REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOHAMED BOUDIAF - M'SILA

FACULTE DES MATHEMATIQUES ET DE L'INFORMATIQUE

DEPARTEMENT D'INFORMATIQUE

N° :……..

DOMAINE : MATHEMATIQUES ET INFORMATIQUE

FILIERE : INFORMATIQUE

OPTION : OUTILS ET METHODES POUR L'INFORMATIQUE DECISIONNELLE

Mémoire présenté pour l'obtention
Du diplôme de Master Académique

Par: KAHLOULA Tarek Yassine

Intitulé

Constructive Algorthims for Manufacturing Scheduling Problems

Soutenu devant le jury composé de :

Mr. BOUGHERARA .S	Université de M'sila	Président
Mr. BELOUADAH .H	Université de M'sila	Rapporteur
Mr. GASMI .A	Université de M'sila	Examinateur

Année universitaire : 2016 /2017

Table of contents

CHAPTER 3 : Constructive Algorithms for Manufacturing Scheduling Problems

Figures :

Tables :

INTRODUCTION

INTRODUCTION

The manufacturing scheduling is an important element in planning and management for every enterprise. Therefore, it has to be taken seriously.

The scheduling problem consists of organizing a set of activities and tasks by determining their dates of launch and dates of ending and their resources assignments, to result chat it is called a schedule.

A task is an activity that relies on certain resources and has certain effects on the project. Tasks may have different characteristics in the same project.

In scheduling, The most algorithms are used to optimize the time criteria, as an example, a good schedule takes less time..etc.

The objective behind this paper is to study the scheduling problems in their practical forms, which means the more complex problems with time-constraints, to build constructive algorithms in order to minimize certain criteria.

The first chapter will study the manufacturing scheduling problem in general, the definitions of the elements and the different types of manufacturing environments.

The second chapter will be dedicated for the most common scheduling problems and the classic algorithms used to solve them.

Throughout the third chapter, we will study the manufacturing scheduling problem in a more complex way, and cite some constructive algorithms that solve those problems. Finally as a result of the study we will develop a program based on the algorithms in order to use it to determine solution for different types of data.

CHAPTER I: MANUFACTURING SCHEDULING PROBLEMS

CHAPTER 1

The Manufacturing Scheduling Problems

1. Introduction:

In this chapter we will discuss the specific problem of manufacturing process management. We will give a definition as well as an introductory example.

Using figures, we will give the characterization of this problem as well as the involvement of each character, and at the end of this first chapter we will cite the different types of the manufacturing process scheduling problem.

2. Manufacturing Scheduling Problem:

2.1. Definition:

A scheduling problem is handling and arranging and controlling workloads of different tasks or procedures with different characteristics using the help of resources[8], by respecting the limitation of the work environment in order to get an optimum or a good schedule. A good schedule is a program of which order the tasks should be and which time each task should process, it optimizes the productivity (cost-wise and time-wise) by assigning each task to the right time and the right resources.

The scheduling problem has various genres domains: Project management and planning, production and manufacturing scheduling, computing process scheduling…

A manufacturing schedule dictates the order in which the jobs go, from rough material to final product. Optimal manufacturing means best quality and minimum cost, while a minimum cost involves essentially a minimum manufacturing time. This specific problem will be explained in details in a more advanced phase in this chapter.

Example: two construction workers ought to help building a house composed of three rooms. The first is a constructor and the second is a painter. A room can’t be constructed and painted at the same time nor can painting be done before construction, so the constructor always does the job first on every room. Each operation (whether painting or constructing) takes a certain period of time. This particular problem can be translated into a scheduling problem, where a schedule has to be made for an assignment of resources/machines (workers) to tasks (the three rooms) to operate (painting and constructing). The schedule must respect all the technical and temporal rules of this situation.

2.2. Manufacturing Scheduling Problem Characterization:

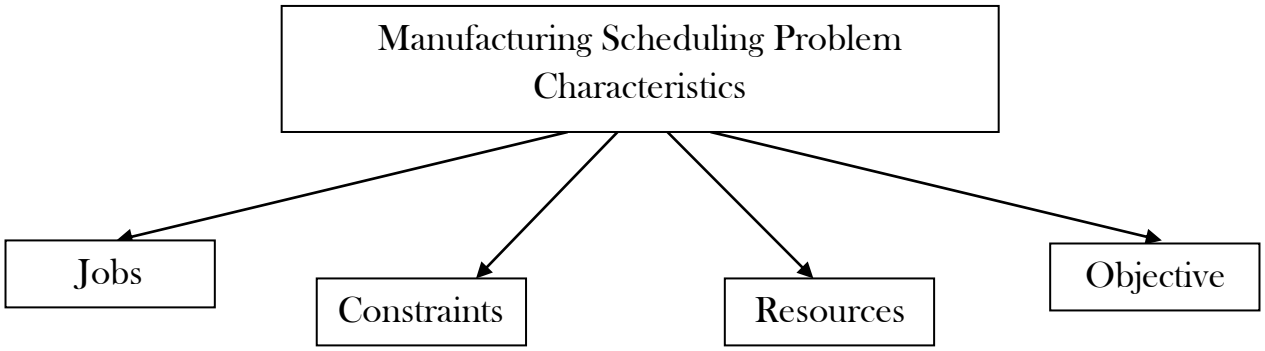


Fig 1.1: Characteristics of MSP

2.3. Resources:

Resources are either renewable or consumable material, required by certain jobs to execute.

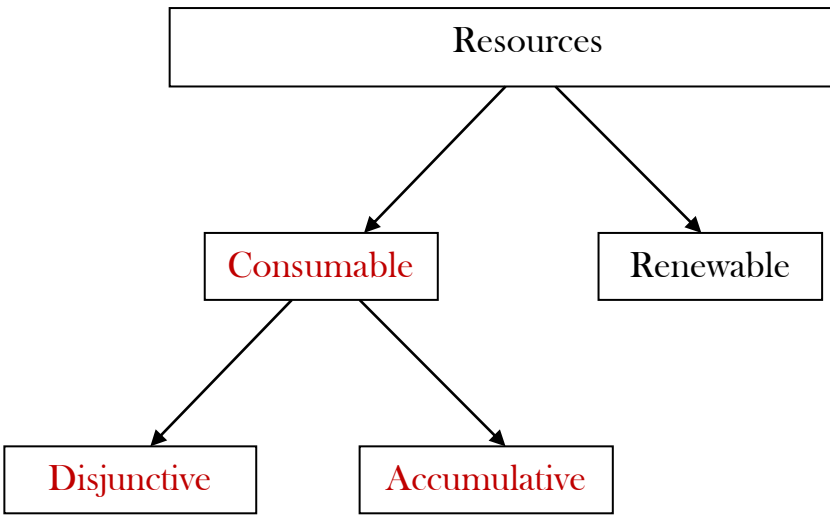


Fig 1.2: Types of resources

In some cases, resources cause resource constraints, in cause of shortage (accumulative constraints) or technical limitations (disjunctive constraints).

2.4. Machines:

In the case of manufacturing scheduling problem, we deal with workshops that have machines as renewable resources[5]. Machines are set to be a support for a job to be executed. A machine can only execute one job at a time.

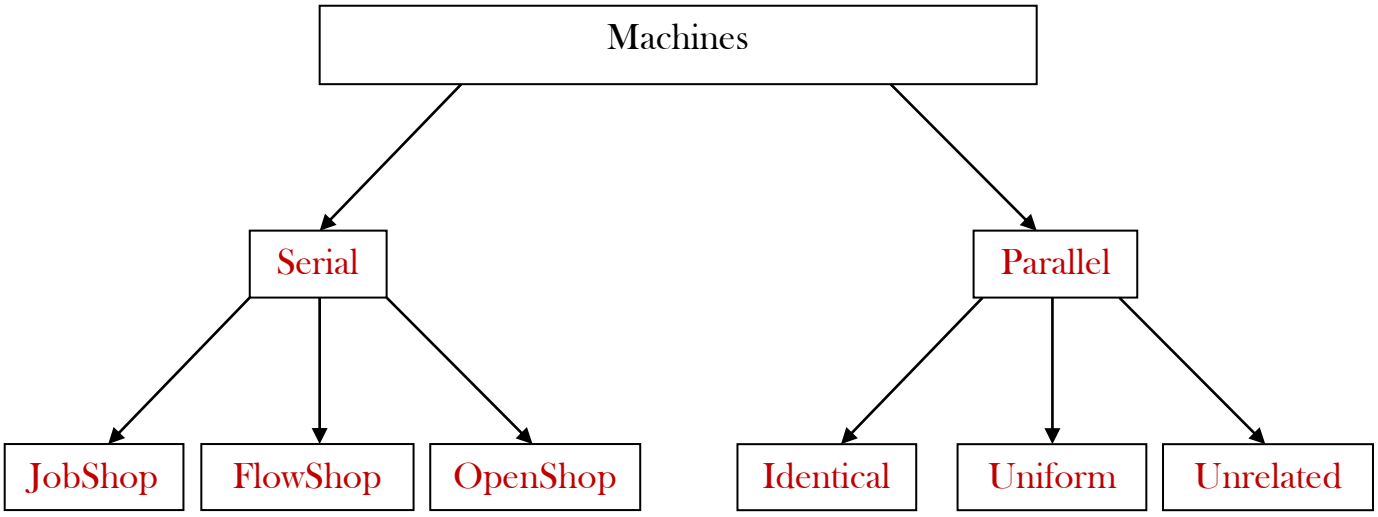


Fig 1.3: Types of machines

In manufacturing management science, machines can be either parallel or serial. In parallel cases, machines are set to have the same functionality which makes them dedicated. If dedicated machines have the same execution speed they're called identical, in contrary, if dedicated parallel machines have different speeds they're called uniform or unrelated, a uniform machine's speed does not change for each task, while a unrelated machine has a different speed for each task. But, in most cases, machines are in series, which means that every N job can be executed on a specific order of machines. In the OpenShop environment, jobs have to operate on all M machines (the number of operations of each job equal the number of machines). While the OpenShop environment is open, where each job is free from any precedence constraints (Constraints section down below), the FlowShop is the same as the OpenShop but in addition, it's limited with precedence constraints, which is the same for all machines (If J_1 is operates before J_2 on M_1, J_1 has to operate first on every other machine in the shop). JobShop protocol has a different concept, precedence affects the operations of the same job, but the order of operating on machines is different from a job to another.

2.5. Jobs:

Jobs (J_j ; j=1::n) are a set of procedures to be done and executed, each on a certain timing, with limited durations. Some jobs require certain resources. In our case, resources are mostly machines, so the usage of a machine M_i by a job J_j is called an $Operation_{ji}$.

Jobs have various parameters:

- p_{ij}: The processing duration of Jj on Mi
- r_j: The release date, or the ready date. It's the timing where a job becomes available and ready to execute.
- d_j: T he due date, it's proffered for a job j to end before this date, otherwise it becomes late.
- d'_j: A job completion date should not pass deadline set to it.
- w_j: Weight are assigned to a tasks by managers for technical reasons of priority.
- C_j: Completion time, a job is completed when it finishes up all its operations.
- L_j: Cj – dj. Lateness. If the job is late, it gives a positive value, if not, it gives a negative one.
- T_j: max {Lj , 0} Tardiness. If a job is late, this gives a positive value, if it's not, it gives a zero.
- U_j: Tardiness indicator: Uj = 0 if (Lj>=0), Uj=1 if (Lj<0). This indicator can help compute the number of late jobs simply by calculating the sum of Uj ($\sum Uj$, j=1::n)
- F_j: Flow Time: Amount of time job j spends in the system. Fj=Cj - rj, where Cj is the completion time of the jth job, and rj is the ready time of the jth job.
- W_j: Waiting Time: Length of time between the ready time of a job and the beginning of processing of a job[9].

2.6. Constraints:

Constraints are restriction and limitations that must be respected in order to have an admissible solution. We note two major categories:

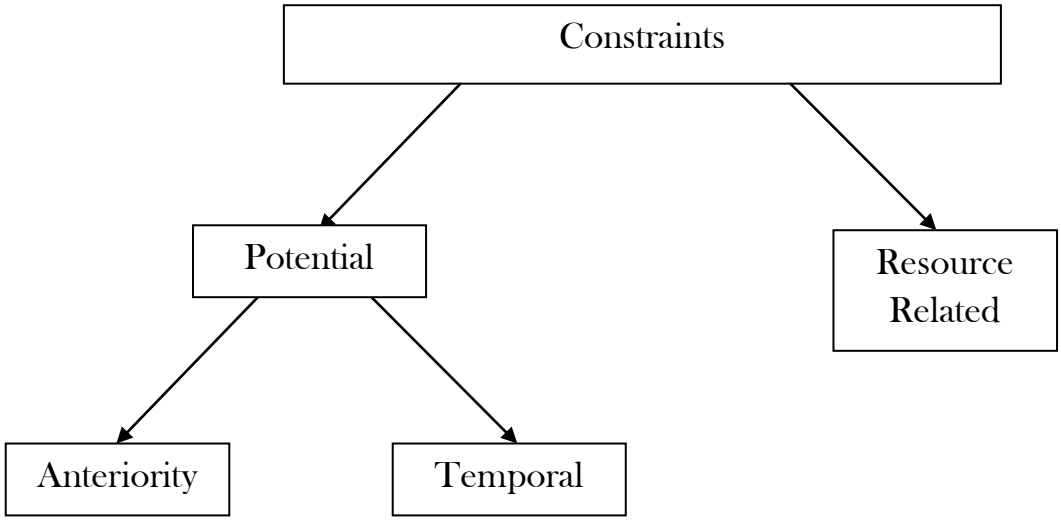


Fig 1.4: Types of constraints

2.6.1. Potential constraints: It has two sorts, anteriority, and temporal constraints. The first is mainly related to technological itinerary and consistency of the project. (Example: Cars can't be painted before a fine white layer) . Temporal constraints limit jobs with availability dates or due dates or even deadlines [Check 2.5 section].

2.6.2. Resource related constraints: Two classifications appear. Consumable resources can cause quantity shortage, or expiring..etc. Renewable resources can cause two types of constraints: accumulative and disjunctive constraints.

Accumulative constraints limit the number of tasks that can be operated at the same time due to the limited number of resources available at the time. For example, in a workshop, there are three machines available, but on the other hand we have four jobs that can be executed at the same time, only three of them can operate on three machines simultaneously.

Disjunctive constraints are caused by technical or security reasons, each two jobs can't operate at the same time in the same environment simultaneously.

2.7. Criteria:

A solution's quality can be rated by how well they've optimized the set of criteria of evaluation related to a problem. Certain criteria are time-related, some are related to resources or to cost\revenue.

As an example, a good solution is a solution that minimizes the quantity of resources used to realize a certain project.

2.8. Objective functions in scheduling problems:

Also known as economic functions are mathematical functions that are desired to optimize certain criteria. A problem can have one or multiple objective functions.

In manufacturing scheduling problems and in most cases, the most common criteria to optimize involves the total duration of the schedule (C_{max} = max{C_j} The makespan, also called the "schedule length"). It's only rational that you *Minimize* C_{max}.

There's also the completion time involvement, as we sometimes need to *Minimize* $\sum C_j$ (j=1::n), or *Minimize* $\frac{1}{n}\sum C_j$ (j=1::n). In some cases, when priorities are involved, *Minimize* $\sum w_j C_j$ (j=1::n).

In many problems, tardiness must be minimized through the objective function *Minimize* T_{max} = *max*{T_j (j=1::n)}. Lateness also can be seen in some problems as *Minimize* L_{max} = *max*{L_j (j=1::n)}. Sometimes it's also reasonable to *Minimize* $\sum U_j$ (j=1::n) which is the number of late jobs in the schedule, maybe even *Minimize* $\sum w_j U_j$ (j=1::n) if the managers ought to optimize the weight of jobs that are late.

2.9. The different types of manufacturing scheduling problems:

Based on the number of machines and the characteristics of workshop environments:

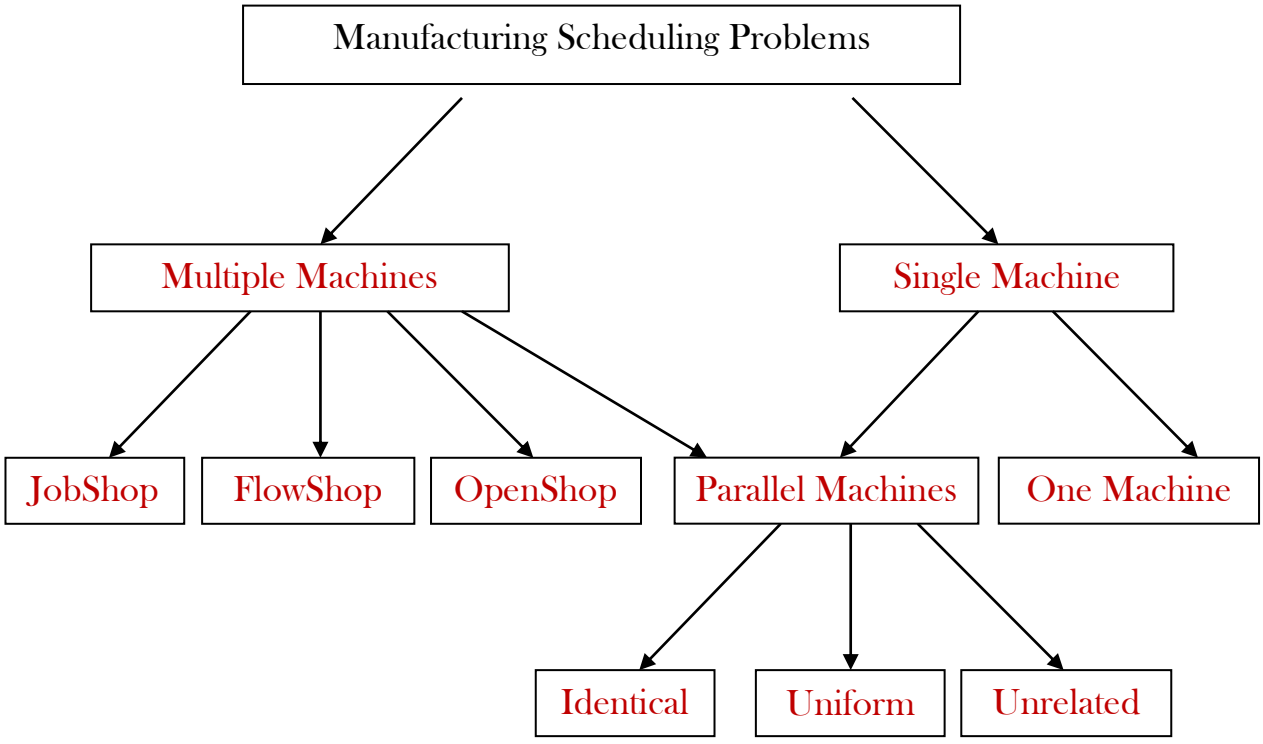


Fig 1.5: Types of MSP

2.8.1. Single machine problems:

Having one single renewable resource in a workshop can only lead to one-machine scheduling, which consists on assigning a group of tasks to a single machine. The jobs are put in order so that a set of measures and criteria can be optimized. This problem is one of the most common in the combinatorial optimization. There are many criteria to optimize within this problem, *Average Flowtime, Lateness, Tardiness,…etc.*

2.2.2. OpenShop:

OpenShop scheduling is a difficult combinatorial optimization problem[8]. In OpenShop environments, jobs are to be executed in any order and are free from precedence constraints.

2.2.3. FlowShop:

FlowShop scheduling is also a difficult combinatorial optimization problem[5]. This type of workshops requires that all jobs must operate on all machines by the same order.

2.2.4. JobShop:

JobShop scheduling is an NP-difficult combinatorial optimization problem.). In JobShop problems, jobs can have different order in which machines they operate on, and the number of operations of each job does not necessarily equal the number of machines M.

3. Variations & Formulation:

The scheduling problems in manufacturing can vary in terms of the nature of the workshop environment, generally restrictions and characteristics of jobs and also the number of resources/machines recruited. For example, the OpenShop environment doesn't have the amount of restrictions as the JobShop or the FlowShop do.

In this case of variations, Graham (1979) has proposed a convenient notation consisting of three fields α, β and γ (α|β |γ) to describe a specific problem of scheduling. Each field may contain different notes separated with commas. The first field α describes the environment of the workshop, which includes the number of machines and the characteristics of the machines environment (FlowShop, JobShop, OpenShop.. etc). The second field β contains the workshop regulations (constraints of different types) as well as the jobs characteristics (preemptions '**PMTN**', availability $\mathbf{r_i}$ of each job..etc). The last field γ points at the criteria or the objective functions to optimize.[1]

Example: $2F||C_{max}$ describes the specific scheduling problem of the FlosShop environment of two machines, with default constraints, and a completion time criteria. $1||\sum C_i$ refers to the classic one machine scheduling problem.

For the first field, machine environments have two classes, single-stage problems and multi-stage problems [Section: Types of problems], each environment can be referred to by a specific symbol, followed or preceded by the number of resources/machines recruited in that environment.

Single-Stage Class	Multi-Stage Class
1 : One single machine	O: OpenShop
P : Identical parallel	F: FlowShop
Q : Uniform	J: JobShop
R : Unrelated identical parallel	H: Hybrid

Table 1.1: Descriptions of environments

The second field has more variations. General constraints that are applied on all jobs are the ones that can fit in the β field. However, in extended problems, where each job has its own characteristic, descriptions are to be followed with another section that has different forms of information (data tables, precedence graphs or a joint piece of texts...etc).

Examples:

$1||\sum C_i$: One machine problem with a mean completion time criteria.

$F3||C_{max}$: Three machines, in a flowshop environment, with makespan criteria.

$F2|ri|C_{max}$: Two machines, in a flowshop environment with availability dates for each job in order to optimize the makespan.

4. Visualizations & Modeling:

In Scheduling in general, work programs (schedules) are displayed in a form of a bar diagram called the Gantt Chart (Henry Gantt, 1910).

The diagram illustrates the dates of the launch dates and finish dates of each activity. It generally has two dimensions. Horizontal vector indicates the time passage, while the vertical one may indicate either the different activities or resources that may be joined to the jobs.

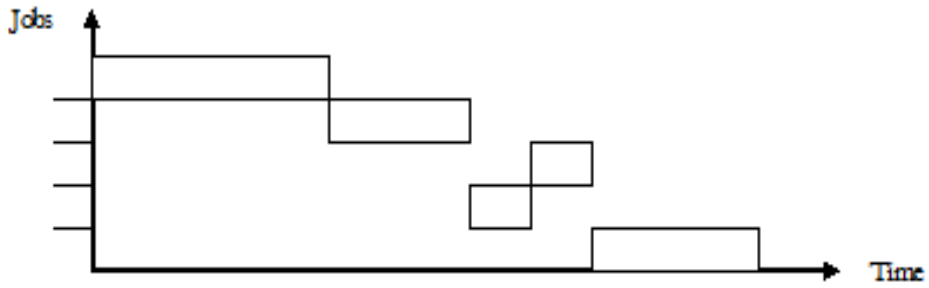


Fig 1.6: Gantt chart

4.1. Bars:

Usually they indicate the starting and the finishing dates of an activity on a certain resource. It may have many sub components that indicate various characteristics that a job may embrace, such as processing durations.

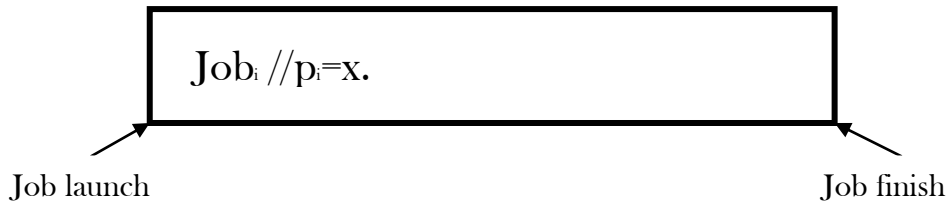


Fig 1.7: Gantt chart bars

4.3. Horizontal Vector:

The horizontal vector simply indicates for the time passage of the activities, read from left to right. The lineup of the left margin of an activity bar indicates the launch date of it, while the lineup of the right margin with the horizontal vector indicates the finish date.

4.2. Vertical vector:

This vertical vector can take three different forms depending on the nature of the problem; Activities only, resources only, and a hybrid form that helps indicate both resources and activities.

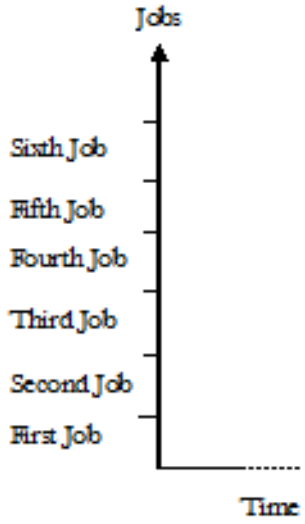


Fig 1.8: Gantt chart activities vector

The form in FIG is generally used when facing the one resource/machine problem, where all jobs process on the same machine 1|β |γ.

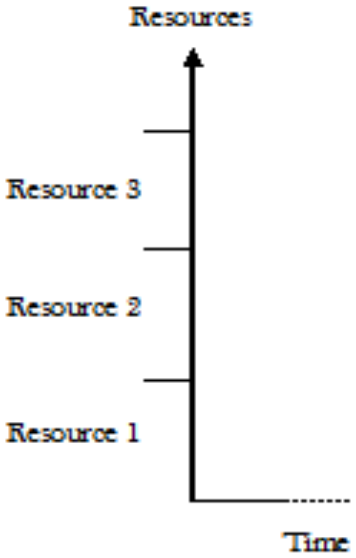


Fig 1.9: Gantt chart resources vector

The form of vertical vector in FIG above indicates for the machines only, while jobs are let to be indicated by the bars. Note that when an activity bar is in the resource line at a certain timing, means that the resource/machine is assigned to the activity at that time. This specific form is used for problems of multiple resources/machines n|β |γ (eg; F2| $|C_{max}$).

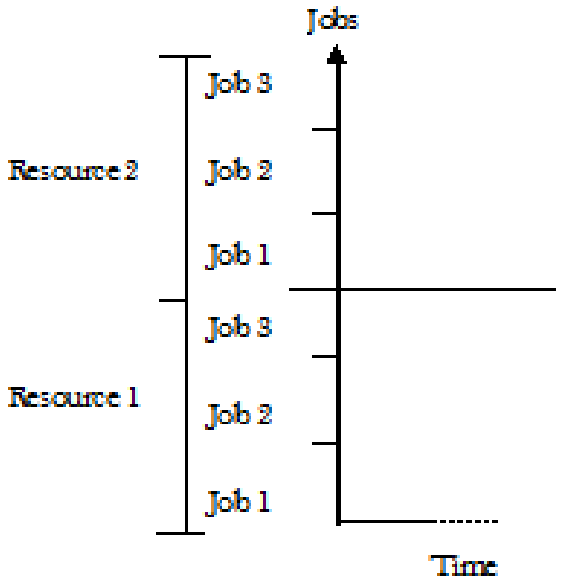


Fig 1.10: Gantt chart resources\activities vector

The form above is used for the same types of problems as the form in FIG used for. However, for other purposes of clarity, it's preferred to use this model, to separate the level of each activity bar from another.

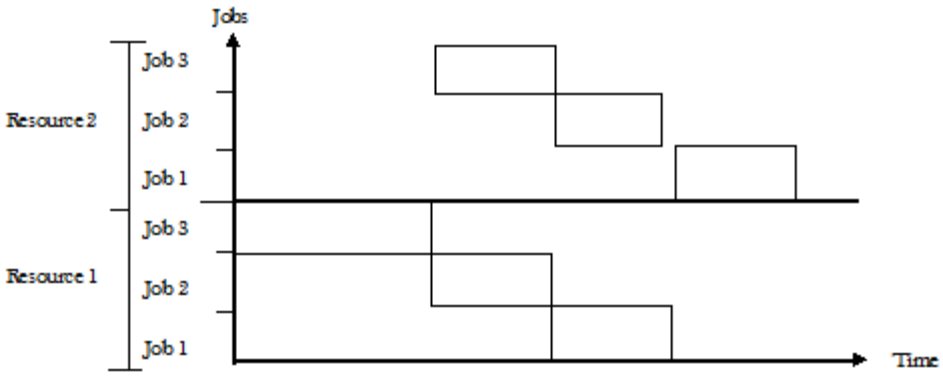


Fig 1.11: Gantt chart example

The figure above models a FlowShop problem, of two machines assigned to three jobs : F2| $|C_{max}$.

4.3. Other components:

Other minor components may mark their appearance in a Gantt chart, for example: colors, anteriority arrows (used to indicate anteriority relations between activities), extra notations…etc.

5. Conclusion

This first chapter contained a clear explanation of how scheduling problems work in general, also cited the different manufacturing planning general problems and how schedules can be modeled in particular cases.

CHAPTER II: SINGLE & MULTI STAGE PROBLEMS & ALGORITHMS

CHAPTER 2

Single & Multi Stage Problems & Algorithms

1. Introduction:

Throughout this chapter, we shall spotlight the most common problems of scheduling in manufacturing, in both single and multi stage cases, followed by the most famous algorithms to solve these problems.

Beginning with single stage problems, we will discuss Mean Flow default problem, as well as the Maximum Lateness problem. Later in this chapter, we will discuss the multi stage problems on both FlowShop and JobShop environments.

2. Single Stage Problems

2.1. Mean Flow Time Single Machine Default Problem $1||\sum C_i$

We assume that all jobs are ready to process at the beginning default time, all jobs J_i (i=1::n) should process on one single machine within a processing duration of p_i for each job J_j. Generally, the mean flowtime is the criteria to be optimized at this case, which is also the mean of completion time of all jobs.

Proof:

Note that:

- *All jobs are ready at launch time, $r_i=0$;*
- *W_i is the waiting time of J_i before it starts at t_i , $W_i=t_i - r_i$.*
- *In this case, $W_i=t_i$ because all jobs have a default ready time value.*

$$\bar{F} = \frac{1}{n}\sum F_i$$

$$= \frac{1}{n}\sum (W_i + p_i)$$

$$= \frac{1}{n}\sum (t_i - r_i + p_i)$$

$$= \frac{1}{n}\sum (t_i + p_i)$$

$$= \frac{1}{n}\sum C_i$$

$$\bar{F} = \bar{C}$$

SPT rule (Shortest Processing Time) (Smith, 1959) minimizes the mean flow time, the mean launch time, and the mean completion time to an optimum value as long as all $r_i = 0$. By ordering jobs according to their processing durations, from minimum to maximum, so a shorter job always processes earlier.

Proof:

Note that: $\sum C_i = \sum(t_i + p_i)$

$$=\sum t_i + \sum p_i$$

And $\sum$pi is constant for whatever order jobs take, which means that minimizing $\sum C_i$ optimizes $\sum t_i$, so an optimum solution for $\mathbf{1}||\sum t_i$ is optimum solution for $\mathbf{1}||\sum \mathbf{C}_i$.

By having the shorter job first, we guarantee that the following job will have an earlier launch time t_i. Applying this method (SPT) for all jobs, we reach a mean start time of an optimum value, which also means an optimal mean completion time.

Example:

Given the following data for a $1||\sum \mathbf{C}_i$

J_i	1	2	3	4	5	6	7	8
p_i	6	4	8	9	4	5	3	7

By following the SPT rule, we get the optimal order S (J_7,J_5,J_2,J_6,J_1,J_8,J_3,J_4)

J_i	7	5	2	6	1	8	3	4
p_i	3	4	4	5	6	7	8	9
C_i	3	7	11	16	22	29	37	46

This schedule S gives an optimum sum of completion times $\sum \mathbf{C}_i$*=171 for this data.*

The SPT algorithm:

0- Initialize Z=0

1- Calculate min p_i of all J_i jobs; i=1::n.

2- Put the Ji job in the first available on the Sequence.

3- Z=Z+pi.

4- Remove J_i from data.

Loop back to 0 until DATA is empty

5- Print Sequence and Z

Complexity:

Data is often taken to be in an array, which allows random access, rather than a list, which only allows sequential access, though often algorithms can be applied with suitable modification to either type of data.

Assume that n is the number of tasks to be sorted. In best and average cases, operations other than sorting are hardly needed. However, in worst cases, comparisons, swaps, shifts, and other needed operations can be processed.

Algorithm	Best	Average	Worst	Method
Quicksort	n*log(n)	n*log(n)	n^2	Partitioning
Mergesort	n*log(n)	n*log(n)	n*log(n)	Merging
Insertionsort	n	n^2	n^2	Inserting

Table 2.1: Complexities of sorting methods[7]

In this case, SPT rule does not need any other operations, therefore, complexity is in most case is *O(nlogn)*.

2.2. Mean Weighted Flow Time Single Machine Problem $1||\sum w_iC_i$

When dealing with the single machine environment, we sometimes are obliged to deal with priorities between jobs, called weights w_i ; i=1::n. The objective in this case is to minimize the mean weighted flowtime and determine an optimal solution for $\mathbf{1||\sum w_iC_i}$.

Theorem:

SWPT rule (Shortest Weighted Processing Time) minimizes the mean weighted flow time, by sorting jobs according to weighted processing time ratio p_i/w_i, from minimum to maximum to form an optimal sequence S.

Proof:

Assume that: Two adjacent jobs J_i & J_j of pi & pj as processing time and w_i & w_j as weights respectively. S is the sequence that schedules J_i before J_i, while S' schedules J_j before J_i

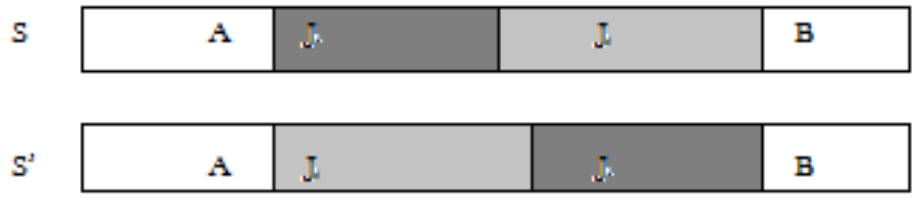


Fig 2.1: SWPT rule swap

$Z = A + w_i (T+p_i) + w_j (T+p_i+p_j) + B$

$Z' = A + w_j (T+p_j) + w_i (T+p_j+pi) + B$

Z & Z' are the values of the objective of $\sum wiC_i$. *for both S and S' respectively.*

$Z - Z' = wi (T+p_i) + w_j (T+p_i+p_j) - w_j (T+p_j) - wi (T+p_j+p_i)$

$= w_i * p_i + w_j (p_i+p_j) - w_j * p_j - wi (p_j+p_i)$

$= w_j * p_i - w_i * p_j$

Let's suppose that : $pi/wi < p_j/w_j$

$Z - Z' = w_j * p_i - w_i * p_j < 0$

$\leftrightarrow \; w_j * p_i < w_i * p_j$

$\leftrightarrow p_i/w_i < p_j/w_j$

$\leftrightarrow$ *Z<Z' and S is better than S', and SWPT is optimal for* $\mathbf{1||\sum w_i C_i}$.

Example:

Given the following data for a $1||\sum \mathbf{w_i C}_i$

J_i	1	2	3	4	5	6	7	8
p_i	6	4	8	9	4	5	3	7
w_i	2	6	4	8	1	0	3	4

In order to get the SWPT order S, we calculate the ratio p_i/w_i

J_i	1	2	3	4	5	6	7	8
p_i	6	4	8	9	4	5	3	7
w_i	2	6	4	8	1	0	3	4
p_i/w_i	3	2/3	2	9/8	4	∞	1	7/4 ←

By ordering the jobs according to the SWPT rule, we get the order S (J_2,J_7,J_4,J_8,J_3,J_1,J_5,J_6)

J_i	2	7	4	8	3	1	5	6
p_i	4	3	9	7	8	6	4	5
w_i	6	3	8	4	4	2	1	0
p_i/w_i	2/3	1	9/8	7/4	2	3	4	∞
C_i	4	7	16	23	31	37	41	46
w_iC_i	24	21	128	92	124	74	41	0 ←

The SWPT rule minimizes the sum of weighted completion times to a value of $\sum \mathbf{w_i C_i}$*=504 for this data.*

The SWPT algorithm:

0- Initialize Z=0.

1- Calculate min p_i/w_i of all J_i jobs; i=1::n.

2- Put the J_i job in the first available on the Sequence.

3- Update Z

4- Remove J_i from data.

Loop back to 0 until DATA is empty

5- Print Sequence and Z

Complexity:

As SWPT rule relies on sorting an array with no major operations needed, its complexity is *O(nlogn).*

2.3. Maximum Lateness Single Machine Problem $1|d_i|Lmax$

Let's consider the following scenario; A single machine recruited to execute all jobs J_j, all are ready to process at launch date, it's preferred that jobs finish processing not long after their due dates d_i, which must lead to minimizing the maximum lateness $L_{max} = \max\{Li = Ci - di ; i=1::n\}$

Theorem:

L_{max} can be minimized by scheduling the jobs that have earlier due dates first. Jackson's EDD rule (Earlier Due Date 1954) gives and optimal sequence to the problem of $\mathbf{1|d_i|L_{max}}$.

Proof:

Suppose a schedule S, which violates EDD, is optimal. In this schedule there must be at least two adjacent jobs i and k such that $d_i > d_k$ and job i precedes job k.

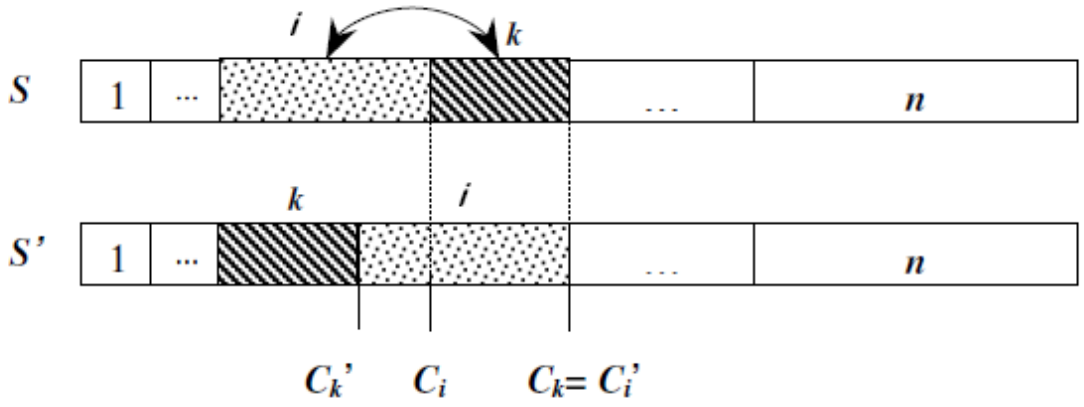


Fig 2.2: EDD rule swap

Swapping jobs i and k leads to another schedule S' such that

$L'_k = C'_k - d_k < C_k - d_k < L_k$

$L'_i = C'_i - d_i = C_i - d_i < L_i$

$L'_j = C'_j - d_j = C_j - d_j = L_j$; for $j \neq i,k$

which implies that

$L_{max}(S') = \max\{L'_k, L'_i, L'_j\} \leq L_{max}(S) = \max\{L_k, L_i, L_j\}$

S' doesn't violate EDD rule => EDD rule brings an optimal solution for the $1|d_i|L_{max}$ problem.

Note that Jackson's rule does not only gives an optimal solution for **$1|d_i|L_{max}$**, but also for **$1|d_i|T_{max}$**, where $T_i = \max\{L_i;0\}$, and $T_{max} = \max\{T_i ; i=1::n\}$. So every optimum schedule for **$1|d_i|L_{max}$** is an optimum solution for **$1|d_i|T_{max}$**.

Example:

Given the following data for a **$1||L_{max}$**

J_i	1	2	3	4	5	6	7	8
p_i	6	4	8	9	4	5	3	7
d_i	16	27	20	16	11	28	14	8

By ordering jobs according to d_i following the EDD rule, we get an order S $(J_8,J_5,J_7,J_1,J_4,J_3,J_2,J_6)$

J_i	8	5	7	1	4	3	2	6
p_i	7	4	3	6	9	8	4	5
d_i	8	11	14	16	16	20	27	28

This order allows us to calculate the lateness of each job

J_i	8	5	7	1	4	3	2	6
p_i	7	4	3	6	9	8	4	5
d_i	8	11	14	16	16	20	27	28
C_i	7	11	14	20	29	37	41	46
L_i	-1	0	0	4	13	17	14	18

After calculating the L_i of each job, we find that *$L_{max}=18$ which represents the optimal solution for the* **$1||L_{max}$** problem on this schedule data.

The EDD algorithm:

0- Initialize Z=0

1- Calculate min d_i of all J_i jobs; i=1::n.

2- Put J_i job in the first available position on the Sequence.

3- Remove J_i from data.

Loop back to 0 until DATA is empty

4- Print Sequence and Z

Complexity:

EDD rule is a simple array sorting, therefore, its complexity is *O(nlogn).*

2.4. Tardy Jobs Single Machine Problem $1|d_i|\sum U_i$

In a similar case to $\mathbf{1|d_i|L_{max}}$, we have the same environment, but the purpose is different. Due to the technical differences, we sometimes find that minimizing the maximum lateness doesn't make much sense, since all late tasks are being penalized equally, which makes us obliged to look into minimizing the number of tardy jobs n_t; facing the $\mathbf{1|d_i|\sum U_i}$ problem.

$n_t = \sum U_i$; i=1::n

Where $U_i = 0$ if ($L_i >= 0$),

$U_i = 1$ if ($L_i < 0$)

Moore & Hodgson [1968] proposed an algorithm based on the EDD rule, in order to find the optimal solution for this problem.

Example:

The following table represents a data for a $1||\sum U_i$ problem. S is the array that defines the order of jobs.

J_i	1	2	3	4	5	6	7	8
p_i	6	4	8	9	4	5	3	7
d_i	16	27	20	16	11	28	14	8

First we order the jobs and calculate L_i of each one until we find the first late one. After obtaining the first late job, we replace it in the last position on the S array, and then we remove it from the data.

J_i	8	5	7	1	4	3	2	6
p_i	7	4	3	6	9	8	4	5
d_i	8	11	14	16	16	20	27	28
C_i	7	11	14	20				
L_i	-1	0	0	4				

We reorder the data until we find the new first late job, and redo the same process by positioning the job in the last position in the array S.

J_i	8	5	7	4	3	2	6
p_i	7	4	3	9	8	4	5
d_i	8	11	14	16	20	27	28
C_i	7	11	14	23			
L_i	-1	0	0	7			

S=(…,J_1,J_4)

J_i	8	5	7	3	2	6
p_i	7	4	3	8	4	5
d_i	8	11	14	20	27	28
C_i	7	11	14	22		
L_i	-1	0	0	2		

S=(…,J_1,J_4,J_3)

J_i	8	5	7	2	6
p_i	7	4	3	4	5
d_i	8	11	14	27	28
C_i	7	11	14	18	23
L_i	-1	0	0	-9	-5

After repeating the same process multiple times we can obtain the ordered data containing of only early jobs J_8, J_5, J_7, J_2 and J_6, t get the final schedule we simply place this sequence $(J_8, J_5, J_7, J_2, J_6)$ at the first position of the array S.

The final schedule $S=(J_8, J_5, J_7, J_2, J_6, J_1, J_4, J_3)$ contains only 3 late jobs, which is the minimal number of late jobs for this data when it comes to minimizing the number of late jobs (the $1||\sum U_i$ problem).

Moore's algorithm:

0- Initialize Z=0

1- Reorder jobs according to EDD rule

2- If no job is tardy, then PRINT Z=0 and BREAK.

If not

3- (Let the first late job be J_i), among the first i jobs, move the largest job to the Sequence

4- Update all completion times and then Loop back to 2 until DATA is empty

5- Print Sequene and Z

Complexity:

Moore's algorithm is an extension to EDD rule, looped *l* times. Due to the limited number of loops, *l* is ignored, hence Moore's algorithm gives solution for $\mathbf{1|d_i|\sum U_i}$ in *O(nlogn) time.*

3. Multi Stage Problems

3.1. The F,2||C_{max} problem:

Generally, in a flow shop problem, there are m machines that should process n jobs. All jobs have the same processing order through the all m machines. In this case; $\mathbf{F,2||C_{max}}$, If there are m=2 machines, so that the makespan C_{max} is minimized ; C_{max} = max {C_i ; i=1::n}.

The main concern of this problem is to determine the optimal order S. This is called permutation schedule.

The problem can be solved by Johnson's algorithm that is based on the SPT rule, starting with the shortest processing time jobs on the first machine, for then the processing on the second machine should start as soon as possible, so it's only reasonable to finish the processing in the second machine with the shortest processing time job. An optimum solution can be built following Johnson's rule[9].

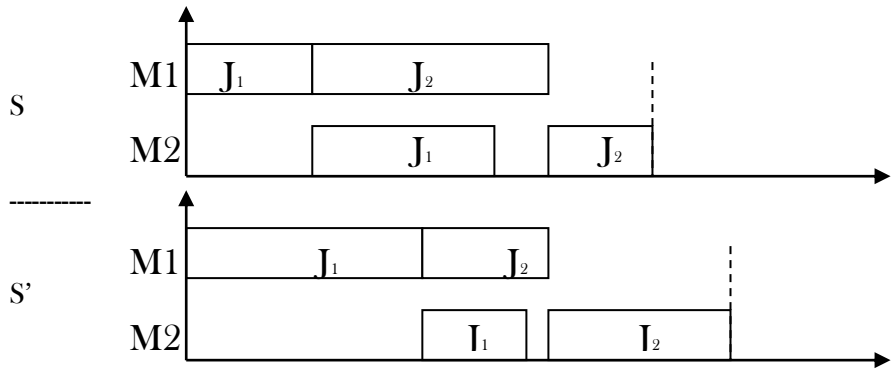


Fig 2.3: Gantt chart for FSP

Whilst schedule S follows SPT rule, schedule S' disobeys it.

$C_{max}<C_{max}'$ → S is better than S'

→ Johnson's algorithm is optimal for **F,2||C_{max}.**

Example:

Given the following data for a **F,2||C_{max}** problem.

M1	J_i	1	2	3	4	5	6	7	8
	p_{i1}	7	5	7	6	4	11	3	5 ←
M2	J_i	1	2	3	4	5	6	7	8
	p_{i2}	11	3	9	7	12	1	2	4 ←

We find the minimum pij in the table. If j=1, place the job on the first available position on the schedule array S. If j=2, place the job on the last available position on the schedule array S. And then we remove the job from the data table.

M1	J_i	1	2	3	4	5	6	7	8
	p_{i1}	7	5	7	6	4	11	3	5
M2	J_i	1	2	3	4	5	6	7	8
	p_{i2}	11	3	9	7	12	1	2	4

S=(…J_6).

M1	J_i	1	2	3	4	5	7	8
	p_{i1}	7	5	7	6	4	3	5
M2	J_i	1	2	3	4	5	7	8
	p_{i2}	11	3	9	7	12	2	4

S=(…J_7,J_6).

M1	J_i	1	2	3	4	5	8
	p_{i1}	7	5	7	6	4	5
M2	J_i	1	2	3	4	5	8
	p_{i2}	11	3	9	7	12	4

S=(… J_2,J_7,J_6).

M1	J_i	1	3	4	5	8
	p_{i1}	7	7	6	4	5
M2	J_i	1	3	4	5	8
	p_{i2}	11	9	7	12	4

S=(J_5,… J_2,J_7,J_6).

M1	J_i	1	3	4	8
	p_{i1}	7	7	6	5
M2	J_i	1	3	4	8
	p_{i2}	11	9	7	4

S=(J_5,… J_8,J_2,J_7,J_6).

M1	J_i	1	3	4
	p_{i1}	7	7	6
M2	J_i	1	3	4
	p_{i2}	11	9	7

S=(J_5 J_4, … J_8,J_2,J_7,J_6).

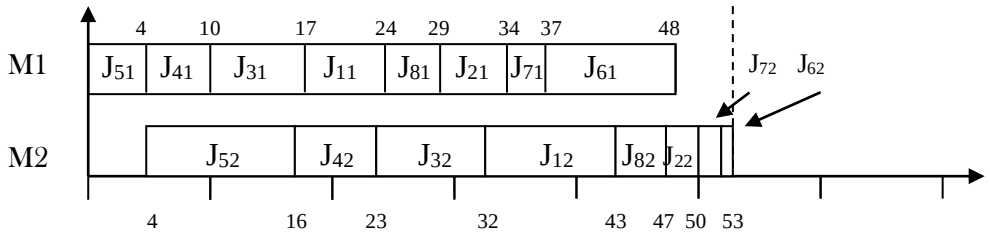
M1	J_i	1	3
	p_{i1}	7	7
M2	J_i	1	3
	p_{i2}	11	9

S=(J_5 J_4,J_3, … J_8,J_2,J_7,J_6)

Being the last unscheduled job, J_1 gets to be in the only available position left in the array S.

S=(J_5 J_4,J_3,J_1,J_8,J_2,J_7,J_6).

Using the Gantt diagram, we determine the makespan of the schedule C_{max}= 53, which is the optimum solution.



Note that it is very important that a job never starts on M2 unless it's done processing on M1, otherwise it will violate the technological constraints of the scheduling problem.

The algorithm of Johnson for the F,2||C_{max} problem:

0- Initialize z=0

1- Find the minimum of all p_{1i} and p_{2i}

2- if j=1 schedule job j_i on the first available position of the sequence

If j=2 schedule job j_i on the last available position of the sequence

3- (Let the first late job be J_i), among the first i jobs, move the largest job to the Sequence

4- Print Sequene and Z

Complexity:

Johnson's algorithm is mostly an array sorting, which can allow solving the **F,2||C_{max}** problem in *O(nlogn)* time.

3.2. The J,2||C_{max} problem:

In order to optimize the makespan C_{max} in a JobShop environment of two machines M1 and M2 and n jobs, we regroup jobs into four sorts according to the technological nature of the JobShop:

Type A: jobs that process on M1only.

Type B: jobs that process on M2 only.

Type C: jobs that process through M1 then M2.

Type D: jobs that process through M2 then M1.

After the repartition, we schedule type A and type B jobs randomly, to obtain the orders S_A and S_B. Then schedule type C and type D jobs according to Johnson's rule for 2F||C_{max} to obtain the orders S_C and S_D. (According to the technical constraints of JobShop, M2 is the first machine for type D jobs and the second machine is M1).

The optimal schedule for 2J||C_{max} is then:

Machine	Processing Order
M1	(S_C, S_A, S_D)
M2	(S_D, S_B, S_C)

Example:

The following example will offer a fair explanation of how a two machines JobShop data is handled when seeking to minimize the makespan C_{max} of the schedule. The jobs on the left start processing on the first machine, while the jobs one the right side of the table are to be processed on the second machine first.

J_i	M1	M2	J_i
J_1	5	4	
J_2	6		
	3	5	J_3
	2	1	J_4
J_5	7		
	7	4	J_6
J_7	9	1	
		2	J_8

The data can be divided by 4 subsets: A, B, C and D.

A= {J_2, J_5} jobs that process on M1.

B= {J_8 } jobs that process on M2.

C= {J_1, J_7} jobs that process on M1 then M2.

D= {J_3, J_4, J_6} jobs that process on M2 then M1.

The tables down below are type A and type B jobs, ordered according to the SPT rule.

Type A		
J_i	2	5
p_i	6	7

Type B	
J_i	8
p_i	2

S_A=(J_2,J_5) S_B=(J_8).

Type C and type D jobs are ordered according to Johnson's method for 2 machines FlowShop problems.

Type C			
M1	J_i	1	7
	p_{i1}	5	9
M2	J_i	1	7
	p_{i2}	4	1

Type D				
M2	J_i	3	4	6
	p_{i2}	5	1	4
M1	J_i	3	4	6
	p_{i1}	3	2	7

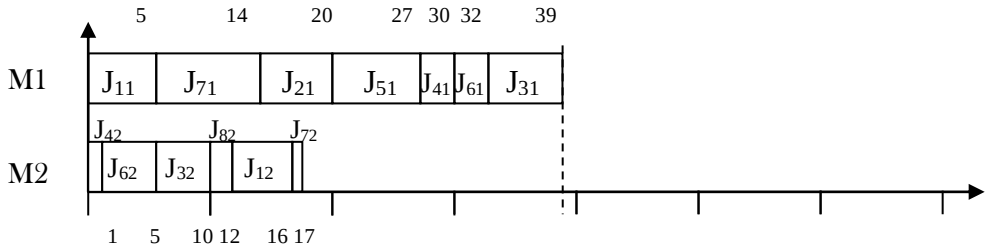
S_C=(J_1,J_7), S_D=(J_4, J_6, J_3).

After obtaining the 4 arrows S_A, S_B, S_C, and S_D, the job sequence on M1 is $S_1=(S_C, S_A, S_D)$ while the sequence on M2 is $S_2=(S_D, S_B, S_C)$

$S_1=(J_1,J_7, J_2,J_5, J_4, J_6, J_3)$.

$S_2=(J_4, J_6, J_3, J_8, J_1,J_7)$.

To determine the makespan of the schedule, we use Gantt diagram, to obtain the optimum solution $C_{max}= 39$.



The algorithm of Johnson for the J,2||C_{max} problem:

0- Initialize Z=0

1- Regroup jobs into 4 sets

TypeA[]=J_i ; where J_i process on M1 only

TypeB[]=J_i ; where J_i process on M2 only

TypeC[]=J_i ; where J_i process on M1 then M2

TypeD[]=J_i ; where J_i process on M2 then M1

2- Sequence jobs from TypeA[] randomly into SA[].

3- Sequence jobs from TypeB[] randomly into SB[].

4- Sort jobs from TypeC[] according to Johnsons rule into SC[];

Where M1 is the first machine then M2 is the second.

5- Jobs from TypeD[] according to Johnsons rule into SD[];

Where M2 is the first machine then M1 is the second.

6- Schedule on M1 ; S1 = SC[] + SA[] + SD[].

7- Schedule on M2 ; S2 = SD[] + SB[] + SC[].

8- Print Z.

Complexity:

Johnson's algorithm extension gives a solution of *O(nlogn)* time for the **J,2||C_{max}** problem.

3.3. The F,3||C_{max} problem:

For a **F,3||C_{max}** problem, there's a special case where the Johnson's rule is optimal, under the two following conditions:

P_{ij} *is the operating duration of Job i on Machine j*

$Min\ \{p_{i1}\} >= Max\{p_{i2}\}$

Or $Min\ \{p_{i3}\} >= Max\{p_{i2}\}\ ;\ i = 1::n$

This means that the maximum of processing duration on the second machine should be minor than the minimum of the processing duration of jobs on either the first or the third machine[9].

The optimal solution is gotten by virtually merging three machines into two, where:

$a_i = p_{i1} + p_{i2}$ *: operating duration of job i on M1*

$b_i = p_{i2} + p_{i3}$ *: operating duration of job i on M2*

Then scheduling jobs as if they're to be processed onto two machines. The final schedule obtained using this rule defines the order of jobs on three machines. That way, the order can be standardized according to the environment characteristics and technological constraints.

The algorithm of Johnson for the F,3||C_{max} problem:

0- Initialize Z

1- Reform DATA from having three machines to two virtual machines :

($a_{1i}=p_{1i}+p_{2i}$) & ($a_{2i}=p_{3i}+p_{2i}$).

2- if j=1 schedule job j_i on the first available position of the sequence

If j=2 schedule job j_i on the last available position of the sequence

3- (Let the first late job be J_i), among the first i jobs, move the largest job to the Sequence

4- Print Sequence and Z

Complexity:

Johnson's algorithm on **F,3||C_{max}** gives a solution by merging 3 machines data into two, then solving the problem as if it's **F,2||C_{max}** on a *O(nlogn)* time.

4. Conclusion

This chapter showcased the different familiar problems followed by their algorithms and some examples that explain how these algorithms get to find optimum and approached solutions.

CHAPTER III: CONSTRUCTIVE ALGORITHMS FOR MANUFACTURING SCHEDULING PROBLEMS

CHAPTER 3

Constructive Algorithms for Manufacturing Scheduling Problems

1. Introduction:

In the past chapter we covered algorithms that solve some of the classic scheduling problems. However, most of those problems are less likely to happen in reality. Real life situations are more complex, as well as scheduling problems, which in real scenarios are more complicated and have a variety of constraints that force us to ditch those classic algorithms and try to find heuristics that are specifically compatible with those complex problems.

In this chapter we will try to cover some of the possible scenarios in manufacturing scheduling problem, with restrictions of release dates (ready dates : r_i) and preemption constraints (PRMN) on various types of environments and numbers of resources.

To test the accuracy of the algorithms, we will try to do an analytic comparison with a lower bound which can be determined by the relaxation of the problems and the disposing of constraints.

2. Single Stage Problems

2.1. The $1|r_i|\sum C_i$ Problem:

Assuming that jobs aren't to be processed on a default start time, where each job J_i (i=1::n) is not allowed to process before its ready date r_i.

A constructive heuristic algorithm based on an adaptation of the SPT rule is used to determine an approached solution for this problem. When a certain number of jobs are ready at a certain date, the job to be processed is determined based on the processing duration of each one according to the SPT rule. The same process is repeated until an array S is fully constructed, which represents the order of jobs in the process schedule.

Example:

Given the following data for $1|r_i|\sum C_i$

We start by the ready job J_2

J_i	1	2	3	4
p_i	6	4	8	9
r_i	2	0	4	3

J_i	1	2	3	4
p_i	6	4	8	9
r_i	4	0	4	4

S=(J_2,…..)

After having scheduled J_2, we update the ready dates, all three ready jobs J_1, J_3, and J_4 get to be scheduled according to the SPT rule.

J_i	1	2	3	4
p_i	6	4	8	9
r_i	2	0	4	4

S=(J_2, J_1, J_3, J_4)

J_i	2	1	3	4
p_i	4	6	8	9
r_i	0	4	4	4
C_i	4	10	18	27

$\sum C_i = 59$.

The algorithm:

0- C=T=0 ; Obj Function and Current time.

1- Get DATA pi, ri (i=1::n)

2- Loop (until DATA is empty)

Ready[]=Find jobs with earliest ri;

Best[]=Find jobs with minimum pi from Ready[];

Schedule Ji on ri;

Update time: T=ri+pi;

Update ObjFun: C=C+T;

Remove job from DATA;

Update ready dates: If (ri<T) Then ri=T;

3- Print C and the schedule S;

Complexity:

In this case, the sorting is repeated a limited number of times which is to be ignored, therefore, the complexity is *O(nlogn)*.

Analytics:

To test the efficiency of this algorithms, we have made a comparison between the objective value for a number of random data and their lower bounds, which we have obtained from the relaxed programs for the $1||\sum C_i$. The results show that this algorithm has a global efficiency ratio of 89.1%

2.2. The $1|r_i|\sum w_iC_i$ Problem:

In order to minimize the sum of weighted completion times $\sum w_iC_i$ in a one machine environment, with the involvement of r_i (i=1::n), we get the help of a constructive heuristic algorithm that builds a minimal solution value based on the SWPT rule

When a certain number of jobs are ready at a certain date, the job to be processed is determined based on the ratio p_i/w_i of each one according to the SWPT rule. The same process is repeated until an array S is fully constructed, which represents the order of jobs in the processing schedule.

Example:

Given the following data for $1|r_i|\sum w_iC_i$

We start by the ready job J_2

J_i	1	2	3	4
p_i	6	4	8	9
w_i	2	4	1	9
r_i	2	0	4	3

S=(J_2,…..)

J_i	1	2	3	4
p_i	6	4	8	9
w_i	2	4	1	9
r_i	4	0	4	4

After having scheduled J_2, we update the ready dates, all three ready jobs J_1, J_3, and J_4 get to be scheduled according to the SWPT rule.

J_i	1	2	3	4
p_i	6	4	8	9
w_i	2	4	1	9
r_i	4	0	4	4

$S=(J_2, J_4, J_1, J_3)$

J_i	2	4	1	3
p_i	4	9	6	8
r_i	0	4	4	4
w_i	2	9	2	1
C_i	4	13	19	27
w_iC_i	8	117	38	27

$\sum w_iC_i=190$.

The algorithm:

0- C=T=0 ; Obj Function and Current time.

1- Get DATA pi, ri and wi (i=1::n)

2- Loop (until DATA is empty)

Ready[]=Find jobs with earliest jobs of minimum ri;

Best[]=Find jobs with minimum ratio pi/wi from Ready[];

Schedule Ji on ri;

Update time: T=ri+pi;

Update ObjFun: C = C + wi*T;

For all jobs : Update ready dates: If (ri<T) Then ri=T;

Remove job Ji from DATA;

3- Print C and the schedule S;

Complexity:

The availability constraints require the sorting to be repeated multiple times m (m=n in worst cases). So the complexity is *O(nlogn)*.

Analytics:

The comparison between the results of this algorithm and the results of the relaxed problem $1||\sum w_i C_i$ show a global efficiency ratio of 81.4%.

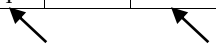
2.3. The $1|r_i$, PRMN$|\sum C_i$ Problem:

In one machine scheduling problems that feature release dates for the jobs, minimal solution for $\sum C_i$ *and* $\sum w_i C_i$ can be found using the past two algorithms. However, in some cases, another characteristic of jobs can be involved, the preemption PRMN, which is when preemption is allowed in the schedule. What we mean by PRMN, is that it is possible that a job can be interrupted at a certain timing in order to allow another job with a better ratio or a minimal duration to be processed instead, with the possibility of continuation of the interrupted job.

Example:

Given the following data for $1|r_i|\sum C_i$

J_i	1	2	3	4
p_i	2	5	8	9
r_i	2	0	4	3



J_2 is ready, starts, but gets interrupted by J_1. p_1 is updated.

J_i	1	2	3	4
p_i	2	3	8	9
r_i	2	2	4	4

J_i	1	2	3	4
p_i	2	3	8	9
r_i	2	4	4	4

S=(J_1,…..)

All other jobs now are ready to process, for that, the SPT rule should be respected.

S=(J_1, J_2, J_3, J_4)

J_i	2 ,	1	2	3	4
p_i	2	2	3	8	9
r_i	0	2	4	4	4
C_i	-----	4	7	15	24

$\sum C_i = 50$.

The algorithm:

0- C=T=0 ; Obj Function and Current time.

1- Get DATA pi, ri (i=1::n)

2- Loop (until DATA is empty)

Ready[]=Find jobs with earliest ri;

Best[]=Find jobs with shortest pi from Ready[];

For [i=1::n]

If(ri+pi>rj+pj)

A=execute only (ri+pi)-rj of Ji on ri;

Update pi=pi-A;

Update ri=ri+A; T=A+ri;

Else

Execute Ji on ri;

Update ri=ri+pi;

T=pi+ri ; C=C+T ;

3- Remove Ji from Data;

4- Update ri for all jobs;

5- Print C and the schedule S;

Complexity:

In worst cases, each job is to be interrupted twice, with the availability requirement of sorting m times, complexity is *O(2×m×nlogn)*. *2×m* are ignored, so the complexity of this algorithms is *O(nlogn)*.

Analytics:

After comparing the results given by this algorithm on the $1|r_i\ PRMN|\sum C_i$ problem and the results given from the relaxed problem $1|r_i\ PRMN|\sum C_i$ on the same data, we found that the efficiency ratio is 90.2%.

2.4. The $1|r_i$, $PRMN|\sum w_iC_i$ Problem:

When weights are assigned to the jobs within the involvement of the PRMN properties, the jobs behave in the same way in the $1|r_i\ ,\ PRMN|\sum w_iC_i$ problem as the past $1|r_i\ ,\ PRMN|\sum w_iC_i$ problem. In this problem, p_i/w_i ratios of ready jobs are compared instead.

However, in this type of problem, we cite two different cases that differ based on the technical characteristics of the jobs. In the first case, when a job gets interrupted, both of the first portion and the other portion are to be considered as individual jobs, while in the second case, the first portion is completely ignored.

Case 1:

Let's assume that when a job J_i gets interrupted, J_{i1} with a processing time p_{i1} and a weight w_{i1}, J_{i2} with a processing time p_{i2} and a weight w_{i2}, are the two portions resulted from the breaking of J_i.

Where

$$p_i = p_{i1} + p_{i2}$$

$$p_i/w_i = p_{i1}/w_{i1} = p_{i2}/w_{i2}$$

$$\leftrightarrow\ w_{i1} = \frac{wi*pi1}{pi}$$

$$\leftrightarrow\ w_{i2} = \frac{wi*pi2}{pi}$$

$$\leftrightarrow\ w_{i1} + w_{i2} = \frac{wi*pi1+wi*pi2}{pi} = \frac{wi*pi}{pi} = w_i$$

$$\leftrightarrow\ w_i = w_{i1} + w_{i2}$$

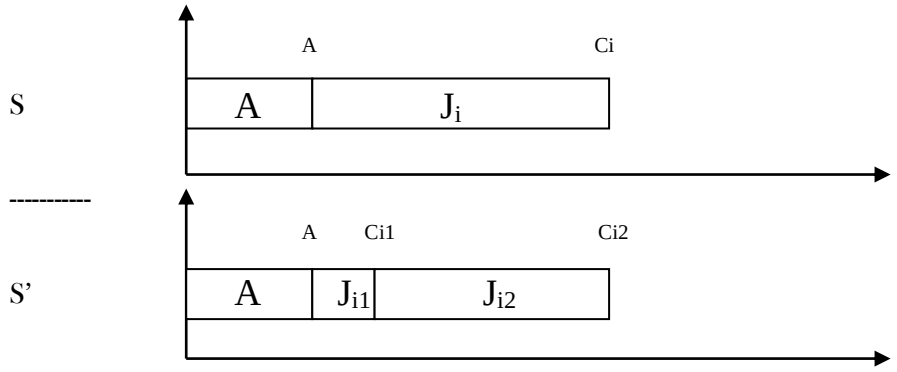


Fig 3.1: Gantt chart in PRMN case 1

Let us find the difference value of the criteria $\sum w_i C_i$ caused by breaking a job J_i.

Let Z be the objective value of the schedule S, while Z' is the objective value of the schedule S'.

$$Z = A + w_i*(p_i + A)$$

$$Z' = A + w_{i1}*(p_{i1} + A) + w_{i2}*(p_{i2} + p_{i1} + A)$$

Let CBRK be the cost of breaking the job Ji

$$CBRK = Z - Z'$$

$$= w_i*(p_i + A) - w_{i1}*(p_{i1} + A) - w_{i2}*(p_{i2} + p_{i1} + A)$$

$$= w_{i1}*(p_{i2} + p_{i1} + A) + w_{i2}*(p_{i2} + p_{i1} + A) - w_{i1}*(p_{i1} + A) - w_{i2}*(p_{i2} + p_{i1} + A)$$

$$= w_{i1}*(p_{i2} + p_{i1} + A) - w_{i1}*(p_{i1} + A)$$

$$CBRK = w_{i1}*p_{i2}$$

The cost of breaking will be added then to Z' after each interruption of a job J_i.

Case 2:

In this case, the first portion is simply ignored, and only the second portion is considered as a job J_i that has a completion time C_i a weight w_i and a duration p_i.

Where:

$$p_i = p_{i2} = p_i - p_{i1}$$

$$w_i = w_{i2} = w_i - wi_1$$

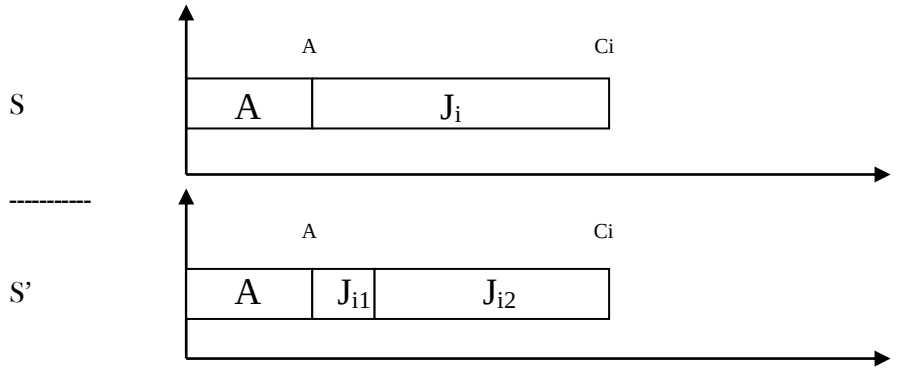


Fig 3.2: Gantt chart for PRMN case 2

$CBRK = Z - Z'$

$= w_i*(p_i + A) - w_{i2}*(p_{i2} + A)$

$= w_i*(p_i + A) - w_i*(p_i + A)$

$CBRK = 0$

Example 1:

Given the following data for $1|r_i, PRMN|\sum w_i C_i$ to cover the second case of the problem

J_i	1	2	3	4
p_i	2	6	8	9
w_i	1	2	1	9
r_i	2	0	4	3

J_2 is ready, starts, but gets interrupted by J_1, because the potential second portion of J_2 has a greater ratio p_2/w_2 than p_1/w_1

J_i	2'	1	2	3	4
p_i	2	2	4	8	9
w_i	2	1	2	1	9
r_i	0	2	2	4	3

J_i	2'	1	2	3	4
p_i	2	2	4	8	9
w_i	2	1	2	1	9
r_i	0	2	2	4	3

J_i	2'	1	2	3	4
p_i	2	2	4	8	9
w_i	2	1	2	1	9
r_i	0	2	4	4	4

$S=(J_1,\ldots..)$

Since all the jobs are available, they process according to the SWPT rule, to obtain the final sequence $S=(J_1, J_4, J_2, J_3)$

J_i	2'	1	4	2	3
p_i	2	2	9	4	8
w_i	2	1	9	2	1
r_i	0	2	4	4	4
C_i	------	4	13	17	25
w_iC_i	------	2	117	34	25

$\sum w_iC_i=178.$

Example 2:

Given the following data for $1|r_i, PRMN|\sum w_iC_i$ to cover the first case of the problem

J_i	1	2	3	4
p_i	2	6	8	9
w_i	1	2	1	9
r_i	2	0	4	3

J_2 is ready, starts, but gets interrupted by J_1, because the ratio p_2/w_2 is greater than p_1/w_1.

J_i	2'	1	2	3	4
p_i	2	2	4	8	9
w_i	0.66	1	1.33	1	9
r_i	0	2	2	4	3

The cost of breaking CRBK gets updated

CRBK=CRBK+0.66*4=2.64.

J_i	2'	1	2	3	4
p_i	2	2	4	8	9
w_i	0.66	1	1.33	1	9
r_i	0	2	4	4	4

S=(J_1,…..)

The rest of the jobs are available, they process according to the SWPT rule, to obtain the final sequence S=(J_1, J_4, J_2, J_3)

J_i	2'	1	4	2	3
p_i	2	2	9	4	8
w_i	0.66	1	9	1.33	1
r_i	0	2	4	4	4
C_i	2	4	13	17	25
w_iC_i	1.33	4	117	22.62	25

$\sum w_iC_i = 169.95 + CBRK$

$\sum w_iC_i = 172.6$

The algorithm:

Case 1:

0- C=T=0 ; Obj Function and Current time.

1- Get DATA pi, ri (i=1::n)

2- Loop (until DATA is empty)

Ready[]=Find jobs with earliest ri;

Best[]=Find jobs with minimum pi/wi from Ready[];

For [i=1::n]

If ((ri+pi>rj+pj) and (pi/wi > pj/wj))

A=execute only (ri+pi)-rj of Ji on ri;

Update pi=pi-A;

Add job Ji' ; pi'= A, wi' = wi*pi'/pi

Calculate CBRK = wi*pi';

Update ri=ri+A; T=A+ri;

Else

Execute Ji on ri;

Update ri=ri+pi;

T=pi+ri ; C=C+T ;

3- Update C;

4- C = C + CBRK

5- Remove Ji from Data;

6- Update ri for all jobs;

7- Print C and the schedule S;

Case 2:

0- C=T=0 ; Obj Function and Current time.

1- Get DATA pi, ri (i=1::n)

2- Loop (until DATA is empty)

Ready[]=Find jobs with earliest ri;

Best[]=Find jobs with minimum pi/wi from Ready[];

For [i=1::n]

If ((ri+pi>rj+pj) and (pi+ri-pj-rj/wi > pj /wj))

A=execute only (ri+pi)-rj of Ji on ri;

Update pi=pi-A;

Update ri=ri+A; T=A+ri;

Else

Execute Ji on ri;

Update ri=ri+pi;

T=pi+ri ; C=C+T ;

Update C;

Remove Ji from Data;

3- Update ri for all jobs;

4- Print C and the schedule S;

Complexity:

Each job is possibly to be interrupted less than two times twice, with the availability requirement of sorting m times, complexity is $O(2\times m\times nlogn)$. $2\times m$ are ignored, so the complexity of this algorithms is $O(nlogn)$.

Analytics:

In the first case, the efficiency ratio tends to 87.65%, whilst the second case has a global efficiency ratio of 87%.

3. Multi Stage Problems:

3.1. The $F,2|r_i|C_{max}$ problem:

In the case of 2 machines, each job has two operation durations p_{i1} and p_{i2} on M1 and M2 respectively, and also two ready dates, r_{i1} and r_{i2} on M1 and M2 respectively. Each ready date r_{ij} determines whether a job J_i is ready to operate on the machine M_j or not.

The adaptation of the SPT rule and the algorithm of Johnson on the $F,2|\ |C_{max}$ resulted a constructive heuristic algorithm based to build an approached solution gradually for the $F,2|r_i|C_{max}$ problem.

The concept of this algorithm to find the jobs with the minimum ready date r_{ij} whether on M1 or M2, when a certain number of jobs are ready at a certain date, the job operate is determined based on the processing duration of each one according to the SPT rule. If the minimum operation duration p_{ij} belongs to the first machine, the job processes at the first available position of the schedule, if it belongs to the second machine, the job processes on the last available position on the schedule. After determining the schedule gradually in a form of an array, the jobs operate on the first machine then on the second machine on their ready dates to assure the technological constraints of the FlowShop environment.

Example:

Given a data for a $F,2|r_i|C_{max}$ problem for 4 jobs.

	M1				M2			
J_i	J_1	J_2	J_3	J_4	J_1	J_2	J_3	J_4
p_i	4	6	4	4	5	7	2	2
r_i	6	0	1	5	0	2	3	7

The available jobs are J_1 and J_2, but the shortest processing time job is J_{12}, so it is scheduled at the very end of the program. S=(……J_1)

	M1				M2			
J_i	J_1	J_2	J_3	J_4	J_1	J_2	J_3	J_4
p_i	4	6	4	4	5	7	2	2
r_i	6	0	1	5	0	2	3	7

The available job is J_{21}, so it is to be scheduled at the very beginning of the program S=(J_2,….J_1). Ready dates are to be updated if necessary.

	M1				M2			
J_i	J_1	J_2	J_3	J_4	J_1	J_2	J_3	J_4
p_i	4	6	4	4	5	7	2	2
r_i	6	0	6	6	0	2	6	7

The available jobs are J_{31}, J_{41} and J_{32}, the shortest processing time of them is p_{32}, so the job J_3 is to be processed at the available position in the array S. S=(J_2,…J_3,J_1). The only available job that is left is J_4, therefore the sequence of the schedule is S=(J_2,J_4,J_3,J_1).

To determine the makespan of the schedule, we ought to utilize the diagram of Gantt.

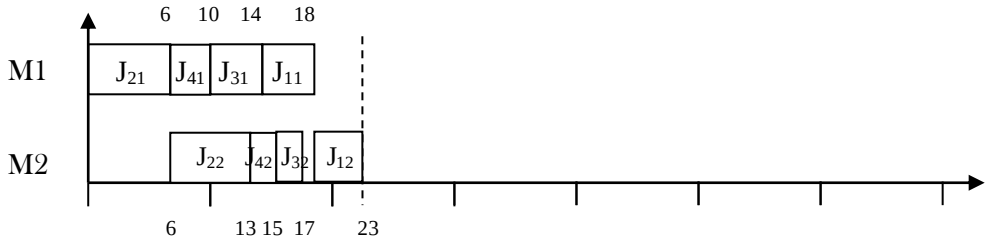


Fig 3.3: Gantt chart for FSP

C_{max}= 23.

The algorithm:

0- C = 0 ; Obj Function

1- Get DATA pij, rij (i=1::n | j=1,2)

2- Loop (until DATA is empty)

Ready[]=Find jobs with earliest rij;

Best[]=Find jobs with minimum pij from Ready[];

If (j=1) : sequence Jij forward; (First available spot on Seq[])

If (j=2) : sequence Jij backwards; (Last available on Seq[])

Update rij for all jobs;

Remove Jij from DATA;

3- Schedule all jobs Jij according to the Sequence on Seq[];

4- Print C and Schedule;

Complexity:

The sorting of the jobs array happens n times, the sorting complexity is *O(nlogn*). Multiplied by the number of iteration makes it tend to $O(n^2 logn)$.

Analytics:

This algorithm has an efficiency ratio of 86.7% when compared to the algorithm of Johnson for the $F,2||C_{max}$ problem.

3.2. The $F,3|r_i|C_{max}$ problem:

For a $\mathbf{F,3|r_i|C_{max}}$ problem, data should respect the particular case, with the following conditions:

P_{ij} *is the operating duration of Job i on Machine j*

$Min\ \{p_{i1}\} >= Max\{p_{i2}\}$

Or $Min\ \{p_{i3}\} >= Max\{p_{i2}\}\ ;\ i = 1::n$

Let's assume that $r_{i1}=r_{i2}=r_{i3}$, which means that a job J_i is ready to operate on the second and the third machines as long as it can operate on the first machine at a certain timing.

A constructive algorithm based on Johnson's algorithms for the $F,3||C_{max}$ that builds a solution gradually is used to determine an approached solution for the $F,3|r_i|C_{max}$ problem.

This algorithm is based on the concept of merging three data of the three machines into a two machine data:

Where:

$a_i = p_{i1} + p_{i2}$ *: operating duration of job i on M1*

$b_i = p_{i2} + p_{i3}$ *: operating duration of job i on M2*

The next steps work exactly the same as the algorithm of the $F,2|r_i|C_{max}$ problem, until a minimum solution is found.

The Algorithm:

0- C = 0 ; Obj Function

1- Get DATA p_{ij}, r_{ij} (i=1::n | j=1,3)

2- Reform DATA from having three machines to two machines:

($a_{i1}=p_{i1}+p_{i2}$) & ($a_{i2}=p_{i3}+p_{i2}$).

3- Loop (until DATA is empty)

Ready[]=Find jobs with earliest r_{ij};

Best[]=Find jobs with minimum a_{ij} from Ready[];

If (j=1) : sequence J_{ij} forward; (First available spot on Seq[])

If (j=2) : sequence J_{ij} backwards; (Last available on Seq[])

Update r_{ij} for all jobs;

Remove J_{ij} from DATA;

4- Schedule all jobs J_{ij} according to the Sequence on Seq[];

5- Print C and Schedule;

Complexity:

The sorting of the jobs array happens to be repeated n times until all jobs are scheduled, the sorting complexity is $O(nlogn)$, being multiplied by the number of iteration makes it tend to $O(n^2logn)$.

Analytics:

This algorithm has an efficiency ratio of 83.4% when compared to the algorithm of Johnson for the $F,3||C_{max}$ problem.

4. Realization & Experimentation:

As a result of the algorithms, and in order to do analytic experiments on every possible case, we ought to develop a program that covers all the problems covered in this chapter, using a tool that takes good advantage of a modern programming language.

4.1. The programming language : Java:

Java is a programming language, widely used and a common computer programming language that is concurrent, class based, and object oriented, specifically designed to have as few implementation dependencies as possible. Java was originally developed by James Gosling at Sun Microsystems and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.[4]

Fig 3.4: Java's logo

4.2. The programming environment: NetBeans IDE :

NetBeans integrated development environment (IDE) is a tool extended from the bundle NetBeans development platform, which is primarily intended to be used by third party developers in order to compile their codes written in Java language.

Fig 3.5: NetBeans' logo

4.3. Presentation of the developed application:

The application developed using the Java programming language and based on the algorithms cited in this chapter is a GUI-based software, that solves the scheduling problems in multiple cases.

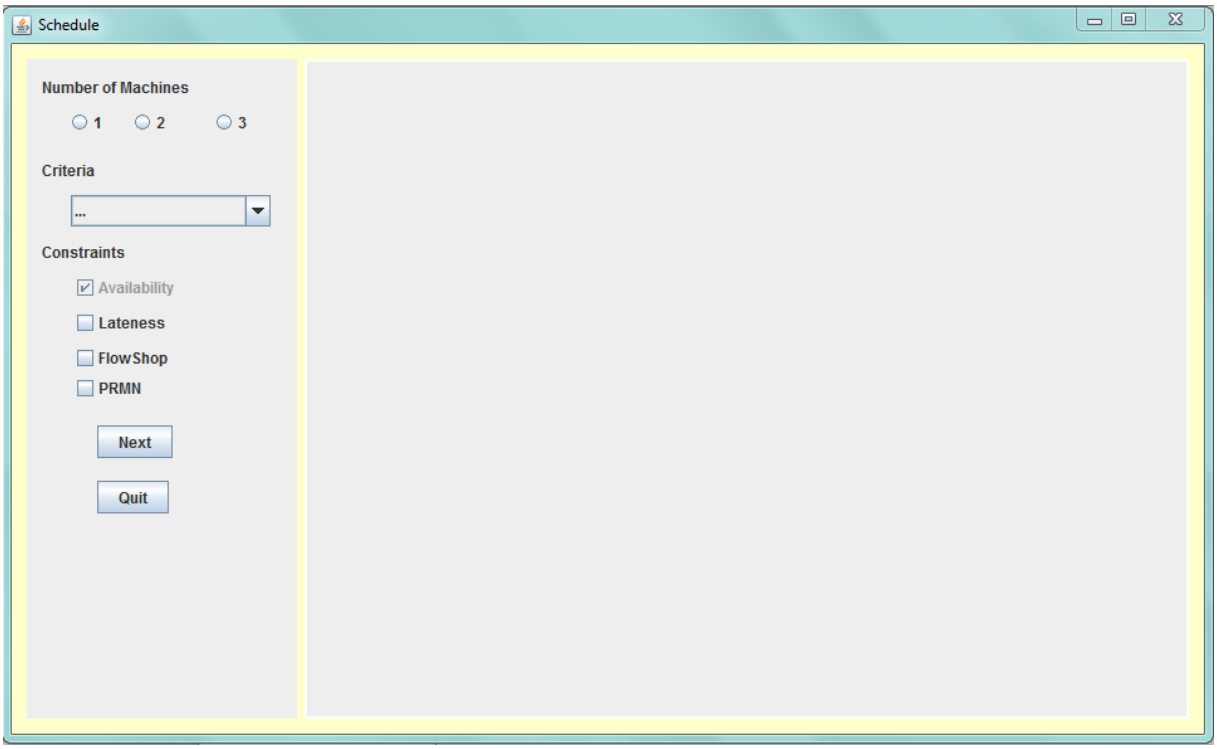


Fig 3.6: GUI 1

The user interface has multiple options in terms of the number of machines, the criteria that is intended to be optimized, and the different constraints and restrictions that can face a problem.

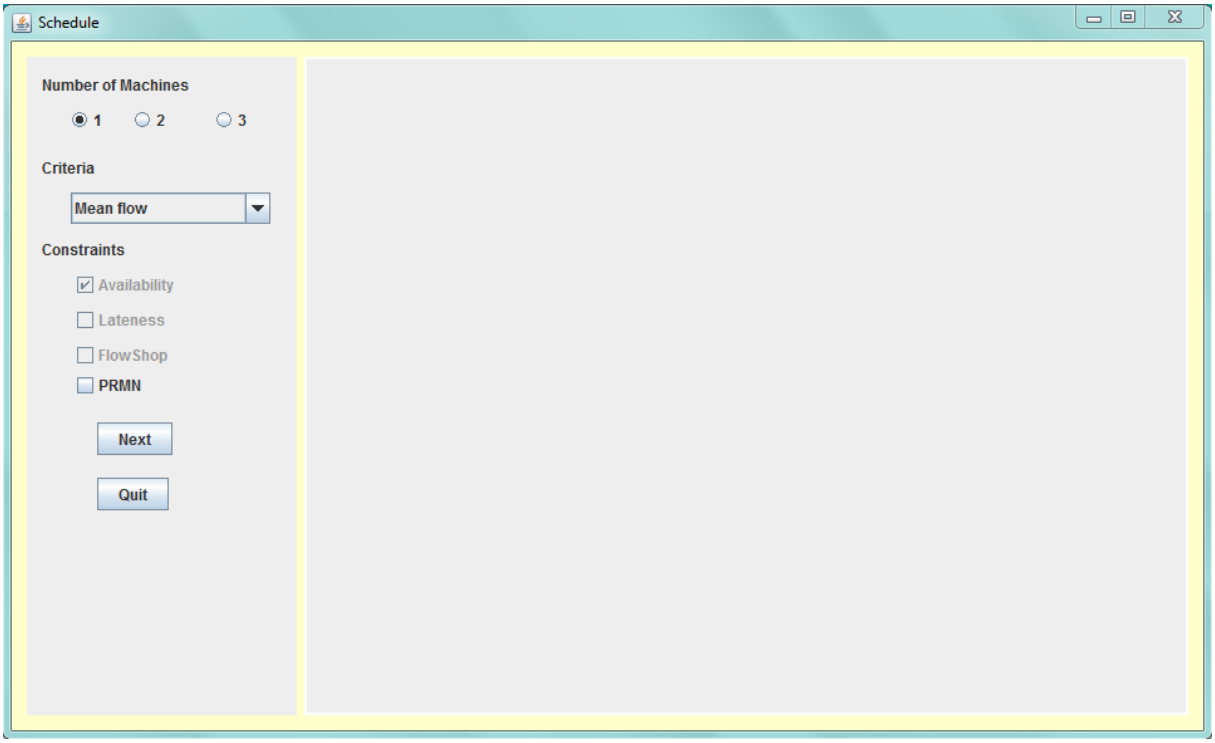


Fig 3.7: GUI 2

After selecting the number of machines, only the criteria options that match this case appear in the criteria selection bar. By selecting a certain criteria, the user can select some constraints that are suited for the type of problem detected by the software. The user should press the Next button in order to continue using, or Quit in order to finish to close the software.

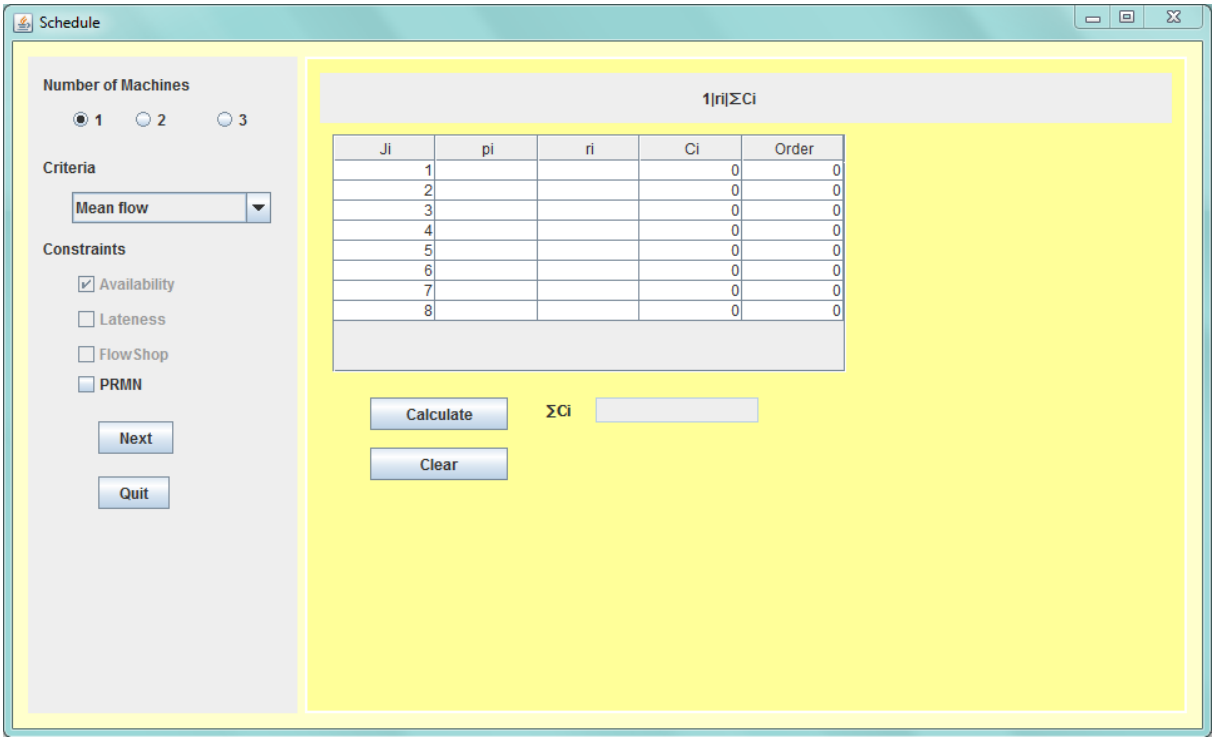


Fig 3.8: GUI 3

When the user chooses to continue running the program, another interface appears to show the table that will be filled with the scheduling problem data by the user manually.

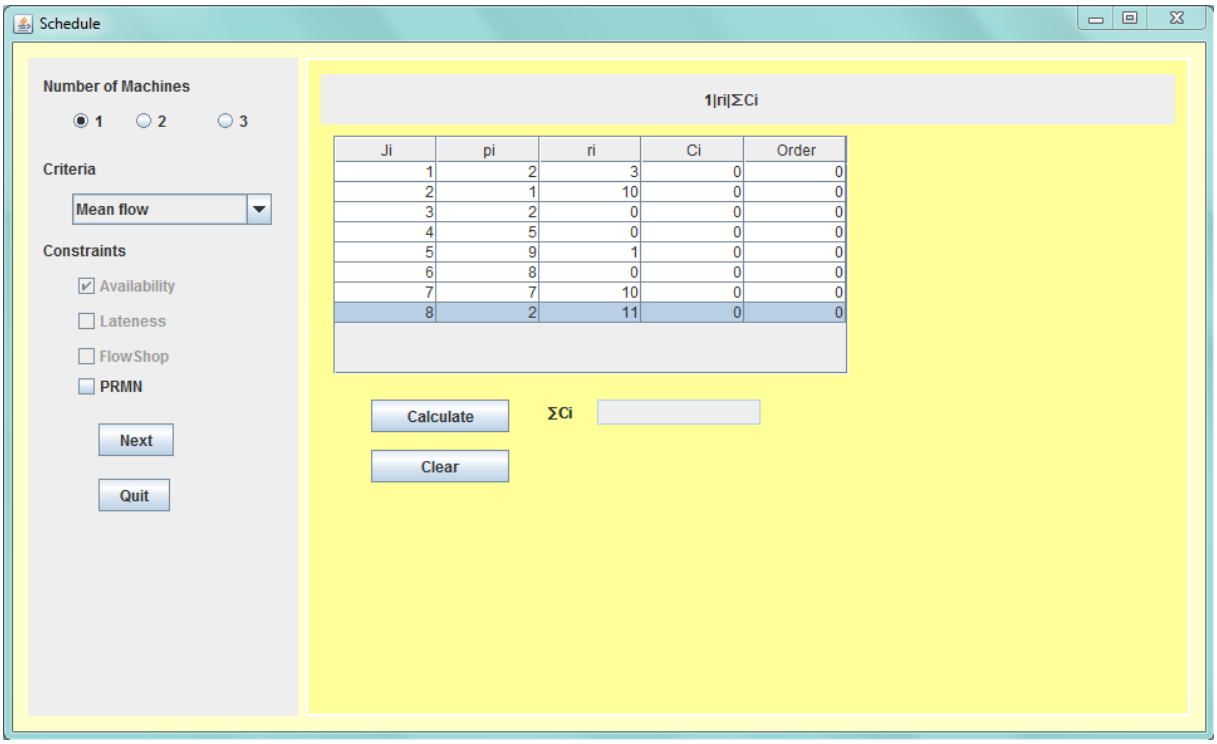


Fig 3.9: GUI 4

The software generates the solution if the user clicks on the Calculate button.

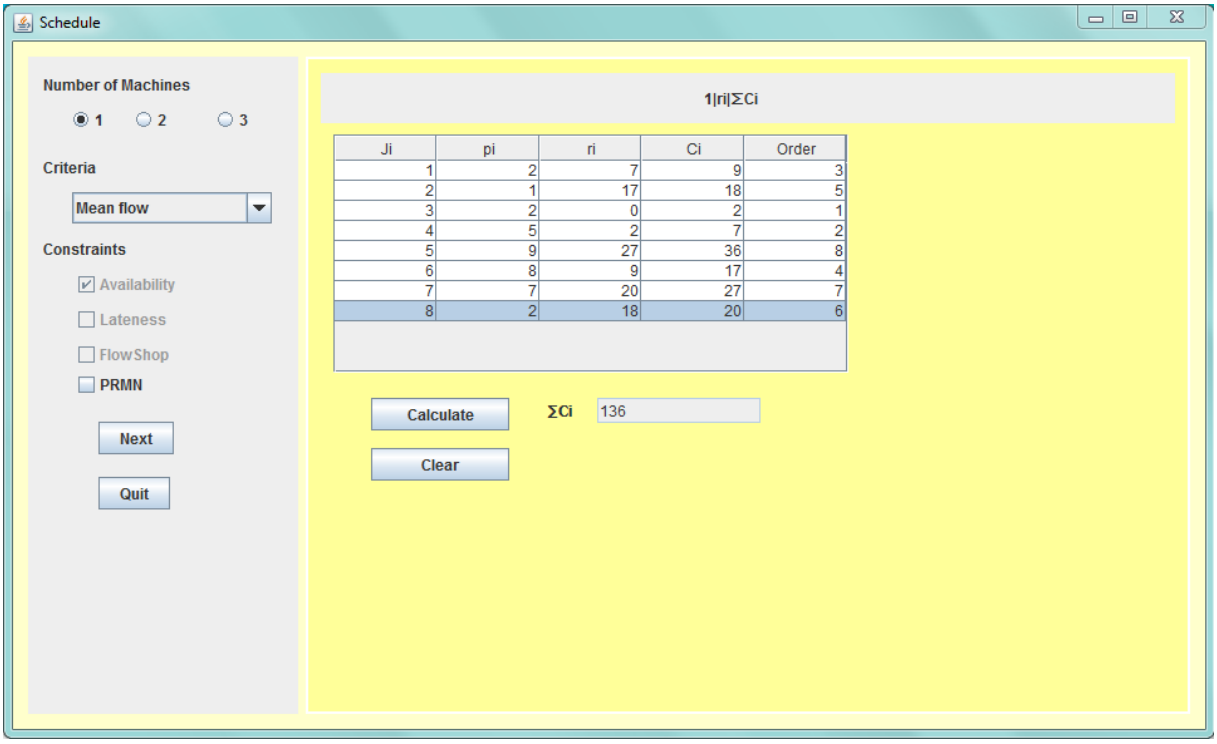


Fig 3.10: GUI 5

The objective value is generated, as well as the sequence and the completion time of each job.

5. Conclusion

In this chapter we covered some of the possible scenarios in manufacturing scheduling problem, with a few restrictions of release dates (ready dates : r_i) and preemption constraints (PRMN) on different types of problems, by proposing constructive heuristics to find good solutions, that were backed-up by a good ratio of efficiency. Those numbers were generated by a program developed in Java programming language.

CONCLUSION

CONCLUSION

The problem of scheduling is very common in the economy and management and a very important element in production planning which consists of optimizing certain criteria and respecting certain limitation and restrictions that the work environments cause.

In this work, the main objective was to study the scheduling problems in more complex ways, to determine minimum solutions that identify minimum costs despite of the different limitation and constraints that could occur in the situation.

Therefore, we determine a number of algorithms that are approached constructive heurists to build good approximate solutions.

This work can be improved in future studies, to find better solutions with less complexity.

BIBLIOGRAPHY

[1] Anand, E. and Panneerselvam, R. Literature Review of Open Shop Scheduling Problem. 2015.

[2] Fatima Sayoti, Mohammed Essaid Rif. Golden Ball Algorithm for solving Flow Shop Scheduling Problem . 2013.

[3] Graham, R. L.; Lawler, E. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey.1979.

[4] Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad; Buckley, Alex. The Java® Language Specification. 2014

[5] Laribi Imane. Heuristics for Solving Flow-Shop Scheduling Problem Under Resources Constraints. 2016.

[6] Lawler, Lenstra, Sequencing & Scheduling : Algorithms and Complexity. 1993.

[7] Nilsson, Stefan. The Fastest Sorting Algorithm. 2000

[8] S. Ahmad. Minimize the Makesapan for Job-Shop Scheduling Problems Using Artificial System Approach. 2015.

[9] S. French. Sequencing and Scheduling: an Introduction to the Mathematics of the Job-Shop. 1981

ملخص

مشكلة الجدولة هي طريقة التعامل و تنظيم و مراقبة أعباء العمل للأنشطة المختلفة مع احترام القيود المفروضة من بيئة العمل من أجل ايجاد الحلول المثلى مع ضمان انتاجية جيدة. هذا العمل هو دراسة لمشاكل الجدولة متعددة المهام في حالات مختلفة. و الهدف الرئيسي من هذا العمل هو ايجاد حلول جيدة لمشاكل جدولة في وضعيات أكثر عملية من خلال استعمال خوارزميات بناءة

الكلمات المفتاحية: جدولة الورشات، المهام، الخوارزميات، القيود

Abstract

A scheduling problem is a way of handling and arranging and controlling workloads of different activities using different resources, respecting the restrictions of the work environment in order to find minimum solutions with good productivity. This paper studies multi-task manufacturing scheduling problems on its different cases. The main objective of this work is to find good solutions for the scheduling problems in more practical scenarios, by building constructive algorithms.

Keys words: Scheduling, jobs, algorithms, constraints

Résumé

Un problème de planification est de gérer et d'organiser et de contrôler les charges de travail de différentes activités en utilisant différentes ressources, en respectant les restrictions de l'environnement de travail afin de trouver des solutions minimales avec une bonne productivité. Cet article étudie les problèmes de planification de production multi-tâches sur des différents cas. L'objectif principal de ce travail est de trouver de bonnes solutions pour les problèmes d'ordonnancement dans des scénarios plus pratiques, en construisant des algorithmes constructifs.

Mots clés : ordonnancement dans l'atelier, tâches, algorithmes, contraintes