce the minimization of total deviation Just-in-Time problem into assignment problem and there by presents an efficient optimization algorithm for this problem. Steiner and Yeomans [33] give a graph theoretic optimization algorithm for minimizing maximum deviation Just-in-Time single level sequencing problem. They also give an algorithm for minimizing multi-level maximum deviation Just-in-Time assembly system under pegging assumption. If the outputs at production level to the final product onto which they will be assembled, the problem with pegging is equivalence to a weighted single level problem.

There exist cyclic schedules for both maximum and minimum deviations. These schedules are optimal and reduce the computational requirement, Steiner and Yeomans [33] and Kubiak [23]. The minimization of maximum deviation (bottleneck) for single level is co-NP, Brauner and Crama [4]. The multilevel problem for two or more production levels is strongly NP-Hard, Kubiak [23]. Bautista, Companys and Corominas [1] linked Just-in-Time sequencing and apportionment problem. Dhamala [10] propose an efficient algorithm which obtains optimal solution for a variation of maximum deviation objective function in single level.

1.2 Motivation

The motivation behind the cyclic scheduling approach in Just-in-Time environment comes from the environment where nearly all the demand requires (or customer purchase) most of the productions from a company in the repetitive manner. This environment is attractive setting for a cyclic schedule to be implemented. Cyclic scheduling is consistent with Just-in-Time in two important respects. First, the cyclic sequence aims to globally minimize inventory by having production ready when it is needed. Second, efforts to reduce setup times are more effective since they can be directed at the specific changeovers that occur in the sequence. Also, the task of coordinating other activities such as raw material delivery, preventive maintenance, and work force schedules becomes simpler.

1.3 Objective of the Study

The specific objective of the study was to study and analyze a cyclic scheduling approach in Just-in-Time environment.

The general objectives of this study were stated as:

- To study different models in Just-in-Time production system.
- To study different objective functions under Just-in-Time production environment.
- To find out possibility of cyclic approach in Just-in-Time System.
- To achieve goal of min-max objective function in cyclic manner

1.4 Organization of the Thesis

Thesis is organized as follows:

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

Chapter 3 sets out the scheduling problems as encountered in the literature. It presents representation of schedule, job characteristics, constraints and criterions. Graham's notation of scheduling problem is provided. Some areas of applications of scheduling problem are mentioned. Min-max and min-sum criteria for scheduling problem are described.

Chapter 4 deals with Just-in-Time system. Mixed model production system is described. Mathematical models are formulated for single level and multi level problems.

Chapter 5 focuses on objective functions for the schedule in Just-in-Time environment. We study min-max objective function which gives maximum overproduction (inventory) / underproduction (shortage) from the desired level of production that occurs at any time during schedule. Min-max problem is reduced to release date/ due date decision problem. Single level just in time sequencing problem is formulated under chain constraints and min-max-absolute-chain-algorithm is covered in this chapter.

Chapter 6 is related with cyclic schedules. Cyclic scheduling problems encountered in manufacturing system are illustrated. A cyclic solution to PRV problem is described. This chapter extends min-max-absolute-chain-algorithm for cyclic sequence. An optimal cyclic approach is used in min-max-absolute-chain-algorithm to increase performance of the algorithm.

Chapter 7 "Implementation and Testing" shows the organization of the program to implement the algorithms. The program is tested for input data set (chains), which represent the demands of products in the mixed model assembly line manufacturing system.

Chapter 8 "Conclusion and Future Recommendation" has concluded the study with some remarks and future recommendation for the study on cyclic paradigm over mixed model Just-in-Time production system with min-max objective.

1.5 Methodology

Papers related to the Scheduling, Optimization problems, Just-in-Time sequencing problems and Cyclic Just-in-Time sequences were collected. Study of these papers encouraged us to work in mixed model Just-in-Time production system. Initially, we studied min-max problems and min-max-absolute-chain-algorithm in detail. Since optimal Just-in-Time sequences are always cyclic, the goal of min-max-absolute-chain-algorithm was tried to achieve in cyclic paradigm.

An application was developed to achieve the goal in Java programming language. The min-max-absolute-chain-algorithm was implemented in cyclic paradigm. Different constraints considered in this work were:

1. Number of cycle and length of cycle is equal in each chain.
2. Number of cycle is equal and length of cycle differs in different chains.
3. Number of cycle differs and length of cycle is same in different chains.

Chapter 2

Computational Complexity

Computational complexity analysis, as a branch of the theory of computation in computer science, investigates the problems related to the amounts of resources required for the execution of algorithms (e.g., execution time), and the inherent difficulty in providing efficient algorithms for specific computational problems. In particular, the theory places practical limits on what computers can accomplish. *Manuel Blum* developed axiomatic approach to measure computational complexity, Blum [3]. He introduced such measures of complexity as the size of a machine and axiomatic complexity measure of recursive functions. The time complexity and space complexity of algorithms are special cases of the axiomatic complexity measure. The axiomatic approach helps to study computational complexity in the most general setting. An important aspect of the theory is to categorize computational problems and algorithms into complexity classes.

2.1 Turing Machines and Algorithms

It all started with a machine. In 1936, *Alan Turing* developed his theoretical computational model. His model is based on his perception of the way mathematicians think. As digital computers were developed in the 40's and 50's, the Turing machine proved itself as the right theoretical model for computation. Basically, Turing machine converts one set of strings to another. A Turing Machine comprises of a finite control, a tape, and a head that can be used for reading and writing on the tape.

There is no precise definition of an algorithm. It is believed that algorithms and Turing machines are equivalent. That is every problem solvable by Turing machine is solvable by algorithm and vice versa. There are other models of computations apart from Turing machines and algorithms. *Church Turing's* thesis states that all these models are equivalent, Deutsch [9]. An algorithm is any well defined computational procedure that takes some value or set of values as input and produces some value or set of values as output.

The basic Turing machine model fails to account for the amount of time or memory needed by a computer, a critical issue today but even more so in those early days of computing. The key idea to measure time and space as a function of the length of the input came in the early 1960's by *Hartmanis and Stearns*, Hartmanis and Stearns [18]. And thus computational complexity was born.

2.2 Computational Resources

Complexity theory analyzes the difficulty of computational problems in terms of many different computational resources. The same problem can be explained in terms of the necessary amounts of many different computational resources, including time, space, randomness, and other less-intuitive measures. A complexity class is the set of all of the computational problems which can be solved using a certain amount of a certain computational resource.

The most well-studied computational resources are time and space. The time complexity of a problem is the number of steps that an algorithm takes to solve an instance of the problem. The space complexity of a problem is a measures the amount of space, or memory required by the algorithm. A good algorithm always takes less time and less space. A better algorithm in bad machine may appear insufficient compared to bad algorithm in good machine. To minimize effect of these considerations, computational complexity deals with instances whose input size is very large, so that machine size can be neglected. To describe behavior of algorithm for large input the concept of asymptotic order is useful.

2.3 Functions

Given two sets A and B , a function f is a binary relation on $A \times B$ such that for all $a \in A$, there exists precisely one $b \in B$ such that $(a, b) \in f$. The set A is called domain of f , and the set B is called co-domain of f . We write $f: A \rightarrow B$ and if $(a, b) \in f$, we write $b=f(a)$, since b is uniquely determined by choice of a . Two functions f and g are equal if they have the same domain and co-domain and if, for all a in the domain, $f(a)=g(a)$.

A finite sequence of length n is a function f whose domain is the set of n integers $\{0, 1, 2, \dots, n-1\}$. Finite sequence is denoted by listing its values: $\{f(0), f(1), f(2), \dots, f(n-1)\}$. An infinite sequence is a function whose domain is set of \mathbb{N} natural numbers. For example, the Fibonacci sequence, defined by recurrence, is the infinite sequence $\{0, 1, 1, 2, 3, 5, 8, \dots\}$.

A function $f(x)$ between two ordered set is unimodal if for some value m (the mode), it is monotonically increasing for $x \leq m$ and monotonically decreasing for $x \geq m$. In that case, the maximum value of $f(x)$ is $f(m)$ and there are no other local maxima. Examples of unimodal function: quadratic polynomial, logistic map, tent map.

A function is convex if and only if its epigraph (the set of points lying on or above the graph) is a convex set. Pictorially, a function is called 'convex' if the function lies below the straight line segment connecting two points, for any two points in the interval. A function f is said to be concave if $-f$ is convex.

2.4 Asymptotic Order of Functions

There are three asymptotic orders that are commonly used. They are big-O, big-omega and big-theta. Let $f: \mathbb{N} \rightarrow \mathbb{R}^+$ and $g: \mathbb{N} \rightarrow \mathbb{R}^+$ be two functions from the set of natural numbers to the set of non-negative real numbers.

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

The function $f(n) = \Omega(g(n))$ (read as “ f of n is omega of g of n ”) if and only if there exists positive constants c and n_0 such that $f(n) \geq c * g(n)$ for all n such that $n \geq n_0$.

The function $f(n) = \Theta(g(n))$ (read as “ f of n is theta of g of n ”) if and only if there exists positive constants c_1, c_2 and n_0 such that $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$ for all n such that $n \geq n_0$.

2.5 Complexity Classes

A complexity class is the set of all of the computational problems which can be solved using a certain amount of a certain computational resource. There are several complexity classes in the theory of computation. The major classes are discussed below.

2.5.1 Class P

The complexity class P is the set 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 $O(n^2)$ time using the quicksort algorithm in worst case. Thus all sorting problems are in P.

2.5.2 Class NP

The complexity class NP is the set of decision problems that can be solved by a non-deterministic 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 $(u, v) \in E$, then $u \in V'$ and $v \in V'$ 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 NP.

2.5.3 The P=NP Question

The question of whether NP is the same set as P (that is whether problems that can be solved in non-deterministic polynomial time can be solved in deterministic polynomial time) is one of the most important open questions in theoretical computer science due to the wide implications a solution would present. If it were true, many important problems

would be shown to have "efficient" solutions. 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 efficiently using computers. The P=NP problem is one of the Millennium Prize Problems proposed by the Clay Mathematics Institute the solution of which is a USD 1,000,000 prize for the first person to provide a solution.

2.5.4 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.

2.5.5 NP-Complete

NP-Complete are the hardest problems among the NP class. The class NP-Complete is the set of decision problems X such that

1. $X \in \text{NP}$
2. Every problem in NP is reducible to X. i.e, NP-Complete are the hardest problems among the NP-class.

2.5.6 NP-Hard

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

Chapter 3

Scheduling Problems

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. A definition quoted by *Carlier and Chretienne*: “Scheduling is to forecast the processing of a work by assigning resources to tasks and fixing their start times. 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 [5].

Another definition put forward by *Pinedo*: “Scheduling concerns the allocation of limited resources to tasks over time. It is decision-making process that has a goal the optimization of one or more objectives.”, Pinedo [31].

In the above definitions, the task (or operation) is the entity to schedule. In this dissertation work we deal with jobs to schedule, each job is broken down into a series of operations. When all jobs contain only a single operation we term mono-operation problem. Else we say multi-operation problem. The operation of a job may be connected by precedence constraints. We deal with the resource or machine. We consider two types of resources: renewable resource (which is available after use e.g.: machine, file, processor, personnel etc.) and non renewable resources (which disappear after use e.g.: money, raw materials etc.). There are two types of optimality criterion: those relating to completion time and those relating to costs. In the category of completion time related criteria we find for example those which measure the completion time of whole schedule and those which measure tardiness of jobs in relation to their due date. In the category of cost related criteria we may cite those which represent cost of machine use and those which represent cost allied to waiting time of operations before and/or after they are processed.

Generally scheduling problems are studied in two research communities:

a) Operations Research

- Job shop and flow-shop problems
- Scheduling of machines, orders, batches, projects etc
- Resources: machines, factory cells, unit processes etc
- Static (off-line) techniques

b) Computer Science

- Schedule tasks in a uni-processor or multi-processor environment.
- Dynamic techniques.

Three activities are common in scheduling:

a) Run-time Dispatching

The actual switching between computations for different events occur at run-time

- Task-based system
- Preemptive or non-preemptive
- Uses knowledge about
 - the current state of the system (e.g. the time)
 - off-line information (e.g. task priorities)

b) Off-line Configuration

The static information for the run-time dispatcher is generated during off-line configuration. There can be very large activities in some approaches – scheduling table; very small activities in other approaches – task priorities.

c) Analysis

In hard real-time systems the deadlines must always be met. Off-line analysis (before the system is started) is required to check so that there are no circumstances that could lead to missed deadlines. There may be a situation where deadlines could be missed. The system is unschedulable in this case. If the scheduler does not find a way to switch between the tasks, then also the system is unschedulable. If all the deadlines are met, the system is

schedulable. A sufficient analysis is an absolute requirement and we like it to be as close to necessary as possible.

3.1 Representation of Schedule

Let there be m number of machines, $M_i, i = 1, 2, \dots, m$ which have to process n jobs, $J_i, i = 1, 2, \dots, n$. The problem is to assign each job one or more time intervals on one or more machines. Such an assignment is called a schedule in general term. A schedule is often represented by Gantt chart. Below is an example of a schedule for a single machine:

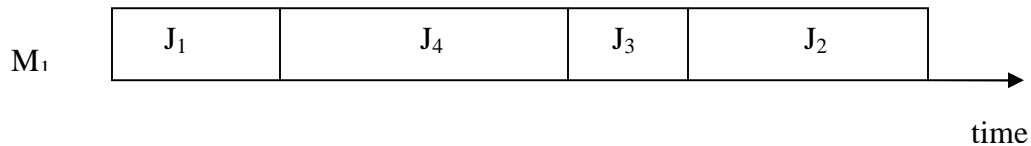


Figure 3.1 Gantt chart for a schedule of four jobs in single machine

The schedule of jobs can be represented as a sequence of jobs. For example, the schedule shown in Figure 3.1 can be written as the sequence $s = (J_1, J_4, J_3, J_2)$. The machine may remain idle for some time interval. We specify idle intervals by writing 'idle' for that time interval.

3.2 Machine Environment

The number of machines may vary according to the production environment. The number of machines may be known or sometime unknown in advance. The simplest machine environment is the single machine environment, on which each n job J_i , each consisting of single operation, have to spend a processing time equal to their given processing requirements $P_i (i=1, 2, \dots, n)$.

In multiple machine environments, a job J_i is a set of n_i number of operations O_i . Here any operation of any job can be processed in any machine. Multiple machine environments are categorized into two broad categories: parallel machines and dedicated

machines. In parallel machine J_i has to spend its processing requirement on any of m machines. These can be *identical*, in which case the machines operate at the same speed; *uniform*, in which case each machine has its own speed; *unrelated*, in which case speed of the machines is job independent. Operations executable on machines is constrained in dedicated machine environment. Flow shops, Open shops and Jobs shops are m machine environments in which each job consists of several operations, each of which has to be executed on a designated machine; no job can undergo more than one operation at a time. In Job shop, the order in which jobs has to be executed is fixed; in a Flow shop, this order is fixed and same for all jobs; and in Open shop, the order is free and hence up to the scheduler.

3.3 Job Characteristics

The job characteristics include the possibility of allowing preemption, and of specifying precedence constraints, release dates, the deadlines, due date and weight. If preemption is allowed, then an operation may be interrupted and resumed at the same time on a different machine or at a later time on any machine; if preemption is not allowed, an operation, once started, must be processed until completion with-out interruption. A precedence constraint stipulates that a certain job cannot start before another has completed. Job availability may be restricted by imposing a release date r_i , before which J_i cannot be started, and a deadline d_i , by which J_i has to be completed. If the job does not complete before its due date, the quality of the output deteriorates. If a job does not complete before its deadline, the output is invalid. Weight of a job means its priority.

Blaziwicz et al. [2] categorize scheduling problems regarding different characteristics.

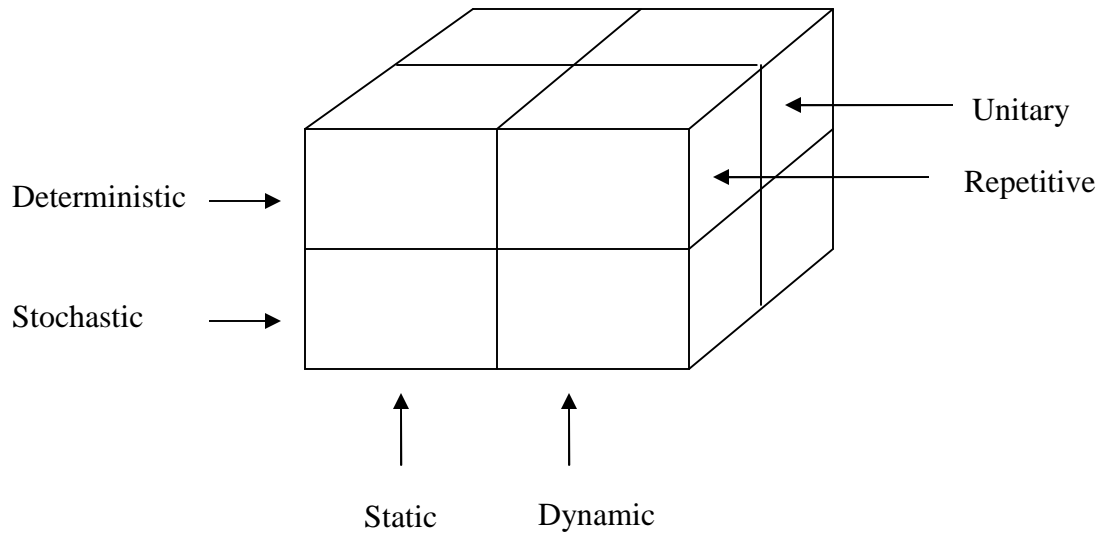


Figure 3.2 Types of scheduling problems according to different characteristics, Blaziwicz et al. [2]

3.3.1 Deterministic vs. Stochastic

In the case where all the characteristics of the problem (processing time of each operation, release dates, etc.) are well known, we speak of a deterministic problem. Conversely, some of these characteristics may be random variables of known probability law. In this case we speak of a stochastic problem, Pinedo [31].

3.3.2 Unitary vs. Repetitive

If the operations are cyclical, we are dealing with a repetitive problem. Conversely, if each operation corresponds to a unique product the problem is said to be unitary.

3.3.3 Static vs. Dynamic

If all the data of the problem are known at the same time we speak of a static problem. For some problems, a schedule may have been calculated and being processed when new

operations arrive in the system. Then the foregoing schedule has to be re-established in realtime. These problems are said to be dynamic.

3.4 Objective Functions in Scheduling

The optimality criteria of the schedule depend upon the objective functions in scheduling. For each job J_j , let release time be r_j , due-date be d_j , and weight be w_j . Following can be the objective function in scheduling:

Completion time C_j

Flow time $F_j = C_j - r_j$

Lateness $L_j = C_j - d_j$

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

Earliness $E_j = \max\{d_j - C_j, 0\}$

Some other objective functions are:

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

Weighted completion time $\sum_j w_j C_j$

Total completion time $\sum_j C_j$

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

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

3.5 The Three Field Notation

Scheduling problems can be described by a three field notation $\alpha | \beta | \gamma$ where α describes the machine environment, β describes the job characteristics, and γ describes the objective criterion to be minimized, Graham et al [16]. A field may contain more than one entry but may also be empty.

Machine environment is described as $\alpha = \alpha_1 \cdot \alpha_2$, where \cdot represents string concatenation. Parameter α_1 characterizes the type of machine used as follows:

- $\alpha_1 = 1$: single machine
- $\alpha_1 = P$: identical machines
- $\alpha_1 = Q$: uniform machines
- $\alpha_1 = R$: unrelated machines
- $\alpha_1 = O$: dedicated machines, Open shop system
- $\alpha_1 = F$: dedicated machines, Flow shop system
- $\alpha_1 = J$: dedicated machines, Job shop system

Parameter α_2 denotes number of machines used.

- $\alpha_2 = \infty$: number of machines is assumed to be variable.
- $\alpha_2 = k$: number of machines is fixed, k number of machines.

Job characteristics is represented as $\beta = \beta_1 \cdot \beta_2 \cdot \beta_3 \cdot \beta_4 \cdot \beta_5 \cdot \beta_6$.

Parameter $\beta_1 \in \{ \emptyset, \text{pmtn} \}$ indicates whether preemption is allowed or not.

- $\beta_1 = \emptyset$: preemption is not allowed.
- $\beta_1 = \text{pmtn}$: preemption is allowed.

Parameter $\beta_2 \in \{ \emptyset, \text{res} \}$ indicates additional resource constraints

- $\beta_2 = \emptyset$: no resource constraints.
- $\beta_2 = \text{res}$: resource constraints are given.

Parameter 3 $\in \{ \text{ , prec, tree } \}$ indicates precedence constraints.

3 = : no precedence constraints.

3 = prec : precedence is in the form of an arbitrary DAG.

3 = tree : precedence is given in the form of tree.

3 =intree : precedence is in the form of intree.

3 = outtree : precedence is in the form of outtree.

Parameter 4 $\in \{ \text{ , } r_j \}$ indicates release dates.

4 = : release date is 0 for all jobs.

4 = r_j : release date is given for each job.

Parameter 5 $\in \{ \text{ , } p_j = p \}$ indicates processing times.

5 = : jobs have arbitrary processing time.

5 = ($p_j = p$) : all jobs have processing time equal to p.

Parameter 6 $\in \{ \text{ , } d \}$ indicates processing times.

6 = : no deadlines.

6 = d : jobs have deadlines.

Objective criterion to be minimized is represented by third field . In this field, the formula that describes the objective function is simply written. One can write $\sum_j C_j$ to indicate total completion time. For example 1 | tree | C_{\max} denotes single machine where the precedence is given in the form of tree and the objective is to minimize the maximum completion time.

3.6 Some Areas of Applications

Scheduling problems are encountered at all levels and in all sectors of activity. Generally we can distinguish between those of manufacturing production and those in computer in computer systems or project management.

3.6.1 Problems Related to Production

We encounter scheduling problems in Flexible Manufacturing Systems (FMS). Numerous definitions of an FMS are found in the literature. Lu and MacCarthy [26], states: “An FMS comprises three principal elements: computer controlled machine tools, an automated transport system and a computer control system”. Besides, this very broad problem encompasses other problems related to Robotic Cell Scheduling and Scheduling of Automated Guided Vehicles (AGV). Electroplating and chemical shops have their own peculiarities in scheduling problems. These shops are characterized by the presence of one or more traveling cranes sharing the same physical area and which are ordered to transport the products for treatment in tanks. In general, the soaking time in a tank is bounded by a minimum and a maximum, transport time is not negligible and the operations must be carried out without waiting time. These problems are very common in industry and the “simple” cases (mono-robot, single batch tanks, etc) have been solved by now.

Scheduling problems in car production lines, so called Car Sequencing Problems, are encountered in assembly shops where certain equipment must be assembled in the different models of vehicles. These problems have constraints and peculiarities of their own. Knowing a sequence of vehicles undergoing treatment, the problem is to determine the type of next vehicle programmed. We have to take account of a group of constraints connected principally to assembly options for these vehicles and to the limited movement of the tools along the production line.

3.6.2 Scheduling Problems in Operating System

Scheduling problems posed by Operating Systems (OS) are online versions of various scheduling problems. In an online version, one does not know processing time and other relevant information of a job until it actually arrives in the system. In an OS, a machine is a processor, and jobs are processes (a process is a program ready for execution. The machine environment has a vast variety. There can be multiple processors, preemption may or may not be allowed, and in almost all situations, the scheduling problems are

resource constrained. OS designers take engineering approach due to this variation. The scheduling algorithms are selected on the basis of simulation experiments. Objective function for OS oriented scheduling is different than those for manufacturing companies. A manufacturing company aims to reduce production cost, where as an OS aims to provide a fair service to all user processes. This leads objective functions like:

1. Processor utilization: This is the average function of time during which the processor is busy.
2. Throughput: This is the number of processes executed per unit time. Throughput is computed by dividing number of processes by schedule length.
3. Average turnaround time: The time that elapses from the moment a program released until it is completed by the system.
4. Average waiting time: The time that a process spends waiting for the processor or some other resources.
5. Average response time: The time taken by a process to response after it is released.

Scheduling problems in computer system is mostly based on the analysis of queuing theory. The basic queuing model is given below.

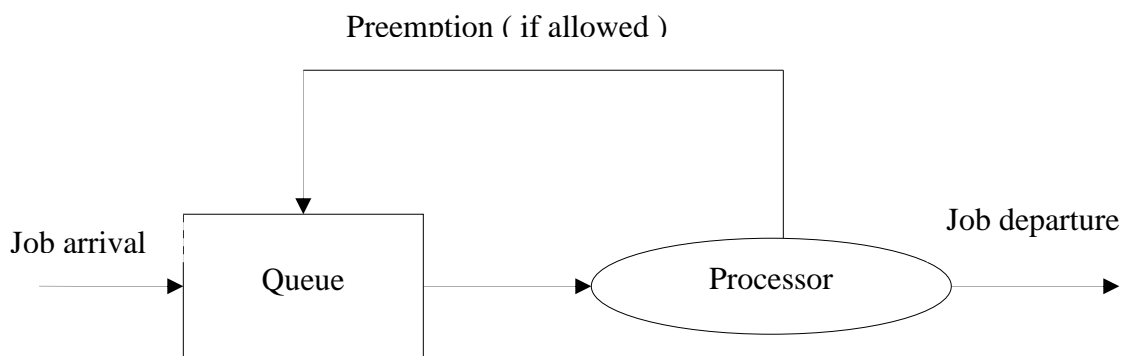


Figure 3.3 The basic queuing model

Jobs arrive and wait in a queue as shown above. The queue is the main memory for an OS. Every scheduling algorithms of an OS follows this model. Some basic algorithms used in OS for uni-processor computers are given below.

1. First Come First Serve (FCFS): At any instant when machine is idle, select available job having least release date.
2. Shortest Processing Time (SPT): When the machine is idle select the available job having least processing time. This rule is also called Shortest Job First (SJF).
3. Shortest Remaining Time Next (SRTN): Select an unfinished job which is having the smallest remaining processing time.
4. Round Robin: Available jobs are stored in a queue according to release dates. Unit processing time is given to each job in a queue in the sorted order. The newly arrived job is appended to the queue. Completed jobs are removed from the queue.

3.6.3 Other Problems

We encounter scheduling problems in computer systems. These problems are studied in different forms by considering mono or multi processor systems, with the constraints of synchronization of operations and resource sharing. In these problems, certain operations are periodic others are not; some are subject to dates, others to deadlines. The objective is to find a feasible solution i.e. a solution which satisfies the constraints. In fact, in spite of appearances they are very close to those encountered in manufacturing systems, Blazewicz et al [2].

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.

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 materials 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.

3.7 Shop Environment

When confronted with a scheduling problem, one has to identify it before tackling it. Acknowledging that the problem is complicated and to know it is already solved in the literature, we must use a recognized notation. For that purpose, shop “models” have been set up, which differ from each other by composition and organization of their resources. *Liu and MacCarthy* generalize types of scheduling problems according to machine environments and operations, Liu and MacCarthy [26].

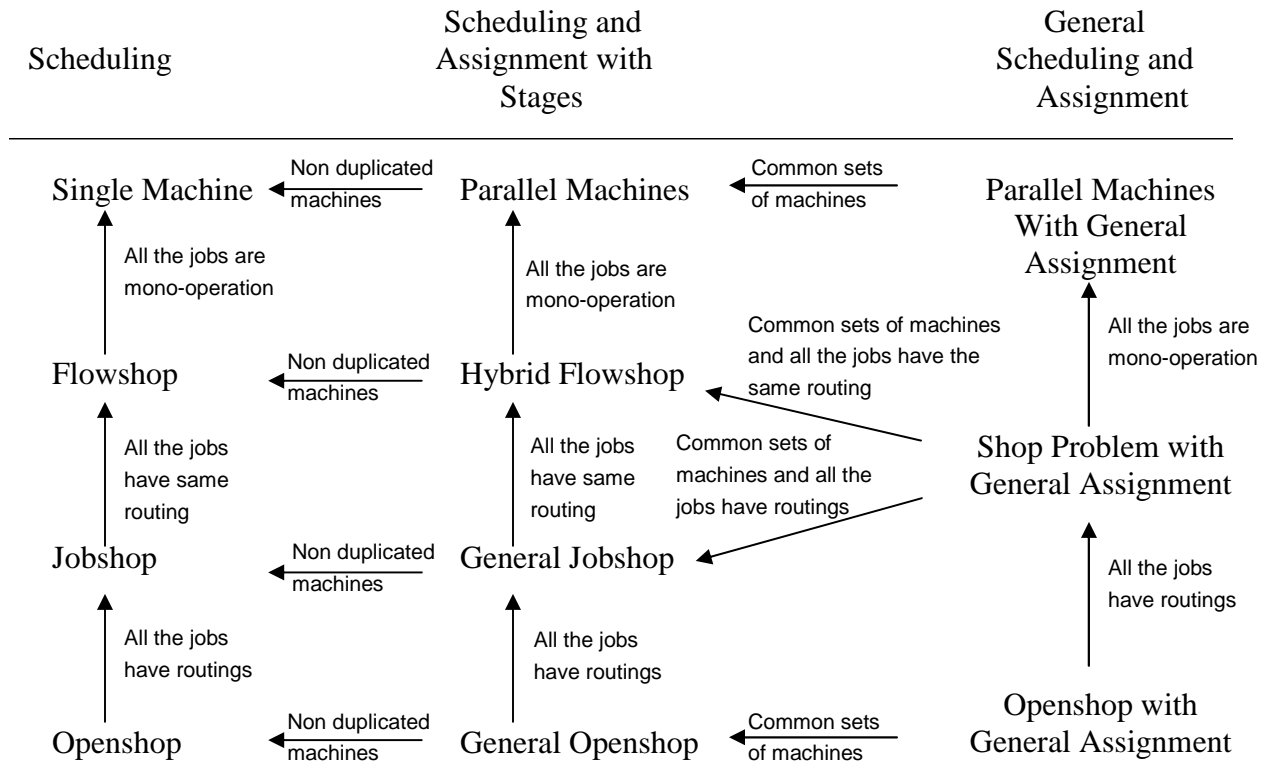


Figure 3.4 Types of scheduling problems according to machine environments and operations, Liu and MacCarthy [26]

3.8 Constraints

A solution of a scheduling problem must always satisfy a certain number of constraints, be they explicit or implicit. For example, in a flow shop problem it is implicit that the jobs are processed according to the routing and therefore an operation can not start while its precedent remains uncompleted. On the other hand, the occurrence of different release dates constitutes a constraint which must be stated precisely.

3.9 Optimality Criteria

In order to evaluate schedules we can use a certain number of criteria. Occasionally we want a criterion to be close to a certain reference value. Here we are at the frontier between the notions of criteria and constraints. If a constraint represents a fact which definitely must be respected, optimizing a criterion allows rather a certain degree of freedom. For example, stating that no job should be late regarding its due date leaves no margin in the schedule calculation. We may even find a situation where no feasible schedule exists. On the other hand, minimizing the number of late jobs allows us to guarantee that there will always be solution even though to achieve this certain operation might be late. From a particular point of view the difference between a criterion and a constraint is only apparent to the decision maker who initiates a schedule calculated by an algorithm.

We can classify criteria into two large families: “min-max” criteria, which represent the maximum value of a set of functions to be minimized, and “min-sum” criteria, which represent a sum of functions to be minimized.

3.9.1 Minimization of a Maximum Function: “min-max” Criteria

Min-max criteria is most frequently presented topic in the literature. The most traditional is the criterion measuring the completion time of the whole jobs. This criterion is denoted by C_{\max} . We define $C_{\max} = \max_{i=1..n} (C_i)$, with C_i being completion time of the job J_i . To

simplify this notation we write “max” for “ $\max_{i=1..n}$ ”. When there is no ambiguity, C_{\max} is

the total length or duration of a schedule, i.e. it is the completion time of the last scheduled job.

We also encounter other criteria based solely on the completion time of jobs such as criteria:

- $F_{\max} = \max (F_i)$, with $F_i = C_i - r_i$: the maximum time spent in the shop, or even yet, the duration of resting with r_i the release date of the job J_i .
- $I_{\max} = \max (I_k)$: with I_k the sum of idle times on resource M_k .

Equally we encounter in the literature criteria which are based on the due dates d_i , for all $i = 1, 2, \dots, n$ of jobs. Notably, we find criteria:

- $L_{\max} = \max (L_i)$: with $L_i = C_i - d_i$: the maximum lateness.
- $T_{\max} = \max (T_i)$: with $T_i = \max (0, C_i - d_i)$: the maximum tardiness.
- $E_{\max} = \max (E_i)$: with $E_i = \max (0, d_i - C_i)$: the maximum earliness.

Generally f_{\max} refers to an ordinary “min-max” criterion, which is a non decreasing function of the completion time of jobs. This is not the case for the criterion E_{\max} .

3.9.1 Minimization of a Sum Function: “min-sum” Criteria

Min-sum criteria are usually more difficult to optimize than “min-max” criteria. This is confronted from a theoretical point of view for certain special problems, Ehrgott [14]. We

write “ ” for “ $\sum_{i=1}^n$ ” when there is no ambiguity. Among min-sum criteria, we meet criteria:

- to designate $\frac{1}{n} \sum C_i$ or $\sum C_i$. This criterion represents the average completion time or total completion time of jobs.

- $\sum w_i C_i$ to designate $\frac{1}{n} \sum w_i C_i$, $\frac{1}{\sum w_i} \sum w_i C_i$ or else $\sum w_i C_i$. This criterion represents the average weighted completion time or total weighted completion time of jobs.
- $\sum w_i E_i$ is the average earliness of jobs.
- $\sum w_i E_i$ is the average weighted earliness of jobs.

In a general way, $\sum w_i C_i$ designates an ordinary “min-sum” criterion which is usually a non decreasing function of the completion times of jobs. This is not the case for criterion $\sum w_i E_i$.

Chapter 4

Just-in-Time Systems

Just-in-Time is an inventory strategy implemented to improve the return on investment of a business by reducing in-process inventory and its associated cost. The Just-in-Time (JIT) production system was first introduced by Toyota Company. The central goal of Just-in-Time systems is to produce only the necessary product in the necessary quantity at the necessary time.

Just-in-Time has been introduced in mixed model assembly line in order to response to the customer demands for a variety of products without holding large inventories or incurring large storages of the products. Such mixed-model must have negligible switch over costs from one model to another and must have a small-lot production. This model aims to hold inventory and shortage cost as small as possible, Dhamala and Khadka [11]. There has been growing interest in Just-in-Time systems research since Monden [30]. Monden [30] states that most important goal of a Just-in-Time system is to keep the schedule as balanced as possible. That is production rate of each type of product per unit time must be as smooth as possible. Miltenburg [28], assuming product require approximately same number and mix of parts, has formulated the problem as a non linear integer programming with objective of minimizing the total deviation between the actual and ideal production.

Just-in-Time philosophy has been used in obtaining an optimal sequence in a mixed model production system where an assembly line is drawn and optimality is tried to achieve within this assembly line. One of the most important optimization problems have been considered is to determine the sequence in which different models are scheduled in the line. The sequence always keep the actual production level and the desired production one as close to each other as possible all the time.

For larger demand sizes, cyclic Just-in-Time sequence model have been introduced and implemented, which is proved to be optimal, Kubiak [22].

4.1 Mixed Model Production

Mixed Model Production is the practice of assembling several distinct models of a product on the same assembly line without changeovers and then sequencing those models in a way that smoothes the demand for upstream components

Each product assembled on the mixed model assembly line requires variety of parts. Often these parts vary from product to product. Scheduling large lots of each product requires large lots of parts. When a part is only needed for certain products, its usage will be high when those products are being assembled and will be low otherwise. This is that Just-in-Time systems wish to avoid. Just-in-Time systems only work when there is constant rate of usage of all parts. To minimize the variation of usage in each part, products will be sequenced in very small number and mix of parts. In this case we can achieve constant rate of part usage by considering only the demand rates for the products. The objective is then to schedule a constant rate of production for each product.

4.2 Mathematical Model Formulations

Just-in-time systems are applicable when there is constant rate of usage of all parts. This is achievable by considering demand rates for products yield. However variability appears between the actual and the ideal production due to integral nature of production. 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. While formulating problem, we assume that the systems have sufficient capacity, negligible switch-over cost and production in unit time. Kubiak [22] refers to single level problem as Product Rate Variation (PRV) problem and multi level problem as Output Rate Variation (ORV) problem.

4.2.1 The PRV Problem Formulation

In Product Rate Variation (PRV) problem, we assume product require approximately the same number and mix of parts. This is a single level case. Let D units of n products be

produced to meet the demands d_i where $i=1, 2, \dots, n$ and $D = \sum_{i=1}^n d_i$ during a specified time horizon. The objective is to maintain cumulative production x_{ik} , a non-negative integer, $i=1, 2, \dots, n$ and $k=1, 2, \dots, D$ of product i during time period 1 through k as close to ideal production kr_i , a non-negative rational number, $i=1, 2, \dots, n$ and $k=1, 2, \dots, D$ with $r_i = d_i/D$ with $\sum_{i=1}^n r_i = 1$ as possible. The specified time horizon is portioned into D equal times of which one unit time is required for a unit of a product to be produced.

The mathematical model of the PRV problem P_1 is as follows:

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

and

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

subject to

$$\sum_{i=1}^n x_{ik} = k, \quad k=1, 2, \dots, D \quad \text{----- (3)}$$

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

$$x_{iD} = d_i; x_{i0} = 0, \quad i=1, 2, \dots, n \quad \text{----- (5)}$$

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

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

This model minimizes the perennial objective functions, the bottleneck measure of deviation F that produces smooth sequence in every time unit and the total measure of deviation G (for min sum) that produces smooth sequence on the average Jost [21].

The exact complexity of the PRV problem still remains open. The problem has been proven to be Co-NP but remains open whether Co-NP-complete or polinomially solvable, Brouner and Crama [4].

4.2.2 The ORV Problem Formulation

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

Consider the system consist of L different production levels $l, l = 1, 2, \dots, L$ with product level l . d_{il} be the demand for part type i of level l , $i = 1, 2, \dots, n_l$, n_l is the number of different part types. Total units of part type i at level l required to produce one unit of product p , $p = 1, 2, \dots, n_l$ be t_{ilp} . $d_{il} = \sum_{p=1}^{n_l} t_{ilp} d_{pl}$, the dependent demand for part i of level l determined by d_{pl} , $p = 1, 2, \dots, n_l$. Note that $t_{ilp} = 1$ for $i = p$ and 0 otherwise. $D_l = \sum_{i=1}^{n_l} d_{il}$ stands for the total demands of level l with demand ratio $r_{il} = d_{il}/D_l$ and $\sum_{i=1}^{n_l} r_{il} = 1$ for $l = 1, 2, \dots, L$.

This is non-preemptive model. The time horizon in the product level is partitioned into D_1 time units and there will be k complete units of various products p at level l during the first k time units. This introduces the concept of stage of a cycle. The pull nature of the systems implies that the lower level parts are pulled forward according to the need of the product level. Let x_{ilk} be the quantity of part i produced at level l in the time units 1 through k and $y_{lk} = \sum_{i=1}^{n_l} x_{ilk}$ be the total quantity produced at level l during the time units

1 through k. Clearly, at level 1, $y_{1k} = \sum_{i=1}^{n_1} x_{i1k} = k$. The required cumulative production

for part i of level l, l = 2 through k time units will be $x_{ilk} = \sum_{p=1}^{n_l} t_{ilp} x_{p1k}$. Consider f_i

unimodal convex function with minimum 0 at 0, $i=1, 2, \dots, n_l$. The mathematical model for the ORV problem is as follows :

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

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

subject to

$$x_{ilk} = \sum_{p=1}^{n_l} t_{ilp} x_{p1k}, i=1, 2, \dots, n_l; l=1, 2, \dots, L \text{ and } k=1, 2, \dots, D_1 \text{-----(9)}$$

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

$$y_{1k} = \sum_{i=1}^{n_1} x_{p1k} = k, k=1, 2, \dots, D_1 \text{-----(11)}$$

$$x_{p1k} = x_{p1(k-1)}; p=1, 2, \dots, n_1 \text{ and } k=1, 2, \dots, D_1 \text{-----(12)}$$

$$x_{p1D_1} = d_1, x_{p10} = 0, p=1, 2, \dots, n_1 \text{-----(13)}$$

$$x_{ilk} = 0, \text{ integer } i=1, 2, \dots, n_l, l=1, 2, \dots, L \text{ and } k=1, 2, \dots, D_1 \text{-----(14)}$$

Constraint (9) ensures that the necessary cumulative production of part i of level l by the end of time unit k is determined explicitly by the quality of products produced at level l. Constraints (10) and (11) show the total cumulative production of level l and level 1, respectively, during the time units 1 through k. Constraint (12) ensures that the total

production of every product over k time units is a non-decreasing function of k . Constraint (13), (11), (12), (14) ensure that exactly one unit of a product is scheduled during one time unit in the product level. ORV problems are NP-hard in general. Two level ORV problems can be solved in pseudo-polynomial time.

Chapter 5

Min-max Problem

The sum of deviations type objective functions produce “smooth” schedules on average. They do not preclude, however, the possibilities of relatively large deviations in certain time periods, contrast the min-max objective function looks for “smooth” schedules in every time period. Min-max objective function has more applicable, physical interpretation than the min-sum functions. The min-max objective function value gives maximum overproduction (inventory) / underproduction (shortage) from the desired level of production that occurs at any time during schedule.

5.1 Graph and Matching Problems

A graph-theoretic approach is used to determine an optimal solution for Just-in-Time problems with non-convex objective function. A graph G is a pair $G = (V, E)$, where V is finite non-empty set of nodes(vertices) and $E \subseteq V \times V$ is a relation set of order pairs (u, v) . An edge between two vertices is denoted by $[u, v]$, consists of pairs (u, v) and (v, u) in the set E . A pair $(u, v) \in E$ is called an arc if pair $(v, u) \notin E$. If all pairs in E are arcs, the graph G is called directed graph. Graph G is called an undirected graph if all pairs in E are edges.

Let $G = (V, E)$ be a graph in which vertex set V can be portioned into two disjoint sets, V_1 and V_2 , and each edge in E has one vertex in V_1 and another in V_2 . In such case G is called bipartite graph. Bipartite graph is denoted by $G = (V_1 \cup V_2, E)$. Otherwise graph is called non-bipartite graph.

A graph $G = (V, E)$ is called a complete graph if $[u, v] \in E$ for all $u, v \in V$ with $u \neq v$. A bipartite graph $G = (V_1 \cup V_2, E)$ is called complete bipartite graph if each $u \in V_1$ is joined to each $v \in V_2$. A graph $G = (V, E)$ with a function $w: E \rightarrow Z$ is called a weighted graph, where Z is usually the set of positive integers.

Given a graph $G = (V, E)$, a matching M in G is a subset of the edge set E with the property that no two edges of M share the same node. A matching M in Graph G is called a maximum matching if no matching in G exists with cardinality more than that of M . The largest possible cardinality of a matching in a graph with $|V|$ nodes is $\lfloor |V|/2 \rfloor$. When the cardinality of a matching M in a graph $G = (V, E)$ is $\lfloor |V|/2 \rfloor$, M is called complete graph or perfect matching.

Steiner and Yeomons [33] study min-max problem reducing to a single machine scheduling decision problem with release time and due dates. They represent the problem as a perfect matching in a V_1 convex bipartite graph $G=(V_1UV_2, E)$ where $V_1=\{1, 2, \dots, D\}$ represents positions and $V_2=\{(i, j) \mid i=1, 2, \dots, n; j=1, 2, \dots, d_i\}$ represents the copies of the products. There exists an edge $\{k, (i, j)\} \in E$ if and only if k lies in the permissible interval $[E(i, j), L(i, j)] \subseteq V_1$ of release time and due date for the j^{th} copy of the product i .

5.2 Release Date/Due Date Decision Problem

As no general solution techniques exist which could handle such large integer programming problems, a special solution procedure is developed for the specific problem under considerations, Miltenburg [28]. Denote a target value for the objective function by the variable B . The goal is to determine the smallest possible B for which a sequence can be created for each $j(i)$ has a completion time k , such that $f_j^i(k) \leq B$ for $k \in [k_j, (k_{j+1}-1)]$. For target value B , $j(i)$ can not start before $k-1$ if $g_j^i - j - kr_i > B$ and can start k if $f_j^i(k+1) = j - (k+1)r_i \leq B$. Therefore, any fixed target value B allows the calculation of a release date and a due date for a specific copy of a product. For a given B early and late starting dates can be calculated for each copy of each product in a one pass procedure and, hence, can be constructed in $O(D)$ time.

The earliest starting time $E(i, j)$ for (i, j) must be the unique integer satisfying $\frac{j-B}{r_i} - 1 \leq E(i, j) < \frac{j-B}{r_i}$ and latest starting time $L(i, j)$ of (i, j) must be the unique integer satisfying $\frac{j-1+B}{r_i} - 1 < L(i, j) \leq \frac{j-1+B}{r_i}$.

Lemma 5.1 Let d_1, d_2, \dots, d_n be any instance of min-max-absolute problem. A sequence $s=s_1, s_2, \dots, s_n$ is B -feasible if and only if for all $i=1, 2, \dots, n$ and $j=1, 2, \dots, d_i$ this sequence assigns the copy (i, j) to the interval $[E(i, j), L(i, j)]$ where

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

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

denote the release date and the due date of the copy (i, j) for given upper bound B , Dhamala [10].

Other measure of deviation is still open. There are various versions of the earliest due date algorithm for scheduling unit time jobs with release times and due dates on a single machine. We can apply a modified version of Glover's EDD algorithm for finding maximum matching in a V_1 convex bipartite graph $G = (V_1UV_2, E)$ such that each ascending $k \in V_1$ is matched to the unmatched copy (i, j) with smallest due date value of $L(i, j)$. The optimal solution can be obtained by using the matching problem and bisection search within the bounds for target value.

5.3 Just-in-Time Sequencing with Input Sequences

Single level just in time sequencing problem is formulated under chain constraints. This sequence is denoted by JIT-Chain, Dhamala/ Kubiak [13].

$$\text{Let } \begin{aligned} u(n_1, D_1) &= u(n_1, D_1)_1 u(n_1, D_1)_2 \dots \dots \dots u(n_1, D_1)_{D_1} \\ u(n_2, D_2) &= u(n_2, D_2)_1 u(n_2, D_2)_2 \dots \dots \dots u(n_2, D_2)_{D_2} \\ &\vdots \\ u(n_t, D_t) &= u(n_t, D_t)_1 u(n_t, D_t)_2 \dots \dots \dots u(n_t, D_t)_{D_t} \end{aligned}$$

$$u(n_m, D_m) = u(n_m, D_m)_1 u(n_m, D_m)_2 \dots u(n_m, D_m)_{D_m}$$

be B_1, B_2, \dots, B_m feasible sequences of lengths D_1, D_2, \dots, D_m , where $D_t = \sum_{i=1}^m d_i^t$ of any given model sets $n_t, t=1, 2, \dots, m$, respectively. Different chains may contain the same type of product models. This is called overlapping system. Dhamala/ Kubiak [13] considers the problem with non-overlapping system.

They derive B-feasible sequence $s=s_1, s_2, \dots, s_n$ where $D = \sum_{t=1}^m d_t$ for min-max-absolute problem such that the restricted mapping satisfy $s|_{u(n_t, D_t)} : s \rightarrow u(n_t, D_t)$ for all $t = 1, 2, \dots, m$ and has the least maximum deviation. It means $F(s) = F(u)$ for any sequence $u = s_1, s_2, \dots, s_n$ satisfying $u|_{u(n_t, D_t)} : u \rightarrow u(n_t, D_t)$. The restriction $s|_{u(n_t, D_t)}$ of the super sequence s to any given sub-sequence $u(n_t, D_t), t=1, 2, \dots, m$ yields the sequence $u(n_t, D_t)$. Therefore, the super sequence s that contains $u(n_t, D_t)$ as its subsequence is order preserving with respect to the m-chain constraints $u(n_t, D_t)_l < u(n_t, D_t)_{l+1}$ for all $l=1, 2, \dots, D_t$ and $t=1, 2, \dots, m$. Such sequence is called order-preserving super sequence. By construction each subsequence represents a chain and there exist at most D constraints all together in these chains.

5.4 An Efficient Scheduling Algorithm

Consider collective demand rates of $n = \sum_{t=1}^m d_t$ models. Total demands is the union of all chains given by $D = \sum_{t=1}^m D_t$. For given bound B , permissible intervals of time windows is given by $[E(i, j), L(i, j)]$, where $i=1, 2, \dots, n$ and $j=1, 2, \dots, d_i$ using known algorithm of Steiner and Yeomans [33]. These time windows must be feasible without chain constraints for this data set.

To ensure B is feasible for the super sequence to be delivered, a test is required. The min-max-absolute-chain sequencing algorithm is reduced to a single machine scheduling decision problem with release times, due dates and chain constraints. Given bound B for min-max-absolute-chain problem, we find out whether a feasible solution of the single processor scheduling problem $1|r_i, \text{chain}|L_{\max}$, with $L_{\max} = 0$ exists or not.

Consider the problem $1| r_i, \text{chain} | L_{\max}$, represent time windows by the intervals $[r_i, d_i] = [E(i, j), L(i, j)]$ calculated as function of given bound B. The chain constraints are given by the subsequence $\bigcup_{i=1}^m \{u(n_i, D_i)^{D_i}\}$ that may be represented by following graph. Define a directed graph $G=(V, E)$ with vertex set $V = \bigcup_{i=1}^m \{u(n_i, D_i)^{D_i}\}$. There exists an arc E from $u(n_i, D_i)_k$ to $u(n_t, D_t)_k'$ if the precedence relation $u(n_i, D_i)_k < u(n_t, D_t)_k'$ is satisfied.

Horn [19] formulated $O(n \log n)$ time algorithm to the single machine scheduling problem $1|r_i, \text{chain}|L_{\max}$. This is called Earliest Due Date (EDD) algorithm and it schedules an available job with the smallest due date at any time. To implement this rule to $1|r_i, \text{chain}|L_{\max}$, the due date need to be modified. In this modification, if job k is the immediate predecessor of job l in any chain and $d_k' = d_l - 1 < d_k$, denoted by $k \rightarrow l$, the due date d_k has to be replaced by modified due date d_k' . A proof on the validity of optimality on L_{\max} makes the use of interchange arguments.

Following algorithm is proposed for the min-max-absolute-chain sequencing problem Dhamala/Kubiak [13].

5.5 Algorithm: min-max-absolute-chain-algorithm

Given: d_{it} for $i=1, 2, \dots, n$ and $t=1, 2, \dots, m$;
an upper bound B for min-max-absolute-chain-problem
 $\text{chain}_1, \text{chain}_2, \dots, \text{chain}_i, \dots, \text{chain}_m$;
Update :

number of demands $n = \sum_{t=1}^m n_t$;

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

total demand $D = \sum_{i=1}^n d_i$.

Step 1 : Calculate windows $[E(i, j), L(i, j)]$, where $i=1, 2, \dots, n$ and $j=1, 2, \dots, d_i$ by Steiner/ Yeomans [33].

Step 2 : Modify due dates $L(i, j)$:

If $(i, j) \prec (i', j')$, then $L(i, j) := \min\{L(i, j), L(i', j') - 1\}$

Step 3 : Schedule the job by EDD-Algorithm by Horn [19].

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

Step 1 and Step 2 requires $O(D)$ time and the Step 3 costs $O(D \log D)$. The overall time complexity of the min-max-absolute-chain-algorithm is $O(D \log D)$. An EDD algorithm in Step 3 applied to modified due dates by Step 2 is called modified EDD algorithm.

5.6 Correctness of min-max-absolute-chain-algorithm

Theorem 5.1 Let B be a target value for the objective function of min-max-absolute-chain sequencing problem. Then, if the modified EDD algorithm finds an optimal solution with $L_{\max} \leq 0$, then min-max-absolute-chain-algorithm finds a B -feasible solution to min-max-absolute-chain sequencing problem, Dhamala [10].

Proof: Suppose $s = s_1, s_2, \dots, s_d$ be a sequence obtained by min-max-absolute-chain-algorithm such that $L_{\max} \leq 0$. That is, each job $k = 1, 2, \dots, D$ is scheduled in the proper window and none of the job is delayed. If s is infeasible to min-max-absolute-chain sequencing problem, then $|x_{ik} - kr_i| > B$ for some product copy (i, j) with $k=1, 2, \dots, D$ and $i=1, 2, \dots, n$. But this is impossible by the construction of time windows.

If the first copy (i, 1) of the product i has to be completed at position k, then it must hold $|x_{ik} - kr_i| = 1 - r_i$. Therefore, the sharp lower bound $1 - r_{\max}$ on a target value B is still valid. An optimal solution to the min-max-absolute-chain problem has to be determined by applying binary search of the target value B in the interval $[1 - r_{\max}, B]$.

Upper bound to the obtained sequence is obtained by putting given sequence:

$\bigcup_{i=1}^m \{u(n_i, D_i)_{l=1}^{D_i}\}$ one after another and then calculate:

$$B = \max_{i,k} \{|x_{ik} - kr_i| : i=1, 2, \dots, n \text{ and } k=1, 2, \dots, D\}.$$

The properties of batch sequence also alter upper bound on target value B of super sequence s. A batch w is a factor of sequence s consisting of the same product copies which cannot be extended either to the right or to the left by same product type copy. |w| represents the batch size of the batch w in s. Clearly, longer batches reduce the number of setups provided sufficiently long buffer size. Given an instance (n, D), we consider a batch sequence s with exactly n-batches, say $s = \sigma_{i_1}, \sigma_{i_2}, \dots, \sigma_{i_n}$, where σ_{i_t} represents a batch with respect to the product type $i_t, t = 1, 2, \dots, n$.

Lemma 5.2 Let $s = \sigma_{i_1}, \sigma_{i_2}, \dots, \sigma_{i_n}$ in be a sequence with batches σ_{i_t} for $t = 1, 2, \dots, n$. Then an upper bound on the target value of s is $d_{\max}(1 - r_{\max})$, Dhamala [10].

Corollary 5.1 An upper bound on the target value of the super sequence s obtained by min-max-absolute-chain-algorithm is $d_{\max}(1 - r_{\max})$. Moreover, the tight lower bound is $1 - r_{\max}$, Dhamala [10].

Proof: An optimal solution to the min-max-absolute-chain problem can be determined by applying binary search in the interval $[1 - r_{\max}, d_{\max}(1 - r_{\max})]$. But a feasibility test requires $O(D \log D)$ time.

As the Horn [19] algorithm works for the problem $1|r_i, \text{prec}|L_{\max}$ this approach is applicable for the min-max-absolute-chain problem with precedence constraint as well. The time complexity of the problem remains intact.

Example 5.1 Given two sequence $u(3,11) = \text{bcbbcebcbbc}$ and $u(2,9) = \text{aadaaaada}$, the super sequence $s = \text{abcabdabcaacbaebacbadbea}$ preserves the orders of the subsequence $s|_{u(3,11)} = \text{bcbbcebcbbc}$ and $s|_{u(2,9)} = \text{aadaaaada}$. Moreover the obtained super sequence s is optimal as $B = 1 - r_{\max} = 1 - 7/20 = 0.65$ is tight, Dhamala [10].

Note the first subsequence of the input subsequence is not optimal for: $|3 - (4 \times 6/11)| = 9/11 > 6/11$ for the third copy product b . But the sequence $u(3,11) = \text{bcbbcebcbbc}$ is optimal with upper bound $6/11$.

Chapter 6

Cyclic Scheduling

A cyclic schedule is a sequence of tasks that are executed repeatedly so that each task is performed exactly once during each cycle. In a multi-machine, multi-product setting, precedence constraints exist between successive tasks on the same lot of a product as well as those defined by the prescribed production sequence on each machine. The use of these schedules has become increasingly prevalent in recent years in various manufacturing environments because they are significantly simpler to describe, understand, and implement than many other scheduling schemes. Communication with the shop floor is improved since the sequence is fixed. There is no need for dispatching decisions. Cyclic scheduling is consistent with the just-in-time or minimum inventory philosophy in two important respects. First, the cyclic sequence aims to globally minimize inventory by having production ready just when it is needed. Second, efforts to reduce setup times are more effective since they can be directed at the specific changeovers that occur in the sequence. Also, the task of coordinating other activities such as raw material delivery, preventive maintenance, and work force schedules becomes simpler. Whybark [34] provides an excellent early description of a cyclic type of production control and the benefits that were realized.

Cyclic scheduling or periodic production has received considerable attention in recent years as an effective technique for repetitive manufacturing. The approach allows for global consideration of inventory costs, setup costs, and work center capacities. The effort to find a good sequence is rewarded by the fact that the sequence will be repeated. Management of the schedule is facilitated by this repetition; it allows for easier communication and planning, and setup reduction efforts can be focused on a much smaller number of transitions.

Fundamental notions of cyclic scheduling for repetitive manufacturing are described in the book by Gessner [15]. Successful practical implementations of the cyclic scheduling paradigm have been reported by Whybark [34] at Kumera Oy (a manufacturer of

transmissions for automobiles) and by Cunninghame-Green [7] in an industrial steelworks. For a more detailed literature survey, refer to Rao [32].

Hall [17] has discussed the numerous advantages of cyclic scheduling. For example, cyclic schedules are non-myopic in the sense that they consider an infinite horizon, and capture within the “cycle length” what happens in steady state over that infinite horizon, while giving us valuable information on system throughput and production lead times. Cyclic schedules result in easier shop floor control (by virtue of the structure and repetitiveness imposed on the facility by cyclic schedules). This allows production planners to concentrate their efforts on reducing manufacturing inefficiencies involving unproductive machine set-up times or yield/quality problems. Cyclic schedules also offer an effective way of modeling the production capacity of an “ideal” facility. In such situations, cyclic scheduling can be looked upon as a “higher level” (tactical) planning and scheduling tool by aggregating over different items with similar routing and processing requirements.

One of the drawbacks of efficient cyclic schedules is that minor variability in the system can disrupt the cyclic schedule, making real-time dynamic control under uncertainty a challenging problem. However, results in McCormick et al [27] show that in most cases a system can recover its steady state after a disruption quite quickly. One possible way to hedge against this contingency is to incorporate some safety time and buffer stock into the system.

The complexity of cyclic scheduling has not been as thoroughly investigated as has the complexity of static scheduling. Just as in static scheduling, one does not have to push cyclic scheduling problems very far before they become intractable. However, cyclic scheduling still excites a lot of interest because of the belief that cyclic scheduling is often superior in practice to static scheduling.

6.1 Cyclic Scheduling Problems

Deterministic scheduling is concerned with entities that need work called jobs, and with the resources that work on jobs called machines. Typically each job consists of several operations, where each operation is required to be done on a particular machine. There is some sort of processing time associated with each operation on each machine, and a machine typically can work on only a single operation at a time. The objective is then to find a schedule that is feasible (does not require a machine to process more than one operation at a time) and which minimizes some objective function. For example, we may wish to minimize the makespan, which is the latest finishing time of any operation of any job.

In a flow shop, each job has the same set of operations, call them 1, 2, . . . , n (although job j 's operation i can have a different processing time than job k 's), and each operation i must be done on the same machine. Furthermore, operations must be done in consecutive order.

In a typical non-cyclic scheduling problem (static scheduling), the set of jobs we must schedule is static and finite. For example, we must schedule all of one day's production in a job shop to minimize makespan. By contrast, cyclic scheduling has a desired pattern of output over time that we want to produce in a steady state. For example, we might want to produce 200 model A's, 400 model B's, and 800 model C's every day for the foreseeable future. We can achieve this by designing a schedule that produces 200 replications of a cycle consisting of one model A, two model B's, and four model C's. Here we are not so much interested in minimizing the makespan of a single cycle, but rather in making sure that consecutive cycles fit together well.

This requires a different sort of objective such as minimizing cycle time (sometimes called cycle length), which is the time between starting the first operation of the first job in a cycle, and starting the first operation of the first job in the next cycle. Cycle time is roughly equivalent to static scheduling's makespan. In the same way that static makespan

problems can be modeled via precedence constraints graphs, where earliest start times correspond to longest paths, cycle time problems in cyclic scheduling can be modeled with precedence constraints graphs that have been “wrapped around a cylinder” so that the final job(s) in a cycle can influence the first job(s) in the next cycle, McCormick et al [27]. Here a smaller cycle time means that we are producing a larger number of finished units per time, i.e., minimizing cycle time is equivalent to maximizing throughput.

However, modern manufacturing is concerned not only with large throughput, but also with minimizing work-in-process (WIP) inventories. We can measure this by counting the elapsed time between when the first operation of a job starts and when its last operation finishes, its flow time. For example, if job j 's flow time in a cyclic schedule spans four cycles then there are four jobs j simultaneously in the system (i.e., $WIP_j = 4$). In more complicated situations we shall see that there is a trade-off between a small cycle time (large throughput) and a small flow time (low WIP).

We now briefly survey previous complexity work in cyclic scheduling. Hsu [20] shows that the Economic Lot Scheduling Problem on a single machine is NP-complete. A different problem that can be seen as a single machine cyclic scheduling problem involves traffic-signal control at a single intersection. Dauscha et al [8] consider the feasibility of scheduling such fixed-duration periodic operations within a given interval of time. They show that their cyclic scheduling problem is strongly NP-complete, but that given the “cyclic sequence type”, the resulting sub problem is polynomially solvable.

6.2 Cyclic Scheduling with Minimal Part Set

Consider an assembly line with m machines. There are n different types of jobs to be assembled. We have a flow shop where each job j visits every machine i in consecutive order and spends time p_{ij} (possibly zero) there. Given a minimum required production rate for each job type, the problem is to operate the assembly line to maximize throughput. We restrict attention to cyclic production schedules in which a Minimal Part Set (MPS) consisting of a known mix of jobs is produced repetitively. Here is an example from McCormick et al [27] for the special case where all p_{ij} are 0 or 1.

Example 6.1 There are four machines and six jobs (A, B, C, D, E & F) in the Minimal Part Set (MPS). The processing requirements matrix is:

$$P_{ij} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Since the largest row sum is five, the minimum possible cycle length is five. Does this instance have a cyclic schedule with cycle length five?

If we order the jobs in the same sequence as the columns in the matrix above, we get an optimal schedule with cycle time five. A complete schedule over an infinite horizon is obtained by repeating the above schedule every five time units.

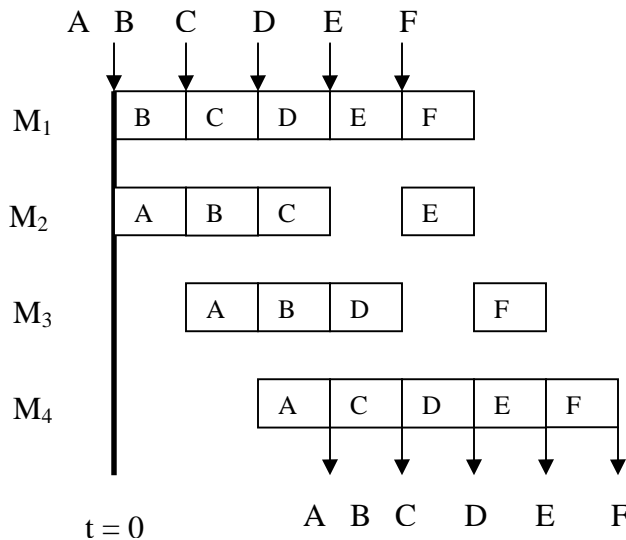


Figure 6.1 Minimum cycle length schedule

The problem of determining the optimal (i.e., maximum throughput) permutation sequence in which to process the jobs in the MPS is called the Sequencing Problem (SP). This problem was studied in McCormick et al [27] where it was claimed that two special cases of SP are strongly NP-hard.

6.3 Cyclic Solution to PRV Problem

The cyclic solution to the PRV problem has received growing attention. The time complexities of all heuristic algorithms are pseudo-polynomial. To reduce the complexity concept of cyclic sequence is introduced. Miltenburg [28] observes existence of cyclic sequence in min-sum problem (sum deviation). Miltenburg states a concatenation of s^m copies of an optimal sequence s for instance (d_1, d_2, \dots, d_n) is optimal for $(md_1, md_2, \dots, md_n)$, $m \geq 1$ to build a sequence for a larger time horizon. Bautista, Companys and Corominas[1] have proven an affirmative answer to optimality of cyclic approach provided that $f_i = f$ for all i such that function f is convex and symmetric with minimum $f(0) = 0$. The cornerstone of their proof is an observation that even with the constraints $x_{iD} = d_i, i=1, 2, \dots, n$, relaxed there still exists an optimal sequence. Kubiak and Kovalyov [24] extend the result to be true if all f_i are convex and symmetric and equal in the interval $(0,1)$ but not true even if a single f_i is asymmetric.

6.4 Cyclic Solution to min-max-absolute-chain Problem

Consider single level just in time sequencing problem formulated under chain constraints.

$$\text{Let } u(n_1, D_1) = u(n_1, D_1)_1 u(n_1, D_1)_2 \dots \dots \dots u(n_1, D_1)_{D_1}$$

$$u(n_2, D_2) = u(n_2, D_2)_1 u(n_2, D_2)_2 \dots \dots \dots u(n_2, D_2)_{D_2}$$

$$u(n_t, D_t) = u(n_t, D_t)_1 u(n_t, D_t)_2 \dots \dots \dots u(n_t, D_t)_{D_t}$$

$$u(n_m, D_m) = u(n_m, D_m)_1 u(n_m, D_m)_2 \dots \dots \dots u(n_m, D_m)_{D_m}$$

be B_1, B_2, \dots, B_m feasible sequences of lengths D_1, D_2, \dots, D_m , where $D_t = \sum_{i=1}^m d_i^t$ of any

given model sets $n_t, t=1, 2, \dots, m$, respectively.

We demonstrate some examples before putting cyclic version of algorithms for the min-max-absolute-chain-algorithm as specified in Chapter 5.5. Consider we have two chains: chain1 and chain2. The number of jobs in these two chains may vary, the number of cycle

that can be formed in each chain may vary and the length of cycle that has been formed in these two jobs may differ. Taking these situations in account, we have tried to make cyclic version of min-max-absolute-chain-algorithm under constraints.

Case 1: Number of cycle and length of cycle is equal

Example 6.1

Suppose two chains,

Chain1 = ababab

Chain2 = fgfgfg

For Chain1:

Number of cycle = 3

Cycle length = 2

For Chain2:

Number of cycle = 3

Cycle length = 2

The number of cycle and length of cycle is equal.

Algorithm 6.1

1. Find the length of cycle c .
2. Take first c elements from each chain and make a jobs sequence holding constraints.
3. Schedule the jobs from step 2 by algorithm 5.5 (min-max-absolute-chain-algorithm) by Dhamala/Kubiak [13].
4. Repeat step 3 until the production satisfies demand.

Notice that Cyclic Scheduling is observed in step 4. Here demand is the ratio (length of chain /cycle length).

Illustration 6.1

For Example 6.1:

Chain1 = ababab

Chain2 = fgfgfg

The Algorithm 6.1 gives length of cycle, $c = 2$.

We take two elements from each chain holding precedence constraint. The output sequence is : abfg abfg abfg

$(abfg)_3$

We execute abfg 3 times.

Case 2: Number of cycle is equal and length of cycle differs in different chain

Example 6.2

Suppose two chains,

Chain1 = abcabcabc

Chain2 = fgfgfg

For Chain1:

Number of cycle = 3

Cycle length = 3

For Chain2:

Number of cycle = 3

Cycle length = 2

The number of cycle is equal and length of cycle differs in different chain. We use the concept of false job in this situation. We insert false job in chains such that the length of cycle in each chain is equal.

The chains are updated as

Chain1 = abcabcabc

Chain2 = fgc'fgc'fgc'

Where c' in chain 2 is false job.

Algorithm 6.2

Update Chains inserting false jobs to make length of cycle same

Use Algorithm 6.1

Remove false jobs

Illustration 6.2

For Example 6.2:

Chain1 = abcabcabc

Chain2 = fgc'fgc'fgc'

The Algorithm 6.2 gives length of cycle, $c = 3$.

We take three elements from each chain holding precedence constraint. The output sequence is: abcfgc' abcfgc' abcfgc'

$(abcfgc')_3$

Remove false jobs c' . We get

$(abcfg)_3$

We execute abcfg 3 times.

Case 3: Number of cycle differs in different chain and length of cycle is same

Example 6.3

Suppose two chains,

Chain1 = ababab

Chain2 = fgfg

For Chain1:

Number of cycle = 3

Cycle length = 2

For Chain2:

Number of cycle = 2

Cycle length = 2

The number of cycle differs in different chain and length of cycle same. We can use the concept of false job in this situation. We insert false job in chains such that the length of cycle in each chain is equal.

The chains are updated as

Chain1 = ababab

Chain2 = fgfga'b'

Where a', b' are false jobs.

Alternative way to solve this sequence is to take minimal number of cycle in the first phase. Then append the remaining cycles in it.

Algorithm 6.3

Update Chains inserting false jobs to make number of cycle same

Use Algorithm 6.1

Remove false jobs

Illustration 6.3

For Example 6.3:

Chain1 = ababab

Chain2 = fgfgf'g'

The Algorithm 6.3 returns length of cycle, $c = 2$.

We take two elements from each chain holding precedence constraint. The output sequence is : abfg abfg abf'g'

(abfg)₃

We execute abfg 3 times.

Remove false jobs f' and g'.

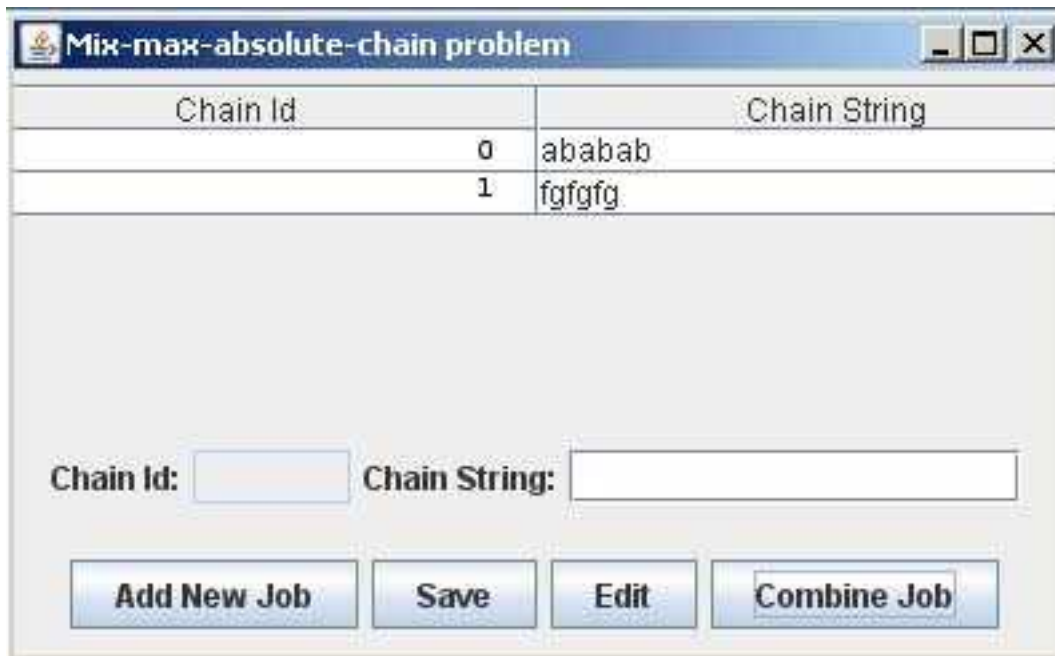
Chapter 7

Implementation and Testing

All the algorithms mentioned in Chapter 6 were implemented for min-max-absolute-chain-algorithm specified in Chapter 5. The program scripts are written in Java Version 1.6.0. The source codes for these programs are included in Appendix. The input data set (chains) represent the demands of products in the mixed model assembly line manufacturing system.

All the situations mentioned in Chapter 6 have different input parameters as follows:

Case 1: Number of cycle and length of cycle is equal



Chain Id	Chain String
0	ababab
1	fgfgfg

Chain Id: Chain String:

Figure 7.1 Input window when number of cycle and length of cycle is equal

Chain Id	Job Name	Earliest Due Date	Late Due Date
0	a	2.0	3.0
0	b	2.0	3.0
0	a	6.0	7.0
0	b	6.0	7.0
0	a	10.0	11.0
0	b	10.0	11.0
1	f	2.0	3.0
1	g	2.0	3.0
1	f	6.0	7.0
1	g	6.0	7.0
1	f	10.0	11.0
1	g	10.0	11.0

Figure 7.2 Calculation of early and late due date

Chain Id	Job Name	Earliest Due Date	Late Due Date
0	a	2.0	3.0
0	b	2.0	3.0
0	a	6.0	7.0
0	b	6.0	7.0
0	a	10.0	11.0
0	b	10.0	11.0
1	f	2.0	3.0
1	g	2.0	3.0
1	f	6.0	7.0
1	g	6.0	7.0
1	f	10.0	11.0
1	g	10.0	11.0

Schedule :
a0 b0 f1 g1 a0 b0 f1 g1 a0 b0 f1 g1

Figure 7.3 Modified due dates and the generated schedule

Case 2: Number of cycle is equal and length of cycle differs in different chain

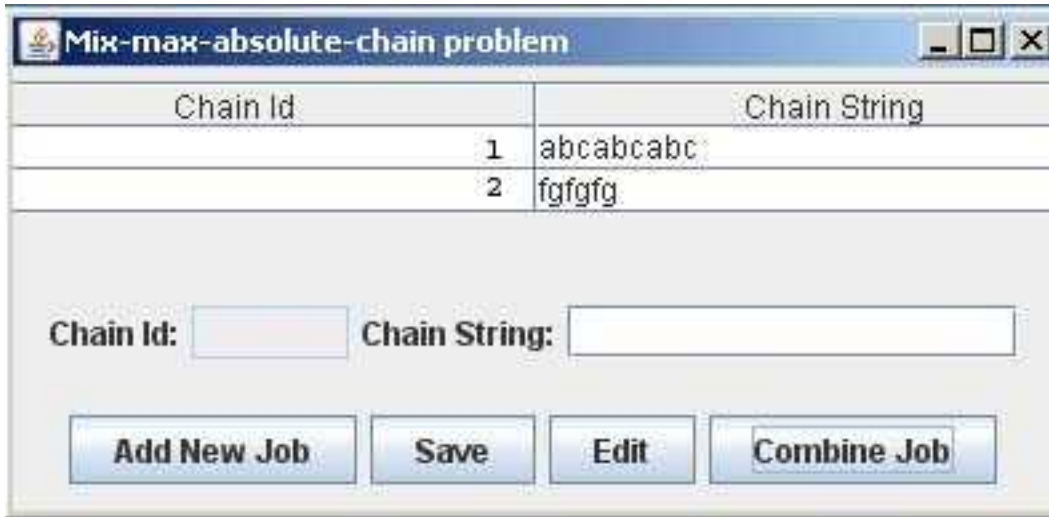


Figure 7.4 Input window for equal number of cycle with different cycle length

Chain Id	Job Name	Earliest Due Date	Late Due Date
0	a	3.0	3.0
0	b	3.0	3.0
0	c	3.0	3.0
0	a	8.0	8.0
0	b	8.0	8.0
0	c	8.0	8.0
0	a	13.0	13.0
0	b	13.0	13.0
0	c	13.0	13.0
1	f	3.0	3.0
1	g	3.0	3.0
1	f	8.0	8.0
1	g	8.0	8.0
1	f	13.0	13.0
1	g	13.0	13.0

Figure 7.5 Calculation of early and late due date

Chain Id	Job Name	Earliest Due Date	Late Due Date
0	a	3.0	3.0
0	b	3.0	3.0
0	c	3.0	3.0
0	a	8.0	8.0
0	b	8.0	8.0
0	c	8.0	8.0
0	a	13.0	13.0
0	b	13.0	13.0
0	c	13.0	13.0
1	f	3.0	3.0
1	g	3.0	3.0
1	f	8.0	8.0
1	g	8.0	8.0
1	f	13.0	13.0
1	g	13.0	13.0

Schedule :
a0 b0 c0 f1 g1 a0 b0 c0 f1 g1 a0 b0 c0 f1 g1

Figure 7.6 Modified due dates and the generated schedule

Case 3: Number of cycle differs in different chain and length of cycle is same

Chain Id	Chain String
1	ababab
2	fgfg

Chain Id: Chain String:

Figure 7.7 Input window when number of cycle differs but same length

Chain Id	Job Name	Earliest Due Date	Late Due Date
0	a	2.0	2.0
0	b	2.0	2.0
0	a	5.0	6.0
0	b	5.0	6.0
0	a	9.0	9.0
0	b	9.0	9.0
1	f	2.0	2.0
1	g	2.0	2.0
1	f	5.0	6.0
1	g	5.0	6.0

Figure 7.8 Calculation of early and late due date

Chain Id	Job Name	Earliest Due Date	Late Due Date
0	a	2.0	2.0
0	b	2.0	2.0
0	a	5.0	6.0
0	b	5.0	6.0
0	a	9.0	9.0
0	b	9.0	9.0
1	f	2.0	2.0
1	g	2.0	2.0
1	f	5.0	6.0
1	g	5.0	6.0

Schedule :
a0 b0 f1 g1 a0 b0 f1 g1 a0 b0

Figure 7.9 Modified due dates and the generated schedule

Chapter 8

Conclusion and Recommendation

8.1. Conclusion

Flexible transfer lines or mixed model assembly lines are capable of diversified small lot production due to negligible switch over costs. It is possible to implement Just-in-Time production with these lines, which produces only the necessary products in the necessary quantities at the necessary time. Many heuristic algorithms to solve the problem have appeared in the literature. Just-in-Time problem can be reduced to assignment problem and then can be solved more efficiently. Our concern in this dissertation, however, is to solve the problem in cyclic paradigm.

We have developed a formulation for solving min-max-absolute-chain problem in cyclic paradigm. Under certain constraints, we have implemented these formulations and shown that it can be used to generate good cyclic schedules. We have not been able to do performance comparison between our approach and other techniques, but the formulation presented here has other clear advantage in terms of our understandability, and in terms of being able to exploit existing constraint reasoning techniques and heuristics.

8.2 Recommendation

It is open whether the min-max problem with such constraints and/or min-max problem with overlapping sequences as constraints are efficiently solvable in cyclic paradigm. Perhaps most importantly, based on these formulations under constraints further research can build in this domain for general case.

References

1. Bautista, J., Companys, R. and Corominas, A., "A note on relation between the product rate variation problem and the apportionment problem", *Journal of the Operations Research Society* 47, 11 (1996) 1410-1414.
2. Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, C. and Weglarz, J., "Scheduling computer and manufacturing processes", Springer, Berlin (1996).
3. Blum, M., "A machine-independent theory of the complexity of recursive functions", *Journal of the ACM* 14, 2 (1967) 322-336.
4. Brouner, N. and Crama, Y., "The maximum deviation just-in-time scheduling problem", *Discrete Applied Mathematics* 134 (2004) 25-50.
5. Carlier, J. and Chretienne, P., "Problemes d'ordonnancement: modelisation / complexite / algorithms", Masson, Paris (1988).
6. Corominas, A. and Moreno, N., "Solving the min-max product rate variation problem as a bottleneck assignment problem", *Computers and Operations Research* 33 (2006) 928-939.
7. Cunninghame Green, R.A., "Minimax algebra, lecture notes in economics and mathematical systems", New York: Springer Verlag 166 (1979).
8. Dauscha, W., Modrow, H. D., and Neumann, A., "On cyclic sequence types for constructing cyclic schedules", *Zeitschrift für Operations Research* 29 (1985) 1-30.

9. Deutsch, D., "Quantum theory, the Church-Turing principle and the universal quantum computer", *Proceedings of the Royal Society of London A*, 400:97, (1985).
10. Dhamala, T. N., "Just-in-time sequencing algorithms for mixed-model production systems", *The Nepali Math. Sci. report* 24, 1 (2005) 25-34.
11. Dhamala, T. N. and Khadka, S.R., "Just-in-time sequencing for mixed-model production systems revisited", submitted to *Discrete Optimization* 2007.
12. Dhamala, T. N. and Kubiak, W., "A brief survey of just-in-time sequencing for mixed-model systems", *International Journal of Operational Research* 2, 2 (2005) 38-47.
13. Dhamala, T. N. and Kubiak, W., "Optimal just-in-time sequences for mixed-model multi-level production", Working Paper, Memorial University of Newfoundland, Canada (2005).
14. Ehrgott, M., "Multiple criteria optimization: classification and methodology", PhD thesis, University of Kaiserslautern, Germany (1997).
15. Gessner, R. A., "Repetitive manufacturing production planning", Wiley-Interscience, New York (1988).
16. Graham, R.E., Lawer, E.L., Lenstra, J.K., and Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling, a survey", *Annals of Discrete Mathematics* 5 (1979) 287-326.
17. Hall, R.W., "Cyclic scheduling for improvement", *International Journal of Production Research* 26, 3 (1988) 457-472.

18. Hartmanis, J. and Stearns, R.E., "On the computational complexity of algorithms", Transactions of the AMS 117 (1965) 285-306.
19. Horn, W.A., "Some simple scheduling algorithms", Naval Research Quarterly 21 (1974) 177-185.
20. Hsu, W-L., "On the general feasibility test of scheduling lot sizes for several products on one machine", Management Science 29 (1983) 93-105.
21. Jost, V. "Deux problèmes d'approximation diophantique: le partage proportionnel en nombres entiers et les pavages équilibrés de z ", DEA ROCO, Laboratoire Leibniz-IMAG (2003).
22. Kubiak, W., "Cyclic just-in-time sequence are optimal", Journal of Global Optimization 27 (2003) 333-347.
23. Kubiak, W., "Minimizing variation of production rates in just-in-time systems: A survey", European Journal of Operational Research 66 (1993) 259-271.
24. Kubiak, W. and Kovalyov M.Y., "Product rate variation problem and greatest common divisor property", Working Paper, Faculty of Business Administration, MUN, St. Johns (1998) 98-51.
25. Kubiak, W. and Sethi, S., "Level schedules for mixed model assembly lines in just-in-time" production system", Management Science 37, 1 (1991) 121-122.
26. Liu, J. L. and MacCarthy, B. L., "The classification of FMS scheduling problems", International Journal of Production Research 34, 3 (1996) 647-656.

27. McCormick, S. T., Pinedo, M. L., Shenker, S. and Wolf, B., "Sequencing in an assembly line with blocking to minimize cycle time", *Operations Research* 37, 6 (1989) 925-935.
28. Miltenburg, J., "Level schedules for mixed-model assembly lines in just-in-time production system", *Management Science* 35 2 (1989) 192-207.
29. 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.
30. Monden, Y., "Toyota production system", Industrial Engineers and Management Press, Norcross, GA (1983).
31. Pinedo, M., "Scheduling - theory, algorithms, and systems", Prentice Hall, Englewood Cliffs (1995).
32. Rao, U. S., "Multi-stage, identical job, cyclic scheduling for repetitive manufacturing", Technical Report 1029, SORIE, Cornell University, Ithaca, NY (1992).
33. Steiner, G. and Yeomans, S., "Level schedules for just-in-time production process", *Management Science* 39 (1993) 728-735.
34. Whybark, D. C., "Production planning and control at Kumera Oy", *Production and Inventory Management* 1 (1984) 71-82.

Bibliography

1. Aigbedo, H., "Some structural properties for the just-in-time level schedule problem", *Production Control and Planning* 11 (2000) 357-362.
2. Bowman, R. A. and Muckstadt, J. A. "Production control of cyclic schedules with demand and process variability", *Production and Operation Management* 4, 2 (1995) 145-162.
3. Ding, F.Y. and Cheng, L., "An effective mixed-model assembly line sequencing heuristic for just-in-time production system", *Journal of Operations Management* 11 (1993) 45-50.
4. Glover, F., "Maximum matching in a convex bipartite graph", *Naval Research Logistics Quarterly* 4 (1967) 313-316.
5. Groefin, H., Luss, H., Rosenwein, M.B. and Wahls, E.T., "Final assembly sequencing for just-in-time manufacturing", *International Journal of Production Research* 27, 2 (1989) 199-213.
6. Inman, R. R. and Bulfin, R. L., "Sequencing just-in-time mixed-model assembly lines", *Management Science* 37, 7 (1991) 901-904.
7. Kubiak, W., and Sethi, S., "Optimal just-in-time schedules for flexible transfer lines", *International Journal of Flexible Manufacturing Systems* 6 (1994) 137-154.
8. Miltenburg, J. and Goldstein, T., "Developing production schedules which balance part usage and smooth production loads for just-in-time production systems", *Naval Research Logistics* 38 (1991) 893-910.

9. Minoux, M., "Mathematical programming, theory and algorithms" Wiley, Newyork (1986).

10. Sumichrastm, Russel, R.T. and Taylo, R.S., "A comparative analysis of sequencing procedure for mixed-model assembly lines in a just-in-time production system", International Journal of Production Research 30, 1 (1992) 199-214.

Appendix A

Basic Mathematical Notations

Set Theory, Sequence and Series

N	Set of natural numbers
R	Set of real numbers
R^+	Set of positive real numbers
$\{a_1, a_2, \dots, a_n\}$	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 of Problems

n	Number of jobs
m	Number of machines
l	Product level
J_i	Job number $i, i = 1, \dots, n$
n_i	Number of operations of job J_i
m^l	Number of machines at stage l
M_j	Machine number $j, j = 1, \dots, m$
$O_{i,j}$	Operation j of job J_i
r_i	Release time of job J_i
d_i	Duedate of job J_i
s_i	Desired start time of job J_i
$p_{i,j}$	Processing time of operation $O_{j,j}$
W_i or w_i	Weight associated to job J_i
D	Total demand

Variable of Problems

$t_{i,j}$	Start time of operation $O_{i,j}$
$C_{i,j}$	Completion time of operation $O_{i,j}$

C_i	Completion time of job J_i
T_i	Tardiness of job J_i
E_i	Earliness of job J_i
L_i	Lateness of job J_i
$E(i, j)$	Release date of the copy (i, j)
$L(i, j)$	Due date of the copy (i, j)

Optimality Criteria

C_{\max}	Makespan or maximum completion time
T_{\max}	Maximum tardiness of jobs
L_{\max}	Maximum lateness of jobs
E_{\max}	Maximum earliness of jobs
F_{\max}	Maximum flowtime of jobs
I_{\max}	Total idle times on resource
	Average completion time of jobs
w	Average weighted completion time of jobs
	Average earliness of jobs
w	Weighted earliness of jobs
f_{\max}	Min-max criterion
	Min-sum criterion

Machine Environment

	Single machine
P	Identical machines
Q	Uniform machines
R	Unrelated machines
F	Flowshop
J	Jobshop
O	Openshop
X	Mixedshop
m	The number of machines or stages is fixed

Appendix B

Program Source Code

1 Package: MinMaxAbsoluteChain.com

1.1 File: RunCombineChain.java

```
package MinMaxAbsoluteChain.com;
import java.util.Vector;
import MinMaxAbsoluteChain.POJO.Chain;
import MinMaxAbsoluteChain.POJO.Job;
import util.*;

public class RunCombineChain {

    int totalChain = 0; //the size of chain
    int n = 0; //total number of job
    int d[] = new int[100]; //demand of job
    double B = 0.5; //constraint
    int D = 0;

    char jobChar[] = new char[100];
    int numberOfJobAtChain[] = new int[10];
    String chain[] = new String[10];
    String validChain[] = new String[10];
    int demandForJob[] = new int[100];
    double r[] = new double[100];
    int totalDemand[] = new int[100];
    int jobDemand[][] = new int[100][100];
    char job[][] = new char[100][100];
    private String combinedChain= null;

    Vector <Job>cat = new Vector<Job>();
    Vector <Job>jobSchedule = new Vector<Job>();

    private ListViewModel Datamodel=new ListViewModel();

    public RunCombineChain(Vector vec){
        try{
            totalChain=vec.size();
            for(int i=0;i<totalChain;i++){
                chain[i]=((Chain)vec.elementAt(i)).getChain_String();
                totalDemand[i]=chain[i].length();
                int jobNo=0;
            }
        }
    }
}
```



```

        for(int l=0;l<totalDemand[i];l++){
            Job testJob=new Job();
            testJob.setJobPosition(l);
            testJob.setJobChar(chain[i].charAt(l));
            testJob.setChainId(i);
            cat.add(testJob);

            if(l==0){
                job[i][jobNo]=chain[i].charAt(0);
                jobDemand[i][jobNo]=1;
                numberOfJobAtChain[i]=1;
                jobNo++;
            }else{
                boolean jobFound=false;
                for(int k=0;k<numberOfJobAtChain[i];k++){
                    if(chain[i].charAt(l)==job[i][k]){
                        jobFound=true;
                        jobDemand[i][k]++;
                        break;
                    }
                }
                if(jobFound==false){
                    job[i][jobNo]=chain[i].charAt(l);
                    jobDemand[i][jobNo]=1;
                    jobNo++;
                    numberOfJobAtChain[i]++;
                }
            }
        }
    }
}

}catch(Exception e){
    e.printStackTrace();
}

for(int i=0;i<totalChain;i++){
    n+=numberOfJobAtChain[i];
    D+=totalDemand[i];
}

int jobId=0;
for(int i=0;i<totalChain;i++){
    for(int j=0;j<numberOfJobAtChain[i];j++){
        d[jobId]=jobDemand[i][j];
        jobChar[jobId]=job[i][j];
        r[jobId]=((double)jobDemand[i][j])/((double)D);
        jobId++;
    }
}

//calculate window

```

```

this.Datamodel.setMatrix(this.totalDemand,4);
String []s={"Chain Id","Job Name","Earliest Due Date","Late Due Date"};
Datamodel.setColName(s);

int jobIndex=0;
for(int i=0;i<totalChain;i++){
    for(int k=0;k<numberOfJobAtChain[i];k++){
        for(int j=1;j<=jobDemand[i][k];j++){
            int jobAtVector=getJobPositionInVec(jobChar[jobIndex],i,j);
            cat.elementAt(jobAtVector).setE(DoubleUtil.getRoundDouble(Math.ceil((j-B)/r[k]),2));

            cat.elementAt(jobAtVector).setL(DoubleUtil.getRoundDouble(Math.floor(((j-
1.0+B)/r[k])+1.0),2));
        }
        jobIndex++;
    }
}

for(int i=0;i<cat.size();i++){
    Job j=cat.elementAt(i);
    Datamodel.setValueAt(i,0,""+j.getChainId());
    Datamodel.setValueAt(i,1,""+j.getJobChar());
    Datamodel.setValueAt(i,2,""+j.getE());
    Datamodel.setValueAt(i,3,""+j.getL());
}

new ShowSchedule(Datamodel,"Calculation of Window Value","");
/*
 * modify due date
 * (i, j)-> (i', j') then L(i,j)=min{L(i,j),L(i',j')-1}
 * in the vector the jobs are saved saving their precedence just check
 * either they fall on same chain or not
 */
for(int i=cat.size()-1;i>0;i--){
    Job j=cat.elementAt(i);
    Job j_prev=cat.elementAt(i-1);
    if(j.getChainId()==j_prev.getChainId()){
        if(j_prev.getL(>)>j.getL()-1){
            cat.elementAt(i-1).setL(j.getL());
        }
    }
}

/* Table Model to show data in window */
for(int i=0;i<cat.size();i++){
    Job j=cat.elementAt(i);
    Datamodel.setValueAt(i,0,""+j.getChainId());
    Datamodel.setValueAt(i,1,""+j.getJobChar());
    Datamodel.setValueAt(i,2,""+j.getE());
    Datamodel.setValueAt(i,3,""+j.getL());
}

```

```

    }

// Combine Chain
try{
    while(cat.size(>0){
        EDD_Algorithm();
    }
}catch(Exception e){
    e.printStackTrace();
}

this.combinedChain=this.getCombinedChain();
new ShowSchedule(Datamodel,"Modified Due Date","<html>Schedule
:<br>" +this.combinedChain+"</html>");
}

public Vector <Job>getModifiedDueDate(){
    return this.cat;
}

public int getJobPositionInVec(char jobChar ,int chainId,int pjobOrder){
    int jobOrder=0;

for(int i=0;i<cat.size();i++){
    Job j=cat.elementAt(i);
    if(j.getChainId()==chainId && j.getJobChar()==jobChar){
        jobOrder++;
        if(jobOrder==pjobOrder)
            return i;
    }
}
return -1;
}

private void EDD_Algorithm(){
    Vector E=new Vector();
    int AI;
    AI=minAvalilableTime();
    E=getJobsAvailableAt(AI);
    Job j=getEarliestDueDateJob(E);

    cat.remove(j);
    this.jobSchedule.add(j);
    System.out.println("EDD ALgorithm");
}

public String getCombinedChain(){
    this.combinedChain="" +this.jobSchedule.elementAt(0).getJobChar()+this.jobSchedule.e
lementAt(0).getChainId()+" ";
    for(int i=1;i<this.jobSchedule.size();i++){
        this.combinedChain+=jobSchedule.elementAt(i).getJobChar()+" "+this.jobSchedule.ele
mentAt(i).getChainId()+" ";
    }
}

```

```

    }
    return this.combinedChain;
}

private Job getEarliestDueDateJob(Vector E){
    int minTime=Integer.MAX_VALUE;
    Job j=null;
    Job temp=null;
    for(int i=0;i<E.size();i++){
        temp=(Job)E.elementAt(i);
        if(temp.getL(<minTime){
            j=(Job)E.elementAt(i);
            minTime=(int)j.getL();
        }
    }
    return j;
}

private int minAvalilableTime(){
    int minTime=Integer.MAX_VALUE;
    for(int i=0;i<cat.size();i++){
        Job j=(Job)cat.elementAt(i);
        if(j.getE(<minTime)
            minTime=(int)j.getE();
    }
    return minTime;
}

private Vector getJobsAvailableAt(int avlTime){
    Vector <Job>v=new Vector<Job>();
    for(int i=0;i<cat.size();i++){
        if(cat.elementAt(i).getE()==avlTime){
            v.addElement((Job)cat.elementAt(i));
        }
    }
    return v;
}
}

```

2 Package: MinMaxAbsoluteChain.GUI

2.1 File: InputChain.java

```

package MinMaxAbsoluteChain.GUI;
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.GridLayout;

```

```

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.JFrame;
import MinMaxAbsoluteChain.POJO.Chain;
import MinMaxAbsoluteChain.com.RunCombineChain;
import MinMaxAbsoluteChain.model.TabelModelForChain;

public class InputChain extends JFrame implements ActionListener{
    JTable table=new JTable();
    TabelModelForChain model=new TabelModelForChain();
    JButton jbtAdd=new JButton("Add New Job");
    JButton jbtSave=new JButton("Save");
    JButton jbtEdit=new JButton("Edit");
    JButton jbtCombineChain=new JButton("Combine Job");
    JTextField jtfChainString=new JTextField(15);
    JTextField jtfChainId=new JTextField(5);

    public static void main(String abc[]){
        InputChain mf=new InputChain();
        mf.setSize(400,400);
        mf.setVisible(true);
        mf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public InputChain(){
        JPanel jpCenter=new JPanel();
        jpCenter.setLayout(new FlowLayout());
        JScrollPane jspTable=new JScrollPane(this.table);
        this.table.setModel(this.model);
        jpCenter.add(jspTable);
        JPanel jpButton =new JPanel();
        jpButton.setLayout(new FlowLayout());
        jpButton.add(this.jbtAdd);
        jpButton.add(this.jbtSave);
        jpButton.add(this.jbtEdit);
        jpButton.add(this.jbtCombineChain);

        JPanel jpData=new JPanel();
        jpData.setLayout(new FlowLayout());
        jpData.add(new JLabel("Chain Id:"));

```

```

    jpData.add(this.jtfChainId);
    jpData.add(new JLabel("Chain String:"));
    jpData.add(this.jtfChainString);
    this.jtfChainId.setEditable(false);
    this.setLayout(new BorderLayout());
    JPanel jpDownHolder=new JPanel();
    jpDownHolder.setLayout(new GridLayout(2,1,5,5));
    jpDownHolder.add(jpData);
    jpDownHolder.add(jpButton);
    JPanel jpDown=new JPanel();
    jpDown.setLayout(new FlowLayout());
    jpDown.add(jpDownHolder);

    this.add(jpDown,BorderLayout.SOUTH);
    this.add(jpCenter,BorderLayout.CENTER);
    this.jbtSave.addActionListener(this);
    this.jbtAdd.addActionListener(this);
    this.jbtEdit.addActionListener(this);
    this.jbtCombineChain.addActionListener(this);
    this.setTitle("Mix-max-absolute-chain problem");

    this.table.addMouseListener(new MouseAdapter(){
    public void mouseClicked(MouseEvent arg0) {
        int i=InputChain.this.table.getSelectedRow();
        Chain c=InputChain.this.model.getChainAt(i);
        InputChain.this.jtfChainId.setText(""+c.getChain_Id());
        InputChain.this.jtfChainString.setText(c.getChain_String());
    }
    });
}

public void actionPerformed(ActionEvent ae) {
    if(ae.getSource().equals(this.jbtAdd)){
        clearBox();
    }
    if(ae.getSource().equals(this.jbtSave)){
        Chain c=new Chain();
        c.setChain_String(this.jtfChainString.getText());
        this.model.addChain(c);
        this.table.updateUI();
        clearBox();
    }

    if(ae.getSource().equals(this.jbtEdit)){
        Chain c=new Chain();
        c.setChain_String(this.jtfChainString.getText());
        c.setChain_Id(Integer.parseInt(this.jtfChainId.getText()));
        this.model.editChain(c);
        this.table.updateUI();
    }
}

```

```

        if(ae.getSource().equals(this.jbtCombineChain)){
            long l1=System.currentTimeMillis();
            RunCombineChain r=new RunCombineChain(this.model.getChain());
            RunCyclicChain(this.model.getChain(),noOfCycle);
            long l2=System.currentTimeMillis();
            JOptionPane.showMessageDialog(this,"Total Run Time :"+(l2-l1)+"
milisecond");
        }
    }

    public void clearBox(){
        this.jtfChainString.setText("");
        this.jtfChainId.setText("");
    }
}

```

3 Package: MinMaxAbsoluteChain.model

3.1 File: ChainDataModel.java

```

package MinMaxAbsoluteChain.model;
import java.util.Vector;
import MinMaxAbsoluteChain.POJO.Chain;
import MinMaxAbsoluteChain.POJO.Job;

public class ChainDataModel {
    static private Vector <Chain>cat=new Vector<Chain>();
    public void addChain(Chain c){
        this.cat.add(c);
    }

    public Vector getChain(){
        return cat;
    }

    public Chain getChain(int chainId){
        try{
            Chain c=cat.elementAt(chainId);
            return c;
        }catch(Exception e){
            return null;
        }
    }

    public void upDateChain(Chain c){
        for(int i=0;i<cat.size();i++){
            if(cat.elementAt(i).equals(c)){

```

```

        cat.elementAt(i).setChain_String(c.getChain_String());
    }
}
}
}

```

3.2 File: TabelModelForChain.java

```

package MinMaxAbsoluteChain.model;
import java.util.Vector;
import javax.swing.table.AbstractTableModel;
import MinMaxAbsoluteChain.POJO.Chain;

public class TabelModelForChain extends AbstractTableModel{
    Vector <Chain>cat=new Vector<Chain>();
    ChainDataModel dataModel=new ChainDataModel();

    public Vector getChain(){
        return this.cat;
    }

    public TabelModelForChain(){
        upDate();
    }

    @Override
    public String getColumnName(int colNo) {
        switch(colNo){
            case 0:
                return "Chain Id";
            case 1:
                return "Chain String";
        }
        return null;
    }

    public int getColumnCount() {
        return 2;
    }

    public int getRowCount() {
        return cat.size();
    }

    public Object getValueAt(int rowIndex, int columnIndex) {
        Chain c= cat.elementAt(rowIndex);
        switch(columnIndex){
            case 0:
                return c.getChain_Id();
            case 1:
                return c.getChain_String();
        }
    }
}

```



```

        }
        return null;
    }

    public void addChain(Chain c){
        c.setChain_Id(dataModel.getChain().size());
        dataModel.addChain(c);
        upDate();
    }

    public void editChain(Chain c){
        dataModel.upDateChain(c);
        upDate();
    }

    public Chain getChainAt(int index){
        return cat.elementAt(index);
    }
    @SuppressWarnings("unchecked")

    public void upDate(){
        this.cat=new ChainDataModel().getChain();
    }
}

```

4 Package: MinMaxAbsoluteChain.POJO

4.1 File: Chain.java

```

package MinMaxAbsoluteChain.POJO;
import java.util.Vector;
public class Chain {
    private int Chain_Id;
    private String Chain_String;

    public int getChain_Id() {
        return Chain_Id;
    }
    public void setChain_Id(int chain_Id) {
        Chain_Id = chain_Id;
    }
    public String getChain_String() {
        return Chain_String;
    }
    public void setChain_String(String chain_String) {
        Chain_String = chain_String;
    }
    @Override
    public boolean equals(Object c) {
        try{

```

```

        Chain temp=(Chain)c;
        return this.Chain_Id==temp.Chain_Id?true:false;
    }catch(Exception e){
        return false;
    }
}
}

```

4.2 File: Job.java

```

package MinMaxAbsoluteChain.POJO;
public class Job {

    private int chainId;
    private int jobPosition;
    private double e;
    private double l;
    private char jobChar;
    private boolean isScheduled;

    public Job(){
        this.isScheduled=false;
    }
    public boolean isScheduled() {
        return isScheduled;
    }
    public void setScheduled(boolean isScheduled) {
        this.isScheduled = isScheduled;
    }
    public int getChainId() {
        return chainId;
    }
    public void setChainId(int chainId) {
        this.chainId = chainId;
    }
    public double getE() {
        return e;
    }
    public void setE(double e) {
        this.e = e;
    }

    public int getJobPosition() {
        return jobPosition;
    }
    public void setJobPosition(int jobPosition) {
        this.jobPosition = jobPosition;
    }
    public double getL() {

```

```
        return l;
    }
    public void setL(double l) {
        this.l = l;
    }
    public char getJobChar() {
        return jobChar;
    }
    public void setJobChar(char jobChar) {
        this.jobChar = jobChar;
    }
}
```