

10/01/202

# COMPTE RENDU PROJET

## MES / ERP

CI-2 GEM



Réalisé par : MAIMOUNI Salah Eddin  
KRAIMI Aya

Encadré par : M.AKIL

Année universitaire 2023/2024

## *Remerciement*

On tient à exprimer notre profonde gratitude envers M. AKIL, qui nous a donné l'opportunité de vivre cette expérience qui marquera notre parcours universitaire en tant qu'élève ingénieur. pour sa contribution exceptionnelle à notre projet de groupe. D'autant plus le rôle crucial qu'a joué pour sa contribution exceptionnelle à notre projet de groupe.

C'est grâce à sa détermination, son expertise et son dévouement que notre équipe a pu surmonter les défis et atteindre nos objectifs avec succès. Merci infiniment pour votre engagement et votre excellence dans ce projet. C'était un plaisir de travailler à vos côtés, et on est reconnaissants de la chance d'avoir un collaborateur aussi exceptionnel.

## Table des matières

CHAPITRE I: LE MODELE “FLOW SHOP”:	7
I.    Introduction générale.....	7
1.    Présentation du modèle « Flow Shop » .....	7
2.    Notion générale .....	7
3.    Diagramme de GANTT .....	8
II.    Modèle de Flow Shop sans contrainte de préparation .....	9
1.    Modèle « Flow Shop » à 2 machines.....	9
2.    Modèle « Flow Shop » à m machines .....	11
III.    Modèle de Flow Shop sous contrainte : .....	13
1.    Modèle « Flow shop » avec préparation et sans blocage .....	13
2.    Modèle « Flow shop » avec blocage et sans préparation .....	15
3.    Modèle « Flow shop » avec blocage et préparation .....	17
CHAPITRE II : ETUDE THEORIQUE.....	19
I.    Data de Problème: .....	19
1.    Tableau de temps de processing .....	19
2.    Tableau de temps de préparation .....	19
3.    Liste des délais de chaque Job.....	20
II.    PFSP (Permutation Flow Shop Scheduling Problem) .....	20
1.    Séquence par algorithme de CDS.....	20
2.    Etapes de résolution de PFSP (Cmax/TFT...) .....	21
3.    Matrice des dates des fins et des débuts de chaque job sur chaque machine .....	22
4.    Performances des machines.....	22
5.    Temps d'attende de chaque job entre chaque deux machines.....	23
6.    Diagramme de Gant, .....	23
III.    PFSP-SDST (Permutation Flow Shop Scheduling Problem with Sequence Depending Setup Time)	24
1.    Séquence par Test de tous les permutations possibles avec min(Cmax) et min(TT).....	24
2.    Etapes de résolution de PFSP-SDST (Cmax/TFT..) .....	24
3.    Matrice des dates des fins et des débuts de chaque job sur chaque machine .....	25
4.    Performances des machines (machines non arrêtées en préparation).....	25
5.    Performances des machines (machines arrêtées en préparation).....	26
6.    Temps d'attendre de chaque job entre chaque deux machines.....	26
7.    Diagramme de Gant, .....	27

IV.	BFSP (Blocking Flow Shop Scheduling Problem).....	27
1.	Séquence par Test de tous les permutations possibles avec min(Cmax) et min(TT).....	27
2.	Etapes de résolution de BFSP (Cmax/TFT...).....	27
3.	Matrices D,C,F .....	30
4.	Performances des machines.....	31
5.	Temps de blocage de chaque job dans chaque machines .....	31
6.	Diagramme de Gantt, .....	32
V.	BFSP-SDST (Blocking Flow Shop Scheduling Problem with Sequence Depending Setup Time) .....	32
1.	Séquence par Test de tous les permutations possibles avec min(Cmax) et min(TT).....	32
2.	Etapes de résolution de BFSP-SDST (Cmax/TFT...) .....	32
3.	Matrices D,C,F .....	36
4.	Performances des machines (machines non arrêtées en préparation).....	36
5.	Performances des machines (machines arrêtées en préparation).....	37
6.	Temps de blocage de chaque job dans chaque machine .....	38
7.	Diagramme de Gantt, .....	38
VI.	NIPFSP (No-Idle Permutation Flow Shop Scheduling Problem).....	38
1.	Séquence par Test de tous les permutations possibles avec min(Cmax) et min(TT).....	38
2.	Etapes de résolution de NIPFSP (Cmax/TFT...) .....	38
3.	Matrice des dates des fins et des débuts de chaque job sur chaque machine .....	41
4.	Performances des machines.....	41
5.	Temps d'attente de chaque job entre chaque deux machines.....	42
6.	Diagramme de Gantt, .....	42
VII.	NWPFSP (No-Wait Permutation Flow Shop Scheduling Problem).....	42
1.	Séquence par Test de tous les permutations possibles avec min(Cmax) et min(TT).....	42
2.	Etapes de résolution de NWPFSP (Cmax/TFT...) .....	42
3.	Matrice des dates des fins et des débuts de chaque job sur chaque machine .....	49
4.	Performances des machines.....	49
5.	Diagramme de Gantt, .....	50
	Chapitre III : Présentation de l'application « MES » : .....	51
I.	Modélisation des algorithme de l'application .....	51
1.	Cas d'utilisation.....	51
2.	Les classes utilisées .....	52
3.	Traitement des contraintes du problème d'ordonnancement .....	52
4.	Choix entre l'algorithme de « Johnson » et « CDS » .....	53
5.	Manipulation de l'application par l'utilisateur .....	54

6.	Génération de la séquence optimale .....	55
7.	Choix de séquence optimale pour « Flow Shop » à m machines .....	56
II.	Code Python de l'application .....	57
1.	Méthode pour traiter le problème « Flow Shop ».....	57
2.	Méthode pour calculer les temps réels de fonctionnement TFR et d'arrêt TAR.....	62
3.	Méthode pour calculer les temps d'attente.....	63
4.	Méthode pour tracer le diagramme « Gant ».....	64
5.	Méthode pour identifier le makespan <i>Cmax</i> .....	68
6.	Méthode pour extraire le makespan et total tardiness .....	68
7.	Méthode pour calculer les combinaisons possibles.....	69
8.	Méthode pour définir les combinaisons où le makespan est égal à sa valeur minimale.....	69
9.	Méthode pour calculer les combinaisons qui donne la valeur minimale du total tardiness..	70
10.	Méthode pour calculer les combinaisons où TT = min(TTs) et Cmax = min(Cmaxs) .....	71
11.	Méthode pour calculer la sequence optimale dans le cas de deux machines.....	72
12.	Méthode pour calculer la sequence optimale dans le cas de plusieurs machines .....	74
13.	Fonction qui traite les problemes flowshop avec condition de blocage avec les temps de préparation.....	75
14.	Fonction qui traite les problemes flowshop avec condition de non attendre.....	79
15.	La fonction qui traite les problemes flowshop avec condition de non attendre .....	81
III.	Guide d'utilisation de l'application .....	83
1.	Présentation de l'application .....	83
2.	Le bouton Home .....	83
3.	Choix de contraintes.....	84
	En fonction du choix de l'utilisateur l'application passe à l'exécution des algorithmes permettant de résoudre le problème à traiter .....	84
4.	Insertion des données .....	84
6.	Diagramme de GANTT .....	87
IV.	Implémentation de l'application.....	90
1.	Flow Shop avec contrainte de préparation et de blocage .....	90
2.	Condition non attendre .....	92
	CONCLUSION .....	95

## Listes des figures

Figure 1: Ordonnancement d'un atelier .....	7
Figure 2: Planification des tâches dans le diagramme .....	8
Figure 3 : Ordonnancement des tâches sur deux machines .....	9
Figure 4 : Diagramme de GANTT du modèle « Flow Shop » à 2 machines .....	10
Figure 5 : Ordonnancement des tâches sur plusieurs machines.....	11
Figure 6 : Diagramme de GANTT du modèle « Flow Shop » à m machines .....	12
Figure 7: Diagramme de GANTT du modèle « Flow shop » avec préparation et sans blocage .....	14
Figure 8 : Atelier "Flow Shop" avec espace de stockage .....	15
Figure 9 : Placement du temps de blocage entre machines.....	15
Figure 10 : Diagramme de GANTT du modèle « Flow shop » avec blocage et sans préparation .....	16
Figure 11 : Diagramme de GANTT du modèle « Flow shop » avec blocage et préparation .....	18
Figure 12 : Diagramme cas d'utilisation.....	51
Figure 13 : Diagramme de classes.....	52
<b>Figure 14: Diagramme d'activité .....</b>	<b>53</b>
<b>Figure 15: Diagramme de séquence.....</b>	<b>54</b>
<b>Figure 16 : Diagramme de séquence .....</b>	<b>54</b>
<b>Figure 17 : Diagramme d'activité en fonction de contraintes.....</b>	<b>55</b>
<b>Figure 18 : Diagramme d'activité pour problème à plusieurs machines.....</b>	<b>56</b>

## Listes des tableaux

Tableau 1: Temps de processing .....	19
Tableau 2 : Temps de préparation de la machine 0.....	19
Tableau 3:Temps de préparation sur la machine 1 .....	19
Tableau 4:Temps de préparation sur la machine 2 .....	20
Tableau 5: Délais de chaque job .....	20
Tableau 6 : Temps de processing des deux machines fictives pou k optimale .....	20
Tableau 7:Séquence trouvée par l'algorithme de Jonhson .....	21
Tableau 8:Dates de fin de chaque job dans chaque machine PFSP .....	22
Tableau 9:dates de début de chaque job dans chaque machine PFSP .....	22
Tableau 10:Performances des machines PFSP .....	23
Tableau 11 : Temps d'attendre de chaque job entre chaque deux machines PFSP .....	23
Tableau 12 : Séquence trouvée par l'algorithme de Test de toutes les séquences.....	24
Tableau 13:dates de fin de chaque job dans chaque machine PFSP-SDST.....	25
Tableau 14:dates de début de chaque job dans chaque machine PFSP-SDST .....	25
Tableau 15 : Péformances des machines PFSP-SDST (machines non arrêtées en préparation) .....	25
Tableau 16 : Péformances des machines PFSP-SDST (machines arrêtées en préparation) .....	26
Tableau 17 : Temps d'attendre de chaque job entre chaque deux machines PFSP-SDST .....	27
Tableau 18 : Séquence trouvée par l'algorithme de Test de toutes les séquences.....	27
Tableau 19: dates de fin de chaque job avec son blocage dans chaque machine BFSP .....	30
Tableau 20 : dates de fin de chaque job dans chaque machine BFSP .....	30
Tableau 21 : dates de début de chaque job dans chaque machine BFSP .....	30
Tableau 22 : Péformances des machines BFSP.....	31
Tableau 23 : Temps de blocage de chaque job dans chaque machines BFSP.....	31
Tableau 24 : Séquence trouvée par l'algorithme de Test de toutes les séquences.....	32
Tableau 25 : dates de fin de chaque job avec son blocage dans chaque machine BFSP-SDST .....	36
Tableau 26 :dates de fin de chaque job dans chaque machine BFSP-SDST .....	36
Tableau 27 : dates de début de chaque job dans chaque machine BFSP-SDST .....	36
Tableau 28 : Péformances des machines BFSP-SDST (machines non arrêtées en préparation).....	36
Tableau 29 : Péformances des machines BFSP-SDST (machines arrêtées en préparation).....	37
Tableau 30 : Temps de blocage de chaque job dans chaque machines BFSP-SDST .....	38
Tableau 31 : Séquence trouvée par l'algorithme de Test de toutes les séquences.....	38
Tableau 32 : Table 32 : dates de fin de chaque job dans chaque machine NIPFSP .....	41
Tableau 33 : dates de début de chaque job dans chaque machine NIPFSP .....	41
Tableau 34 : Performances des machines NIPFSP .....	41
Tableau 35 : Temps d'attendre de chaque job entre chaque deux machines NIPFSP .....	42
Tableau 36 : Séquence trouvée par l'algorithme de Test de toutes les séquences.....	42
Tableau 37 : dates de fin de chaque job dans chaque machine NWPFSP .....	49
Tableau 38 : dates de début de chaque job dans chaque machine NWPFSP .....	49
Tableau 39 : Péformances des machines NWPFSP .....	49

# CHAPITRE I: LE MODELE “FLOW SHOP”:

## I. Introduction générale

### 1. Présentation du modèle « Flow Shop »

Le problème du flow shop, souvent appelé "flow shop scheduling problem", est un problème d'optimisation dans le domaine de la planification de la production. Il se pose lorsque plusieurs tâches (ou jobs) doivent être traitées à travers une série de machines ou de postes de travail, et chaque tâche suit une séquence spécifique de ces machines.

Ce problème a fait l'objet de plusieurs études récemment et on peut le considérer avec ou sans temps de préparation sinon avec ou sans temps de blocage.

L'objectif principal est d'ordonnancer ces tâches de manière à minimiser certains critères, tel que le temps total de production, les coûts, ou d'autres mesures de performance spécifiques au contexte.

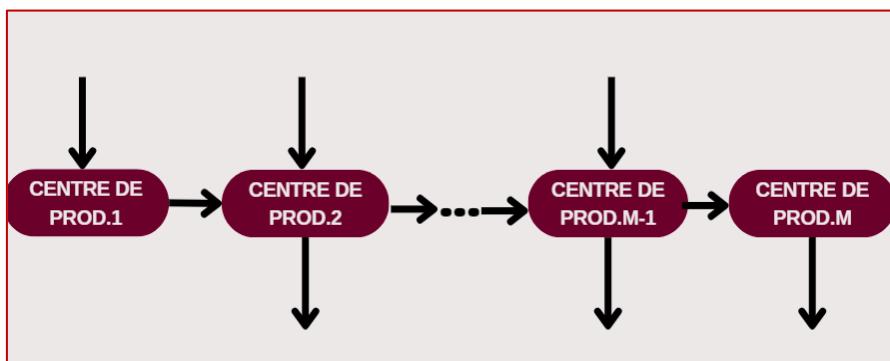


Figure 1: Ordonnancement d'un atelier

### 2. Notion générale

- $\sigma$  : une séquence se réfère à l'ordre spécifique dans lequel les jobs « j » doivent être exécutées à travers les différents postes de travail.
- $j$  : « jobs » Chaque job suit une séquence prédéterminée de machine  $M_i$  implantées en série
- $P_{i,\sigma_j}$ : Durée de traitement du job « j » sur la machine «  $M_i$  ».

- $C_{i,\sigma_j}$  : Date de fin du travail «  $\sigma_j$  » sur la machine «  $M_i$  ».
- $s_{k,\sigma_j}$  : Temps de préparation de la machine avant le début du traitement d'une tâche  $M_i$  spécifique.
- $D_{k,\sigma_j}$  : Date de fin du job « j » après blocage à la position de la machine «  $M_i$  ».
- **Condition de blocage:** C'est la condition de blocage peut survenir lorsque la progression d'un job est entravée en raison de certaines contraintes ou dépendances entre les tâches et les machines.
- **Condition no-wait** : fait référence à une situation où un job ou une tâche peut passer immédiatement à la phase suivante sans devoir attendre à un poste de travail

### 3. Diagramme de GANTT

C'est une technique de visualisation de l'utilisation de moyens productifs et/ou de l'avancement de l'exécution de tâches popularisée par Gantt et est classiquement utilisée en ordonnancement en atelier. Une tâche « j » est représentée sur un axe, habituellement horizontal, par un segment dont la longueur est, en principe, proportionnelle au temps d'exécution. Lorsque l'on étudie l'évolution de l'utilisation de plusieurs facteurs productifs des machines  $M_i$  l'utilisation de chaque facteur productif est portée sur un axe différent.

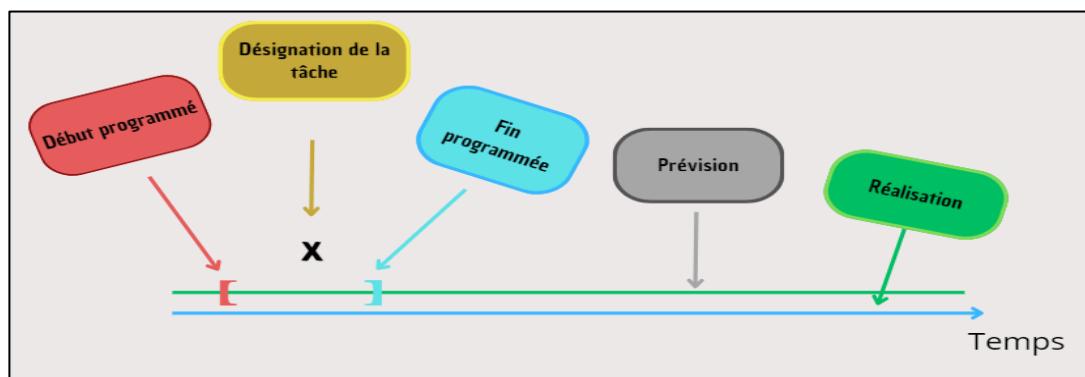


Figure 2: Planification des tâches dans le diagramme

## II. Modèle de Flow Shop sans contrainte de préparation

Une version intéressante de ce problème est le « flow shop » sans préparation dans laquelle les tâches ne sont pas autorisées à attendre entre deux ressources. Dans ce scénario, les machines peuvent passer d'une tâche à l'autre instantanément, sans nécessiter de temps d'ajustement ou de configuration. Cette simplification du modèle est souvent utilisée dans des contextes théoriques ou dans des situations où les temps de préparation sont négligeables par rapport aux temps de traitement réels des tâches. Cela permet de simplifier les calculs et les analyses associés à la planification et à l'ordonnancement des tâches dans le flow shop.

### 1. Modèle « Flow Shop » à 2 machines

#### **1.1. Description du modèle :**

Un ensemble de « job »  $J = \{J_1, J_2, \dots, J_n\}$  ayant la même gamme est exécutée sur deux machines  $M = \{M_1, M_2\}$  implantée en série.

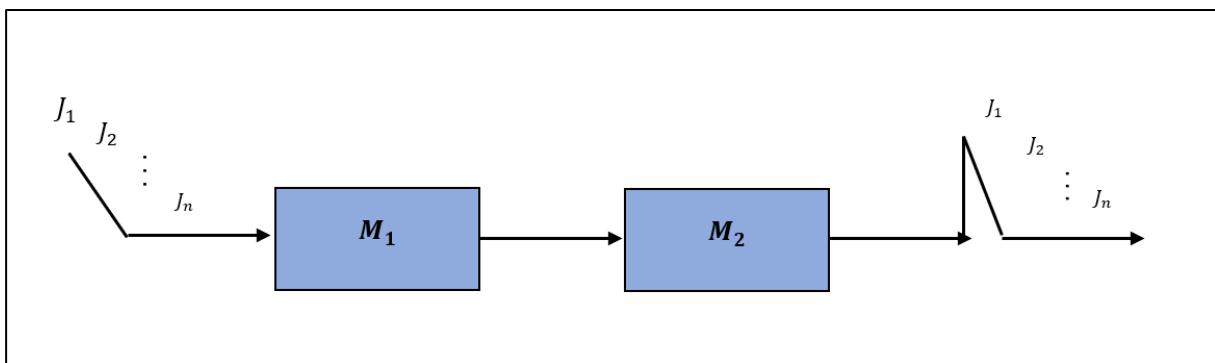


Figure 3 : Ordonnancement des tâches sur deux machines

#### **1.2. Modèle mathématique : algorithme de « JOHNSON »**

$$C_{1,\sigma_1} = p_{1,\sigma_1} \quad (1)$$

$$C_{i,\sigma_1} = C_{(i-1),\sigma_1} + p_{i,\sigma_1} \quad (2)$$

$$2 \leq i \leq m$$

$$C_{1,\sigma_j} = C_{1,\sigma_{(j-1)}} + p_{1,\sigma_j} \quad (3)$$

$$2 \leq j \leq n$$

$$C_{i,\sigma_j} = \max \{C_{(i-1),\sigma_j}, C_{i,\sigma_{(j-1)}}\} + p_{i,\sigma_1} \quad (4)$$

j = 2, ..., n et i = 2, ..., m

$$C_{max} = \max_{1 \leq j \leq n} \{C_{m,\sigma_1}\} \quad (5)$$

### **1.3. Diagramme de GANTT**

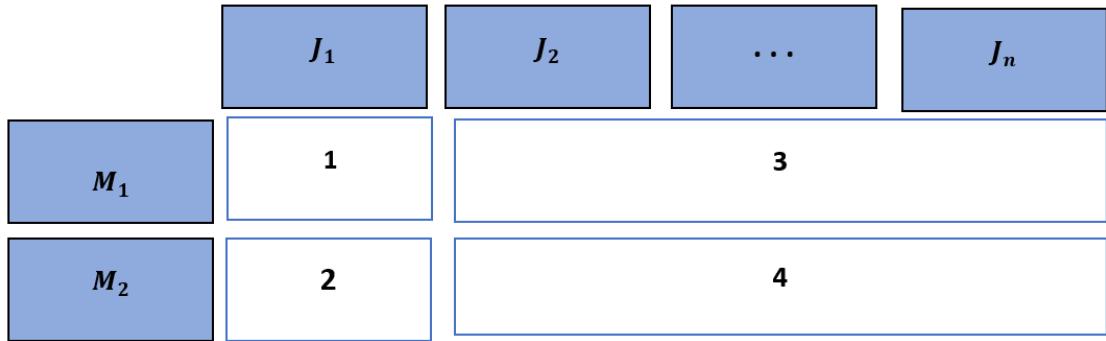


Figure 4 : Diagramme de GANTT du modèle « Flow Shop » à 2 machines

### **1.4. Performances de la machine**

- Taux de fonctionnement réel de la machine  $M_i$  :

$$TFR_i = \frac{\sum_1^n p_{i,\sigma_j}}{C_{max}}$$

- Taux d'arrêt réel de la machine  $M_i$  :

$$TAR_i = \frac{C_{max} - \sum_1^n p_{i,\sigma_j}}{C_{max}}$$

## 2. Modèle « Flow Shop » à m machines

### 2.1. Description du modèle :

Un ensemble de « job »  $J = \{J_1, J_2, \dots, J_n\}$  ayant la même gamme est exécutée sur deux machines  $M = \{M_1, M_2, \dots, M_m\}$  implantée en série.

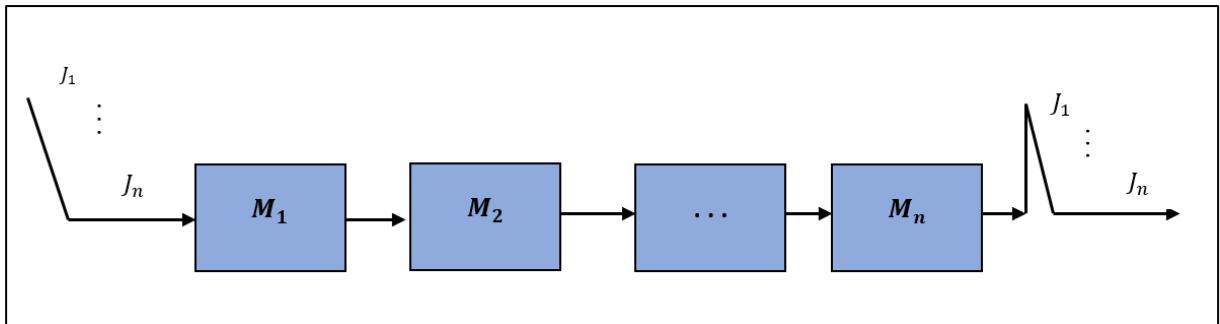


Figure 5 : Ordonnancement des tâches sur plusieurs machines

### 2.2. Modèle mathématique

#### 2.2.1. Algorithme « CDS » :

Cette procédure est développée par CAMPBELL, DUDEK et SMITH (CDS). Elle s'appuie sur la règle de Johnson. Elle consiste à générer  $(m-1)$  solutions en appliquant l'algorithme de Johnson sur deux machines fictives. La première regroupe les premières machines, la deuxième regroupe les  $k$  dernières machines,  $k$  varie de 1 à  $(m-1)$ . Les temps opératoires de chaque job  $j$  sur ces deux machines fictives sont la somme des temps opératoires sur les machines qu'ils regroupent

Nous pouvons les définir ainsi :

$$p_{1,j}^k = \sum_{i=1}^k p_{i,j}^k, p_{2,j}^k = \sum_{i=m-k+1}^m p_{i,j}^k : j = 1, \dots, n \quad (6)$$

On applique « CDS » selon les étapes suivantes :

Soit  $1 \leq k \leq m - 1$

- Etape 1 : On regroupe  $k$  les premières lignes du modèle .
- Etape 2 : On regroupe les  $k$  dernières ligne du modèle.
- Etape 3 : On forme l'ensemble  $U = \{j / p_{1,j} < p_{2,j}\}$
- Etape 4 : On forme l'ensemble  $V = \{j / p_{1,j} \geq p_{2,j}\}$
- Etape 5 : On classe  $U$  par ordre croissant sur la machine  $M_1$ .
- Etape 6 : On classe  $V$  par ordre des croissant sur la machine  $M_2$ .
- Etape 7 : La concaténation des sous séquences  $U$  et  $V$  tel que  $\sigma = [UV]$ .

### 2.2.2. Algorithme de « JOHNSON » :

La Formation de la séquence optimale grâce à l'algorithme « CDS » permet la transition d'un problème à plusieurs machines à un problème à deux machines ce qui va nous permettre d'appliquer l'algorithme de « JOHNSON » afin de résoudre le problème Flow Shop.

$$C_{1,\sigma_1} = p_{1,\sigma_1} \quad (7)$$

$$C_{i,\sigma_1} = C_{(i-1),\sigma_1} + p_{i,\sigma_1} \quad (8)$$

$$i = 2, \dots, m$$

$$C_{1,\sigma_j} = C_{1,\sigma_{(j-1)}} + p_{1,\sigma_j} \quad (9)$$

$$j=2, \dots, n$$

$$C_{i,\sigma_j} = \max \{C_{(i-1),\sigma_j}, C_{i,\sigma_{(j-1)}}\} + p_{i,\sigma_1} \quad (10)$$

$$j=2, \dots, n \quad i=2, \dots, m$$

$$C_{max} = \max_{1 \leq j \leq n} \{C_{m,\sigma_1}\} \quad (11)$$

### 2.3. Diagramme de GANTT :

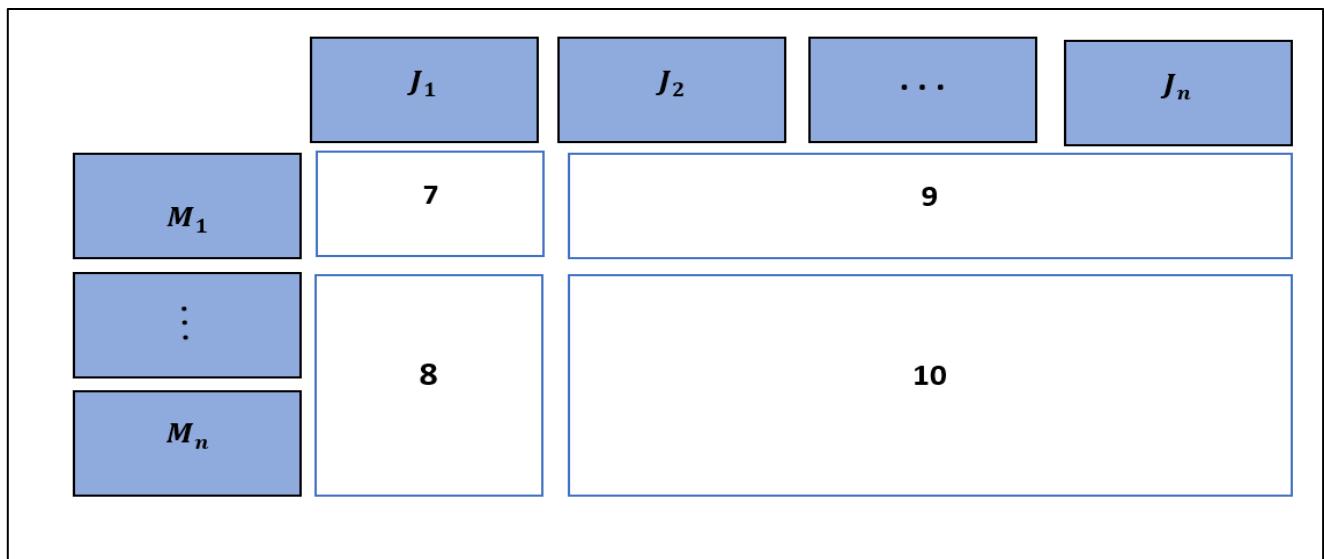


Figure 6 : Diagramme de GANTT du modèle « Flow Shop » à  $m$  machines

## **2.4. Performances de la machine**

- Taux de fonctionnement réel de la machine  $M_i$  :

$$TFR_i = \frac{\sum_1^n p_{i,\sigma_j}}{C_{max}}$$

- Taux d'arrêt réel de la machine  $M_i$  :

$$TAR_i = \frac{c_{max} - \sum_1^n p_{i,\sigma_j}}{c_{max}}$$

## III. Modèle de Flow Shop sous contrainte :

### 1. Modèle « Flow shop » avec préparation et sans blocage

#### **1.1. Description du modèle :**

Dans la version avec temps de préparation, ces derniers ne sont pas soumis aux contraintes sans attente. Les avantages potentiels d'un flow shop avec préparation comprennent une réduction des temps d'arrêt entre les tâches, une meilleure utilisation des machines, et une plus grande flexibilité dans l'ordonnancement des tâches. Les machines doivent souvent être ajustées ou reconfigurées pour passer d'une tâche à une autre, ce qui peut avoir un impact significatif sur la planification et l'efficacité globale du processus de production.

#### **1.2. Modèle mathématique**

$$C_{1,\sigma_1} = p_{1,\sigma_1} + s_{\sigma_1,\sigma_1,1} \quad (12)$$

$$C_{i,\sigma_1} = \max\{S_{\sigma_1,\sigma_1,i}, C_{(i-1),\sigma_1}\} + p_{\sigma_1,i} \quad (13)$$

$$1 \leq i \leq m$$

$$C_{1,\sigma_j} = C_{1,\sigma_{(j-1)}} + S_{\sigma_{(j-1)},\sigma_j,1} + p_{1,\sigma_j} \quad (14)$$

$$2 \leq j \leq n$$

$$C_{i,\sigma_j} = \max \left\{ C_{(i-1),\sigma_1}, C_{i,\sigma_{(j-1)}} + S_{\sigma_{(j-1)},\sigma_j,i} \right\} + p_{i,\sigma_j} \quad (15)$$

$$1 \leq i \leq m \text{ et } 2 \leq j \leq n$$

$$C_{max} = \max_{1 \leq j \leq m} \{C_{m,\sigma_j}\} \quad (16)$$

### 1.3. Diagramme de GANTT

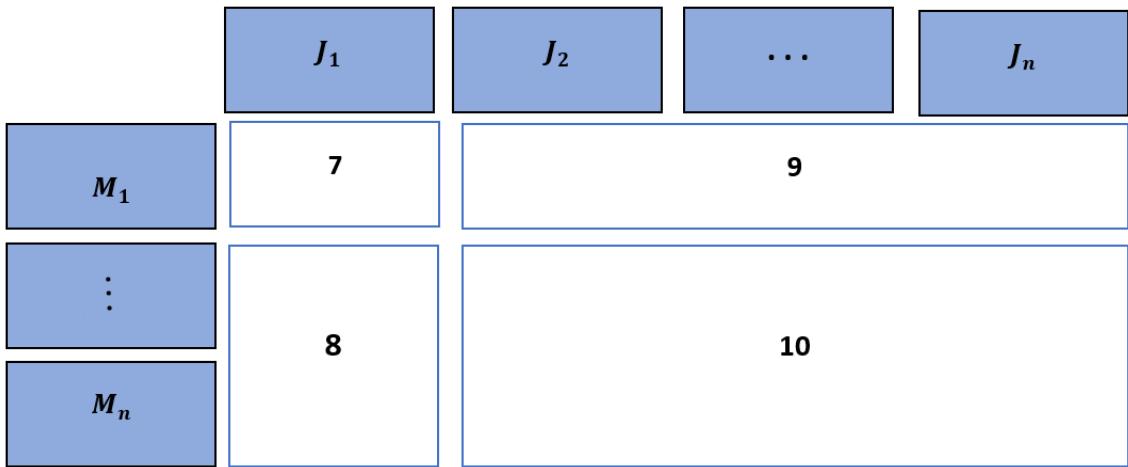


Figure 7: Diagramme de GANTT du modèle « Flow shop » avec préparation et sans blocage

### 1.4. Performances des machines :

- Taux de fonctionnement réel de la machine  $M_i$  :

$$TFR_i = \frac{\sum_1^n p_{i,\sigma_j}}{C_{max}}$$

- Taux d'arrêt de préparation de chaque machines  $M_i$

$$TAP_i = \frac{s_{\sigma_1,\sigma_1,i} + \sum_1^n s_{\sigma_{(j-1)},\sigma_j,i}}{C_{max}}$$

- Taux d'arrêt réel de chaque machines  $M_i$

$$TAP_i = \frac{C_{max} - (\sum_1^n p_{i,\sigma_j} + s_{\sigma_1,\sigma_1,i} + \sum_2^n s_{\sigma_{(j-1)},\sigma_{j,i}})}{C_{max}}$$

## 2. Modèle « Flow shop » avec blocage et sans préparation

### 2.1. Description du modèle :

Le blocage est une contrainte qui consiste à maintenir un « job » dans une machine malgré son achèvement. Cette contrainte se présente lorsque le problème de stockage est de capacité nulle.

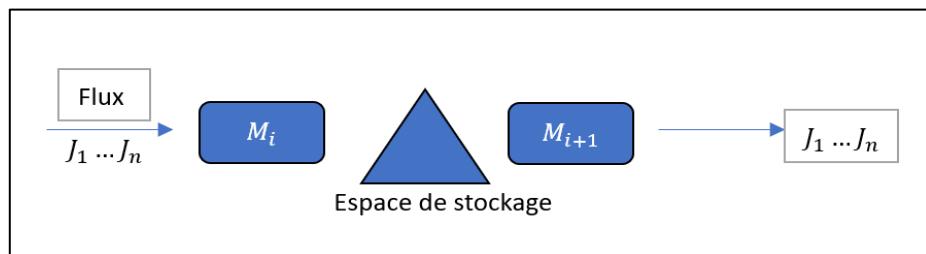


Figure 8 : Atelier "Flow Shop" avec espace de stockage

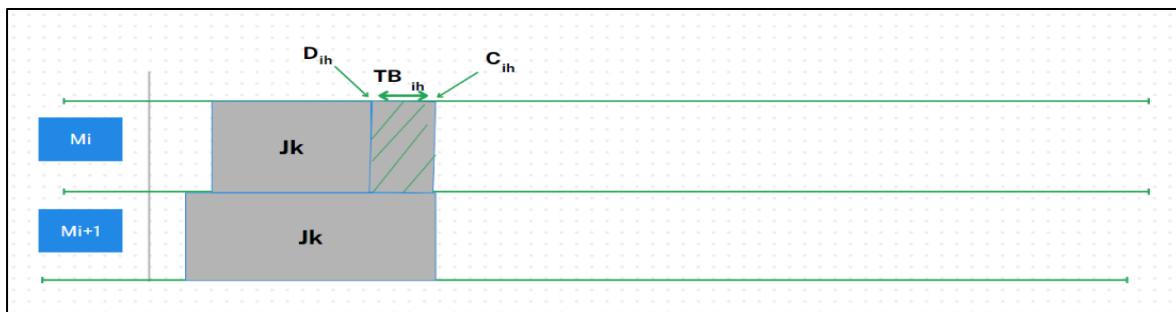


Figure 9 : Placement du temps de blocage entre machines

Avec :

$D_{ih}$  : Date de départ des jobs  $h$  de la machine  $M_i$

$C_{ih}$  : Date de fin du job  $h$  dans la machine  $M_i$

$TB_{ih}$  : Temps de blocage des jobs dans la machine  $M_i$

### 2.1. Modèle mathématique :

$$D_{0,\sigma_1} = 0 \quad (17)$$

$$D_{i,\sigma_1} = D_{(i-1),\sigma_1} + p_{i,\sigma_1} \quad (18)$$

$$i = 1, \dots, (m-1)$$

$$D_{0,\sigma_j} = D_{0,\sigma_{(j-1)}} \quad (19)$$

$$j = 2, \dots, n$$

$$D_{i,\sigma_j} = \max \{D_{(i-1),\sigma_j} + p_{i,\sigma_j}; D_{(i+1),\sigma_{(j-1)}}\} \quad (20)$$

$$i = 1, \dots, (m-1) \text{ et } j = 2, \dots, n$$

$$D_{m,\sigma_j} = D_{m-1,\sigma_j} + p_{m,\sigma_j} \quad (21)$$

$$C_{max} = \max_{1 \leq j \leq m} \{D_{m,\sigma_j}\} \quad (22)$$

## 2.2. Diagramme de GANTT :

Ordonnancement des tâches sur le diagramme de Gantt en fonction de leur ordre d'exécution selon les équations mathématiques. Cela dépendra de la séquence optimale déterminée.

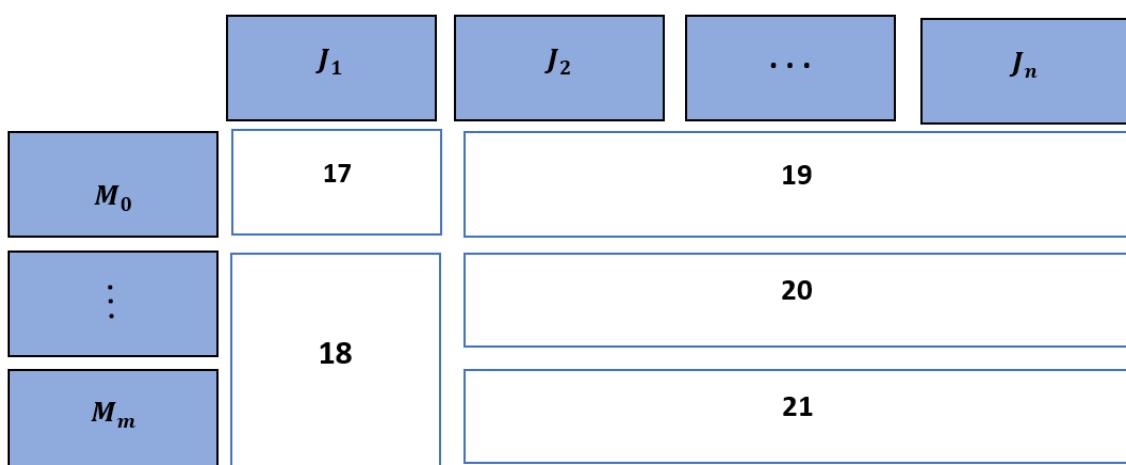


Figure 10 : Diagramme de GANTT du modèle « Flow shop » avec blocage et sans préparation

- Temps de blocage :

$$TB_{i,\sigma_j} = D_{i,\sigma_j} - (D_{(i-1),\sigma_j} + p_{i,\sigma_j})$$

- Total Flow Time (TFT):

$$TFT = \sum_1^n D_{m,\sigma_j}$$

- Total tardiness (TT) :

$$T_{m,\sigma_j} = \max \{D_{m,\sigma_j} - d_{\sigma_j}; 0\}$$

$$TT = \sum_1^n T_{m,\sigma_j}$$

### 3. Modèle « Flow shop » avec blocage et préparation

#### 3.1. Description du modèle

Ce modèle permet de déterminer les performances des machines dans des ateliers représentant le cas de flow shop sous la contrainte du blocage (Blocking flow shop scheduling problem with sequence dependent setup time :BFSP-SDST). Le blocage est une contrainte rencontrée lorsque l'espace de stockage est insuffisant devant les files d'attente des machines. Afin de coder le programme précédent nous aurons besoin d'une machine virtuelle initiale pour pouvoir démarrer notre code de la fonction objectif et un job virtuel , dans la séquence.

#### 3.2. Modèle mathématique

$$D_{0,\sigma_0} = 0 \quad (23)$$

$$D_{i,0} = 0 \quad (24)$$

$$i = 1, \dots, n$$

$$D_{0,\sigma_1} = D_{1,\sigma_0} + S_{\sigma_1,\sigma_1,1} \quad (25)$$

$$D_{i,\sigma_1} = \max \{D_{(i-1),\sigma_1} + P_{i,\sigma_1}; D_{i+1} + S_{\sigma_1,\sigma_1,i}\} \quad (26)$$

$$D_{0,\sigma_j} = D_{1,\sigma_{j-1}} + S_{\sigma_{j-1},\sigma_j,1} \quad (27)$$

$$j = 1, 2, \dots, n$$

$$D_{i,\sigma_j} = \max \{D_{(i-1),\sigma_j} + P_{i,\sigma_j}; D_{i+1,\sigma_{j-1}} + S_{\sigma_{j-1},\sigma_j,i+1}\} \quad (28)$$

$$\begin{aligned} 1 \leq j \leq n \\ 1 \leq i \leq m-1 \end{aligned}$$

$$D_{m,\sigma_j} = D_{(m-1),\sigma_j} + P_{m,\sigma_j} \quad (29)$$

$$C_{max} = \max_{1 \leq j \leq m} \{D_{m,\sigma_j}\} \quad (30)$$

### 3.3. Performances des machines :

- Temps de blocage :

$$TB_{i,\sigma_j} = D_{i,\sigma_j} - (D_{(i-1),\sigma_j} + p_{i,\sigma_j})$$

- Total Flow Time (TFT):

$$TFT = \sum_1^n D_{m,\sigma_j}$$

- Total tardiness (TT) :

$$T_{m,\sigma_j} = \max \{D_{m,\sigma_j} - d_{\sigma_j}; 0\}$$

$$TT = \sum_1^n T_{m,\sigma_j}$$

### 3.4. Diagramme de GANTT :

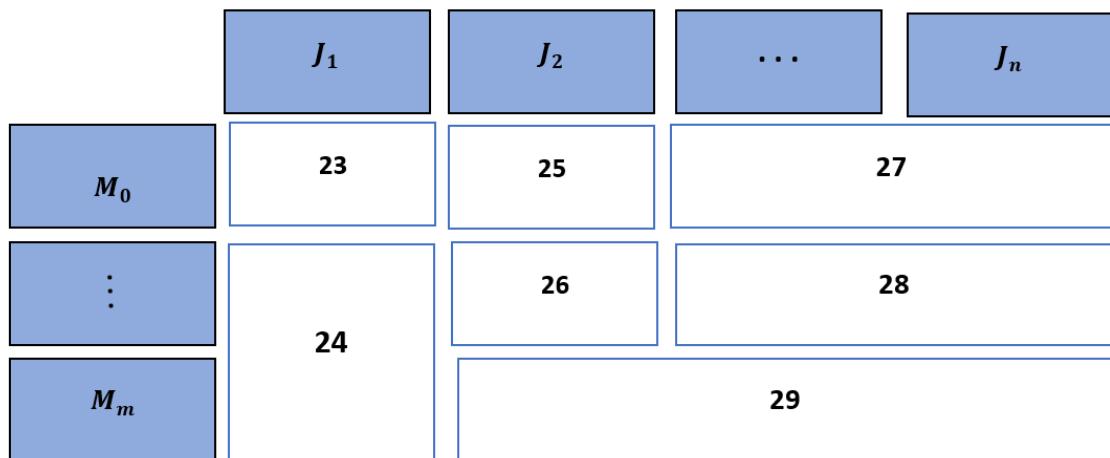


Figure 11 : Diagramme de GANTT du modèle « Flow shop » avec blocage et préparation

## CHAPITRE II : ETUDE THEORIQUE

### I. Data de Problème:

#### 1. Tableau de temps de processing

$P[i][S[j]]$ : matrice des temps de processing

*Tableau 1: Temps de processing*

P	J0	J1	J2	J3	J4	J5
M0	7	9	5	9	8	4
M1	3	8	4	5	4	9
M2	5	5	8	7	6	7

#### 2. Tableau de temps de préparation

$TP[i][S[j1]][S[j2]]$ : Tenseur des temps de préparation

TP0	J0	J1	J2	J3	J4	J5
J0	3	2	3	2	3	2
J1	2	2	2	3	2	3
J2	4	2	2	3	2	4
J3	3	3	3	5	4	3
J4	3	2	3	2	4	3
J5	2	4	3	4	3	2

*Tableau 2 : Temps de préparation de la machine 0*

TP1	J0	J1	J2	J3	J4	J5
J0	4	3	3	2	3	4
J1	2	3	2	4	2	3
J2	2	2	2	5	3	2
J3	3	2	3	3	5	4
J4	2	4	3	4	5	2
J5	3	2	3	4	5	3

*Tableau 3: Temps de préparation sur la machine 1*

TP2	J0	J1	J2	J3	J4	J5
J0	5	4	3	3	2	3
J1	3	2	2	4	2	4
J2	2	2	3	2	3	3
J3	3	2	3	3	2	3
J4	3	2	3	3	4	4
J5	2	2	3	2	3	3

Tableau 4: Temps de préparation sur la machine 2

### 3. Liste des délais de chaque Job

$d[S[j]]$ : List des délais de chaque job

	J0	J1	J2	J3	J4	J5
d	4	2	0	3	1	5

Tableau 5: Délais de chaque job

## II. PFSP (Permutation Flow Shop Scheduling Problem)

### 1. Séquence par algorithme de CDS

Après le teste de tous les k possibles de  $k = 1, \dots, 2^n$  on a trouvé que le meilleur k qui nous donne Cmax le plus optimale est  $k=2$

$P[i][S[j]]$ : matrice des temps de processing trouvée par CDS

P_CDS	J0	J1	J2	J3	J4	J5
M0	10	17	9	14	12	13
M1	8	13	12	12	10	16

Tableau 6 : Temps de processing des deux machines fictives pour k optimale

$S[j]$ : Séquence trouvée par l'algorithme de Jonhson

$U[j]$ : Jobs tel que  $P[0][S[j]] < P[1][S[j]]$ : [2, 5]

$V[j]$ : Jobs tel que  $P[0][S[j]] \geq P[1][S[j]]$ : [0, 1, 3, 4]

$U_{SPT}[j]$ :  $U[j]$  avec Jobs ordonancer par SPT(short processing Time), temps de processing croissant:[2, 5]

$V_{LPT}[j]$ :  $V[j]$  avec Jobs ordonancer par LPT(long processing Time), temps de processing décroissant:[1, 3, 4, 0]

Alors la séquence trouvée par Jonhson est la suivante

S	J2	J5	J1	J3	J4	J0
---	----	----	----	----	----	----

Tableau 7:Séquence trouvée par l'algortithme de Jonhson

## 2. Etapes de résolution de PFSP (Cmax/TFT...)

$C[i][S[j]]$ : matrice des dates de fin de chaque job dans chaque machine

Sans préparation et sans aucune condition

équation 1: pour  $i = 0$  et  $j = 0$

$$C[0][J2] = P[0][J2] = 5$$

équation 2: pour  $i = 0, \dots, 2$  et  $j = 0$

$$C[1][J2] = C[0][J2] + P[1][J2] = 5 + 4 = 9$$

$$C[2][J2] = C[1][J2] + P[2][J2] = 9 + 8 = 17$$

équation 3: pour  $i = 0$  et  $j = 0, \dots, 5$

$$C[0][J5] = C[0][J2] + P[0][J5] = 5 + 4 = 9$$

$$C[0][J1] = C[0][J5] + P[0][J1] = 9 + 9 = 18$$

$$C[0][J3] = C[0][J1] + P[0][J3] = 18 + 9 = 27$$

$$C[0][J4] = C[0][J3] + P[0][J4] = 27 + 8 = 35$$

$$C[0][J0] = C[0][J4] + P[0][J0] = 35 + 7 = 42$$

équation 4: pour  $i = 1, \dots, 2$  et  $j = 1, \dots, 5$

$$C[1][J5] = \max(C[0][J5], C[1][J2]) + P[1][J5] = \max(9, 9) 9 = 9 + 9 = 18$$

$$C[1][J1] = \max(C[0][J1], C[1][J5]) + P[1][J1] = \max(18, 18) 8 = 18 + 8 = 26$$

$$C[1][J3] = \max(C[0][J3], C[1][J1]) + P[1][J3] = \max(27, 26) 5 = 27 + 5 = 32$$

$$C[1][J4] = \max(C[0][J4], C[1][J3]) + P[1][J4] = \max(35, 32) 4 = 35 + 4 = 39$$

$$C[1][J0] = \max(C[0][J0], C[1][J4]) + P[1][J0] = \max(42, 39) 3 = 42 + 3 = 45$$

$$C[2][J5] = \max(C[1][J5], C[2][J2]) + P[2][J5] = \max(18, 17) 7 = 18 + 7 = 25$$

$$C[2][J1] = \max(C[1][J1], C[2][J5]) + P[2][J1] = \max(26, 25) 5 = 26 + 5 = 31$$

$$C[2][J3] = \max(C[1][J3], C[2][J1]) + P[2][J3] = \max(32, 31) 7 = 32 + 7 = 39$$

$$C[2][J4] = \max(C[1][J4], C[2][J3]) + P[2][J4] = \max(39, 39) 6 = 39 + 6 = 45$$

$$C[2][J0] = \max(C[1][J0], C[2][J4]) + P[2][J0] = \max(45, 45) 5 = 45 + 5 = 50$$

Total flow time TFT =  $C[2][J2] + C[2][J5] + C[2][J1] + C[2][J3] + C[2][J4] +$

$$C[2][J0] =$$

$$17 + 25 + 31 + 39 + 45 + 50 = 207.0$$

Tardiness T:

$$T[J2] = \max(C[2][J2] - d[J2], 0) = \max(17 - 0, 0) = 17$$

$$T[J5] = \max(C[2][J5] - d[J5], 0) = \max(25 - 5, 0) = 20$$

$$T[J1] = \max(C[2][J1] - d[J1], 0) = \max(31 - 2, 0) = 29$$

$$T[J3] = \max(C[2][J3] - d[J3], 0) = \max(39 - 3, 0) = 36$$

$$T[J4] = \max(C[2][J4] - d[J4], 0) = \max(45 - 1, 0) = 44$$

$$T[J0] = \max(C[2][J0] - d[J0], 0) = \max(50 - 4, 0) = 46$$

$$\text{Tardiness TT} = \sum T[S[j]] : j = 0, \dots, 5$$

$$TT = 192$$

### 3. Matrice des dates des fins et des débuts de chaque job sur chaque machine

C	J2	J5	J1	J3	J4	J0
M0	5	9	18	27	35	42
M1	9	18	26	32	39	45
M2	17	25	31	39	45	50

Tableau 8: Dates de fin de chaque job dans chaque machine PFSP

$F[i][S[j]]$ : dates de début de chaque job dans chaque machine PFSP pour trouver les dates de débuts de chaque job:  $F[i][S[j]] = C[i][S[j]] - P[i][S[j]]$

F	J2	J5	J1	J3	J4	J0
M0	0.0	5	9	18	27	35
M1	5	9	18	27	35	42
M2	9	18	26	32	39	45

Tableau 9: dates de début de chaque job dans chaque machine PFSP

### 4. Performances des machines

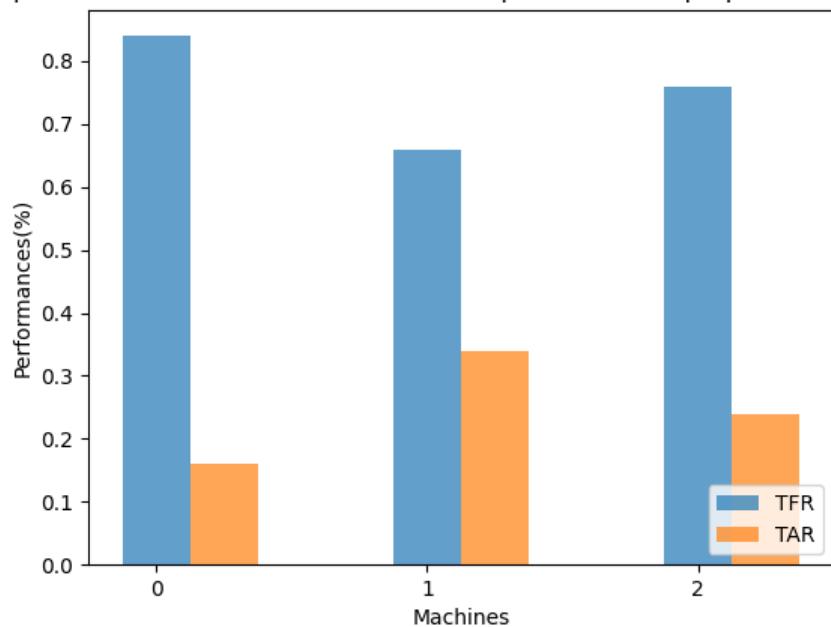
TFR: (teaux de fonctionnement réel)

TAR: (teaux d'arrêt réel)

	TFR	TAR
M0	84.0%	16.0%
M1	66.0%	34.0%
M2	76.0%	24.0%

Tableau 10: Performances des machines PFSP

Temps de fonctionnement réel(TFR) et Temps d'arrêt réel/préparation (TAR/TA)



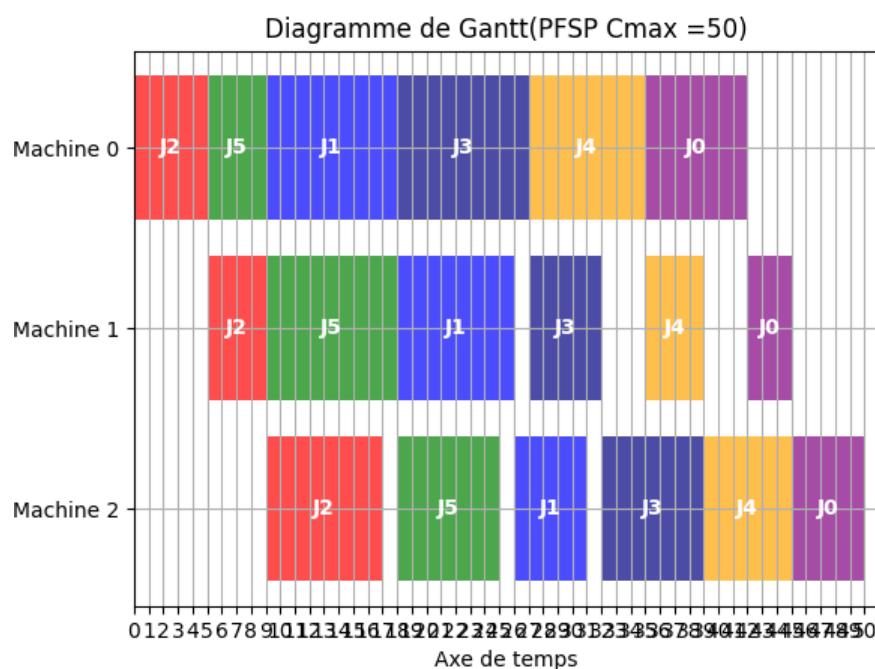
##### 5. Temps d'attente de chaque job entre chaque deux machines

TW : (waiting Time) temps d'attente de chaque job entre chaque deux machines

TW	J2	J5	J1	J3	J4	J0
M0/1	0	0	0	0	0	0
M1/2	0	0	0	0	0	0

Tableau 11 : Temps d'attendre de chaque job entre chaque deux machines PFSP

##### 6. Diagramme de Gantt,



### III. PFSP-SDST (Permutation Flow Shop Scheduling Problem with Sequence Depending Setup Time)

1. Séquence par Test de tous les permutations possibles avec min(Cmax) et min(TT)

S	J2	J4	J5	J0	J1	J3
---	----	----	----	----	----	----

Tableau 12 : Séquence trouvée par l'algorithme de Test de toutes les séquences

2. Etapes de résolution de PFSP-SDST (Cmax/TFT...)

$C[i][S[j]]$ : matrice des dates de fin de chaque job dans chaque machine

$M[i][S[j]][S[j]]$ : Tenseur des temps de préparation

Avec préparation et sans aucune condition

équation 1: pour  $i = 0$  et  $j = 0$

$$C[0][J2] = P[0][J2] + M[0][J2][J2] = 5+2 = 7$$

équation 2: pour  $i = 1, \dots, 2$  et  $j = 0$

$$C[1][J2] = \max(C[0][J2], M[1][J2][J2]) + P[1][J2] = \max(7, 2) + 4 = 11$$

$$C[2][J2] = \max(C[1][J2], M[2][J2][J2]) + P[2][J2] = \max(11, 2) + 8 = 19$$

équation 3: pour  $i = 0$  et  $j = 1, \dots, 5$

$$C[0][J4] = C[0][J2] + P[0][J2] + M[0][J2][J4] = 7+8+2 = 17$$

$$C[0][J5] = C[0][J4] + P[0][J4] + M[0][J4][J5] = 17+4+3 = 24$$

$$C[0][J0] = C[0][J5] + P[0][J5] + M[0][J5][J0] = 24+7+2 = 33$$

$$C[0][J1] = C[0][J0] + P[0][J0] + M[0][J0][J1] = 33+9+2 = 44$$

$$C[0][J3] = C[0][J1] + P[0][J1] + M[0][J1][J3] = 44+9+3 = 56$$

équation 4: pour  $i = 1, \dots, 2$ ,  $j = 1, \dots, 5$

$$C[1][J4] = \max(C[0][J4], C[1][J2] + M[1][J2][J4]) + P[1][J4] = \max(17, 11+3) + 4 = 21$$

$$C[1][J5] = \max(C[0][J5], C[1][J4] + M[1][J4][J5]) + P[1][J5] = \max(24, 21+2) + 4 = 33$$

$$C[1][J0] = \max(C[0][J0], C[1][J5] + M[1][J5][J0]) + P[1][J0] = \max(33, 33+3) + 4 = 39$$

$$C[1][J1] = \max(C[0][J1], C[1][J0] + M[1][J0][J1]) + P[1][J1] = \max(44, 39+3) + 4 = 52$$

$$C[1][J3] = \max(C[0][J3], C[1][J1] + M[1][J1][J3]) + P[1][J3] = \max(56, 52+4) + 4 = 61$$

$$C[2][J4] = \max(C[1][J4], C[2][J2] + M[2][J2][J4]) + P[2][J4] = \max(21, 19+3) + 8 = 28$$

$$C[2][J5] = \max(C[1][J5], C[2][J4] + M[2][J4][J5]) + P[2][J5] = \max(33, 28+4) + 8 = 40$$

$$C[2][J0] = \max(C[1][J0], C[2][J5] + M[2][J5][J0]) + P[2][J0] = \max(39, 40+2) + 8 = 47$$

$$C[2][J1] = \max(C[1][J1], C[2][J0] + M[2][J0][J1]) + P[2][J1] = \max(52, 47+4) + 8 = 57$$

$$C[2][J3] = \max(C[1][J3], C[2][J1] + M[2][J1][J3]) + P[2][J3] = \max(61, 57+4) + 8 = 68$$

Total flow time TFT = C[2][J2] + C[2][J4] + C[2][J5] + C[2][J0] + C[2][J1] + C[2][J3] =

$$19 + 28 + 40 + 47 + 57 + 68 = 259.0$$

Tardiness T:

$$T[J2] = \max( C[2][J2] - d[J2], 0 ) = \max( 19 - 0, 0 ) = 19$$

$$T[J4] = \max( C[2][J4] - d[J4], 0 ) = \max( 28 - 1, 0 ) = 27$$

$$T[J5] = \max( C[2][J5] - d[J5], 0 ) = \max( 40 - 5, 0 ) = 35$$

$$T[J0] = \max( C[2][J0] - d[J0], 0 ) = \max( 47 - 4, 0 ) = 43$$

$$T[J1] = \max( C[2][J1] - d[J1], 0 ) = \max( 57 - 2, 0 ) = 55$$

$$T[J3] = \max( C[2][J3] - d[J3], 0 ) = \max( 68 - 3, 0 ) = 65$$

Tardiness TT =  $\sum T[S[j]] : j = 0, \dots, 5$

$$TT = 244$$

### 3. Matrice des dates des fins et des débuts de chaque job sur chaque machine

C	J2	J4	J5	J0	J1	J3
M0	7	17	24	33	44	56
M1	11	21	33	39	52	61
M2	19	28	40	47	57	68

Tableau 13:dates de fin de chaque job dans chaque machine PFSP-SDST

F[i][S[j]]:dates de début de chaque job dans chaque machine PFSP-SDST

pour trouver les dates de débuts de chaque job: F[i][S[j]] = C[i][S[j]] - P[i][S[j]]

F	J2	J4	J5	J0	J1	J3
M0	2	9	20	26	35	47
M1	7	17	24	36	44	56
M2	11	22	33	42	52	61

Tableau 14:dates de début de chaque job dans chaque machine PFSP-SDST

### 4. Performances des machines (machines non arrêtées en préparation)

TFR: (teaux de fonctionnement réel)

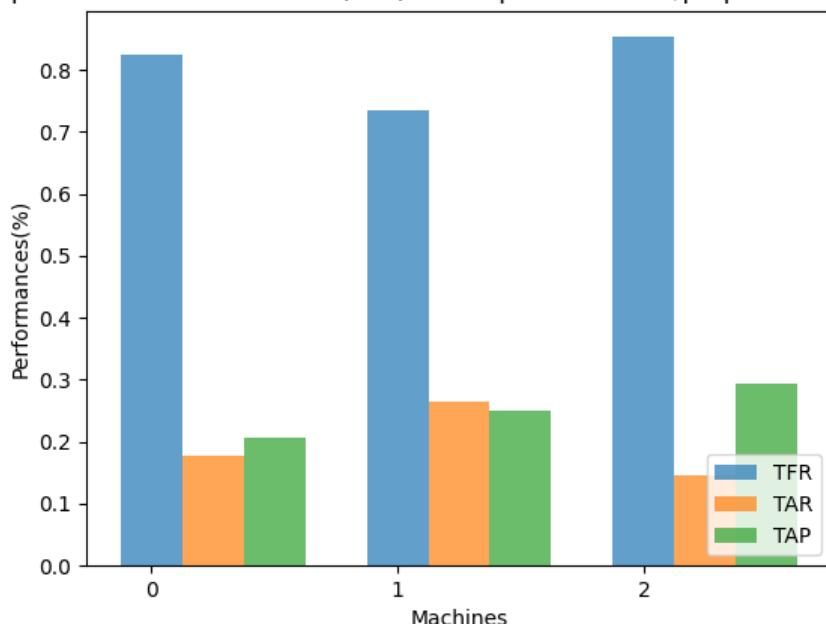
TAR: (teaux d'arrêt réel)

TAP: (teaux d'arrêt de préparation)

	TFR	TAR	TAP
M0	82.35%	17.65%	20.59%
M1	73.53%	26.47%	25.0%
M2	85.29%	14.71%	29.41%

Tableau 15 : Pérformances des machines PFSP-SDST (machines non arrêtées en préparation)

Temps de fonctionnement réel(TFR) et Temps d'arrêt réel/préparation (TAR/TA)



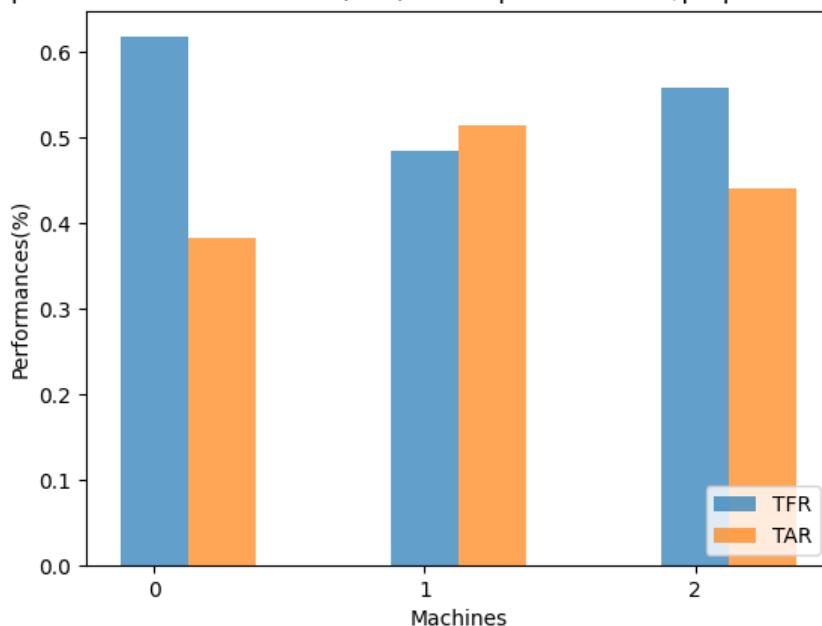
### 5. Performances des machines (machines arrêtées en préparation)

	TFR	TAR
M0	82.35%	17.65%
M1	73.53%	26.47%
M2	85.29%	14.71%

Tableau 16 : Performances des machines PFSP-SDST (machines arrêtées en préparation)

### 6. Temps d'attendre de chaque job entre chaque deux machines

Temps de fonctionnement réel(TFR) et Temps d'arrêt réel/préparation (TAR/TA)

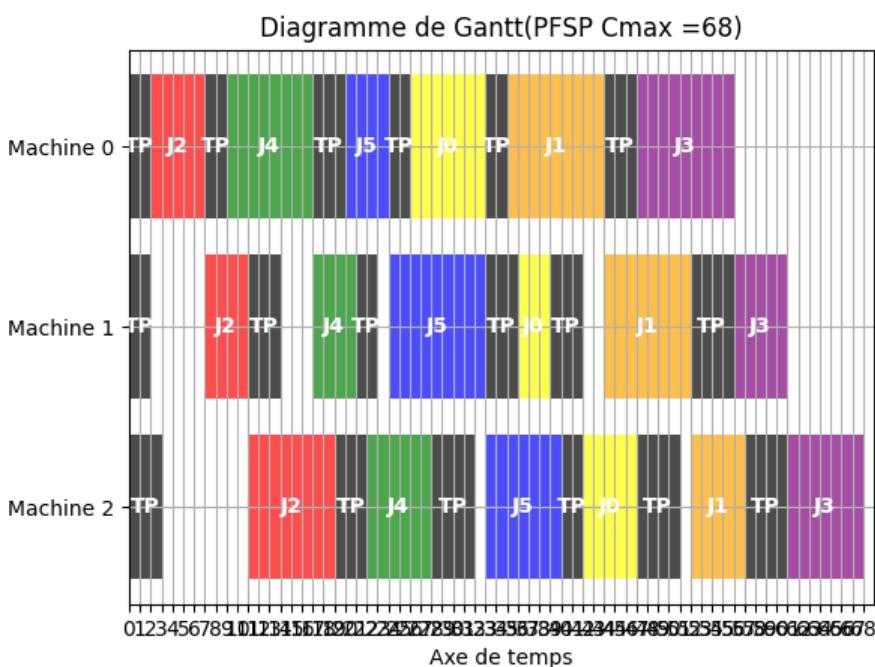


TW: (waiting Time) temps d'attente de chaque job entre chaque deux machines

TW	J2	J4	J5	J0	J1	J3
M0/1	0	0	0	3	0	0
M1/2	0	1	0	3	0	0

Tableau 17 : Temps d'attendre de chaque job entre chaque deux machines PFSP-SDST

## 7. Diagramme de Gantt,



#### IV. BFSP (Blocking Flow Shop Scheduling Problem)

1. Séquence par Test de tous les permutations possibles avec  $\min(C_{\max})$  et  $\min(TT)$

S	J2	J5	J1	J4	J0	J3
---	----	----	----	----	----	----

Tableau 18 : Séquence trouvée par l'algotritisme de Test de toutes les séquences

- ## 2. Etapes de résolution de BFSP (Cmax/TFT...)

$D[i][S[j]]$ : matrice des dates de fin de chaque job avec son blocage dans chaque machine, aussi début des jobs dans la machine précédente  
 pour la matrice D l'indexation des machines réel est de 1 à 3

Sans préparation et avec blocage

on Ajout une machine virtuel pour calculer la matrice D

indéxation des machines réel de 1 à3

équation 1: pour  $i = 0$  et  $j = 0$

$$D[0][J2] = 0$$

équation 2: pour  $i = 1, \dots, 2$  et  $j = 0$

$$D[1][J2] = D[0][J2] + P[0][J2] = 0.0 + 5 = 5.0$$

$$D[2][J2] = D[1][J2] + P[1][J2] = 5.0 + 4 = 9.0$$

équation 5: pour  $i = 2$  et  $j = 0$

$$D[3][J2] = D[2][J2] + P[2][J2] = 9.0 + 8 = 17.0$$

équation 3: pour  $i = 0$  et  $j = 1$

$$D[0][J5] = D[1][J2] = 5.0$$

équation 4: pour  $i = 1$  et  $j = 1$

$$D[1][J5] = \max(D[0][J5] + P[0][J9.0]$$

équation 4: pour  $i = 2$  et  $j = 1$

$$D[2][J5] = \max(D[1][J5] + P[1][J18.0]$$

équation 5: pour  $i = 2$  et  $j = 1$

$$D[3][J5] = D[2][J5] + P[2][J5] = 18.0 + 7 = 25.0$$

équation 3: pour  $i = 0$  et  $j = 2$

$$D[0][J1] = D[1][J5] = 9.0$$

équation 4: pour  $i = 1$  et  $j = 2$

$$D[1][J1] = \max(D[0][J1] + P[0][J18.0]$$

équation 4: pour  $i = 2$  et  $j = 2$

$$D[2][J1] = \max(D[1][J1] + P[1][J26.0]$$

équation 5: pour  $i = 2$  et  $j = 2$

$$D[3][J1] = D[2][J1] + P[2][J1] = 26.0 + 5 = 31.0$$

équation 3: pour  $i = 0$  et  $j = 3$

$$D[0][J4] = D[1][J1] = 18.0$$

équation 4: pour  $i = 1$  et  $j = 3$

$$D[1][J4] = \max(D[0][J4] + P[0][J26.0]$$

équation 4: pour  $i = 2$  et  $j = 3$

$$D[2][J4] = \max(D[1][J4] + P[1][J31.0]$$

équation 5: pour  $i = 2$  et  $j = 3$

$$D[3][J4] = D[2][J4] + P[2][J4] = 31.0 + 6 = 37.0$$

équation 3: pour  $i = 0$  et  $j = 4$

$$D[0][J0] = D[1][J4] = 26.0$$

équation 4: pour  $i = 1$  et  $j = 4$

$$D[1][J0] = \max( D[0][J0] + P[0][J33.0]$$

équation 4: pour  $i = 2$  et  $j = 4$

$$D[2][J0] = \max( D[1][J0] + P[1][J37.0]$$

équation 5: pour  $i = 2$  et  $j = 4$

$$D[3][J0] = D[2][J0] + P[2][J0] = 37.0 + 5 = 42.0$$

équation 3: pour  $i = 0$  et  $j = 5$

$$D[0][J3] = D[1][J0] = 33.0$$

équation 4: pour  $i = 1$  et  $j = 5$

$$D[1][J3] = \max( D[0][J3] + P[0][J42.0]$$

équation 4: pour  $i = 2$  et  $j = 5$

$$D[2][J3] = \max( D[1][J3] + P[1][J47.0]$$

équation 5: pour  $i = 2$  et  $j = 5$

$$D[3][J3] = D[2][J3] + P[2][J3] = 47.0 + 7 = 54.0$$

Calcul des temps de blocage

$$TB[0][J2] = D[1][J2] - D[0][J2] - P[0][J2] = 5.0 - 0.0 - 5 = 0.0$$

$$TB[0][J5] = D[1][J5] - D[0][J5] - P[0][J5] = 9.0 - 5.0 - 4 = 0.0$$

$$TB[0][J1] = D[1][J1] - D[0][J1] - P[0][J1] = 18.0 - 9.0 - 9 = 0.0$$

$$TB[0][J4] = D[1][J4] - D[0][J4] - P[0][J4] = 26.0 - 18.0 - 8 = 0.0$$

$$TB[0][J0] = D[1][J0] - D[0][J0] - P[0][J0] = 33.0 - 26.0 - 7 = 0.0$$

$$TB[0][J3] = D[1][J3] - D[0][J3] - P[0][J3] = 42.0 - 33.0 - 9 = 0.0$$

$$TB[1][J2] = D[2][J2] - D[1][J2] - P[1][J2] = 9.0 - 5.0 - 4 = 0.0$$

$$TB[1][J5] = D[2][J5] - D[1][J5] - P[1][J5] = 18.0 - 9.0 - 9 = 0.0$$

$$TB[1][J1] = D[2][J1] - D[1][J1] - P[1][J1] = 26.0 - 18.0 - 8 = 0.0$$

$$TB[1][J4] = D[2][J4] - D[1][J4] - P[1][J4] = 31.0 - 26.0 - 4 = 1.0$$

$$TB[1][J0] = D[2][J0] - D[1][J0] - P[1][J0] = 37.0 - 33.0 - 3 = 1.0$$

$$TB[1][J3] = D[2][J3] - D[1][J3] - P[1][J3] = 47.0 - 42.0 - 5 = 0.0$$

$$TB[2][J2] = D[3][J2] - D[2][J2] - P[2][J2] = 17.0 - 9.0 - 8 = 0.0$$

$$TB[2][J5] = D[3][J5] - D[2][J5] - P[2][J5] = 25.0 - 18.0 - 7 = 0.0$$

$$TB[2][J1] = D[3][J1] - D[2][J1] - P[2][J1] = 31.0 - 26.0 - 5 = 0.0$$

$$TB[2][J4] = D[3][J4] - D[2][J4] - P[2][J4] = 37.0 - 31.0 - 6 = 0.0$$

$$TB[2][J0] = D[3][J0] - D[2][J0] - P[2][J0] = 42.0 - 37.0 - 5 = 0.0$$

$$TB[2][J3] = D[3][J3] - D[2][J3] - P[2][J3] = 54.0 - 47.0 - 7 = 0.0$$

Total flow time TFT =  $D[3][J2] + D[3][J5] + D[3][J1] + D[3][J4] + D[3][J0] +$

$$D[3][J3] =$$

$$17.0 + 25.0 + 31.0 + 37.0 + 42.0 + 54.0 = 206.0$$

Tardiness T:

$$T[J2] = \max(D[3][J2] - d[J2], 0) = \max(17.0 - 0, 0) = 17.0$$

$$T[J5] = \max(D[3][J5] - d[J5], 0) = \max(25.0 - 5, 0) = 20.0$$

$$T[J1] = \max(D[3][J1] - d[J1], 0) = \max(31.0 - 2, 0) = 29.0$$

$$T[J4] = \max(D[3][J4] - d[J4], 0) = \max(37.0 - 1, 0) = 36.0$$

$$T[J0] = \max(D[3][J0] - d[J0], 0) = \max(42.0 - 4, 0) = 38.0$$

$$T[J3] = \max(D[3][J3] - d[J3], 0) = \max(54.0 - 3, 0) = 51.0$$

Tardiness TT =  $\sum T[S[j]] : j = 0, \dots, 5$

$$TT = 191.0$$

### 3. Matrices D,C,F

D	J2	J5	J1	J4	J0	J3
M0	0.0	5.0	9.0	18.0	26.0	33.0
M1	5.0	9.0	18.0	26.0	33.0	42.0
M2	9.0	18.0	26.0	31.0	37.0	47.0
M3	17.0	25.0	31.0	37.0	42.0	54.0

Tableau 19: dates de fin de chaque job avec son blocage dans chaque machine BFSP

C[i][S[j]]:dates de fin de chaque job dans chaque machine BFSP

C	J2	J5	J1	J4	J0	J3
M0	5.0	9.0	18.0	26.0	33.0	42.0
M1	9.0	18.0	26.0	30.0	36.0	47.0
M2	17.0	25.0	31.0	37.0	42.0	54.0

Tableau 20 : dates de fin de chaque job dans chaque machine BFSP

F[i][S[j]]:dates de début de chaque job dans chaque machine BFSP

pour trouver les dates de débuts de chaque job:  $F[i][S[j]] = C[i][S[j]] - P[i][S[j]]$

F	J2	J5	J1	J4	J0	J3
M0	0.0	5.0	9.0	18.0	26.0	33.0
M1	5.0	9.0	18.0	26.0	33.0	42.0
M2	9.0	18.0	26.0	31.0	37.0	47.0

Tableau 21 : dates de début de chaque job dans chaque machine BFSP

#### 4. Performances des machines

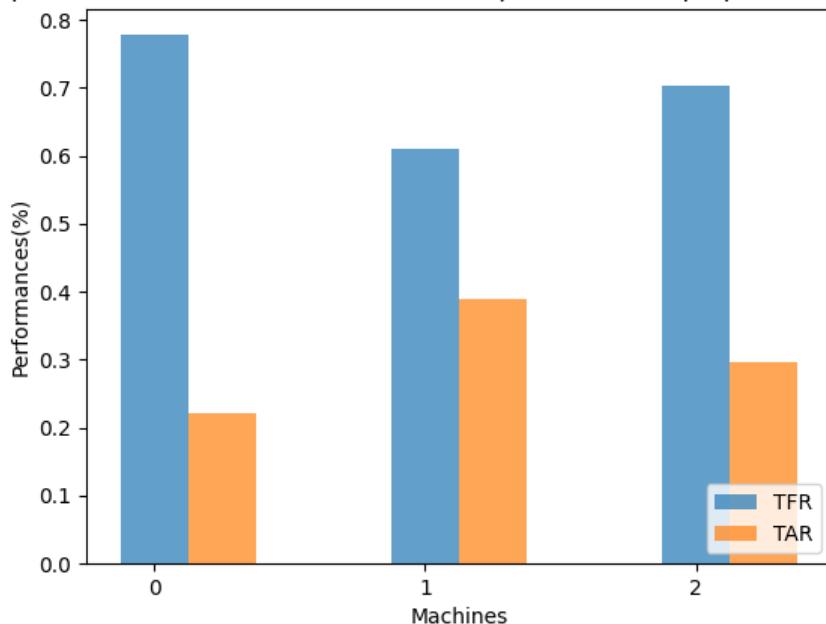
TFR: (teaux de fonctionnement réel)

TAR: (teaux d'arrêt réel)

	TFR	TAR
M0	77.78%	22.22%
M1	61.11%	38.89%
M2	70.37%	29.63%

Tableau 22 : Performances des machines BFSP

Temps de fonctionnement réel(TFR) et Temps d'arrêt réel/préparation (TAR/TA)



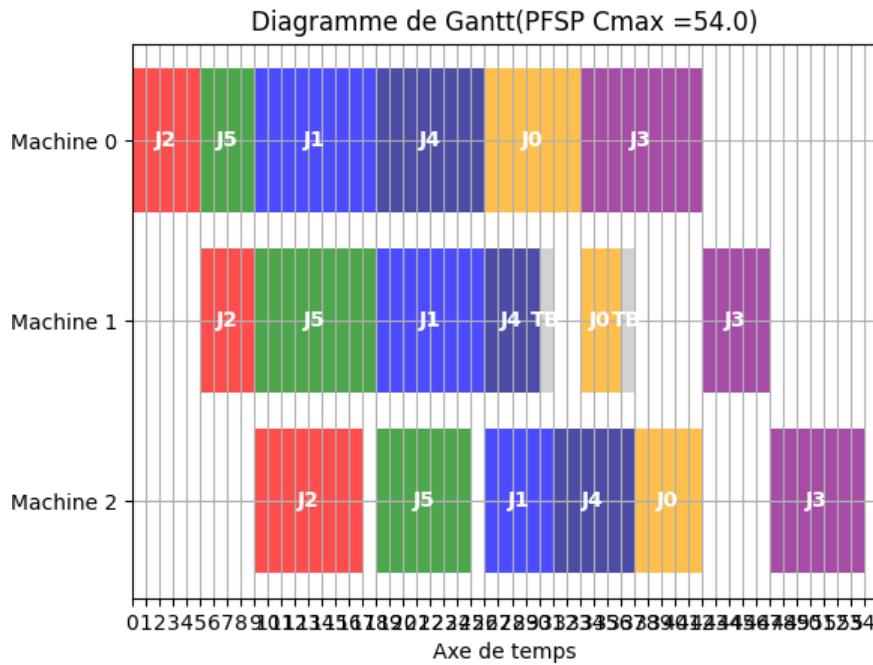
#### 5. Temps de blocage de chaque job dans chaque machines

TB: (Blocking Time) temps de blocage de chaque job dans chaque machines

TB	J2	J5	J1	J4	J0	J3
M0	0.0	0.0	0.0	0.0	0.0	0.0
M1	0.0	0.0	0.0	1.0	1.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0

Tableau 23 : Temps de blocage de chaque job dans chaque machines BFSP

## 6. Diagramme de Gantt,



## V. BFSP-SDST (Blocking Flow Shop Scheduling Problem with Sequence Depending Setup Time)

1. Séquence par Test de tous les permutations possibles avec min(Cmax) et min(TT)

S	J2	J4	J5	J3	J0	J1
---	----	----	----	----	----	----

Tableau 24 : Séquence trouvée par l'algorithme de Test de toutes les séquences

### 2. Etapes de résolution de BFSP-SDST (Cmax/TFT...)

D[i][S[j]]: matrice des dates de fin de chaque job avec son blocage dans chaque machine, aussi début des jobs dans la machine précédente  
 pour la matrice D l'indexation des machines réel est de 1 à 3 et l'indexation des jobs réel de 1 à 6

Avec préparation et avec blocage

on Ajout une machine virtuel et un job virtuel pour calculer la matrice D  
 indéxation des machines réel de 1 à 3 ; et des jobs de 1 à 6  
 équation 1: pour  $i = 0, \dots, 3$  et  $j = 0$

$$D[i][J0] = 0$$

équation 2: pour  $i = 0$  et  $j = 1$

$$D[0][J3] = D[1][J0] + M[1][J3][J3] = 0.0 + 2 = 2.0$$

équation 4: pour  $i = 1$  et  $j = 1$

$$\begin{aligned} D[1][J3] &= \max( D[0][J3] + P[1][J3] , D[2][J0] + M[1][J3][J3] ) = \max( 2.0 + 5.0 , 0.0 + 2 ) \\ &= 7.0 \end{aligned}$$

équation 4: pour  $i = 2$  et  $j = 1$

$$\begin{aligned} D[2][J3] &= \max( D[1][J3] + P[2][J3] , D[3][J0] + M[2][J3][J3] ) = \max( 7.0 + 4.0 , 0.0 + 3 ) \\ &= 11.0 \end{aligned}$$

équation 6: pour  $i = 3$  et  $j = 1$

$$D[3][J3] = D[2][J3] + P[3][J3] = 11.0 + 8.0 = 19.0$$

équation 3: pour  $i = 0$  et  $j = 2$

$$D[0][J5] = D[1][J3] + M[1][J3][J5] = 7.0 + 2 = 9.0$$

équation 5: pour  $i = 1$  et  $j = 2$

$$\begin{aligned} D[1][J5] &= \max( D[0][J5] + P[1][J5] , D[2][3] + M[1][3][5] ) = \max( 9.0 + 8.0 , 11.0 + 3 ) \\ &= 17.0 \end{aligned}$$

équation 5: pour  $i = 2$  et  $j = 2$

$$D[2][J5] = \max( D[1][J5] + P[2][J5] , D[3][3] + M[2][3][5] ) = \max( 17.0 + 4.0 , 19.0 + 3 )$$

équation 6: pour  $i = 3$  et  $j = 2$

$$D[3][J5] = D[2][J5] + P[3][J5] = 22.0 + 6.0 = 28.0$$

équation 3: pour  $i = 0$  et  $j = 3$

$$D[0][J6] = D[1][J5] + M[1][J5][J6] = 17.0 + 3 = 20.0$$

équation 5: pour  $i = 1$  et  $j = 3$

$$\begin{aligned} D[1][J6] &= \max( D[0][J6] + P[1][J6] , D[2][5] + M[1][5][6] ) = \max( 20.0 + 4.0 , 22.0 + 2 ) \\ &= 24.0 \end{aligned}$$

équation 5: pour  $i = 2$  et  $j = 3$

$$D[2][J6] = \max( D[1][J6] + P[2][J6] , D[3][5] + M[2][5][6] ) = \max( 24.0 + 9.0 , 28.0 + 4 )$$

équation 6: pour  $i = 3$  et  $j = 3$

$$D[3][J6] = D[2][J6] + P[3][J6] = 33.0 + 7.0 = 40.0$$

équation 3: pour  $i = 0$  et  $j = 4$

$$D[0][J4] = D[1][J6] + M[1][J6][J4] = 24.0 + 4 = 28.0$$

équation 5: pour  $i = 1$  et  $j = 4$

$$\begin{aligned} D[1][J4] &= \max( D[0][J4] + P[1][J4] , D[2][6] + M[1][6][4] ) = \max( 28.0 + 9.0 , 33.0 + 4 ) \\ &= 37.0 \end{aligned}$$

équation 5: pour  $i = 2$  et  $j = 4$

$$D[2][J4] = \max( D[1][J4] + P[2][J4] , D[3][6] + M[2][6][4] ) = \max( 37.0 + 5.0 , 40.0 + 2 )$$

équation 6: pour  $i = 3$  et  $j = 4$

$$D[3][J4] = D[2][J4] + P[3][J4] = 42.0 + 7.0 = 49.0$$

équation 3: pour  $i = 0$  et  $j = 5$

$$D[0][J1] = D[1][J4] + M[1][J4][J1] = 37.0 + 3 = 40.0$$

équation 5: pour  $i = 1$  et  $j = 5$

$$D[1][J1] = \max( D[0][J1] + P[1][J1] , D[2][4] + M[1][4][1] ) = \max( 40.0 + 7.0 , 42.0 + 3 )$$

= 47.0équation 5: pour  $i = 2$  et  $j = 5$

$$D[2][J1] = \max( D[1][J1] + P[2][J1] , D[3][4] + M[2][4][1] ) = \max( 47.0 + 3.0 , 49.0 + 3 )$$

= 52.0équation 6: pour  $i = 3$  et  $j = 5$

$$D[3][J1] = D[2][J1] + P[3][J1] = 52.0 + 5.0 = 57.0$$

équation 3: pour  $i = 0$  et  $j = 6$

$$D[0][J2] = D[1][J1] + M[1][J1][J2] = 47.0 + 2 = 49.0$$

équation 5: pour  $i = 1$  et  $j = 6$

$$D[1][J2] = \max( D[0][J2] + P[1][J2] , D[2][1] + M[1][1][2] ) = \max( 49.0 + 9.0 , 52.0 + 3 )$$

= 58.0équation 5: pour  $i = 2$  et  $j = 6$

$$D[2][J2] = \max( D[1][J2] + P[2][J2] , D[3][1] + M[2][1][2] ) = \max( 58.0 + 8.0 , 57.0 + 4 )$$

= 66.0équation 6: pour  $i = 3$  et  $j = 6$

$$D[3][J2] = D[2][J2] + P[3][J2] = 66.0 + 5.0 = 71.0$$

Indexation normale pour TB et C

Calcul de la matrice C

$$C[0][J2] = D[1][J0] + P[0][J2] = 2.0 + 5.0 = 7.0$$

$$C[0][J4] = D[1][J3] + P[0][J4] = 9.0 + 8.0 = 17.0$$

$$C[0][J5] = D[1][J5] + P[0][J5] = 20.0 + 4.0 = 24.0$$

$$C[0][J3] = D[1][J6] + P[0][J3] = 28.0 + 9.0 = 37.0$$

$$C[0][J0] = D[1][J4] + P[0][J0] = 40.0 + 7.0 = 47.0$$

$$C[0][J1] = D[1][J1] + P[0][J1] = 49.0 + 9.0 = 58.0$$

$$C[1][J2] = D[2][J0] + P[1][J2] = 7.0 + 4.0 = 11.0$$

$$C[1][J4] = D[2][J3] + P[1][J4] = 17.0 + 4.0 = 21.0$$

$$C[1][J5] = D[2][J5] + P[1][J5] = 24.0 + 9.0 = 33.0$$

$$C[1][J3] = D[2][J6] + P[1][J3] = 37.0 + 5.0 = 42.0$$

$$C[1][J0] = D[2][J4] + P[1][J0] = 47.0 + 3.0 = 50.0$$

$$C[1][J1] = D[2][J1] + P[1][J1] = 58.0 + 8.0 = 66.0$$

$$C[2][J2] = D[3][J0] + P[2][J2] = 11.0 + 8.0 = 19.0$$

$$C[2][J4] = D[3][J3] + P[2][J4] = 22.0 + 6.0 = 28.0$$

$$C[2][J5] = D[3][J5] + P[2][J5] = 33.0 + 7.0 = 40.0$$

$$C[2][J3] = D[3][J6] + P[2][J3] = 42.0 + 7.0 = 49.0$$

$$C[2][J0] = D[3][J4] + P[2][J0] = 52.0 + 5.0 = 57.0$$

$$C[2][J1] = D[3][J1] + P[2][J1] = 66.0 + 5.0 = 71.0$$

Calcul des temps de blocage

$$TB[0][J2] = D[1][J0] - C[0][J2] = 7.0 - 7.0 = 0.0$$

$$TB[0][J4] = D[1][J3] - C[0][J4] = 17.0 - 17.0 = 0.0$$

$$TB[0][J5] = D[1][J5] - C[0][J5] = 24.0 - 24.0 = 0.0$$

$$TB[0][J3] = D[1][J6] - C[0][J3] = 37.0 - 37.0 = 0.0$$

$$TB[0][J0] = D[1][J4] - C[0][J0] = 47.0 - 47.0 = 0.0$$

$$TB[0][J1] = D[1][J1] - C[0][J1] = 58.0 - 58.0 = 0.0$$

$$TB[1][J2] = D[2][J0] - C[1][J2] = 11.0 - 11.0 = 0.0$$

$$TB[1][J4] = D[2][J3] - C[1][J4] = 22.0 - 21.0 = 1.0$$

$$TB[1][J5] = D[2][J5] - C[1][J5] = 33.0 - 33.0 = 0.0$$

$$TB[1][J3] = D[2][J6] - C[1][J3] = 42.0 - 42.0 = 0.0$$

$$TB[1][J0] = D[2][J4] - C[1][J0] = 52.0 - 50.0 = 2.0$$

$$TB[1][J1] = D[2][J1] - C[1][J1] = 66.0 - 66.0 = 0.0$$

$$TB[2][J2] = D[3][J0] - C[2][J2] = 19.0 - 19.0 = 0.0$$

$$TB[2][J4] = D[3][J3] - C[2][J4] = 28.0 - 28.0 = 0.0$$

$$TB[2][J5] = D[3][J5] - C[2][J5] = 40.0 - 40.0 = 0.0$$

$$TB[2][J3] = D[3][J6] - C[2][J3] = 49.0 - 49.0 = 0.0$$

$$TB[2][J0] = D[3][J4] - C[2][J0] = 57.0 - 57.0 = 0.0$$

$$TB[2][J1] = D[3][J1] - C[2][J1] = 71.0 - 71.0 = 0.0$$

$$\text{Total flow time TFT} = D[4][J0] + D[4][J3] + D[4][J5] + D[4][J6] + D[4][J4] + D[4][J1] + D[4][J2] =$$

$$0.0 + 19.0 + 28.0 + 40.0 + 49.0 + 57.0 + 71.0 = 264.0$$

Tardiness T:

$$T[J0] = \max( D[4][J0] - d[J-1], 0 ) = \max( 0.0 - 5, 0 ) = 0$$

$$T[J3] = \max( D[4][J3] - d[J2], 0 ) = \max( 19.0 - 0, 0 ) = 19.0$$

$$T[J5] = \max( D[4][J5] - d[J4], 0 ) = \max( 28.0 - 1, 0 ) = 27.0$$

$$T[J6] = \max( D[4][J6] - d[J5], 0 ) = \max( 40.0 - 5, 0 ) = 35.0$$

$$T[J4] = \max( D[4][J4] - d[J3], 0 ) = \max( 49.0 - 3, 0 ) = 46.0$$

$$T[J1] = \max( D[4][J1] - d[J0], 0 ) = \max( 57.0 - 4, 0 ) = 53.0$$

$$T[J2] = \max( D[4][J2] - d[J1], 0 ) = \max( 71.0 - 2, 0 ) = 69.0$$

Tardiness TT =  $\sum T[S[j]] : j = 0, \dots, 6$

TT = 249.0

### 3. Matrices D,C,F

D	J0	J3	J5	J6	J4	J1	J2
M0	0.0	2.0	9.0	20.0	28.0	40.0	49.0
M1	0.0	7.0	17.0	24.0	37.0	47.0	58.0
M2	0.0	11.0	22.0	33.0	42.0	52.0	66.0
M3	0.0	19.0	28.0	40.0	49.0	57.0	71.0

Tableau 25 : dates de fin de chaque job avec son blocage dans chaque machine BFSP-SDST

C[i][S[j]]:dates de fin de chaque job dans chaque machine BFSP-SDST

C	J2	J4	J5	J3	J0	J1
M0	7.0	17.0	24.0	37.0	47.0	58.0
M1	11.0	21.0	33.0	42.0	50.0	66.0
M2	19.0	28.0	40.0	49.0	57.0	71.0

Tableau 26 :dates de fin de chaque job dans chaque machine BFSP-SDST

F[i][S[j]]:dates de début de chaque job dans chaque machine BFSP-SDST

pour trouver les dates de débuts de chaque job: F[i][S[j]] = C[i][S[j]] - P[i][S[j]]

F	J2	J4	J5	J3	J0	J1
M0	2.0	9.0	20.0	28.0	40.0	49.0
M1	7.0	17.0	24.0	37.0	47.0	58.0
M2	11.0	22.0	33.0	42.0	52.0	66.0

Tableau 27 : dates de début de chaque job dans chaque machine BFSP-SDST

### 4. Performances des machines (machines non arrêtées en préparation)

TFR: (teaux de fonctionnement réel)

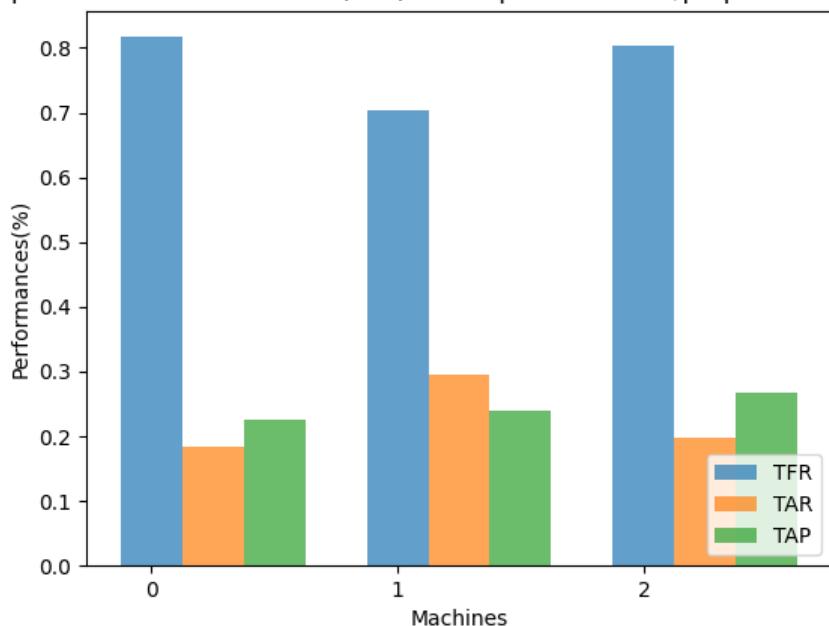
TAR: (teaux d'arrêt réel)

TAP: (teaux d'arrêt de préparation)

	TFR	TAR	TAP
M0	81.69%	18.31%	22.54%
M1	70.42%	29.58%	23.94%
M2	80.28%	19.72%	26.76%

Tableau 28 : Performances des machines BFSP-SDST (machines non arrêtées en préparation)

Temps de fonctionnement réel(TFR) et Temps d'arrêt réel/préparation (TAR/TA)

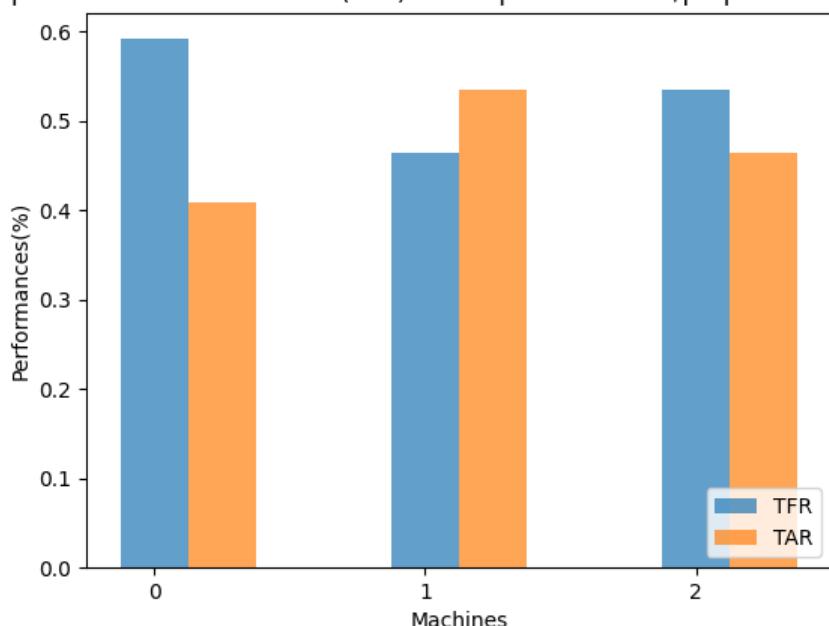


## 5. Performances des machines (machines arrêtées en préparation)

	TFR	TAR
M0	81.69%	18.31%
M1	70.42%	29.58%
M2	80.28%	19.72%

Tableau 29 : Pérformances des machines BFSP-SDST (machines arrêtées en préparation)

Temps de fonctionnement réel(TFR) et Temps d'arrêt réel/préparation (TAR/TA)



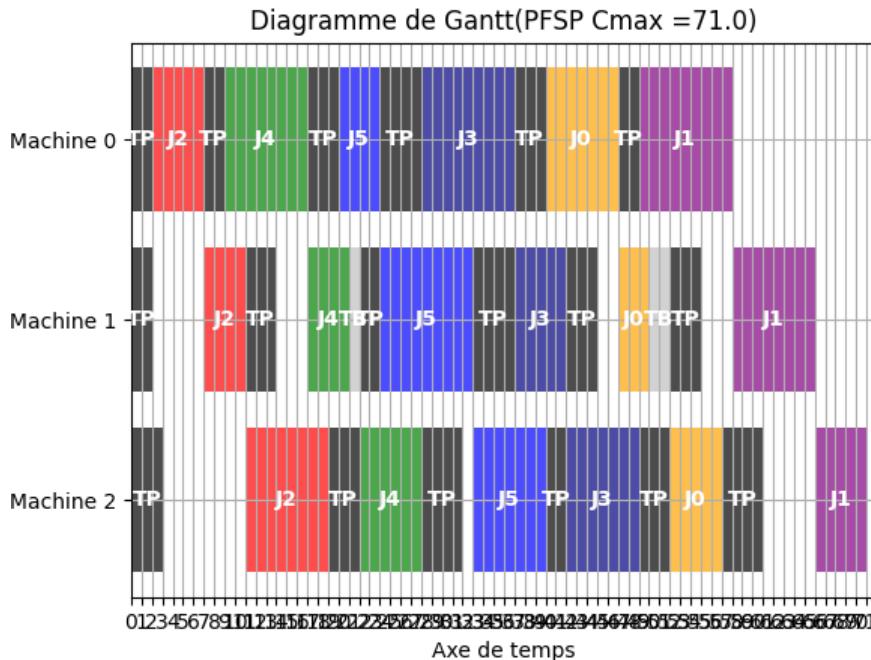
## 6. Temps de blocage de chaque job dans chaque machine

TB: (Blocking Time) temps de blocage de chaque job dans chaque machines

TB	J2	J4	J5	J3	J0	J1
M0	0.0	0.0	0.0	0.0	0.0	0.0
M1	0.0	1.0	0.0	0.0	2.0	0.0
M2	0.0	0.0	0.0	0.0	0.0	0.0

Tableau 30 : Temps de blocage de chaque job dans chaque machines BFSP-SDST

## 7. Diagramme de Gantt,



## VI. NIPFSP

(No-Idle Permutation Flow Shop Scheduling Problem)

- Séquence par Test de tous les permutations possibles avec min(Cmax) et min(TT)

S	J0	J3	J2	J5	J1	J4

Tableau 31 : Séquence trouvée par l'algorithme de Test de toutes les séquences

- Etapes de résolution de NIPFSP (Cmax/TFT...)

C[i][S[j]]: matrice des dates de fin de chaque job dans chaque machine

Sans préparation et dans arrêts

équation 1 : pour i = 0 :

$$L[0] = 0$$

équation 3 et 4 : pour i = 0 et j = 0,...,5:

$$C[0][J0] = L[0] + P[0][J0] = 0.0 + 7 = 7.0$$

$$C[0][J3] = C[0][J0] + P[0][J3] = 7.0 + 9 = 16.0$$

$$C[0][J2] = C[0][J3] + P[0][J2] = 16.0 + 5 = 21.0$$

$$C[0][J5] = C[0][J2] + P[0][J5] = 21.0 + 4 = 25.0$$

$$C[0][J1] = C[0][J5] + P[0][J1] = 25.0 + 9 = 34.0$$

$$C[0][J4] = C[0][J1] + P[0][J4] = 34.0 + 8 = 42.0$$

équation 2: pour i=1

$$\begin{aligned} \text{maxs}[0] &= P[0][J0] + P[0][J3] + P[0][J2] + P[0][J5] + P[0][J1] + P[0][J4] - P[1][J0] - \\ &P[1][J3] - P[1][J2] - P[1][J5] - P[1][J1] = 7 + 9 + 5 + 4 + 9 + 8 - 3 - 5 - 4 - 9 - 8 = 13 \end{aligned}$$

$$\begin{aligned} \text{maxs}[1] &= P[0][J0] + P[0][J3] + P[0][J2] + P[0][J5] + P[0][J1] - P[1][J0] - P[1][J3] - \\ &P[1][J2] - P[1][J5] = 7 + 9 + 5 + 4 + 9 - 3 - 5 - 4 - 9 = 13 \end{aligned}$$

$$\begin{aligned} \text{maxs}[2] &= P[0][J0] + P[0][J3] + P[0][J2] + P[0][J5] - P[1][J0] - P[1][J3] - P[1][J2] = 7 + 9 + \\ &5 + 4 - 3 - 5 - 4 = 13 \end{aligned}$$

$$\text{maxs}[3] = P[0][J0] + P[0][J3] + P[0][J2] - P[1][J0] - P[1][J3] = 7 + 9 + 5 - 3 - 5 = 13$$

$$\text{maxs}[4] = P[0][J0] + P[0][J3] - P[1][J0] = 7 + 9 - 3 = 13$$

$$\text{maxs}[5] = P[0][J0] - = 7 - = 7$$

$$\begin{aligned} L[1] &= L[0] + \max(\text{maxs}[0], \text{maxs}[1], \text{maxs}[2], \text{maxs}[3], \text{maxs}[4], \text{maxs}[5]) = 0.0 + \\ &\max(13, 13, 13, 13, 13, 7) = 0.0 + 13 = 13.0 \end{aligned}$$

équation 3 et 4 : pour i = 1 et j = 0,...,5:

$$C[1][J0] = L[1] + P[1][J0] = 13.0 + 3 = 16.0$$

$$C[1][J3] = C[1][J0] + P[1][J3] = 16.0 + 5 = 21.0$$

$$C[1][J2] = C[1][J3] + P[1][J2] = 21.0 + 4 = 25.0$$

$$C[1][J5] = C[1][J2] + P[1][J5] = 25.0 + 9 = 34.0$$

$$C[1][J1] = C[1][J5] + P[1][J1] = 34.0 + 8 = 42.0$$

$$C[1][J4] = C[1][J1] + P[1][J4] = 42.0 + 4 = 46.0$$

équation 2: pour i=2

$$\begin{aligned} \text{maxs}[0] &= P[1][J0] + P[1][J3] + P[1][J2] + P[1][J5] + P[1][J1] + P[1][J4] - P[2][J0] - \\ &P[2][J3] - P[2][J2] - P[2][J5] - P[2][J1] = 3 + 5 + 4 + 9 + 8 + 4 - 5 - 7 - 8 - 7 - 5 = 1 \end{aligned}$$

$$\begin{aligned} \text{maxs}[1] &= P[1][J0] + P[1][J3] + P[1][J2] + P[1][J5] + P[1][J1] - P[2][J0] - P[2][J3] - \\ &P[2][J2] - P[2][J5] = 3 + 5 + 4 + 9 + 8 - 5 - 7 - 8 - 7 = 2 \end{aligned}$$

$$\begin{aligned} \text{maxs}[2] &= P[1][J0] + P[1][J3] + P[1][J2] + P[1][J5] - P[2][J0] - P[2][J3] - P[2][J2] = 3 + 5 + \\ &4 + 9 - 5 - 7 - 8 = 1 \end{aligned}$$

$$\text{maxs}[3] = P[1][J0] + P[1][J3] + P[1][J2] - P[2][J0] - P[2][J3] = 3 + 5 + 4 - 5 - 7 = 0$$

$$\text{maxs}[4] = P[1][J0] + P[1][J3] - P[2][J0] = 3 + 5 - 5 = 3$$

$$\text{maxs}[5] = P[1][J0] - = 3 - = 3$$

$L[2] = L[1] + \max( \text{maxs}[0], \text{maxs}[1], \text{maxs}[2], \text{maxs}[3], \text{maxs}[4], \text{maxs}[5] ) = 13.0 + \max(1, 2, 1, 0, 3, 3) = 13.0 + 3 = 16.0$

équation 3 et 4 : pour  $i = 2$  et  $j = 0, \dots, 5$ :

$$C[2][J0] = L[2] + P[2][J0] = 16.0 + 5 = 21.0$$

$$C[2][J3] = C[2][J0] + P[2][J3] = 21.0 + 7 = 28.0$$

$$C[2][J2] = C[2][J3] + P[2][J2] = 28.0 + 8 = 36.0$$

$$C[2][J5] = C[2][J2] + P[2][J5] = 36.0 + 7 = 43.0$$

$$C[2][J1] = C[2][J5] + P[2][J1] = 43.0 + 5 = 48.0$$

$$C[2][J4] = C[2][J1] + P[2][J4] = 48.0 + 6 = 54.0$$

Calcul des temps d'attendre :

$$TA[0/1][J0] = C[1][J0] - P[1][J0] - C[i][J0] = 16.0 - 3 - 7.0 = 6.0$$

$$TA[0/1][J3] = C[1][J3] - P[1][J3] - C[i][J3] = 21.0 - 8 - 16.0 = 0.0$$

$$TA[0/1][J2] = C[1][J2] - P[1][J2] - C[i][J2] = 25.0 - 4 - 21.0 = 0.0$$

$$TA[0/1][J5] = C[1][J5] - P[1][J5] - C[i][J5] = 34.0 - 5 - 25.0 = 0.0$$

$$TA[0/1][J1] = C[1][J1] - P[1][J1] - C[i][J1] = 42.0 - 4 - 34.0 = 0.0$$

$$TA[0/1][J4] = C[1][J4] - P[1][J4] - C[i][J4] = 46.0 - 9 - 42.0 = 0.0$$

$$TA[1/2][J0] = C[2][J0] - P[2][J0] - C[i][J0] = 21.0 - 5 - 16.0 = 0.0$$

$$TA[1/2][J3] = C[2][J3] - P[2][J3] - C[i][J3] = 28.0 - 5 - 21.0 = 0.0$$

$$TA[1/2][J2] = C[2][J2] - P[2][J2] - C[i][J2] = 36.0 - 8 - 25.0 = 3.0$$

$$TA[1/2][J5] = C[2][J5] - P[2][J5] - C[i][J5] = 43.0 - 7 - 34.0 = 2.0$$

$$TA[1/2][J1] = C[2][J1] - P[2][J1] - C[i][J1] = 48.0 - 6 - 42.0 = 1.0$$

$$TA[1/2][J4] = C[2][J4] - P[2][J4] - C[i][J4] = 54.0 - 7 - 46.0 = 2.0$$

Total flow time TFT =  $C[2][J0] + C[2][J3] + C[2][J2] + C[2][J5] + C[2][J1] + C[2][J4] =$

$$21.0 + 28.0 + 36.0 + 43.0 + 48.0 + 54.0 = 230.0$$

Tardiness T:

$$T[J0] = \max( C[2][J0] - d[J0], 0 ) = \max( 21.0 - 4, 0 ) = 17.0$$

$$T[J3] = \max( C[2][J3] - d[J3], 0 ) = \max( 28.0 - 3, 0 ) = 25.0$$

$$T[J2] = \max( C[2][J2] - d[J2], 0 ) = \max( 36.0 - 0, 0 ) = 36.0$$

$$T[J5] = \max( C[2][J5] - d[J5], 0 ) = \max( 43.0 - 5, 0 ) = 38.0$$

$$T[J1] = \max( C[2][J1] - d[J1], 0 ) = \max( 48.0 - 2, 0 ) = 46.0$$

$$T[J4] = \max( C[2][J4] - d[J4], 0 ) = \max( 54.0 - 1, 0 ) = 53.0$$

Tardiness TT =  $\sum T[S[j]] : j = 0, \dots, 5$

$$TT = 215.0$$

### 3. Matrice des dates des fins et des débuts de chaque job sur chaque machine

C	J0	J3	J2	J5	J1	J4
M0	7.0	16.0	21.0	25.0	34.0	42.0
M1	16.0	21.0	25.0	34.0	42.0	46.0
M2	21.0	28.0	36.0	43.0	48.0	54.0

Tableau 32 : Table 32 : dates de fin de chaque job dans chaque machine NIPFSP

$F[i][S[j]]$ :dates de début de chaque job dans chaque machine NIPFSP  
pour trouver les dates de débuts de chaque job:  $F[i][S[j]] = C[i][S[j]] - P[i][S[j]]$

F	J0	J3	J2	J5	J1	J4
M0	0.0	7.0	16.0	21.0	25.0	34.0
M1	13.0	16.0	21.0	25.0	34.0	42.0
M2	16.0	21.0	28.0	36.0	43.0	48.0

Tableau 33 : dates de début de chaque job dans chaque machine NIPFSP

### 4. Performances des machines

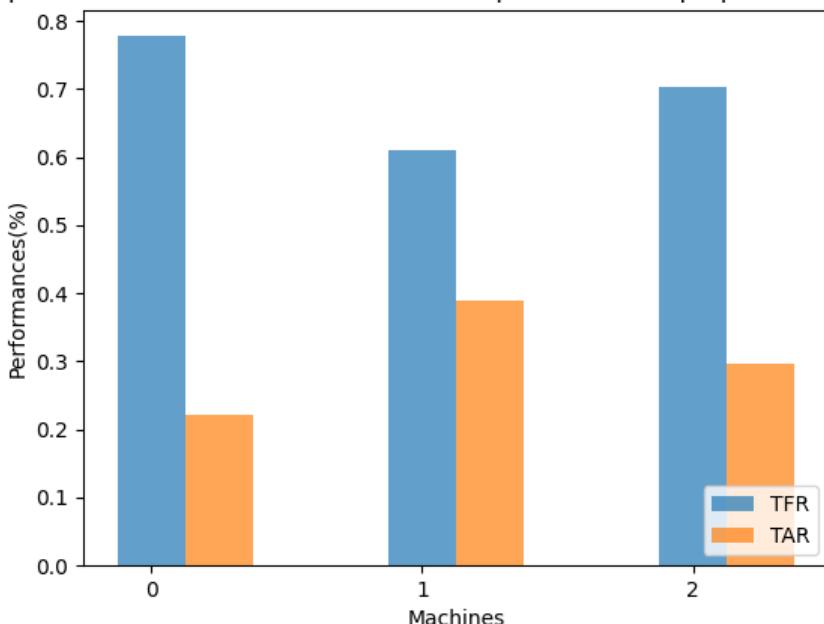
TFR: (taux de fonctionnement réel)

TAR: (taux d'arrêt réel)

	TFR	TAR
M0	77.78%	22.22%
M1	61.11%	38.89%
M2	70.37%	29.63%

Tableau 34 : Performances des machines NIPFSP

Temps de fonctionnement réel(TFR) et Temps d'arrêt réel/préparation (TAR/TA)



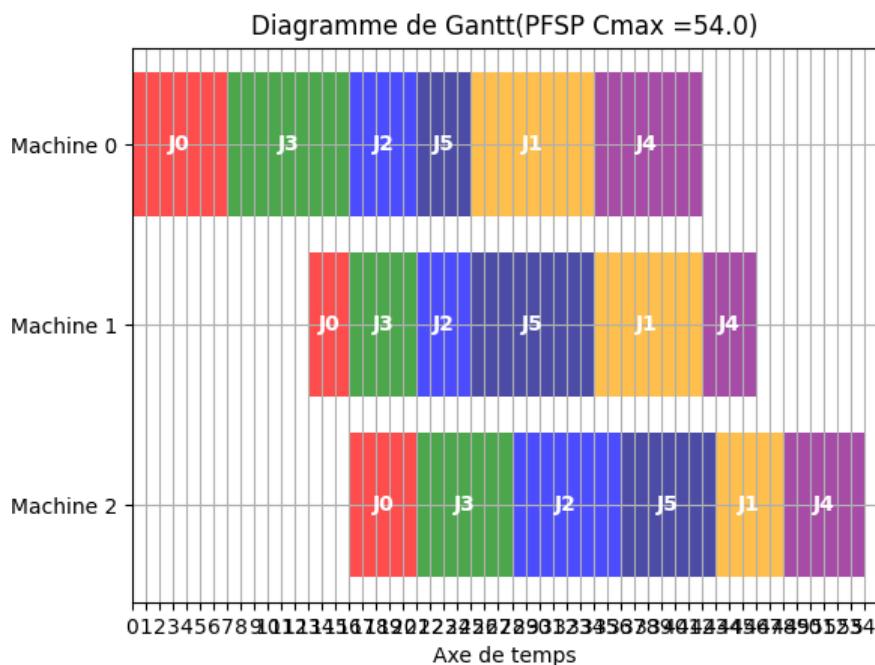
## 5. Temps d'attende de chaque job entre chaque deux machines

TW: (waiting Time) temps d'attende de chaque job entre chaque deux machines

TW	J0	J3	J2	J5	J1	J4
M0/1	6.0	0.0	0.0	0.0	0.0	0.0
M1/2	0.0	0.0	3.0	2.0	1.0	2.0

Tableau 35 : Temps d'attendre de chaque job entre chaque deux machines NIPFSP

## 6. Diagramme de Gantt,



## VII. NWPFSP

(No-Wait Permutation Flow Shop Scheduling Problem)

1. Séquence par Test de tous les permutations possibles avec min(Cmax) et min(TT)

S	J0	J2	J5	J1	J3	J4

Tableau 36 : Séquence trouvée par l'algorithme de Test de toutes les séquences

2. Etapes de résolution de NWPFSP (Cmax/TFT...)

C[i][S[j]]: matrice des dates de fin de chaque job dans chaque machine

Sans préparation et dans attendre

équation 1 : Calcul de la matrice des délais D entre chaque deux job tel que  $j_1 = 0, \dots, 5$   $j_2 = 0, \dots, 5$  :

$\text{maxs}[0] = 0 - 0 = 0$   
 $\text{maxs}[1] = P[1][J0] - P[0][J0] = 3 - 7 = -4$   
 $\text{maxs}[2] = P[1][J0] + P[2][J0] - P[0][J0] - P[1][J0] = 3 + 5 - 7 - 3 = 8 - 10 = -2$   
 $\text{maxs} = [0, -4, -2]$   
 $D[J0][J0] = P[0][J0] + \max(\max(\text{maxs}), 0) = 7 + 0 = 7$   
 $\text{maxs}[0] = 0 - 0 = 0$   
 $\text{maxs}[1] = P[1][J0] - P[0][J1] = 3 - 9 = -6$   
 $\text{maxs}[2] = P[1][J0] + P[2][J0] - P[0][J1] - P[1][J1] = 3 + 5 - 9 - 8 = 8 - 17 = -9$   
 $\text{maxs} = [0, -6, -9]$   
 $D[J0][J1] = P[0][J0] + \max(\max(\text{maxs}), 0) = 7 + 0 = 7$   
 $\text{maxs}[0] = 0 - 0 = 0$   
 $\text{maxs}[1] = P[1][J0] - P[0][J2] = 3 - 5 = -2$   
 $\text{maxs}[2] = P[1][J0] + P[2][J0] - P[0][J2] - P[1][J2] = 3 + 5 - 5 - 4 = 8 - 9 = -1$   
 $\text{maxs} = [0, -2, -1]$   
 $D[J0][J2] = P[0][J0] + \max(\max(\text{maxs}), 0) = 7 + 0 = 7$   
 $\text{maxs}[0] = 0 - 0 = 0$   
 $\text{maxs}[1] = P[1][J0] - P[0][J3] = 3 - 9 = -6$   
 $\text{maxs}[2] = P[1][J0] + P[2][J0] - P[0][J3] - P[1][J3] = 3 + 5 - 9 - 5 = 8 - 14 = -6$   
 $\text{maxs} = [0, -6, -6]$   
 $D[J0][J3] = P[0][J0] + \max(\max(\text{maxs}), 0) = 7 + 0 = 7$   
 $\text{maxs}[0] = 0 - 0 = 0$   
 $\text{maxs}[1] = P[1][J0] - P[0][J4] = 3 - 8 = -5$   
 $\text{maxs}[2] = P[1][J0] + P[2][J0] - P[0][J4] - P[1][J4] = 3 + 5 - 8 - 4 = 8 - 12 = -4$   
 $\text{maxs} = [0, -5, -4]$   
 $D[J0][J4] = P[0][J0] + \max(\max(\text{maxs}), 0) = 7 + 0 = 7$   
 $\text{maxs}[0] = 0 - 0 = 0$   
 $\text{maxs}[1] = P[1][J0] - P[0][J5] = 3 - 4 = -1$   
 $\text{maxs}[2] = P[1][J0] + P[2][J0] - P[0][J5] - P[1][J5] = 3 + 5 - 4 - 9 = 8 - 13 = -5$   
 $\text{maxs} = [0, -1, -5]$   
 $D[J0][J5] = P[0][J0] + \max(\max(\text{maxs}), 0) = 7 + 0 = 7$   
 $\text{maxs}[0] = 0 - 0 = 0$   
 $\text{maxs}[1] = P[1][J1] - P[0][J0] = 8 - 7 = 1$   
 $\text{maxs}[2] = P[1][J1] + P[2][J1] - P[0][J0] - P[1][J0] = 8 + 5 - 7 - 3 = 13 - 10 = 3$   
 $\text{maxs} = [0, 1, 3]$

$D[J1][J0] = P[0][J1] + \max(\max(\text{maxs}), 0) = 9 + 3 = 12$   
 $\text{maxs}[0] = 0 - 0 = 0$   
 $\text{maxs}[1] = P[1][J1] - P[0][J1] = 8 - 9 = -1$   
 $\text{maxs}[2] = P[1][J1] + P[2][J1] - P[0][J1] - P[1][J1] = 8 + 5 - 9 - 8 = 13 - 17 = -4$   
 $\text{maxs} = [0, -1, -4]$   
 $D[J1][J1] = P[0][J1] + \max(\max(\text{maxs}), 0) = 9 + 0 = 9$   
 $\text{maxs}[0] = 0 - 0 = 0$   
 $\text{maxs}[1] = P[1][J1] - P[0][J2] = 8 - 5 = 3$   
 $\text{maxs}[2] = P[1][J1] + P[2][J1] - P[0][J2] - P[1][J2] = 8 + 5 - 5 - 4 = 13 - 9 = 4$   
 $\text{maxs} = [0, 3, 4]$   
 $D[J1][J2] = P[0][J1] + \max(\max(\text{maxs}), 0) = 9 + 4 = 13$   
 $\text{maxs}[0] = 0 - 0 = 0$   
 $\text{maxs}[1] = P[1][J1] - P[0][J3] = 8 - 9 = -1$   
 $\text{maxs}[2] = P[1][J1] + P[2][J1] - P[0][J3] - P[1][J3] = 8 + 5 - 9 - 5 = 13 - 14 = -1$   
 $\text{maxs} = [0, -1, -1]$   
 $D[J1][J3] = P[0][J1] + \max(\max(\text{maxs}), 0) = 9 + 0 = 9$   
 $\text{maxs}[0] = 0 - 0 = 0$   
 $\text{maxs}[1] = P[1][J1] - P[0][J4] = 8 - 8 = 0$   
 $\text{maxs}[2] = P[1][J1] + P[2][J1] - P[0][J4] - P[1][J4] = 8 + 5 - 8 - 4 = 13 - 12 = 1$   
 $\text{maxs} = [0, 0, 1]$   
 $D[J1][J4] = P[0][J1] + \max(\max(\text{maxs}), 0) = 9 + 1 = 10$   
 $\text{maxs}[0] = 0 - 0 = 0$   
 $\text{maxs}[1] = P[1][J1] - P[0][J5] = 8 - 4 = 4$   
 $\text{maxs}[2] = P[1][J1] + P[2][J1] - P[0][J5] - P[1][J5] = 8 + 5 - 4 - 9 = 13 - 13 = 0$   
 $\text{maxs} = [0, 4, 0]$   
 $D[J1][J5] = P[0][J1] + \max(\max(\text{maxs}), 0) = 9 + 4 = 13$   
 $\text{maxs}[0] = 0 - 0 = 0$   
 $\text{maxs}[1] = P[1][J2] - P[0][J0] = 4 - 7 = -3$   
 $\text{maxs}[2] = P[1][J2] + P[2][J2] - P[0][J0] - P[1][J0] = 4 + 8 - 7 - 3 = 12 - 10 = 2$   
 $\text{maxs} = [0, -3, 2]$   
 $D[J2][J0] = P[0][J2] + \max(\max(\text{maxs}), 0) = 5 + 2 = 7$   
 $\text{maxs}[0] = 0 - 0 = 0$   
 $\text{maxs}[1] = P[1][J2] - P[0][J1] = 4 - 9 = -5$   
 $\text{maxs}[2] = P[1][J2] + P[2][J2] - P[0][J1] - P[1][J1] = 4 + 8 - 9 - 8 = 12 - 17 = -5$

maxs = [0, -5, -5]

$$D[J2][J1] = P[0][J2] + \max(\max(maxs), 0) = 5 + 0 = 5$$

maxs[0] = 0 - 0 = 0

$$\text{maxs}[1] = P[1][J2] - P[0][J2] = 4 - 5 = -1$$

$$\text{maxs}[2] = P[1][J2] + P[2][J2] - P[0][J2] - P[1][J2] = 4 + 8 - 5 - 4 = 12 - 9 = 3$$

maxs = [0, -1, 3]

$$D[J2][J2] = P[0][J2] + \max(\max(maxs), 0) = 5 + 3 = 8$$

maxs[0] = 0 - 0 = 0

$$\text{maxs}[1] = P[1][J2] - P[0][J3] = 4 - 9 = -5$$

$$\text{maxs}[2] = P[1][J2] + P[2][J2] - P[0][J3] - P[1][J3] = 4 + 8 - 9 - 5 = 12 - 14 = -2$$

maxs = [0, -5, -2]

$$D[J2][J3] = P[0][J2] + \max(\max(maxs), 0) = 5 + 0 = 5$$

maxs[0] = 0 - 0 = 0

$$\text{maxs}[1] = P[1][J2] - P[0][J4] = 4 - 8 = -4$$

$$\text{maxs}[2] = P[1][J2] + P[2][J2] - P[0][J4] - P[1][J4] = 4 + 8 - 8 - 4 = 12 - 12 = 0$$

maxs = [0, -4, 0]

$$D[J2][J4] = P[0][J2] + \max(\max(maxs), 0) = 5 + 0 = 5$$

maxs[0] = 0 - 0 = 0

$$\text{maxs}[1] = P[1][J2] - P[0][J5] = 4 - 4 = 0$$

$$\text{maxs}[2] = P[1][J2] + P[2][J2] - P[0][J5] - P[1][J5] = 4 + 8 - 4 - 9 = 12 - 13 = -1$$

maxs = [0, 0, -1]

$$D[J2][J5] = P[0][J2] + \max(\max(maxs), 0) = 5 + 0 = 5$$

maxs[0] = 0 - 0 = 0

$$\text{maxs}[1] = P[1][J3] - P[0][J0] = 5 - 7 = -2$$

$$\text{maxs}[2] = P[1][J3] + P[2][J3] - P[0][J0] - P[1][J0] = 5 + 7 - 7 - 3 = 12 - 10 = 2$$

maxs = [0, -2, 2]

$$D[J3][J0] = P[0][J3] + \max(\max(maxs), 0) = 9 + 2 = 11$$

maxs[0] = 0 - 0 = 0

$$\text{maxs}[1] = P[1][J3] - P[0][J1] = 5 - 9 = -4$$

$$\text{maxs}[2] = P[1][J3] + P[2][J3] - P[0][J1] - P[1][J1] = 5 + 7 - 9 - 8 = 12 - 17 = -5$$

maxs = [0, -4, -5]

$$D[J3][J1] = P[0][J3] + \max(\max(maxs), 0) = 9 + 0 = 9$$

maxs[0] = 0 - 0 = 0

$$\text{maxs}[1] = P[1][J3] - P[0][J2] = 5 - 5 = 0$$



$$\text{maxs}[2] = P[1][J3] + P[2][J3] - P[0][J2] - P[1][J2] = 5 + 7 - 5 - 4 = 12 - 9 = 3$$

$$\text{maxs} = [0, 0, 3]$$

$$D[J3][J2] = P[0][J3] + \max(\max(\text{maxs}), 0) = 9 + 3 = 12$$

$$\text{maxs}[0] = 0 - 0 = 0$$

$$\text{maxs}[1] = P[1][J3] - P[0][J3] = 5 - 9 = -4$$

$$\text{maxs}[2] = P[1][J3] + P[2][J3] - P[0][J3] - P[1][J3] = 5 + 7 - 9 - 5 = 12 - 14 = -2$$

$$\text{maxs} = [0, -4, -2]$$

$$D[J3][J3] = P[0][J3] + \max(\max(\text{maxs}), 0) = 9 + 0 = 9$$

$$\text{maxs}[0] = 0 - 0 = 0$$

$$\text{maxs}[1] = P[1][J3] - P[0][J4] = 5 - 8 = -3$$

$$\text{maxs}[2] = P[1][J3] + P[2][J3] - P[0][J4] - P[1][J4] = 5 + 7 - 8 - 4 = 12 - 12 = 0$$

$$\text{maxs} = [0, -3, 0]$$

$$D[J3][J4] = P[0][J3] + \max(\max(\text{maxs}), 0) = 9 + 0 = 9$$

$$\text{maxs}[0] = 0 - 0 = 0$$

$$\text{maxs}[1] = P[1][J3] - P[0][J5] = 5 - 4 = 1$$

$$\text{maxs}[2] = P[1][J3] + P[2][J3] - P[0][J5] - P[1][J5] = 5 + 7 - 4 - 9 = 12 - 13 = -1$$

$$\text{maxs} = [0, 1, -1]$$

$$D[J3][J5] = P[0][J3] + \max(\max(\text{maxs}), 0) = 9 + 1 = 10$$

$$\text{maxs}[0] = 0 - 0 = 0$$

$$\text{maxs}[1] = P[1][J4] - P[0][J0] = 4 - 7 = -3$$

$$\text{maxs}[2] = P[1][J4] + P[2][J4] - P[0][J0] - P[1][J0] = 4 + 6 - 7 - 3 = 10 - 10 = 0$$

$$\text{maxs} = [0, -3, 0]$$

$$D[J4][J0] = P[0][J4] + \max(\max(\text{maxs}), 0) = 8 + 0 = 8$$

$$\text{maxs}[0] = 0 - 0 = 0$$

$$\text{maxs}[1] = P[1][J4] - P[0][J1] = 4 - 9 = -5$$

$$\text{maxs}[2] = P[1][J4] + P[2][J4] - P[0][J1] - P[1][J1] = 4 + 6 - 9 - 8 = 10 - 17 = -7$$

$$\text{maxs} = [0, -5, -7]$$

$$D[J4][J1] = P[0][J4] + \max(\max(\text{maxs}), 0) = 8 + 0 = 8$$

$$\text{maxs}[0] = 0 - 0 = 0$$

$$\text{maxs}[1] = P[1][J4] - P[0][J2] = 4 - 5 = -1$$

$$\text{maxs}[2] = P[1][J4] + P[2][J4] - P[0][J2] - P[1][J2] = 4 + 6 - 5 - 4 = 10 - 9 = 1$$

$$\text{maxs} = [0, -1, 1]$$

$$D[J4][J2] = P[0][J4] + \max(\max(\text{maxs}), 0) = 8 + 1 = 9$$

$$\text{maxs}[0] = 0 - 0 = 0$$

$$\text{maxs}[1] = P[1][J4] - P[0][J3] = 4 - 9 = -5$$

$$\text{maxs}[2] = P[1][J4] + P[2][J4] - P[0][J3] - P[1][J3] = 4 + 6 - 9 - 5 = 10 - 14 = -4$$

$$\text{maxs} = [0, -5, -4]$$

$$D[J4][J3] = P[0][J4] + \max(\max(\text{maxs}), 0) = 8 + 0 = 8$$

$$\text{maxs}[0] = 0 - 0 = 0$$

$$\text{maxs}[1] = P[1][J4] - P[0][J4] = 4 - 8 = -4$$

$$\text{maxs}[2] = P[1][J4] + P[2][J4] - P[0][J4] - P[1][J4] = 4 + 6 - 8 - 4 = 10 - 12 = -2$$

$$\text{maxs} = [0, -4, -2]$$

$$D[J4][J4] = P[0][J4] + \max(\max(\text{maxs}), 0) = 8 + 0 = 8$$

$$\text{maxs}[0] = 0 - 0 = 0$$

$$\text{maxs}[1] = P[1][J4] - P[0][J5] = 4 - 4 = 0$$

$$\text{maxs}[2] = P[1][J4] + P[2][J4] - P[0][J5] - P[1][J5] = 4 + 6 - 4 - 9 = 10 - 13 = -3$$

$$\text{maxs} = [0, 0, -3]$$

$$D[J4][J5] = P[0][J4] + \max(\max(\text{maxs}), 0) = 8 + 0 = 8$$

$$\text{maxs}[0] = 0 - 0 = 0$$

$$\text{maxs}[1] = P[1][J5] - P[0][J0] = 9 - 7 = 2$$

$$\text{maxs}[2] = P[1][J5] + P[2][J5] - P[0][J0] - P[1][J0] = 9 + 7 - 7 - 3 = 16 - 10 = 6$$

$$\text{maxs} = [0, 2, 6]$$

$$D[J5][J0] = P[0][J5] + \max(\max(\text{maxs}), 0) = 4 + 6 = 10$$

$$\text{maxs}[0] = 0 - 0 = 0$$

$$\text{maxs}[1] = P[1][J5] - P[0][J1] = 9 - 9 = 0$$

$$\text{maxs}[2] = P[1][J5] + P[2][J5] - P[0][J1] - P[1][J1] = 9 + 7 - 9 - 8 = 16 - 17 = -1$$

$$\text{maxs} = [0, 0, -1]$$

$$D[J5][J1] = P[0][J5] + \max(\max(\text{maxs}), 0) = 4 + 0 = 4$$

$$\text{maxs}[0] = 0 - 0 = 0$$

$$\text{maxs}[1] = P[1][J5] - P[0][J2] = 9 - 5 = 4$$

$$\text{maxs}[2] = P[1][J5] + P[2][J5] - P[0][J2] - P[1][J2] = 9 + 7 - 5 - 4 = 16 - 9 = 7$$

$$\text{maxs} = [0, 4, 7]$$

$$D[J5][J2] = P[0][J5] + \max(\max(\text{maxs}), 0) = 4 + 7 = 11$$

$$\text{maxs}[0] = 0 - 0 = 0$$

$$\text{maxs}[1] = P[1][J5] - P[0][J3] = 9 - 9 = 0$$

$$\text{maxs}[2] = P[1][J5] + P[2][J5] - P[0][J3] - P[1][J3] = 9 + 7 - 9 - 5 = 16 - 14 = 2$$

$$\text{maxs} = [0, 0, 2]$$

$$D[J5][J3] = P[0][J5] + \max(\max(\text{maxs}), 0) = 4 + 2 = 6$$

$$\text{maxs}[0] = 0 - 0 = 0$$

$$\text{maxs}[1] = P[1][J5] - P[0][J4] = 9 - 8 = 1$$

$$\text{maxs}[2] = P[1][J5] + P[2][J5] - P[0][J4] - P[1][J4] = 9 + 7 - 8 - 4 = 16 - 12 = 4$$

$$\text{maxs} = [0, 1, 4]$$

$$D[J5][J4] = P[0][J5] + \max(\max(\text{maxs}), 0) = 4 + 4 = 8$$

$$\text{maxs}[0] = 0 - 0 = 0$$

$$\text{maxs}[1] = P[1][J5] - P[0][J5] = 9 - 4 = 5$$

$$\text{maxs}[2] = P[1][J5] + P[2][J5] - P[0][J5] - P[1][J5] = 9 + 7 - 4 - 9 = 16 - 13 = 3$$

$$\text{maxs} = [0, 5, 3]$$

$$D[J5][J5] = P[0][J5] + \max(\max(\text{maxs}), 0) = 4 + 5 = 9$$

équation 2 : Calcul de la dernière ligne de la matrice C pour  $i = 2$  et  $j = 0, \dots, 5$  :

$$C[2][J0] = P[0][J0] + P[1][J0] + P[2][J0] = 7 + 3 + 5 = 15 = 15$$

$$C[2][J2] = D[J0][J2] + P[0][J2] + P[1][J2] + P[2][J2] = 7.0 + 5 + 4 + 8 = 7.0 + 17 = 24.0$$

$$C[2][J5] = D[J0][J2] + D[J2][J5] + P[0][J5] + P[1][J5] + P[2][J5] = 7.0 + 5.0 + 4 + 9 + 7 = 12.0 + 20 = 32.0$$

$$C[2][J1] = D[J0][J2] + D[J2][J5] + D[J5][J1] + P[0][J1] + P[1][J1] + P[2][J1] = 7.0 + 5.0 + 4.0 + 9 + 8 + 5 = 16.0 + 22 = 38.0$$

$$C[2][J3] = D[J0][J2] + D[J2][J5] + D[J5][J1] + D[J1][J3] + P[0][J3] + P[1][J3] + P[2][J3] = 7.0 + 5.0 + 4.0 + 9.0 + 9 + 5 + 7 = 25.0 + 21 = 46.0$$

$$C[2][J4] = D[J0][J2] + D[J2][J5] + D[J5][J1] + D[J1][J3] + D[J3][J4] + P[0][J4] + P[1][J4] + P[2][J4] = 7.0 + 5.0 + 4.0 + 9.0 + 9.0 + 8 + 4 + 6 = 34.0 + 18 = 52.0$$

équation 3 : Calcul de la matrice C pour  $i = 1, \dots, 0$  et  $j = 0, \dots, 5$  :

$$C[1][J0] = C[2][J0] - P[2][J0] = 15.0 - 5 = 10.0$$

$$C[1][J2] = C[2][J2] - P[2][J2] = 24.0 - 8 = 16.0$$

$$C[1][J5] = C[2][J5] - P[2][J5] = 32.0 - 7 = 25.0$$

$$C[1][J1] = C[2][J1] - P[2][J1] = 38.0 - 5 = 33.0$$

$$C[1][J3] = C[2][J3] - P[2][J3] = 46.0 - 7 = 39.0$$

$$C[1][J4] = C[2][J4] - P[2][J4] = 52.0 - 6 = 46.0$$

$$C[0][J0] = C[1][J0] - P[1][J0] = 10.0 - 3 = 7.0$$

$$C[0][J2] = C[1][J2] - P[1][J2] = 16.0 - 4 = 12.0$$

$$C[0][J5] = C[1][J5] - P[1][J5] = 25.0 - 9 = 16.0$$

$$C[0][J1] = C[1][J1] - P[1][J1] = 33.0 - 8 = 25.0$$

$$C[0][J3] = C[1][J3] - P[1][J3] = 39.0 - 5 = 34.0$$

$$C[0][J4] = C[1][J4] - P[1][J4] = 46.0 - 4 = 42.0$$

$$\text{Total flow time TFT} = C[2][J0] + C[2][J2] + C[2][J5] + C[2][J1] + C[2][J3] + C[2][J4] =$$

$$15.0 + 24.0 + 32.0 + 38.0 + 46.0 + 52.0 = 207.0$$

Tardiness T:

$$T[J0] = \max(C[2][J0] - d[J0], 0) = \max(15.0 - 4, 0) = 11.0$$

$$T[J2] = \max(C[2][J2] - d[J2], 0) = \max(24.0 - 0, 0) = 24.0$$

$$T[J5] = \max(C[2][J5] - d[J5], 0) = \max(32.0 - 5, 0) = 27.0$$

$$T[J1] = \max(C[2][J1] - d[J1], 0) = \max(38.0 - 2, 0) = 36.0$$

$$T[J3] = \max(C[2][J3] - d[J3], 0) = \max(46.0 - 3, 0) = 43.0$$

$$T[J4] = \max(C[2][J4] - d[J4], 0) = \max(52.0 - 1, 0) = 51.0$$

$$\text{Tardiness TT} = \sum T[S[j]] : j = 0, \dots, 5$$

$$TT = 192.0$$

### 3. Matrice des dates des fins et des débuts de chaque job sur chaque machine

C	J0	J2	J5	J1	J3	J4
M0	7.0	12.0	16.0	25.0	34.0	42.0
M1	10.0	16.0	25.0	33.0	39.0	46.0
M2	15.0	24.0	32.0	38.0	46.0	52.0

Tableau 37 : dates de fin de chaque job dans chaque machine NWPFPSP

F[i][S[j]]:dates de début de chaque job dans chaque machine NWPFPSP

pour trouver les dates de débuts de chaque job:  $F[i][S[j]] = C[i][S[j]] - P[i][S[j]]$

F	J0	J2	J5	J1	J3	J4
M0	0.0	7.0	12.0	16.0	25.0	34.0
M1	7.0	12.0	16.0	25.0	34.0	42.0
M2	10.0	16.0	25.0	33.0	39.0	46.0

Tableau 38 : dates de début de chaque job dans chaque machine NWPFPSP

### 4. Performances des machines

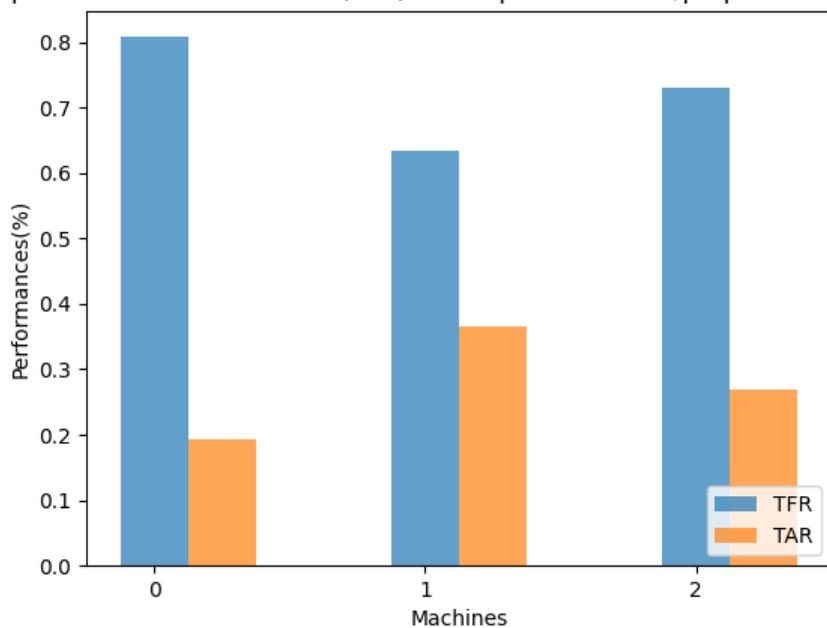
TFR: (taux de fonctionnement réel)

TAR: (taux d'arrêt réel)

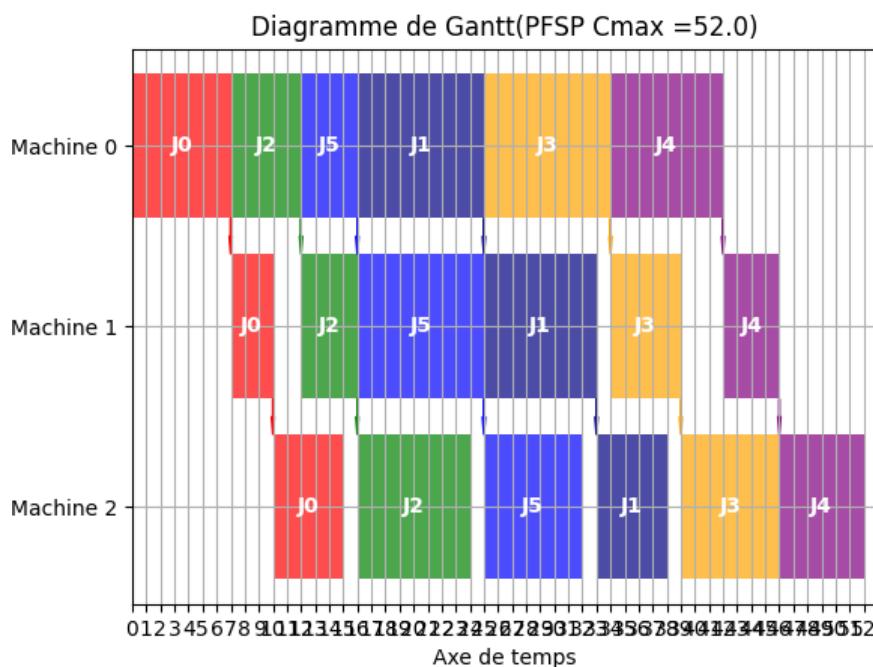
	TFR	TAR
M0	80.77%	19.23%
M1	63.46%	36.54%
M2	73.08%	26.92%

Tableau 39 : Performances des machines NWPFPSP

Temps de fonctionnement réel(TFR) et Temps d'arrêt réel/préparation (TAR/TA)



## 5. Diagramme de Gantt,



## Chapitre III : Présentation de l'application « MES » :

### I. Modélisation des algorithme de l'application

#### 1. Cas d'utilisation

Le diagramme de cas d'utilisation représente deux fonctionnalités principales de l'application d'ordonnancement Flow Shop. La première fonction, "Générer Rapport", permet à l'utilisateur d'obtenir des rapports détaillés sur les résultats des algorithmes. L'utilisateur peut visualiser et sauvegarder ces rapports. La deuxième fonction, "Visualisation des Données", offre à l'utilisateur une interface graphique pour explorer visuellement les performances des algorithmes en testant diverses combinaisons. En ajustant les paramètres de visualisation, l'utilisateur peut générer des graphiques interactifs et exporter les visualisations. Ces deux fonctions offrent un ensemble complet d'outils pour analyser et comprendre les résultats de l'application d'ordonnancement Flow Shop.

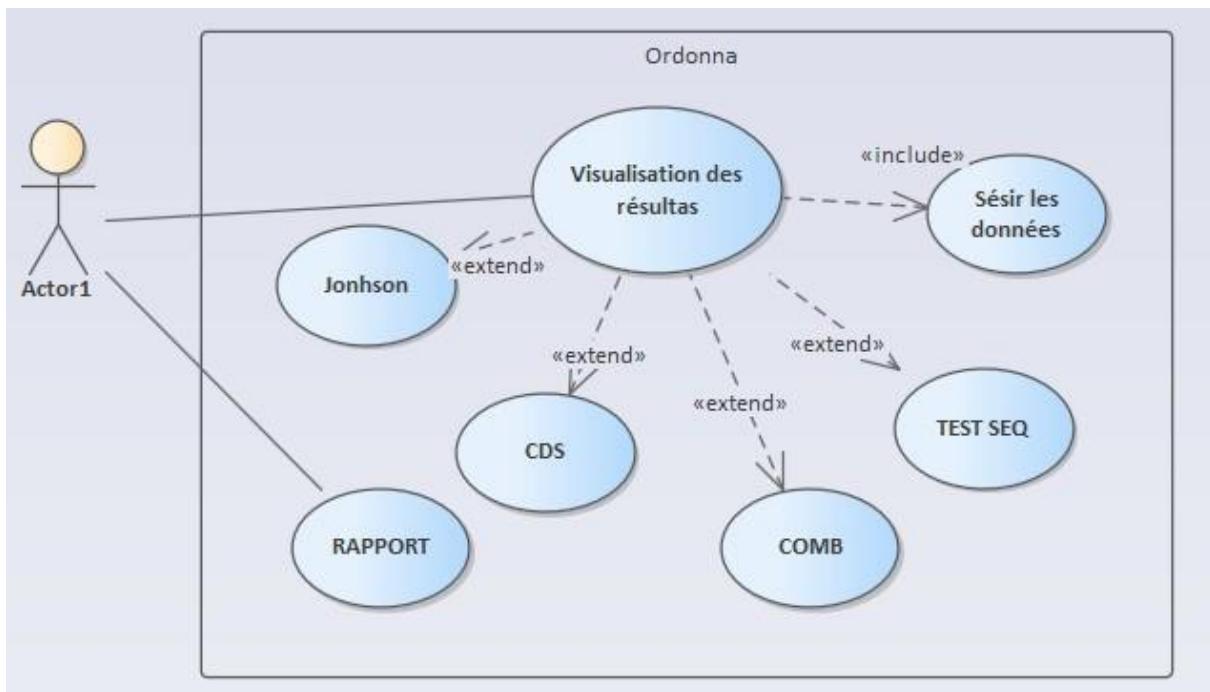


Figure 12 : Diagramme cas d'utilisation

## 2. Les classes utilisées

Le diagramme de classe représente l'architecture d'une application d'ordonnancement Flow Shop avec quatre classes principales : App, Ordonnancement, Report, et Tabs. Les classes App et Tabs héritent de la classe Custom\_Tkinter pour gérer l'affichage. Chaque classe possède des attributs spécifiques (en rouge) et des méthodes appropriées (en vert). La classe App initialise l'application, utilisant Ordonnancement pour résoudre les problèmes et Tabs pour afficher les onglets. La classe Ordonnancement a une méthode pour résoudre les problèmes de Flow Shop, tandis que la classe Report génère un rapport en extrayant et traitant les données de l'instance App. La classe Tabs affiche le diagramme de GANTT et les performances des machines en utilisant les résultats de l'ordonnancement.

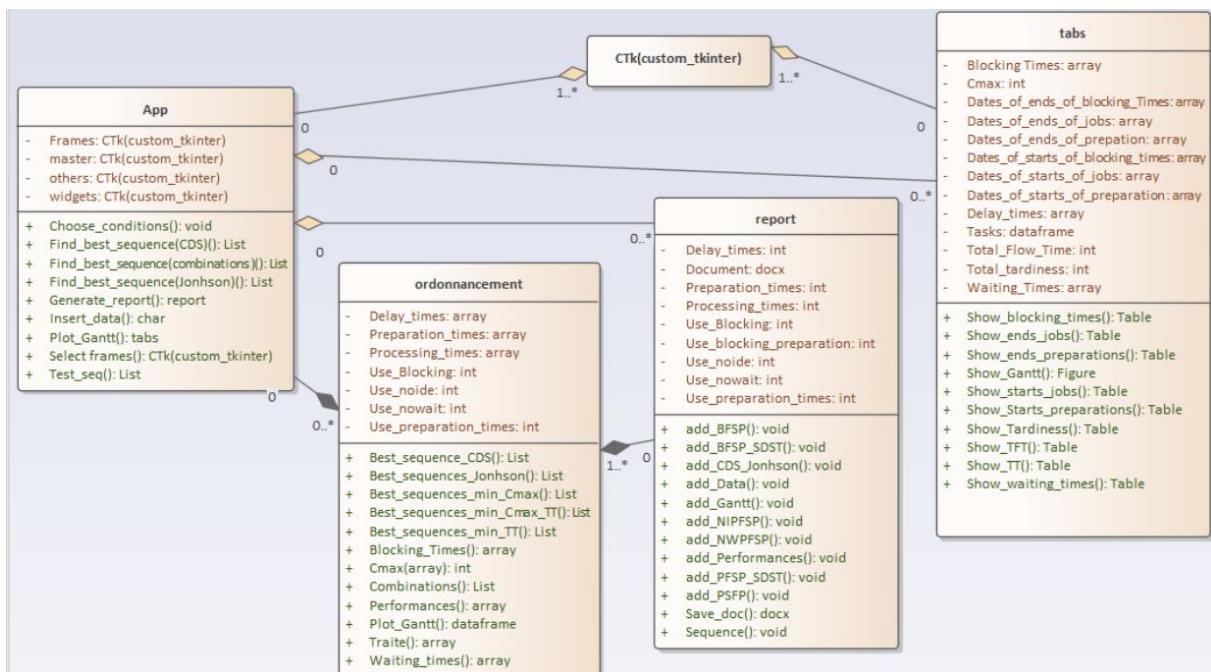
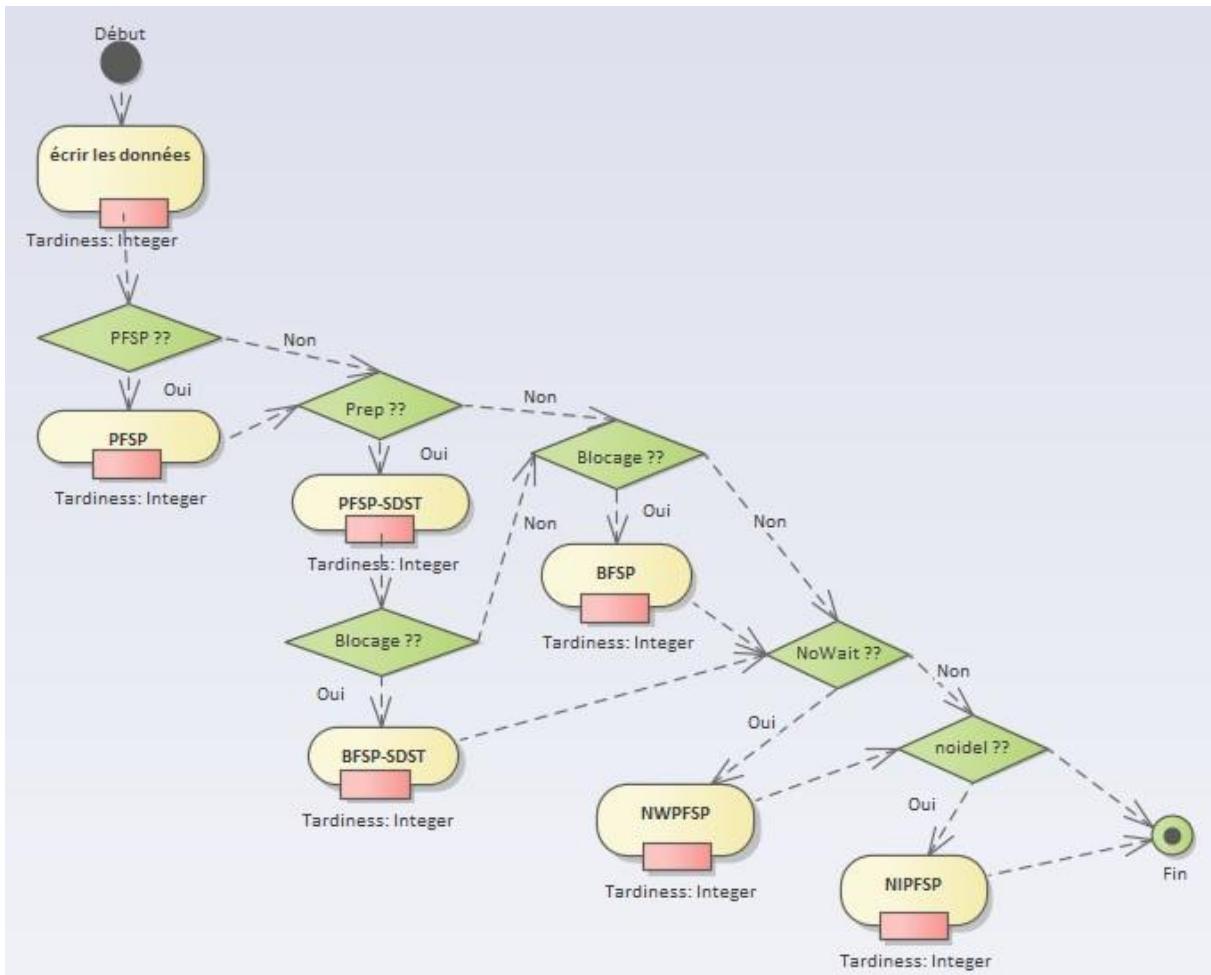


Figure 13 : Diagramme de classes

## 3. Traitement des contraintes du problème d'ordonnancement

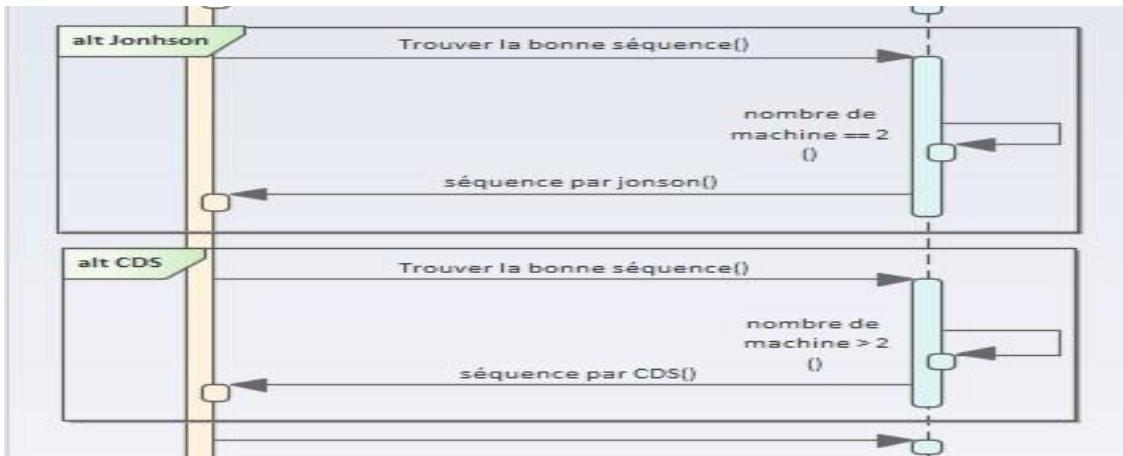
Le diagramme d'activité décrit le processus de génération de rapport pour les problèmes de Flow Shop. Il débute par la collecte des données nécessaires et passe ensuite par une vérification pour identifier le type de problème de Flow Shop à traiter. En fonction de la contrainte spécifique, comme la préparation, le blocage total, la tardiness, le no wait, ou le no idle, le processus suit des chemins distincts pour traiter chaque cas. Une fois le problème spécifique résolu, le rapport est généré en incluant les résultats et les métriques de performance. Le processus se termine avec succès, prêt à présenter ou à sauvegarder le rapport généré.



*Figure 14: Diagramme d'activité*

#### 4. Choix entre l'algorithme de « Johnson » et « CDS »

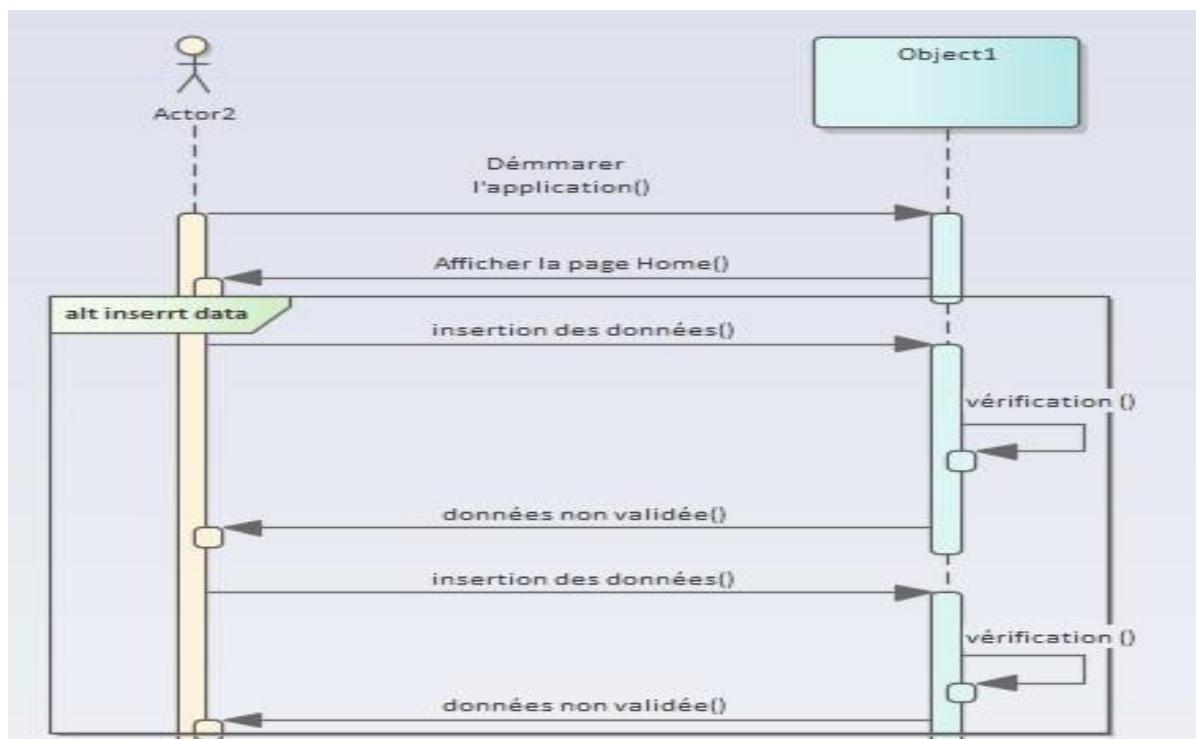
Le diagramme de séquence décrit comment l'application réagit lorsque l'utilisateur clique sur le bouton "Trouver la Bonne Séquence". L'application vérifie d'abord le nombre de machines dans le problème d'ordonnancement. Si le nombre est égal à 2, elle applique directement l'algorithme de Johnson pour générer la séquence optimale. Sinon, si le nombre de machines est supérieur à 2, l'application opte pour l'algorithme CDS. Dans les deux cas, la séquence optimale est ensuite affichée à l'utilisateur. Ce processus dynamique permet à l'application de choisir la meilleure approche en fonction du contexte du problème d'ordonnancement.



*Figure 15: Diagramme de séquence*

## 5. Manipulation de l'application par l'utilisateur

Le diagramme de séquence décrit le processus lorsqu'un utilisateur démarre l'application. Après l'affichage de l'écran d'accueil, l'utilisateur saisit des données. L'application vérifie ces données, et si elles sont valides, elles sont traitées avec succès. En cas d'erreurs dans les données, l'application affiche un message d'erreur et guide l'utilisateur pour effectuer les corrections nécessaires, répétant le processus de vérification jusqu'à ce que les données soient correctes. En fin de compte, le traitement des données s'effectue sans erreurs.



*Figure 16 : Diagramme de séquence*

## 6. Génération de la séquence optimale

Le diagramme d'activité détaille le processus de génération de la meilleure séquence en prenant en compte diverses conditions et contraintes. Il démarre en vérifiant la présence de conditions spécifiques telles que "No Idle" et "No Wait". Si ces conditions ne s'appliquent pas, il examine la contrainte de préparation, distinguant entre le blocage et l'absence de blocage. En appliquant ces contraintes, le processus détermine la séquence optimale tout en effectuant les calculs de performances nécessaires. Enfin, la séquence optimale, accompagnée des performances calculées, est affichée à l'utilisateur. Le processus se termine une fois que la séquence optimale est générée avec succès.

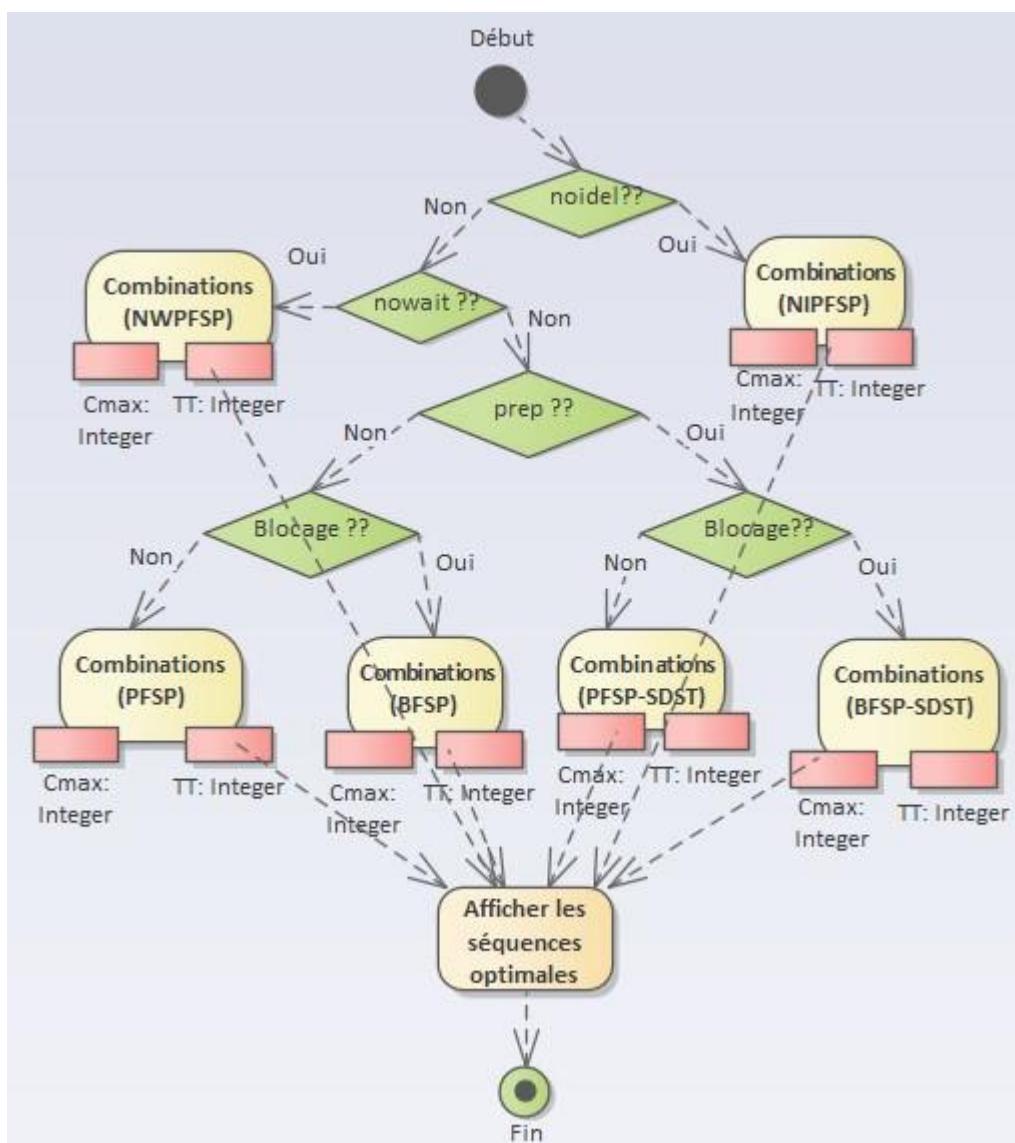


Figure 17 : Diagramme d'activité en fonction de contraintes

## 7. Choix de séquence optimale pour « Flow Shop » à m machines

Le diagramme d'activité décrit le processus de génération et de sélection de la séquence optimale en fonction des performances ( $C_{max}$ ,  $TT$ , ou les deux) pour un nombre donné de jobs. Il débute par la génération de toutes les combinaisons possibles, puis teste chaque combinaison en enregistrant les performances ( $C_{max}$ ) pour chacune. Une fois toutes les combinaisons testées, le processus choisit la séquence optimale en fonction des performances spécifiées. La séquence optimale est ensuite affichée à l'utilisateur, marquant la fin du processus.

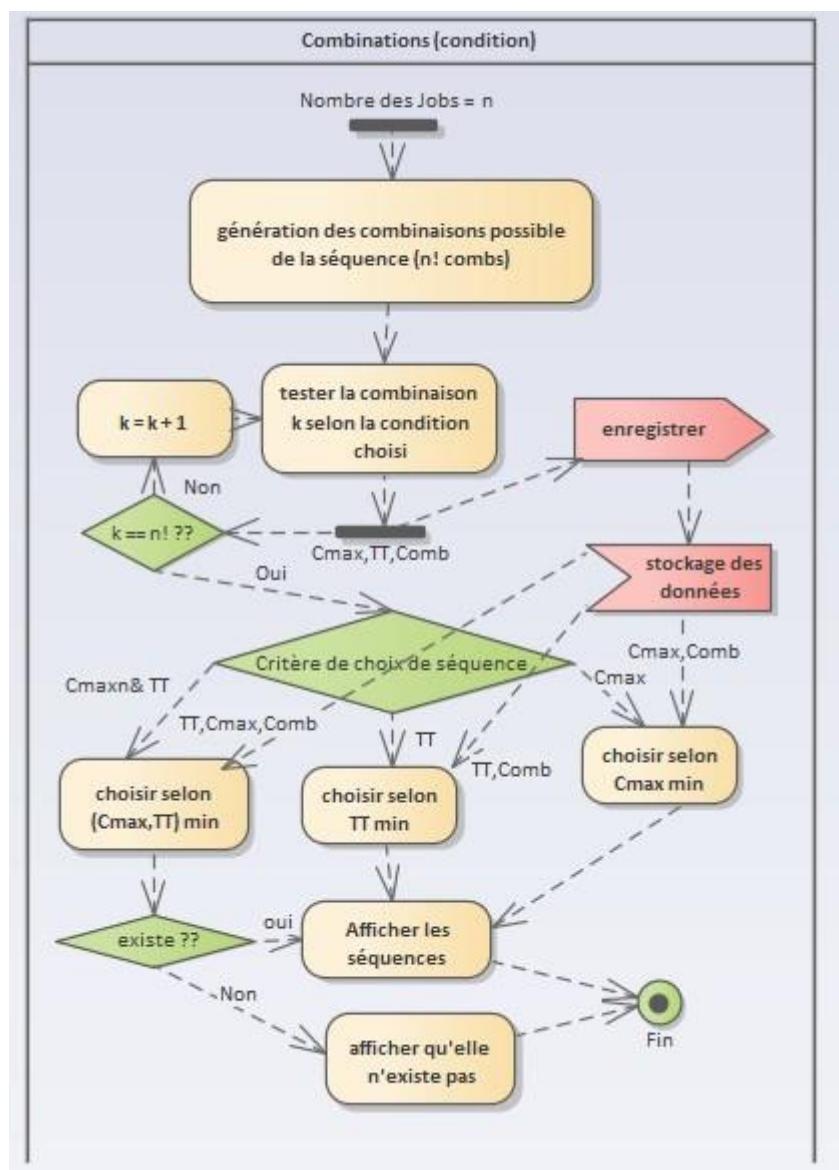


Figure 18 : Diagramme d'activité pour problème à plusieurs machines

## II. Code Python de l'application

### 1. Méthode pour traiter le problème « Flow Shop »

Cette méthode effectue le traitement principal des données, pour résoudre le problème de planification, selon différentes conditions. Elle commence par initialiser quatre listes vides : self.O, self.OP, self.F, et self.FP. Ces listes seront utilisées pour stocker les dates de fin (O), les dates de fin des temps de préparation (OP), les dates de début des jobs (F), et les dates de début des temps de préparation (FP). Ensuite elle traite les conditions de Non-Arrêt (noidel) et les conditions de Non-Attente (nowait) :

- Si **self.noidel = 1**, la méthode appelle une fonction externe **noidel\_probleme\_solving** avec certains paramètres, et le résultat est renvoyé.
- Si **self.nowait = 1**, la méthode appelle une fonction externe **nowait\_probleme\_solving** avec certains paramètres, et le résultat est renvoyé

```
28     def traite (self):  
29         # initialisation de variables  
30         self.O,self.OP,self.F,self.FP=[],[],[],[]  
31         # avec condition de non arret  
32         if self.noidel == 1:  
33             return noidel_probleme_solving(self.M,self.S,self.dj,self.TT)  
34         # avec condition de non attendre  
35         if self.nowait == 1:  
36             return nowait_probleme_solving(self.M,self.S,self.dj,self.TT)
```

Dans le même contexte, la méthode traite tout les cas du problème Flow Shop commençant par :

- Sans Préparation et Sans Blocage (**self.prep == 0** et **self.block == 0**):

On initialise les listes self.O et self.F avec des zéros. On introduit 4 équations pour calculer les dates de fin (O) et de début (F) des jobs en utilisant la matrice des temps de processus (self.M) et la séquence des jobs (self.S).

Si **self.TT =1**, elle calcule également le Total Flow Time (TFT), les temps de retard d'un job (Tm), et le Total Tardiness (TT).

```

38     if self.prep == 0 and self.block == 0:
39         # initialisation
40         for i in range(len(self.M)):
41             self.O.append([0.0]*len(self.M[0]))
42             self.F.append([0.0]*len(self.M[0]))
43         # equation 1
44         self.O[0][0] = self.M[0][self.S[0]]
45         # equation 3
46         for i in range(1,len(self.M)):
47             self.O[i][0] = self.O[i-1][0]+self.M[i][self.S[0]]
48             self.F[i][0] = self.O[i-1][0]
49         # equation 2
50         for j in range(1,len(self.S)):
51             self.O[0][j] = self.O[0][j-1]+self.M[0][self.S[j]]
52             self.F[0][j] = self.O[0][j-1]
53         # equation 4
54         for i in range(1,len(self.M)):
55             for j in range(1,len(self.S)):
56                 self.O[i][j] = max(self.O[i-1][j],self.O[i][j-1])+self.M[i][self.S[j]]
57                 self.F[i][j] = max(self.O[i-1][j],self.O[i][j-1])
58         if self.TT == 1:
59             TFT,Tm,TT=0.0,[],0.0
60             #TFT : total flow time
61             #Tm : temps de retard d'un job "tardiness"
62             #TT : total tardiness
63             for j in range(len(self.M[0])):
64                 TFT+=self.O[-1][j]
65                 Tm.append(max(self.O[-1][j]-self.dj[self.S[j]],0))
66             TT=sum(Tm)
67             # renvoi des dates des fins et des debuts de chaque job
68             return [self.O,self.F,TFT,Tm,TT]
69         else:
70             TFT=0.0
71             for j in range(len(self.M[0])):
72                 TFT+=self.O[-1][j]
73             return [self.O,self.F,TFT]

```

➤ Avec Préparation et Sans Blocage (self.prep == 1 et self.block == 0):

Dans le cas où il y a de la préparation (self.prep == 1) mais pas de blocage (self.block == 0), la méthode traite applique différentes équations pour calculer les dates de fin (O), de début (F), de fin des temps de préparation (OP), et de début des temps de préparation (FP).

- Initialisation des listes : La méthode initialise les listes self.O, self.F, self.OP, et self.FP avec des zéros pour stocker les résultats.
- Équation 1 (Jobs et Préparation) : Pour le premier job (j=0), la date de fin du premier job avec la préparation (O[0][0]) est calculée en ajoutant le temps de processus du premier job (self.M[0][self.S[0]]) et le temps de préparation correspondant (self.Mprep[0][self.S[0]][self.S[0]]). La date de début du premier job avec la préparation (F[0][0]) est égale au temps de préparation correspondant.

- Équation 2 (Jobs et Préparation) :Pour les autres jobs ( $i > 0$ ), la date de fin du job avec la préparation ( $O[i][0]$ ) est calculée en prenant le maximum entre la date de fin du job précédent et le temps de préparation correspondant, puis en ajoutant le temps de processus du job en cours.
- La date de début du job avec la préparation ( $F[i][0]$ ) est égale au maximum entre la date de fin du job précédent et le temps de préparation correspondant.
- Équation 3 (Jobs et Préparation) :Pour la première machine ( $j=0$ ), la date de fin du job avec la préparation ( $O[0][j]$ ) est calculée en ajoutant le temps de processus du premier job sur la première machine ( $self.M[0][self.S[j]]$ ) et le temps de préparation correspondant entre les jobs ( $self.Mprep[0][self.S[j-1]][self.S[j]]$ ). La date de début du job avec la préparation ( $F[0][j]$ ) est égale à la date de fin du job précédent.
- Équation 4 (Jobs et Préparation) : Pour les autres machines ( $j > 0$ ), la date de fin du job avec la préparation ( $O[i][j]$ ) est calculée en prenant le maximum entre la date de fin du job précédent sur la même machine et la date de fin du job sur la machine précédente, puis en ajoutant le temps de processus du job en cours. La date de début du job avec la préparation ( $F[i][j]$ ) est égale au maximum entre la date de fin du job précédent sur la même machine et la date de fin du job sur la machine précédente.

```

75     elif self.prep == 1 and self.block == 0:
76         # initialisation
77         for i in range(len(self.M)):
78             self.O.append([0.0]*len(self.M[0]))
79             self.F.append([0.0]*len(self.M[0]))
80             self.OP.append([0.0]*len(self.M[0]))
81             self.FP.append([0.0]*len(self.M[0]))
82         # equation 1
83         self.O[0][0] = self.M[0][self.S[0]]+self.Mprep[0][self.S[0]][self.S[0]]
84         self.F[0][0] = self.Mprep[0][self.S[0]][self.S[0]]
85         self.OP[0][0] = self.F[0][0]
86         # equation 2
87         for i in range(1,len(self.M)):
88             self.O[i][0] = max( self.O[i-1][0] , self.Mprep[i][self.S[0]][self.S[0]] )+self.M[i][self.S[0]]
89             self.F[i][0] = max( self.O[i-1][0] , self.Mprep[i][self.S[0]][self.S[0]] )
90             self.OP[i][0] = self.Mprep[i][self.S[0]][self.S[0]]
91         # equation 3
92         for j in range(1,len(self.S)):
93             self.O[0][j] = self.O[0][j-1]+self.M[0][self.S[j]]+self.Mprep[0][self.S[j-1]][self.S[j]]
94             self.F[0][j] = self.O[0][j-1]+self.Mprep[0][self.S[j-1]][self.S[j]]
95             self.OP[0][j] = self.F[0][j]
96             self.FP[0][j] = self.O[0][j-1]
97         # equation 4
98         for i in range(1,len(self.M)):
99             for j in range(1,len(self.S)):
100                 self.O[i][j] = max(self.O[i-1][j],self.O[i][j-1]+self.Mprep[i][self.S[j-1]][self.S[j]])
101                 self.F[i][j] = max(self.O[i-1][j],self.O[i][j-1]+self.Mprep[i][self.S[j-1]][self.S[j]])
102                 self.OP[i][j] = self.O[i][j-1]+self.Mprep[i][self.S[j-1]][self.S[j]]
```

```

103         self.FP[i][j]= self.O[i][j-1]
104     if self.TT == 1:
105         TFT,Tm,TT=0.0,[],0.0
106         #TFT : total flow time
107         #Tm : temps de retard d'un job "tardiness"
108         #TT : total tardiness
109         for j in range(len(self.M[0])):
110             TFT+=self.O[-1][j]
111             Tm.append(max(self.O[-1][j]-self.dj[self.S[j]],0))
112         TT=sum(Tm)
113         # renvoi des dates des fins et des debuts de chaque job et temp de preparation
114         return [self.O,self.F,self.OP,self.FP,TFT,Tm,TT]
115     else:
116         TFT=0.0
117         for j in range(len(self.M[0])):
118             TFT+=self.O[-1][j]
119         return [self.O,self.F,self.OP,self.FP,TFT]

```

➤ Sans Préparation et Avec Blocage (self.prep == 0 et self.block == 1):

La méthode traite effectue le calcul des dates de fin (O) et de début (F) des jobs, ainsi que des temps de blocage (TB), des dates de début des temps de blocage (TBF), et des dates de fin des temps de blocage (TBO) :

- Initialisation des structures de données : La méthode initialise une matrice self.D qui représente les dates de fin des jobs avec leur blocage, et des listes pour stocker les résultats tels que self.F, self.DN, self.TB, self.O, self.TBO, et self.TBF.
- Équation 1 (Jobs et Blocage) : la première colonne de self.D est remplie en ajoutant les temps de processus des jobs successifs sur la même machine.
- Équation 2, 3, 4 (Jobs et Blocage) : pour les colonnes suivantes, le calcul est effectué en prenant en compte les temps de processus des jobs et les temps de blocage entre les jobs sur la même machine ou entre différentes machines. Cela permet de déterminer les dates de fin des jobs avec leur blocage (self.D[i][j]).
- Extraction des données : les données nécessaires, telles que les dates de fin des jobs (self.F), les nouvelles dates de début des jobs (self.DN), les temps de blocage (self.TB), les dates de début des temps de blocage (self.TBF), et les dates de fin des temps de blocage (self.TBO), sont extraites de la matrice self.D.

Les résultats sont stockés dans les listes correspondantes et peuvent être utilisés pour obtenir les informations sur le temps d'exécution des jobs, les temps de blocage, et pour calculer le

Total Flow Time (TFT), les temps de retard d'un job (Tm), et le Total Tardiness (TT) dans le cas où self.TT=1

```

121     elif self.prep == 0 and self.block == 1:
122         # self.D est la matrice des fins des jobs avec leur blockage qui nous donne plusieurs données
123         # initialisation et équation 1
124         self.D = np.array([[0.0]*len(self.M[0])]*(len(self.M)+1))
125         # équation 2
126         for i in range(1,len(self.M)):
127             self.D[i][0]=self.D[i-1][0]+self.M[i-1][self.S[0]]
128             # équation 5
129             self.D[len(self.M)][0]=self.D[len(self.M)-1][0]+self.M[len(self.M)-1][self.S[0]]
130             # équation 2,3,4
131             for j in range(1,len(self.D[0])):
132                 for i in range(len(self.D)):
133                     if i == 0:# équation 3
134                         self.D[0][j]=self.D[1][j-1]
135                     elif i!=0 and i!=len(self.M):# équation 4
136                         self.D[i][j]=max(self.D[i-1][j]+self.M[i-1][self.S[j]],self.D[i+1][j-1])
137                     else:# équation 5
138                         self.D[i][j]=self.D[len(self.M)-1][j]+self.M[len(self.M)-1][self.S[j]]
139
140             self.F = np.array([[0.0]*len(self.M[0])]*len(self.M)) # dates de débuts des jobs
141             self.DN= np.array([[0.0]*len(self.M[0])]*len(self.M)) # nouvelle matrice D
142             self.TB= np.array([[0.0]*len(self.M[0])]*len(self.M)) # temps de blockage
143             self.O = np.array([[0.0]*len(self.M[0])]*len(self.M)) # dates des fins des jobs
144             self.TBO = np.array([[0.0]*len(self.M[0])]*len(self.M)) # dates de fins des temps de blockage
145             self.TBF = np.array([[0.0]*len(self.M[0])]*len(self.M)) # dates des débuts des temps de blockage
146             # extraction des données depuis la matrice D
147             for i in range(len(self.D)-1):
148                 for j in range(len(self.D[0])):
149                     self.F[i][j]=self.D[i][j]
150                     self.DN[i][j]=self.D[i+1][j]
151                     self.TB[i][j]=self.DN[i][j]-self.F[i][j]-self.M[i][self.S[j]]
152                     self.O[i][j]=self.DN[i][j]-self.TB[i][j]
153                     if self.TB[i][j]!=0:
154                         self.TBO[i][j]=self.DN[i][j]
155                         self.TBF[i][j]=self.O[i][j]
156             if self.TT == 1:
157                 TFT,Tm,TT=0.0,[],0.0
158                 #TFT : total flow time
159                 #Tm : temps de retard d'un job "tardiness"
160                 #TT : total tardiness
161                 for j in range(len(self.M[0])):
162                     TFT+=self.D[-1][j]
163                     Tm.append(max(self.D[-1][j]-self.dj[self.S[j]],0))
164                     TT+=sum(Tm)
165                     # renvoie des dates des fins et des débuts de chaque job et temp de blockage
166                     return [self.O,self.F,self.TBO,self.TBF,self.TB,TFT,Tm,TT]
167             else:
168                 TFT=0.0
169                 for j in range(len(self.M[0])):
170                     TFT+=self.O[-1][j]
171             return [self.O,self.F,self.TBO,self.TBF,self.TB,TFT]

```

➤ Avec Préparation et Avec Blocage (self.prep == 1 et self.block == 1):

La méthode appelle une fonction externe block\_prep pour effectuer le calcul des dates de fin (O), de début (F), des dates de fin des temps de préparation (OP), des dates de début des temps de préparation (FP), des temps de blocage (TB), des dates de début des temps de blocage (TBF), et des dates de fin des temps de blocage (TBO) suivant les étapes suivantes :

```

173     elif self.prep == 1 and self.block == 1:
174         return block_prep(self.M,list(self.S).copy(),self.Mprep,self.dj,self.TT)

```

## 2. Méthode pour calculer les temps réels de fonctionnement TFR et d'arrêt TAR

Cette méthode permet de calculer les performances des machines, il s'agit du taux de fonctionnement réel et du taux d'arrêt ainsi que l'extraction  $C_{max}$

- Initialisation des listes TFR et TAR : Deux listes vides sont créées pour stocker les résultats des calculs.

```
176     def TFR_TAR(self):  
177         TFR , TAR = [] , []
```

- Extraction du makespan  $C_{max}$  : Le makespan est extrait en appelant une fonction Cmax avec la séquence des jobs S comme argument.

```
178     ... , ... , ... , ...  
179     #extraction de makspane  
     Cmax = self.Cmax(self.S)
```

- Calcul de TFR et TAR pour le cas 1 (préparation = arrêt) : Pour chaque machine (i), on calcule le temps de fonctionnement réel (tfr) en sommant les temps de processus des jobs dans la séquence S. Ensuite, on divise tfr par le makespan (Cmax) pour obtenir le ratio de temps de fonctionnement réel (cst). TFR est alors ce ratio, et TAR est simplement 1 - cst.

```
181     for i in range(len(self.M)):  
182         tfr = 0  
183         for j in range(len(self.M[0])):  
184             tfr += self.M[i][self.S[j]]  
185         cst = tfr/Cmax  
186         TFR.append(cst)  
187         TAR.append(1- cst)
```

- Calcul de TFR, TAR, et TAP pour le cas 2 (préparation = travail) : Si la condition prep est égale à 1 (préparation activée), on calcule TFR, TAR et TAP pour chaque machine en considérant les temps de préparation (Mprep). Le calcul est similaire à celui du premier cas, mais avec l'ajout des temps de préparation.

```

189     if self.prep == 1:
190         TFR_2 , TAR_2 , TAP = [] , [] , []
191         for i in range(len(self.M)):
192             tfr , tap = 0 , 0
193             for j in range(len(self.M[0])):
194                 if j != 0:
195                     tfr += self.M[i][self.S[j]]+self.Mprep[i][self.S[j-1]][self.S[j]]
196                     tap += self.Mprep[i][self.S[j-1]][self.S[j]]
197                 else:
198                     tfr += self.M[i][self.S[j]]+self.Mprep[i][self.S[j]][self.S[j]]
199                     tap += self.Mprep[i][self.S[j]][self.S[j]]
200             cst = tfr/Cmax
201             cst2 = tap/Cmax
202             TFR_2.append(cst)
203             TAR_2.append(1- cst)
204             TAP.append(cst2)

```

- Renvoi des résultats : Si la condition préparation égale à l'arrêt (cas 1) , ce qui signifie qu'il n'y a pas de préparation, la méthode renvoie simplement les résultats (TFR et TAR). Cependant si préparation égale au travail (cas 2) la méthode renvoie

```

205     # si cas 2 renvoi tous les cas
206     return [TFR,TAR,TFR_2,TAR_2,TAP]
207     #si non renvoi le cas1
208     return [TFR,TAR]
209     # si cas 2 renvoi tous les cas
210     return [TFR,TAR,TFR_2,TAR_2,TAP]
211     #si non renvoi le cas1
212     return [TFR,TAR]

```

### 3. Méthode pour calculer les temps d'attente

Cette méthode TW (Time Waiting) est utilisée pour calculer différents aspects des temps d'attente dans le contexte du problème « Flow Shop » :

- Équation 0 : Calcul du temps d'attente total (TW) pour chaque job sur chaque machine. Le temps d'attente est la différence entre la fin du job précédent et le début du job en cours, moins la durée du job en cours.

```

216     # équation 0
217     for i in range(len(self.M)-1):
218         TW.append([])
219         for j in range(len(self.M[0])):
220             TW[i].append(trait[0][i+1][j]-self.M[i+1][self.S[j]]-trait[0][i][j])

```

- Équation 1 : Calcul du temps d'attente non arrêt (TWNA) pour la première machine. Cela représente le temps d'attente entre le début du premier job sur la première machine et la fin du dernier job sur la machine précédente.

```

221     # équation 1
222     for j in range(1,len(self.M[0])):
223         TWNA[0].append(max(self.O[1][j-1]-self.O[0][j],0.0))

```

- Équation 2 : Calcul du temps d'attente non arrêt (TWNA) pour les machines suivantes (sauf la dernière). Cela représente le temps d'attente entre le début du premier job sur chaque machine et la fin du dernier job sur la machine précédente.

```
224     # équation 2
225     for i in range(1,len(self.M)-1):
226         TWNA.append([])
227         TWNA[i].append(self.F[i+1][0]-self.O[i][0])
```

- Équation 3 : Calcul du temps d'attente non arrêt (TWNA) pour les machines suivantes (sauf la dernière) et pour chaque job. Cela représente le temps d'attente entre le début du premier job sur chaque machine et la fin du dernier job sur la machine précédente.

```
228     # équation 3
229     for i in range(1,len(self.M)-1):
230         TWNA.append([])
231         for j in range(1,len(self.M[0])):
232             TWNA[i].append(max(0.0,self.O[i+1][j-1]-self.O[i][j]))
```

- Équation 4 : Suppression de la dernière ligne de la matrice TWNA, car elle représente le temps d'attente non-arrêt pour la dernière machine, qui n'est pas nécessaire pour le calcul du temps d'attente en arrêt (TWA) par C'est la différence entre le temps d'attente total (TW) et le temps d'attente non-arrêt (TWNA).

```
233     # équation 4
234     TWNA.remove(TWNA[len(self.M)-1])
235     for i in range(len(self.M)-1):
236         TWA.append([])
237         for j in range(len(self.M[0])):
238             TWA[i].append(TW[i][j]-TWNA[i][j])
239     return [TW,TWA,TWNA]
```

#### 4. Méthode pour tracer le diagramme « Gant »

C'est un code qui génère un diagramme de Gantt pour visualiser l'ordonnancement des tâches sur différentes machines. Voici une explication générale de chaque partie du code :

- Initialisation : Ce code permet d'initier le diagramme de GANTT en stockant plusieurs data à savoir :
- Global color\_names : Ceci déclare une liste globale color\_names qui semble être utilisée pour stocker les couleurs des blocs de temps sur le diagramme de Gantt (sans préparation ou avec noidel/nowait).
  - Global color\_names\_prep : De manière similaire, cela déclare une autre liste globale color\_names\_prep pour stocker les couleurs des blocs de temps avec préparation.

- Matrix = self.traite() : Appelle la méthode traite de l'objet self (qui doit être une instance de quelque chose) pour obtenir des données traitées et les stocke dans la variable Matrix.
- If type(Matrix)==str: return Matrix : Vérifie si le type de Matrix est une chaîne de caractères (probablement utilisé pour signaler une erreur dans le traitement). Si c'est le cas, il retourne cette chaîne.
- START, END, Machines, colors, texts = [] , [] , [] , [] , [] : Initialise plusieurs listes vides (START, END, Machines, colors, texts) qui seront utilisées pour stocker les données nécessaires à la construction du diagramme de Gantt.
- Gantt = {'Machines' : None, 'Start' : None, 'End' : None , 'Color' : None, 'Label': None } : Initialise un dictionnaire Gantt avec des clés représentant différentes parties des données nécessaires pour le diagramme de Gantt.

```

241     def Gantt (self):
242         global color_names      # liste des couleur utilusée sans préparaion ou avec noidel ou avec nowai
243         global color_names_prep # liste des couleur utilusée avec préparaion
244         Matrix = self.traite()    # données traitées
245         # s'il y'a une erreur dans le traitement
246         if type(Matrix)==str:
247             return Matrix
248         START , END , Machines , colors , texts= [] , [] , [] , [] , []
249         # START : list des temps de débuts de chaque block soit temps de processus ou de preparaison ou de
250         # END : list des temps de fin de chaque block soit temps de processus ou de preparaison ou de bloca
251         # Machines : list des labels de chaque machine
252         # colors : list des couleur de chaque block soit temps de processus ou de preparaison ou de blocage
253         # noir pour les temps de preparation / gris pour blocage / autres pour les temps de processus
254         # text : labels de chaque block
255         # Jj pour les jobs / TP pour les préparaion / TB pour les blocages
256         Gantt = {'Machines' : None , 'Start' : None,'End' : None , 'Color' : None, 'Label': None }

```

- Construction du dictionnaire Gantt : dictionnaire des données qui sera utilisé pour afficher le diagramme de Gantt. Une boucle « for i in range(len(self.M)-1,-1,-1) » itère sur les indices des machines en sens inverse, c'est-à-dire de la dernière à la première machine. Pour chaque itération de la boucle, un label est créé pour la machine correspondante. Le label est de la forme "Machine i" où i est l'indice de la machine. Ce label est ajouté à la liste Machines.

```

257     # Gantt : dictionnaire des données utilusée pour afficher le gaint
258     for i in range(len(self.M)-1,-1,-1): # remplissage des labels pour les machines
259         Machines.append("Machine "+str(i))

```

- Génération du diagramme de Gantt pour différents cas :

- Sans préparation et sans blocage (nowait ou nodel) : les listes START, END, colors, et texts pour chaque job (J) en utilisant les temps de début et de fin de la matrice Matrix[0] et Matrix[1].

```

261     if (self.prep == 0 and self.block == 0) or self.nodel==1 or self.nowait == 1:
262         if len(Matrix[0][0])> len(color_names):
263             color_names = color_names*((len(Matrix[0][0])/len(color_names))+1)
264         for lst1,lst2,k in zip(Matrix[0],Matrix[1],range(len(self.M))):
265             START.append([])
266             END.append([])
267             colors.append([])
268             texts.append([])
269             for e1,e2,c,i in zip(lst1,lst2,color_names,range(len(lst1))):
270                 START[k].append([e2])
271                 END[k].append([e1])
272                 colors[k].append(c)
273                 texts[k].append("J"+str(self.S[i]))

```

- Avec préparation et sans blocage : il utilise les données de Matrix[0], Matrix[1], Matrix[2], et Matrix[3] pour représenter les temps de préparation (TP) et les jobs.

```

275     elif self.prep == 1 and self.block == 0:
276         M1 , M2 = [] , []
277         for lst1,lst2,lst3,lst4 in zip(Matrix[0],Matrix[1],Matrix[2],Matrix[3]):
278             lst5 , lst6 = lst1+lst3 , lst2+lst4
279             lst5.sort()
280             lst6.sort()
281             M1.append(lst5)
282             M2.append(lst6)
283         if len(M1[0])> len(color_names_prep):
284             color_names_prep = color_names_prep*((len(Matrix[0][0])/len(color_names_prep))+1)
285         for lst1,lst2,k in zip(M1,M2,range(len(M1))):
286             START.append([])
287             END.append([])
288             colors.append([])
289             texts.append([])
290             for e1,e2,c,i in zip(lst1,lst2,color_names_prep,range(len(lst1))):
291                 START[k].append([e2])
292                 END[k].append([e1])
293                 colors[k].append(c)
294                 if i%2 == 1:
295                     texts[k].append("J"+str(self.S[i//2]))
296                 else:
297                     texts[k].append("TP")

```

- Sans préparation et avec blocage : Il utilise les informations de Matrix[0], Matrix[1], Matrix[2], Matrix[3], et Matrix[4] pour représenter les temps de blocage (TB) et les jobs.

```

299     elif self.prep == 0 and self.block == 1:
300         if len(Matrix[0][0]) > len(color_names):
301             color_names = color_names*((len(Matrix[0][0])/len(color_names))+1)
302         for Jends,Jstarts,TBends,TBstarts,TBs,i in zip(Matrix[0],Matrix[1],Matrix[2],Matrix[3],Matrix[4]):
303             START.append([])
304             END.append([])
305             colors.append([])
306             texts.append([])
307             for Jend,Jstart,TBend,TBstart in zip(Jends,Jstarts,TBends,TBstarts):
308                 START[i].append([float(Jstart)])
309                 END[i].append([float(Jend)])
310                 if TBstart!=0:
311                     START[i].append([float(TBstart)])
312                     END[i].append([float(TBend)])
313             START[i].sort()
314             END[i].sort()
315             index_color = 0
316             colors[i].append(color_names[index_color])
317             texts[i].append("J"+str(self.S[index_color]))
318             index_color+=1
319             for j in range(1,len(START[i])):
320                 if START[i][j] in TBstarts:
321                     colors[i].append('silver')
322                     texts[i].append("TB")
323                 else:
324                     colors[i].append(color_names[index_color])
325                     texts[i].append("J"+str(self.S[index_color]))
326                     index_color+=1

```

- Avec préparation et avec blocage : les temps de préparation, les jobs et les blocages sont pris en compte à l'aide des données de Matrix[0], Matrix[1], Matrix[2], Matrix[3], Matrix[4], et Matrix[5].

```

328     elif self.prep == 1 and self.block == 1:
329         if len(Matrix[0][0]) > len(color_names):
330             color_names = color_names*((len(Matrix[0][0])/len(color_names))+1)
331         for Jends,Jstarts,Pends,Pstarts,TBends,TBstarts,i in zip(Matrix[0],Matrix[1],Matrix[2],Matrix[3],Matrix[4],Matrix[5]):
332             START.append([])
333             END.append([])
334             colors.append([])
335             texts.append([])
336             for Jend,Jstart,Pend,Pstart,TBend,TBstart,j in zip(Jends,Jstarts,Pends,Pstarts,TBends,TBstarts):
337                 print("done")
338                 START[i].append([float(Pstart)])
339                 texts[i].append("TP")
340                 colors[i].append("black")
341                 START[i].append([float(Jstart)])
342                 texts[i].append("J"+str(self.S[j]))
343                 colors[i].append(color_names[j])
344                 END[i].append([float(Pend)])
345                 END[i].append([float(Jend)])
346                 if TBstart!=0:
347                     print("added")
348                     START[i].append([float(TBstart)])
349                     texts[i].append("TB")
350                     colors[i].append('silver')
351                     END[i].append([float(TBend)])

```

- Construction du dictionnaire de Gantt et conversion en DataFrame Pandas : Les listes sont inversées pour obtenir l'ordre chronologique, les données sont organisées dans un dictionnaire Gantt et le dictionnaire est converti en un DataFrame Pandas pour une manipulation plus facile.

```

352     START.reverse()
353     END.reverse()
354     texts.reverse()
355     colors.reverse()
356     Gantt['Machines'] = Machines
357     Gantt['Start'] = START
358     Gantt['End'] = END
359     Gantt['Color'] = colors
360     Gantt['Label'] = texts
361     tasks_df = pd.DataFrame(Gantt)
362     tasks_df = tasks_df.set_index('Machines')
363     return tasks_df

```

## 5. Méthode pour identifier le makespan $C_{max}$

Cette fonction Cmax calcule la valeur maximale de la séquence finale traitée. La fonction prend en argument une séquence Seq. Elle attribue la séquence à l'attribut S de l'objet actuel (instance de la classe). Ensuite, elle appelle la méthode traite () pour obtenir la matrice traitée. À partir de cette matrice, elle extrait la première partie de la première liste (matrice), correspondant aux temps de fin des différents jobs sur la dernière machine (dernière colonne de la matrice). Elle utilise ensuite la fonction max () pour trouver la valeur maximale parmi ces temps de fin.

```

365     def Cmax(self,Seq):
366         self.S = Seq
367         return max(self.traite()[0][len(self.M)-1])

```

## 6. Méthode pour extraire le makespan et total tardiness

Cette fonction Cmax\_TT est une extension de la fonction Cmax. Elle retourne à la fois la valeur maximale des temps de fin des jobs sur la dernière machine (Cmax) et la liste des temps totaux de traitement (TT) pour chaque job dans le cas où TT==1. La fonction prend en argument une séquence Seq. Elle attribue la séquence à l'attribut S de l'objet actuel (instance de la classe). Ensuite, elle appelle la méthode traite() pour obtenir la matrice traitée. Elle extrait la première partie de la première liste de la matrice traitée, correspondant aux temps de fin des différents jobs sur la dernière machine (dernière colonne de la matrice). Elle utilise ensuite la fonction max() pour trouver la valeur maximale parmi ces temps de fin. Elle retourne une liste contenant

la valeur maximale trouvée (Cmax) et la dernière liste de la matrice traitée, qui semble représenter les temps totaux de traitement (TT) pour chaque job.

```
369     def Cmax_TT(self,Seq):
370         self.S = Seq
371         donees = self.traite()
372         return [max(donees[0][len(self.M)-1]),donees[-1]] # retourner Cmax et TT
```

## 7. Méthode pour calculer les combinaisons possibles

Cette fonction combinations génère toutes les combinaisons possibles des indices des jobs sur les machines et calculer le Cmax correspondant pour chacune de ces combinaisons. Elle obtient la longueur n de la première machine (self.M[0]), représentant le nombre de jobs. Elle appelle une fonction generate\_combinations(n) pour générer toutes les combinaisons possibles des indices de jobs. Elle initialise une liste vide Cmaxs pour stocker les valeurs de Cmax.

- Si l'attribut TT est égal à 1, elle initialise une liste vide TTs pour stocker les temps totaux de traitement. Ensuite, elle boucle à travers chaque combinaison générée :
  - Si l'attribut TT est égal à 1, elle appelle la fonction Cmax\_TT pour obtenir la valeur de Cmax et les temps totaux de traitement correspondants. Elle ajoute la valeur de Cmax à la liste Cmaxs et les temps totaux de traitement à la liste TTs.
  - Si l'attribut TT n'est pas égal à 1, elle appelle la fonction Cmax pour obtenir la valeur de Cmax et l'ajoute à la liste Cmaxs.

Elle retourne une liste contenant les valeurs de Cmax, les combinaisons générées, et éventuellement les temps totaux de traitement si l'attribut TT est égal à 1.

## 8. Méthode pour définir les combinaisons où le makespan est égal à sa valeur minimale

Cette fonction « choosed\_combinations » sélectionne la combinaison qui minimise la valeur de Cmax parmi toutes les combinaisons générées.

- Elle appelle la fonction combinations pour obtenir la liste des valeurs de Cmax et des combinaisons générées.

- Puis, elle convertit la liste des valeurs de Cmax en un tableau NumPy.
- Elle utilise la fonction np.where pour trouver les indices où la valeur de Cmax est minimale.
- Elle initialise une liste vide choosed pour stocker les combinaisons correspondant à la valeur minimale de Cmax.
- Elle boucle à travers les indices trouvés et ajoute les combinaisons correspondantes à la liste choosed.
- Elle retourne une liste contenant la valeur minimale de Cmax et les combinaisons correspondantes.

```

374     def combinations (self) :
375         n = len(self.M[0])
376         combinations = generate_combinations(n)
377         Cmaxs = []
378         if self.TT ==1:
379             TTs = []
380             for comb in combinations:
381                 cmax_tt = self.Cmax_TT(comb)
382                 Cmaxs.append(cmax_tt[0])
383                 TTs.append(cmax_tt[1])
384             return [Cmaxs,combinations,TTs]
385         else:
386             for comb in combinations:
387                 Cmaxs.append(self.Cmax(comb))
388             return [Cmaxs,combinations]
```

## 9. Méthode pour calculer les combinaisons qui donne la valeur minimale du total tardiness

Cette fonction choosed\_combinations\_by\_TT est similaire à la fonction précédente, mais elle sélectionne la combinaison qui minimise le temps total (TT) plutôt que la valeur de Cmax.

- Elle appelle la fonction combinations pour obtenir la liste des valeurs de Cmax, des combinaisons générées et des valeurs de TT.
- Elle convertit la liste des valeurs de TT en un tableau NumPy.

- Elle utilise la fonction np.where pour trouver les indices où la valeur de TT est minimale.
- Elle initialise une liste vide choosed pour stocker les combinaisons correspondant à la valeur minimale de TT.
- Elle boucle à travers les indices trouvés et ajoute les combinaisons correspondantes à la liste choosed.
- Elle retourne une liste contenant la valeur minimale de TT et les combinaisons correspondantes.

```

399     def choosed_combinations_by_TT(self):
400         lst = self.combinations()
401         TTs = np.array(lst[2])
402         indexs = np.where( TTs == min(TTs))
403         choosed = []
404         for i in indexs[0]:
405             choosed.append(lst[1][i])
406         return [min(TTs),choosed]

```

## 10. Méthode pour calculer les combinaisons où $TT = \min(TTs)$ et $Cmax = \min(Cmaxs)$

Cette fonction choosed\_combinations\_by\_TT semble être similaire à la fonction précédente, mais elle sélectionne la combinaison qui minimise le temps total (TT) plutôt que la valeur de Cmax.

- Elle appelle la fonction combinations pour obtenir la liste des valeurs de Cmax, des combinaisons générées et des valeurs de TT.
- Elle convertit la liste des valeurs de TT en un tableau NumPy.
- Elle utilise la fonction np.where pour trouver les indices où la valeur de TT est minimale.
- Elle initialise une liste vide choosed pour stocker les combinaisons correspondant à la valeur minimale de TT.

- Elle boucle à travers les indices trouvés et ajoute les combinaisons correspondantes à la liste choosed.
- Elle retourne une liste contenant la valeur minimale de TT et les combinaisons correspondantes.

```

408     def choosed_combinations_by_TT_Cmax(self):
409         lst = self.combinations()
410         TTs = np.array(lst[2])
411         Cmaxs = np.array(lst[0])
412         indexes_tt = np.where( TTs == min(TTs))
413         indexes_cmax = np.where(Cmaxs == min(Cmaxs))
414         choosed_by_cmax = []
415         choosed_by_tt = []
416         for i in indexes_cmax[0]:
417             choosed_by_cmax.append(lst[1][i])
418         for i in indexes_tt[0]:
419             choosed_by_tt.append(lst[1][i])
420         choosed = []
421         for comb in choosed_by_tt:
422             if comb in choosed_by_cmax:
423                 choosed.append(comb)
424         if choosed!=[]:
425             return [(min(TTs),min(Cmaxs)),choosed]
426         else:
427             return "il n'y a pas de séquence vérifiant min(TT) et min(Cmax)"

```

## 11. Méthode pour calculer la séquence optimale dans le cas de deux machines

Cette fonction semble traiter les problèmes flowshop avec des conditions de blocage, y compris les temps de préparation. Elle retourne diverses informations liées au problème, notamment les dates de début et de fin des travaux, les temps de préparation, les temps de blocage, et d'autres données. Voici une explication des parties principales du code :

La fonction prend en paramètres des listes P, S, TP, dj et T :

P : matrice de temps de traitement des travaux.

S : séquence des travaux.

TP : matrice de temps de préparation entre les travaux.

dj : liste des dates d'échéance des travaux.

T : un paramètre booléen qui indique si le calcul des retards (tardiness) doit être effectué.

- La fonction commence par préparer les données, ajuste les indices, ajoute des lignes et colonnes de zéros à la matrice de traitement P.
- Ensuite, elle calcule les dates de début et de fin des travaux, les temps de préparation, les temps de blocage, etc.
- La partie finale de la fonction retourne une liste d'informations, notamment les dates de début et de fin des travaux, les temps de préparation, les temps de blocage, le temps total de flow, et d'autres informations selon la valeur du paramètre T.

```

429     def Johnson(self): #(Algorithme de jonhson)
430         if len(self.M) != 2 :
431             print("Jonhson Algorithme ne fonctionne qu'avec deux machines")
432             return None
433         # etape 1 et 2
434         jobs = list(range(len(self.M[0])))
435         U,V = [],[]
436         for j,t1,t2 in zip(jobs,self.M[0],self.M[1]):
437             if t1 < t2:
438                 U.append(j)
439             else:
440                 V.append(j)
441         X,Y = U.copy(),V.copy()
442         # etape 3
443         SPT,LPT,M1,M2 = [],[],[],[]
444         for j in U:
445             M1.append(self.M[0][j])
446         for j in V:
447             M2.append(self.M[1][j])
448         x = 0
449         n= len (U)
450         MA,MB = [],[]
451         while 1:
452             if len(SPT)==n:
453                 break
454             if M1[x] == min(M1):
455                 SPT.append(U[x])
456                 MA.append(M1[x])
457                 M1.remove(M1[x])
458                 U.remove(U[x])
459                 x=0
460             else:
461                 x+=1
462         x = 0
463         n= len (V)
464         while 1:
465             if len(LPT)==n:
466                 break
467             if M2[x] == max(M2):
468                 LPT.append(V[x])
469                 MB.append(M2[x])
470                 M2.remove(M2[x])
471                 V.remove(V[x])
472                 x=0
473             else:
474                 x+=1
475         print(SPT,LPT)
476         # etape 4
477         A1=find_combinations(SPT, MA)
478         A2=find_combinations(LPT, MB)
479         RESULT =combine_lists(A1,A2)
480         return [X,Y,SPT,LPT,RESULT]

```

## 12. Méthode pour calculer la séquence optimale dans le cas de plusieurs machines

Cette fonction CDS (Campbell, Dudek et Smith) est une extension de l'algorithme de Johnson pour résoudre des problèmes de « Flow Shop » avec plus de deux machines.

- Si le nombre de machines est inférieur ou égal à 2, la fonction imprime un message et renvoie None. Si le nombre de machines est égal à 2, elle applique l'algorithme de Johnson (la fonction self.Johnson()) et retourne le résultat.
- Si le nombre de machines est supérieur à 2, la fonction construit une nouvelle matrice Matrix en additionnant les temps de traitement des jobs sur les premières k machines dans la première ligne de Matrix et les temps de traitement des jobs sur les dernières machines (à partir de la k+1ème) dans la deuxième ligne de Matrix.
- La matrice self.M est ensuite mise à jour avec la nouvelle matrice Matrix et l'algorithme de Johnson est appliqué sur cette nouvelle matrice.
- La fonction retourne le résultat de l'algorithme de Johnson appliqué à la matrice modifiée self.M.

```
482     def CDS(self,k): # (CAMPBELL, DUDEK et SMITH)
483         if len(self.M) <= 2 :
484             print("CDS ne fonctionne qu'avec plus de deux machines")
485             return None
486         if len(self.M) == 2 :
487             print("on va appliquer Jonhson car il y'a deux machines ")
488             return self.Johnson()
489         Matrix = [[],[]]
490         for j in range(len(self.M[0])):
491             som = 0
492             for i in range(k):
493                 som = som + self.M[i][j]
494             Matrix[0].append(som)
495         for j in range(len(self.M[0])):
496             som = 0
497             for i in range(len(self.M)-k,len(self.M)):
498                 som = som + self.M[i][j]
499             Matrix[1].append(som)
500         self.M = Matrix
501         return self.Johnson()
```

### 13.Fonction qui traite les problèmes flowshop avec condition de blocage avec les temps de préparation

Cette fonction `block_prep` semble implémenter un modèle pour résoudre des problèmes de flowshop avec des contraintes de blocage et des temps de préparation. Voici une explication de la fonction :

- Préparation des données : Les indices de la liste S sont incrémentés de 1 (pour s'adapter à une indexation à partir de 1). La matrice P est modifiée en ajoutant une ligne de zéros au début (représentant le temps de préparation initial). La matrice P est modifiée en ajoutant une colonne de zéros à gauche (représentant le temps de préparation initial).
- Calcul des dates de fin (O) et de début (F) des jobs : Les équations 3, 4 et 5 sont utilisées pour calculer les dates de fin et de début des jobs.
- Calcul des dates de fin (OP) et de début (FP) des temps de préparation : Les dates de fin et de début des temps de préparation sont calculées.
- Calcul des dates de fin (OB) et de début (FB) des temps de blocage :Les dates de fin et de début des temps de blocage sont calculées.
- Calcul de la partie supérieure droite de la matrice D (OOB) :La partie supérieure droite de la matrice D est extraite.
- Calcul du temps de blocage (TB) :Le temps de blocage pour chaque job est calculé.
- Calcul des temps de blocage dus à la préparation (TBP), à d'autres blocages (TBB) et au travail (TBF) : Les temps de blocage sont décomposés en temps de blocage dus à la préparation, à d'autres blocages et au travail.
- Calcul du Total Flow Time (TFT) :Le temps total d'écoulement des jobs est calculé.

- Calcul du Total Tardiness (TT) : Le retard total et le Total Tardiness sont calculés si l'argument T est égal à 1. La fonction retourne une liste contenant différentes informations sur la planification, notamment les dates de fin et de début des jobs, des temps de préparation et des temps de blocage. Si T est égal à 1, elle inclut également des informations sur le Total Flow Time, le retard de chaque job et le Total Tardiness.

```

503 def block_prep(P,S,TP,dj,T):
504     #préparation des variables et équation 1 et 2 "adéquation des systèmes d'indexation"
505     for i in range(len(S)):
506         S[i]=S[i]+1
507     S = np.array(S)
508     S = np.hstack(([0], S))
509     new_line = [0.0]*len(P[0])
510     P = np.vstack((new_line, P))
511     new_column = [[0.0]]*len(P)
512     P = np.hstack((new_column,P))
513     D = np.array([[0.0]*(len(P[0])+1)]*(len(P)))
514     #équation 3 4 et 5
515     for j in range(1,len(D[0])):
516         for i in range(len(D)):
517             if i == 0:
518                 # équation 5
519                 if j == 1:
520                     D[0][j] = D[1][j-1] + TP[0][S[1]-1][S[1]-1]
521                 else:
522                     D[0][j] = D[1][j-1] + TP[0][S[j-1]-1][S[j]-1]
523             elif i == len(D)-1:
524                 D[-1][j] = D[-2][j] + P[-1][S[j]]
525             else:
526                 # équation 3
527                 if j == 1:
528                     D[i][1] = max(D[i-1][1] + P[i][S[1]], D[i+1][0] + TP[i][S[1]-1][S[1]-1])
529                 # équation 4
530                 else:
531                     D[i][j] = max(D[i-1][j] + P[i][S[j]], D[i+1][j-1] + TP[i][S[j-1]-1][S[j]-1])
532     O,F,OP,FP,OB,FB,OOB,TB = [],[],[],[],[],[],[],[]
533     # O: dates des fins des jobs (out of job)
534     # F: dates des debuts des jobs (first of job)
535     # OP : dates des fins des temps de préparation (out of preparation time)
536     # FP : dates des debuts des temps de préparation (firt of preparation time)
537     # FB : dates des debuts des temps de blocage (first of blocing time)
538     # OB : dates des find des temps de blocage (first of job blocking time)
539     # OOB : partie supérieur à droite de D
540     # TB : temps de blocage (blocking time)
541     #préparation pour le calcul
542     for i in range(len(D)):
543         Flst,FPlst,OOBlst=[],[],[]
544         for j in range(len(D[0])):
545             if i != 0 and j != len(D[0])-1:
546                 FPlst.append(D[i][j])
547             if i != len(D)-1 and j != 0:
548                 Flst.append(D[i][j])
549             if i !=0 and j !=0:
550                 OOBlst.append(D[i][j])
551         if i != 0 :
552             FP.append(FPlst)
553             OOB.append(OOBlst)
554         if i != len(D)-1:
555             F.append(Flst)
556     # calcul des matrices nécessaire pour tracer le gant
557     for i in range(len(F)):
558         O.append([])
559         OP.append([])
560         TB.append([])
```

```

561     OB.append([])
562     FB.append([])
563     for j in range(len(F[0])):
564         O[i].append(F[i][j]+P[i+1][S[j+1]])
565         if j == 0:
566             OP[i].append(FP[i][0]+TP[i][S[1]-1][S[1]-1])
567         else:
568             OP[i].append(FP[i][j]+TP[i][S[j]-1][S[j+1]-1])
569         TB[i].append(OOB[i][j]-O[i][j])
570         if TB[i][j] == 0:
571             OB[i].append(0.0)
572             FB[i].append(0.0)
573         else:
574             FB[i].append(O[i][j])
575             if j != len(F[0])-1:
576                 OB[i].append(FP[i][j+1])
577             else:
578                 OB[i].append(O[i][j]+TB[i][j])
579     TBP,TBB,TBF = [],[],[]
580     # TBB : temps de blocage à cause d'un autre blocage
581     # TBP : temps de blocage à cause de préparation
582     # TBF : temps de blocage à cause de travail
583     # calcul des temps de blocage
584     for i in range(len(O)-1):
585         TBP.append([])
586         TBB.append([0.0])
587         TBF.append([0.0])
588         if TB[i][0] != 0:
589             TBP[i].append(TB[i][0])
590         else:
591             TBP[i].append(0.0)
592     for i in range(len(O)-1):
593         for j in range(1,len(O[0])):
594             if TB[i][j] != 0:
595                 if F[i+1][j]-FP[i+1][j] >= TB[i][j]:
596                     TBP[i].append(TB[i][j])
597                     TBF[i].append(0.0)
598                     TBB[i].append(0.0)
599                 else:
600                     TBP[i].append(TP[i+1][S[j]-1][S[j+1]-1])
601                     TBF[i].append(O[i+1][j-1] - O[i][j])
602                     if TB[i+1][j-1] != 0:
603                         TBB[i].append(TB[i+1][j-1])
604                     else:
605                         TBB[i].append(0.0)
606                 else:
607                     TBP[i].append(0.0)
608                     TBB[i].append(0.0)
609                     TBF[i].append(0.0)
610     if T==1:
611         TFT,Tm,TT=0.0,[],0.0
612         #TFT : total flow time
613         #Tm : temps de retard d'un job "tardiness"
614         #TT : total tardiness
615         for j in range(len(P[0])):
616             TFT+=D[-1][j]

```

```

617         if j!=0:
618             Tm.append(max(D[-1][j]-dj[s[j]-1],0))
619             TT+=sum(Tm)
620             return [O,F,OP,FP,OB,FB,TB,TBF,TBB,TBP,TFT,Tm,TT]
621     else:
622         TFT=0.0
623         for j in range(len(P[0])):
624             TFT+=D[-1][j]
625         return [O,F,OP,FP,OB,FB,TB,TBF,TBB,TBP,TFT]
626 # fonction qui traite les problemes flowshop avec condition de non arret
627 def noidele_probleme_solving(P,S,dj,T):
628     L,C,F,TW=[0.0,[],[],[]]
629     # L list des temps mort dans chaque machine et équation 1
630     # C matrice des temps de fin de chaque job
631     # F matrice des temps des début de chaque job
632     # TW matrice des temps d'attente de chaque job entre machine i et i+1
633     n,m=len(P[0]),len(P)
634     # n est le nombre total des jobs et m et le nombre total des machines
635     # application des formules mathématique
636     for i in range(m):
637         C.append([])
638         F.append([])
639         # équation 2
640         if i!=0:
641             maxs = []
642             for k in range(n):
643                 mx1,mx2 = 0,0
644                 for j in range(n-k):
645                     mx1+=P[i-1][s[j]]
646                     for j in range(n-k-1):
647                         mx2+=P[i][s[j]]
648                         maxs.append(mx1-mx2)
649                         L.append(L[i-1]+max(maxs))
650             for j in range(n):
651                 # équation 3
652                 if j==0:
653                     C[i].append(L[i]+P[i][s[j]])
654                     F[i].append(L[i])
655                 # équation 4
656                 else:
657                     C[i].append(C[i][j-1]+P[i][s[j]])
658                     F[i].append(C[i][j-1])
659             # calcul des temps d'attendre de chaque job de la machine i à i+1
660             for i in range(m-1):
661                 TW.append([])
662                 for j in range(n):
663                     TW[i].append(F[i+1][j]-C[i][j])
664             if T==1:
665                 TFT,Tm,TT=0.0,[],0.0
666                 #TFT : total flow time
667                 #Tm :temps de retard d'un job "tardiness"
668                 #TT : total tardiness
669                 for j in range(len(P[0])):
670                     TFT+=C[-1][j]
671                     Tm.append(max(C[-1][j]-dj[s[j]],0))
672                 TT+=sum(Tm)
673                 return [C,F,TW,TFT,Tm,TT]
674             else:
675                 TFT=0.0
676                 for j in range(len(P[0])):
677                     TFT+=C[-1][j]
678                 return [C,F,TW,TFT]

```

## 14.Fonction qui traite les problemes flowshop avec condition de non attendre

La fonction « noidel\_probleme\_solving » résoud un problème de « Flow shop3 sans délais entre les jobs sur chaque machine :

- Initialisation : « L » est une liste des temps morts dans chaque machine , « C » est une matrice des temps de fin de chaque job ,« F » est une matrice des temps de début de chaque job et « TW » est une matrice des temps d'attente de chaque job entre deux machines successives.
- Équation 1 - Calcul des temps morts (L) :La liste « L » des temps morts dans chaque machine est calculée en utilisant l'équation 1. Chaque élément de la liste « L » représente le temps mort accumulé dans la machine correspondante.
- Équation 2 - Calcul des temps de fin (C) et de début (F) :Les matrices « C » et « F » sont calculées en utilisant les équations 2, 3, et 4, qui prennent en compte les temps d'exécution des jobs et les temps morts accumulés.
- Calcul des temps d'attente (TW) : La matrice « TW » est calculée en mesurant les temps d'attente de chaque job entre deux machines successives.
- Calcul du Total Flow Time (TFT) :Le temps total d'écoulement des jobs est calculé.
- Calcul du Total Tardiness (TT) : Le retard total et le Total Tardiness sont calculés si l'argument T est égal à 1.

```

627 def nodel_probleme_solving(P,S,dj,T):
628     L,C,F,TW=[0.0,[],[],[]]
629     # L list des temps mort dans chaque machine et équation 1
630     # C matrice des temps de fin de chaque job
631     # F matrice des temps des début de chaque job
632     # TW matrice des temps d'attente de chaque job entre machine i et i+1
633     n,m=len(P[0]),len(P)
634     # n est le nombre total des jobs et m et le nombre total des machines
635     # application des formules mathématique
636     for i in range(m):
637         C.append([])
638         F.append([])
639         # équation 2
640         if i!=0:
641             maxs = []
642             for k in range(n):
643                 mx1,mx2 = 0,0
644                 for j in range(n-k):
645                     mx1+=P[i-1][S[j]]
646                     for j in range(n-k-1):
647                         mx2+=P[i][S[j]]
648                         maxs.append(mx1-mx2)
649             L.append(L[i-1]+max(maxs))
650         for j in range(n):
651             # équation 3
652             if j==0:
653                 C[i].append(L[i]+P[i][S[j]])
654                 F[i].append(L[i])
655             # équation 4
656             else:
657                 C[i].append(C[i][j-1]+P[i][S[j]])
658                 F[i].append(C[i][j-1])
659     # calcul des temps d'attendre de chaque job de la machine i à i+1
660     for i in range(m-1):
661         TW.append([])
662         for j in range(n):
663             TW[i].append(F[i+1][j]-C[i][j])
664     if T==1:
665         TFT,Tm,TT=0.0,[],0.0
666         #TFT : total flow time
667         #Tm :temps de retard d'un job "tardiness"
668         #TT : total tardiness
669         for j in range(len(P[0])):
670             TFT+=C[-1][j]
671             Tm.append(max(C[-1][j]-dj[S[j]],0))
672             TT+=sum(Tm)
673         return [C,F,TW,TFT,Tm,TT]
674     else:
675         TFT=0.0
676         for j in range(len(P[0])):
677             TFT+=C[-1][j]
678         return [C,F,TW,TFT]

```

## 15.La fonction qui traite les problèmes flowshop avec condition de non attendre

La fonction « nowait\_probleme\_solving » semble résoudre un problème de « Flow Shop » sans attente entre les jobs sur chaque machine :

- Initialisation : « n » est le nombre total de jobs, « m » est le nombre total de machines, « D » est une matrice des délais entre deux jobs  $j_1$  et  $j_2$ , « C » est une matrice des temps de fin de chaque job (au début, elle est simplement la dernière ligne), « F » est une matrice des temps de début de chaque job, « TW » est une matrice des temps d'attente entre deux jobs successifs dans chaque machine.
- Équation 1 - Calcul de la matrice des délais (D) : La matrice D est calculée en utilisant l'équation 1 qui prend en compte les temps d'exécution des jobs sur chaque machine.
- Équation 2 - Calcul des temps de fin (C) et de début (F) : Les temps de fin C et de début F sont calculés en utilisant l'équation 2, qui prend en compte les délais et les temps d'exécution.
- Calcul des temps d'attente (TW) : La matrice TW est calculée en mesurant les temps d'attente entre deux jobs successifs sur chaque machine.
- Calcul du Total Flow Time (TFT) : Le temps total d'écoulement des jobs est calculé.
- Calcul du Total Tardiness (TT) : Le retard total et le Total Tardiness sont calculés si l'argument T est égal à 1.

La fonction retourne une liste contenant différentes informations sur la planification, notamment les matrices des temps de fin et de début, des temps d'attente et le Total Flow Time. Si T est égal à 1, elle inclut également des informations sur le retard de chaque job et le Total Tardines

```

680 def nowait_probleme_solving(P,S,dj,T):
681     n,m=len(P[0]),len(P) # n est le nombre total des jobs et m est le nombre total des machines
682     D = np.array([[0.0]*n]*n) # matrice des delais entre deux jobs j1 et j2
683     C = [] # C matrice des temps de fin de chaque job "au début elle est seulement la dernière ligne"
684     # équation 1 : calcul de la matrice des delais
685     for j1 in range(n):
686         for j2 in range(n):
687             mxs = []
688             for r in range(m):
689                 mx1,mx2=0,0
690                 for k in range(1,r+1):
691                     mx1+=P[k][j1]
692                     for k in range(r):
693                         mx2+=P[k][j2]
694                     mxs.append(mx1-mx2)
695                 D[j1][j2] = P[0][j1] + max(max(mxs),0)
696     # équation 2 : calcul des temps de fins et de début de chaque job dans la dernière machine
697     for j in range(n):
698         v1,v2=0,0
699         for k in range(1,j+1):
700             v1+=D[S[k-1]][S[k]]
701         for k in range(m):
702             v2+=P[k][S[j]]
703         C.append(v1+v2)
704     C=np.array([[0.0]*n]*(m-1)+[C]) # C devien une matrice
705     F=np.array([[0.0]*n]*(m)) # F matrice des temps des début de chaque job
706     for i in range(m-1,-1,-1):
707         for j in range(n):
708             if i!=m-1:
709                 C[i][j]=C[i+1][j]-P[i+1][S[j]]
710                 F[i][j] = C[i][j] - P[i][S[j]]
711     TW=np.array([[0.0]*(n-1)]*(m))
712     # TW : matrice des temps d'attente entre deux jobs successif dans chaque machine
713     for i in range(m):
714         for j in range(n-1):
715             TW[i][j] = F[i][j+1]-C[i][j]
716     if T ==1:
717         TFT,Tm,TT=0.0,[],0.0
718         #TFT : total flow time
719         #Tm : temps de retard d'un job "tardiness"
720         #TT : total tardiness
721         for j in range(len(P[0])):
722             TFT+=C[-1][j]
723             Tm.append(max(C[-1][j]-dj[S[j]],0))
724         TT=sum(Tm)
725         return [C,F,TW,TFT,Tm,TT]
726     else:
727         TFT=0.0
728         for j in range(len(P[0])):
729             TFT+=C[-1][j]
730         return [C,F,TW,TFT]

```

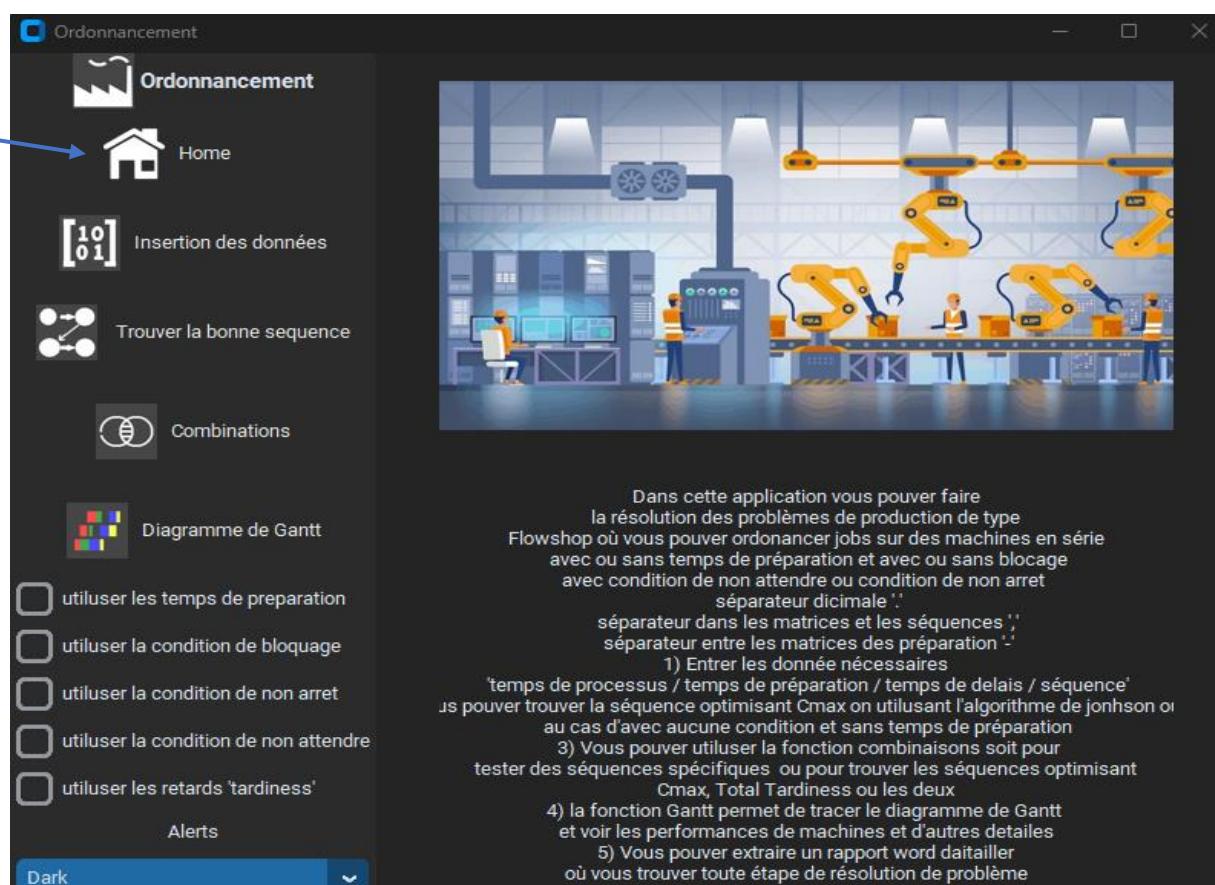
### III. Guide d'utilisation de l'application

#### 1. Présentation de l'application

Notre application d'ordonnancement "Flow Shop" vise à optimiser les processus de production en résolvant efficacement le problème flow shop. Elle offre une solution complète, allant de la génération de matrices de jobs à la création de séquences optimales, la construction du diagramme de Gantt, et l'extraction des métriques clés telles que le Temps d'Attente Total (TAT), le Temps de Flux Réduit (TFR) et le Temps Total de Production (TT). Les fonctionnalités principales de l'application consistent à définir facilement des ensembles de tâches avec des temps d'exécution spécifiques en utilisant des algorithmes d'ordonnancement avancés pour déterminer la séquence optimale des jobs. Cela permet de minimiser les temps d'attente en optimisant l'utilisation des ressources, améliorant ainsi l'efficacité de la production. Or, l'utilisateur entre les paramètres tels que les temps d'exécution des tâches dans l'interface conviviale de l'application.

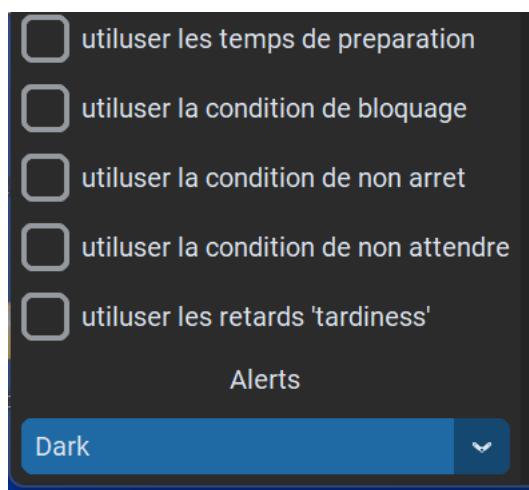
#### 2. Le bouton Home

Le bouton « Home » explique l'utilisation de l'application : c'est un guide qui facilite aux utilisateurs la manipulation de l'application selon leurs besoins



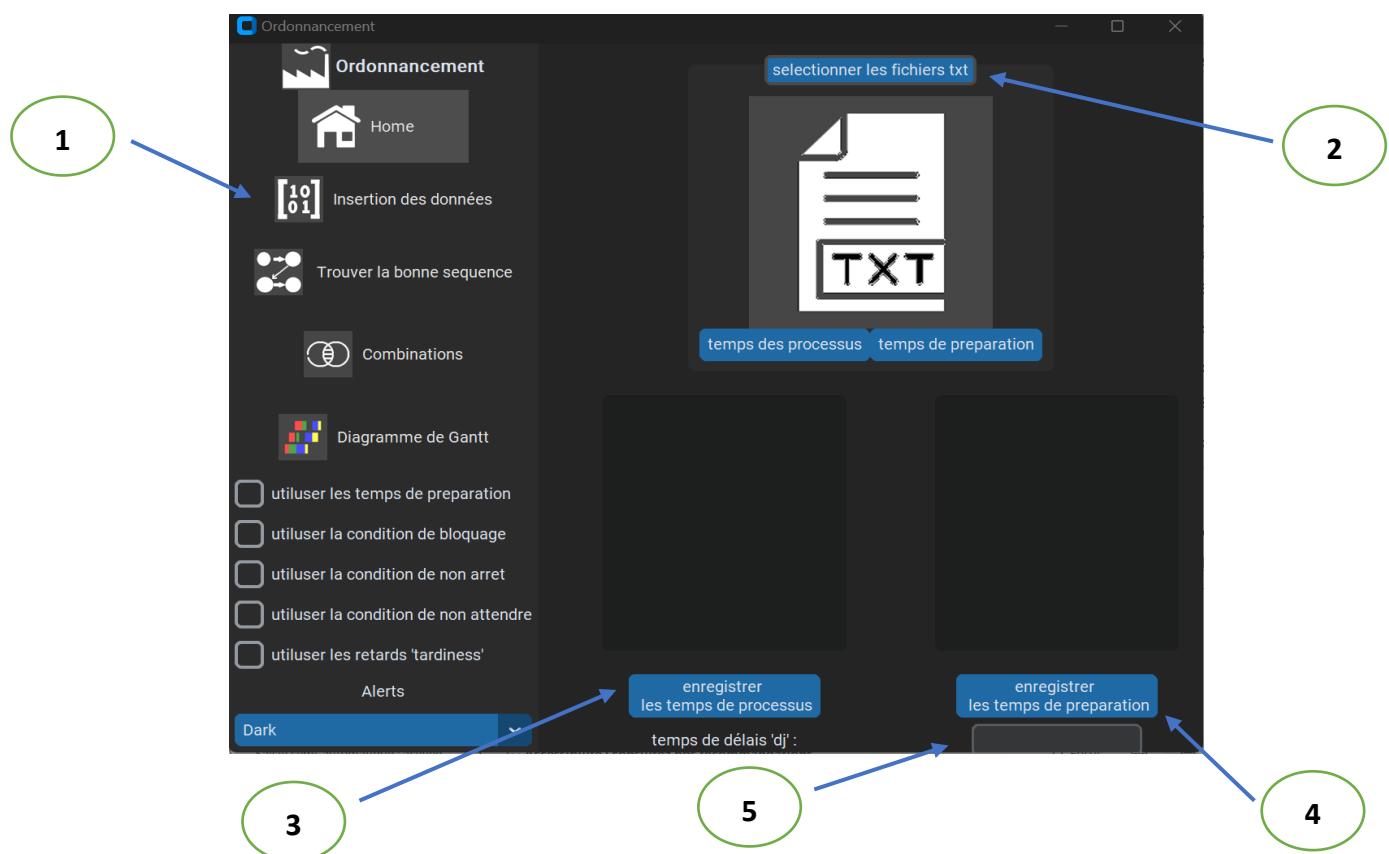
### 3. Choix de contraintes

Cette application permet de choisir la contrainte sous laquelle le problème « Flow Shop » est soumis grâce au « option menu » suivant :



En fonction du choix de l'utilisateur l'application passe à l'exécution des algorithmes permettant de résoudre le problème à traiter

### 4. Insertion des données



- Le bouton « Insertion des données » affiche l'interface conviviale pour saisir les paramètres du problème qu'on veut traiter.

Si le fichier est endommagé, on a l'apparition d'un message erreur comme suit :



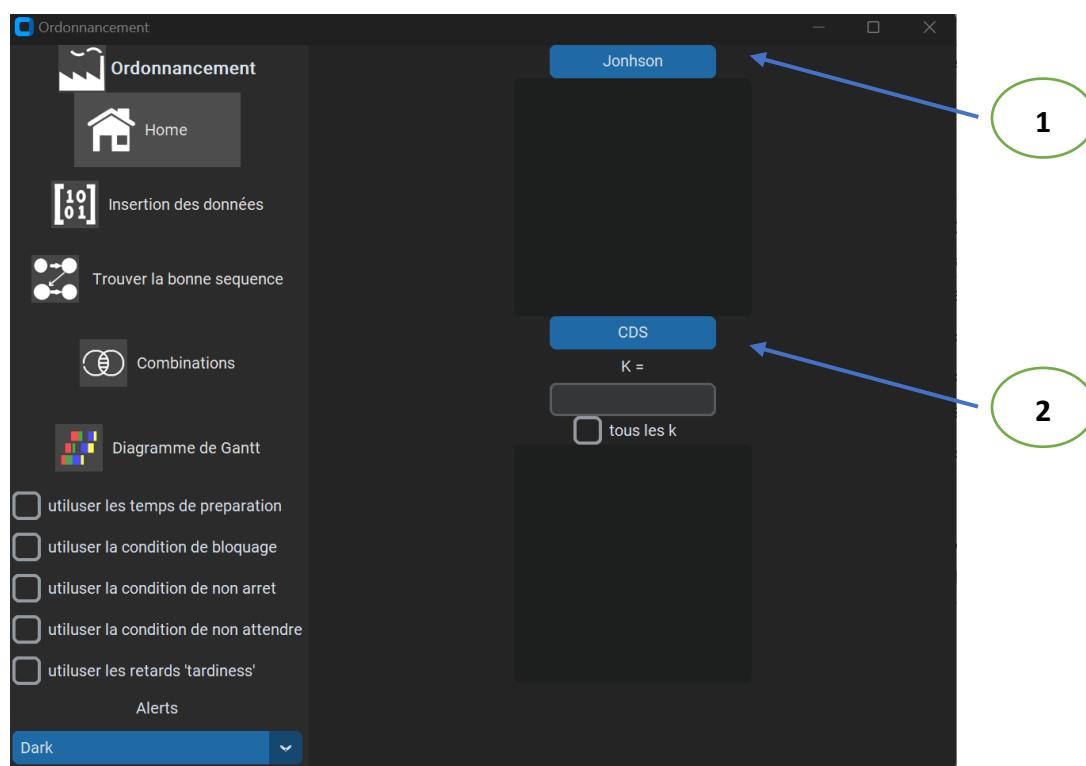
- Le bouton « sélectionner le fichier txt » permet d'insérer le fichier de matrice qu'on veut traiter
- Le bouton « temps de process » permet d'insérer le fichier de la matrice des temps de process des jobs qu'on veut traiter sur l'ensembles de machines qu'on a.
- Le bouton « temps de préparation » permet d'insérer les matrices de préparation des jobs sur chaque machine.
- C'est une case texte intru qui permet d'insérer les délais de chaque job.

#### **NOTE :**

L'application utilise par défaut les syntaxes suivantes :

- La séparation décimale par (.)
- La séparation entre temps de process (,),
- La séparation entre matrice (-)

#### 5. Choix d'algorithme à utiliser

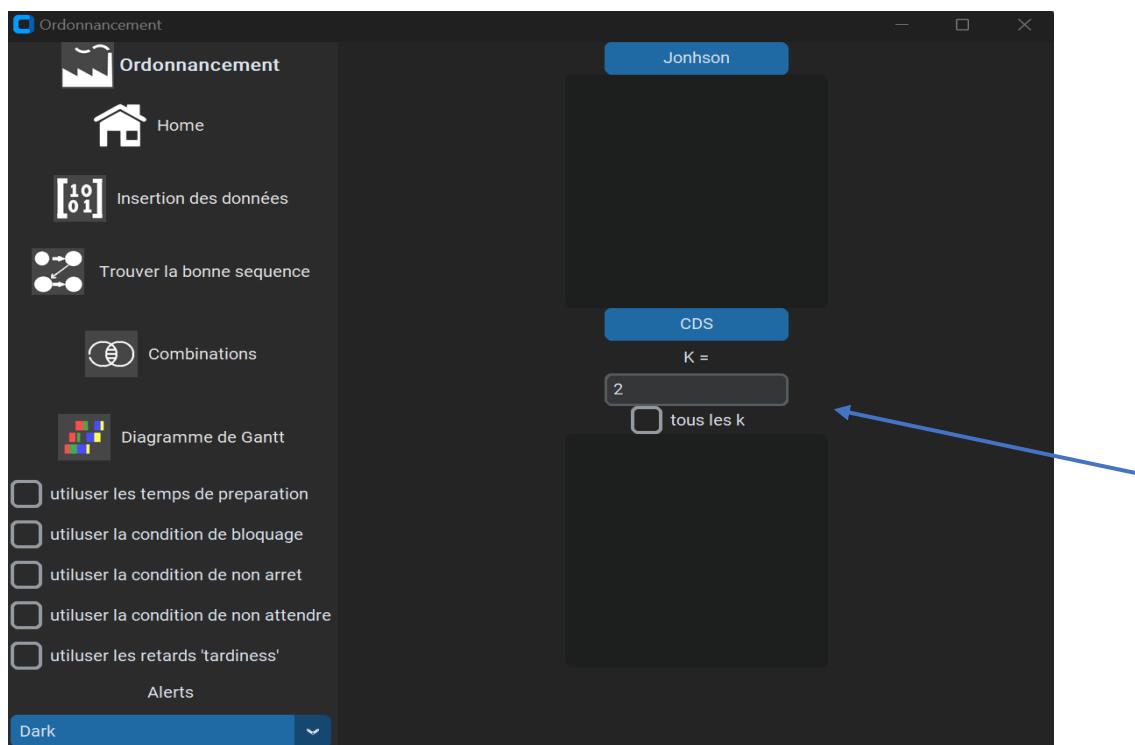


1. Le bouton « Johnson » permet l'application cet algorithme pour résoudre le problème Flow Shop à 2 machines
2. Le bouton « CDS » permet l'application de l'algorithme CDS pour résoudre le problème Flow Shop à m machines en insérant la constante k  
Si on n'insère pas la constante k, on a l'apparition d'un message erreur comme suit :

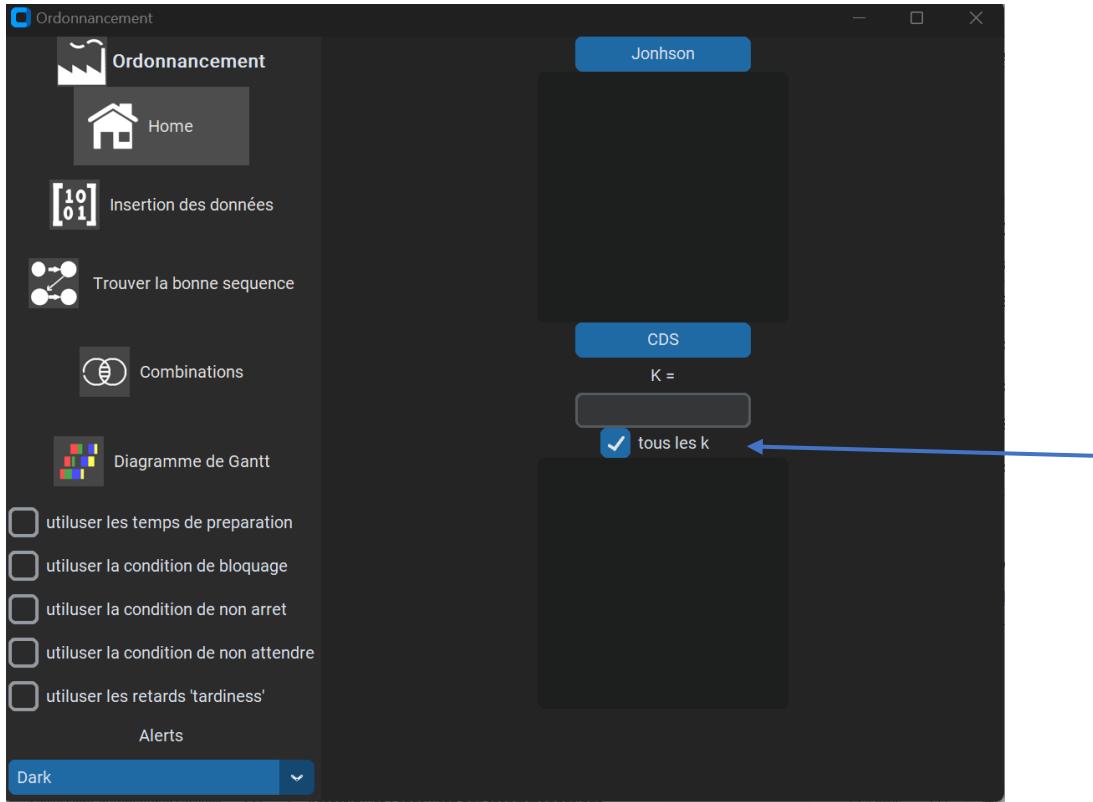
k n'est pas valide

Pour résoudre ce problème on a deux alternatives :

1. On insère la constante k dans la case texte intru



## 2. On appuie sur le check box « tous les k »



Le choix de l'incrémentation de l'algorithme se base sur le nombre de machine. Or, si nombre de machine est égale à 2 on a application directe de l'algorithme de « JOHNSON » sinon si le nombre de machine est supérieure à 1 on a application de l'algorithme du « CDS » pour générer la séquence optimale qui va nous permettre d'appliquer l'algorithme de « JOHNSON »

## 6. Diagramme de GANTT

Le bouton « Diagramme de GANTT » permet de tracer le diagramme en insérant la séquence des jobs dans le texte intru (1).

Le traçage du diagramme est accompagné par les performances des machines en fonction des contraintes sous lesquelles les machines sont soumises, à savoir :

- Option « utiliser le retards **tardiness** » désactivée : L'application affiche seulement le  $C_{max}$
- Option « **utiliser le temps de tardiness** » activée :

L'application affiche en plus du  $C_{max}$  une colonne (T) de « Tardinesse » de chaque job

The screenshot shows the main window of the 'Ordonnancement' application. On the left, a sidebar lists various features: 'Ordonnancement' (selected), 'Home', '[10/1] Insertion des données', 'Trouver la bonne sequence', 'Combinations', 'Diagramme de Gantt' (selected), 'utiliser les temps de préparation', 'utiliser la condition de blocage', 'utiliser la condition de non arrêt', 'utiliser la condition de non attendre', and 'utiliser les retards 'tardiness''. Below this is an 'Alerts' section and a 'Dark' theme switch.

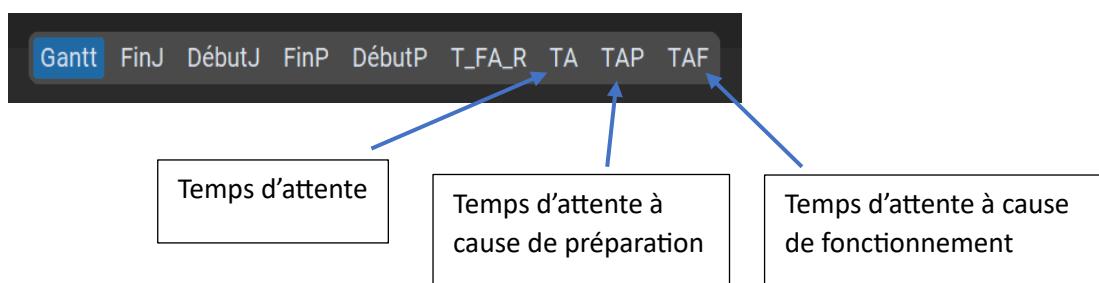
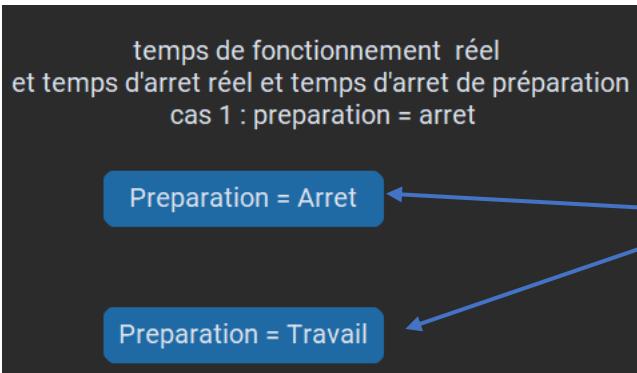
The main area contains a 'Diagramme de Gantt' button with a blue arrow pointing to it from the left sidebar. Above the Gantt chart, there is a text input field labeled 'sequence =' with a small grey rectangle next to it. To the right of the Gantt chart, a large green circle contains the number '1'. Below the Gantt chart, a 'Génération de rapport:' section lists several scheduling problems with checkboxes:

- PFSP  
(Permutation Flow Shop Scheduling Problem)
- PFSP-SDST  
(Permutation Flow Shop Scheduling Problem with Sequence Depending Setup Time)
- BFSP  
(Blocking Flow Shop Scheduling Problem)
- BFSP-SDST  
(Blocking Flow Shop Scheduling Problem with Sequence Depending Setup Time)
- NIPFSP  
(No-Idle Permutation Flow Shop Scheduling Problem)
- NWPFSP  
(No-Wait Permutation Flow Shop Scheduling Problem)
- Utiliser les retards 'tardiness'
- Utiliser la séquence spécifique'

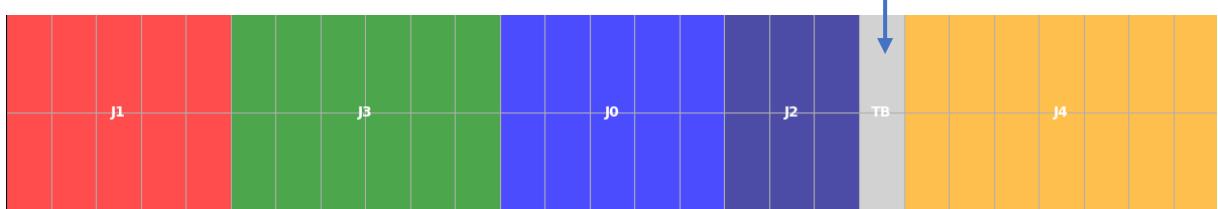
Below the report generation section, a Gantt chart window is open. The title bar includes tabs: Gantt, FinJ, DébutJ, T\_FA\_R, TA, and T. The main content displays the following text: 'total flow time (TFT=114.0) / Tardiness of jobs (Tj) / Total tardness (TT=34.0)' followed by a table:

Machines	Job 1	Job 3	Job 0	Job 2	Job 4
M2	0	12.0	3.0	19.0	0

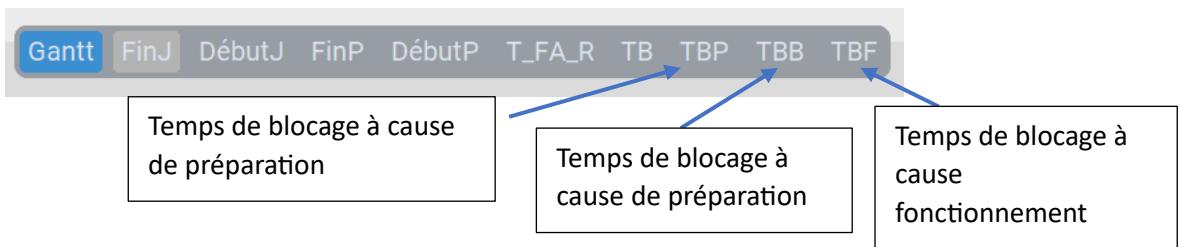
- Option « **utiliser temps de préparation** » activée : L'application affiche le taux de fonctionnement réel ainsi que le taux d'arrêt de chaque machine dans le cas où la machine est considérée en préparation et le cas où la machine est considérée en fonctionnement lors de sa préparation et le temps d'attente dans les 2 cas.



- Option « **utilise la condition de blocage** » activée : On a le traçage de temps de blocage « TB » en gris



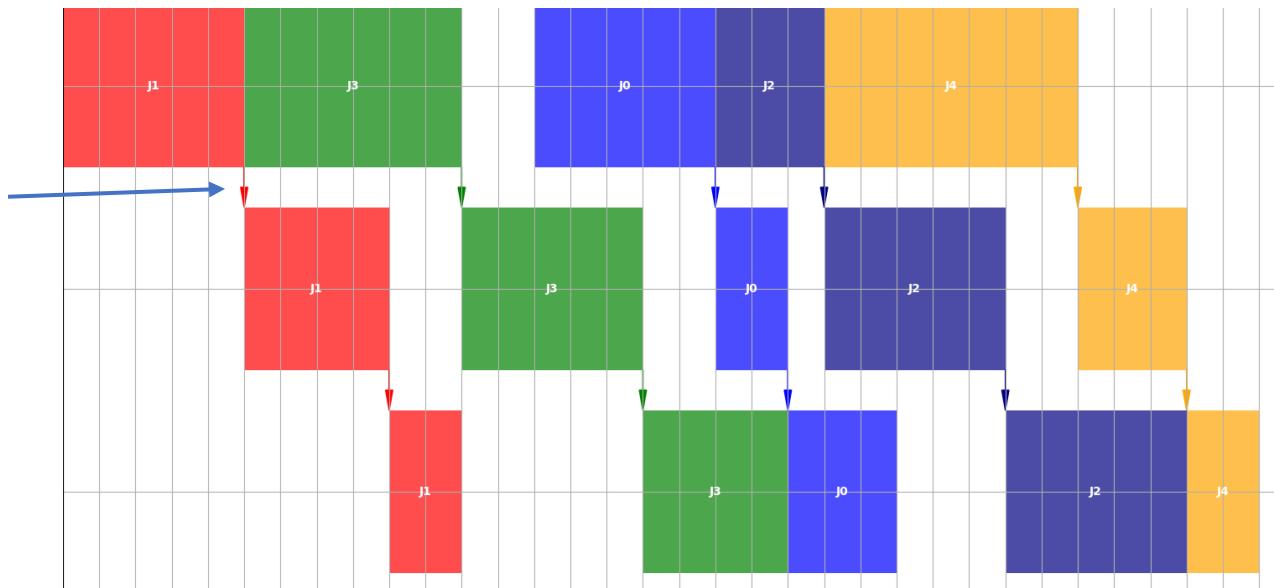
- Option « **utilise la condition de blocage** » et « **utiliser temps de préparation** » activées : l'application affiche dans la barre le temps de blocage dans 3 cas, il s'agit du temps de blocage à cause de la préparation, temps de blocage à cause du fonctionnement et temps de blocage à cause de blocage



- Option « **utiliser la condition non-arrêt** » :  
L'application affiche le temps d'attente (TA) de chaque jobs



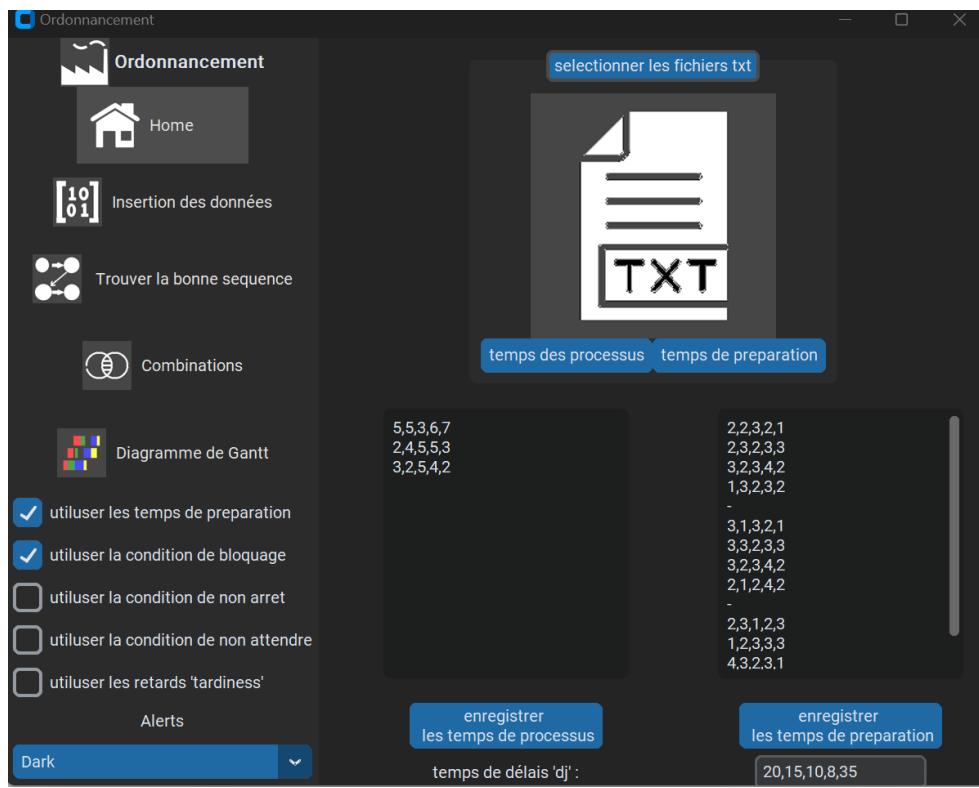
- Option « utiliser la condition de non attendre »



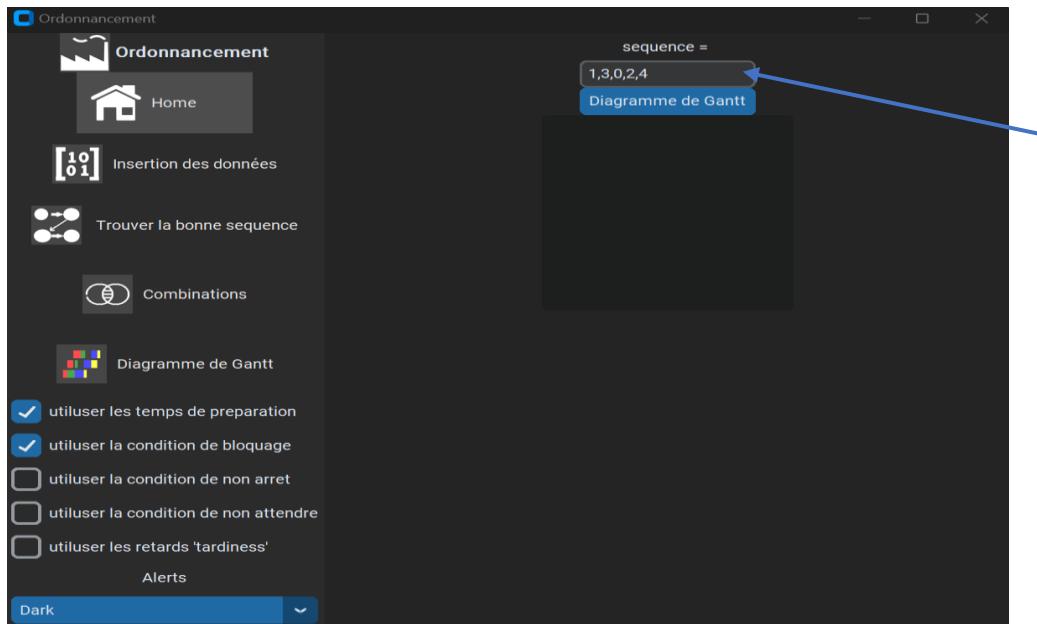
## IV. Implémentation de l'application

### 1. Flow Shop avec contrainte de préparation et de blocage

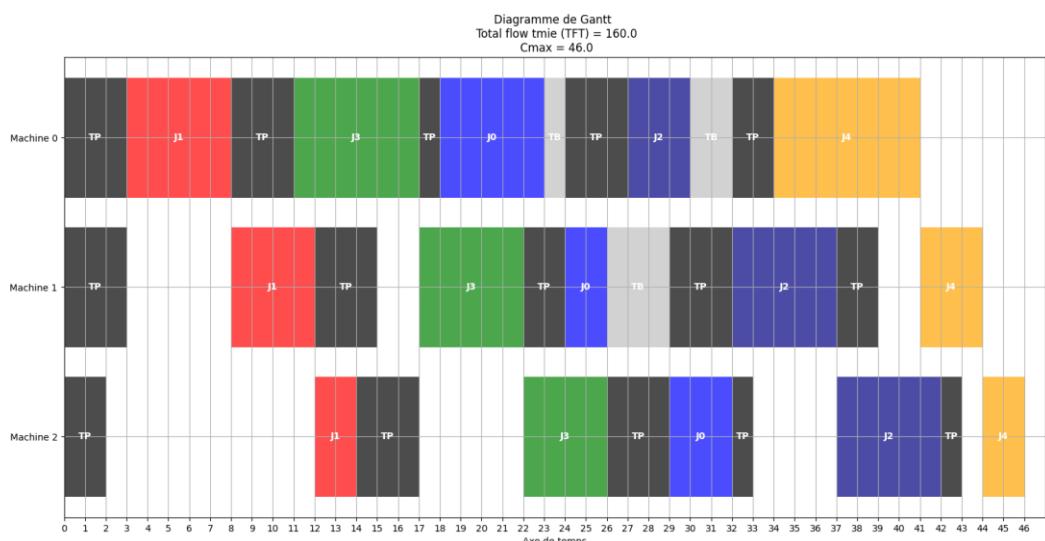
- On insère les matrices de temps de processus et temps de préparation ainsi de le délai de chaque job après avoir choisi les options « utiliser le temps de préparation » et « utiliser le temps de blocage »



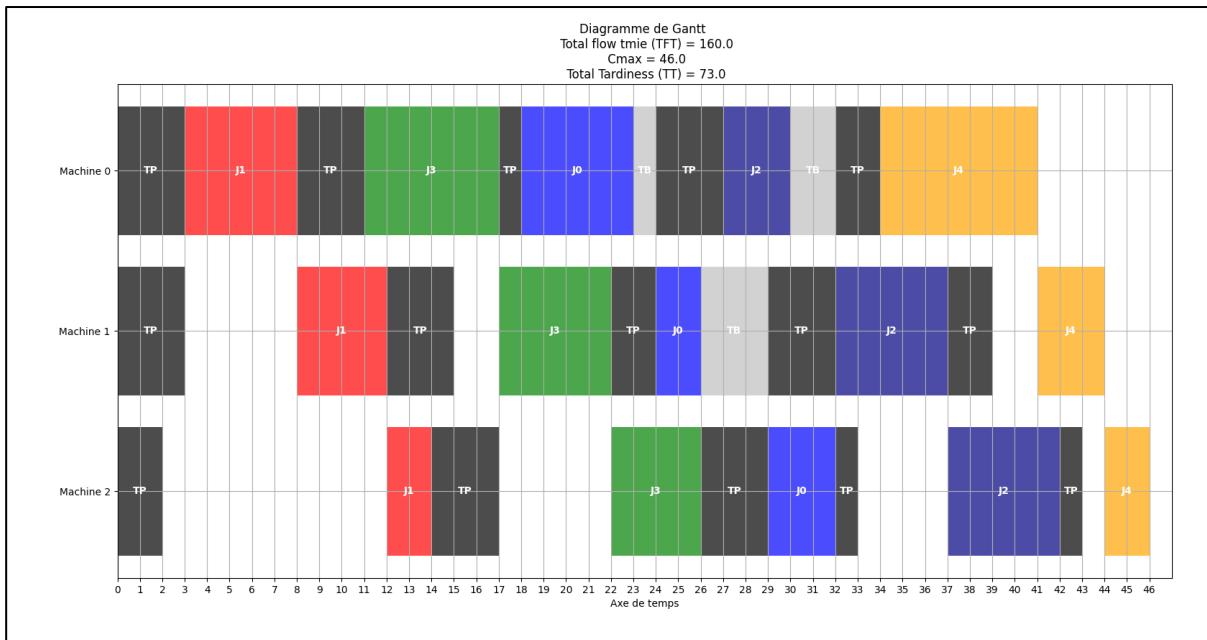
- On insère la séquence des jobs qu'on veut traiter



- Traçage de diagramme de GANTT pour cette séquence et affichage du  $C_{max}$  et TFT



- Si on ajoute l'option « utiliser le temps tardiness » on aura aussi la valeur du total tardiness (TT)



## 2. Condition non attendre

On travaille sur les mêmes données en cliquant sur la condition non attendre

Ordonnancement

selectionner les fichiers txt

temps des processus	temps de préparation
5,5,3,6,7 2,4,5,5,3 3,2,5,4,2	2,2,3,2,1 2,3,2,3,3 3,2,3,4,2 -
	3,1,3,2,1 3,3,2,3,3 3,2,3,4,2 2,1,2,4,2 -
	2,3,1,2,3 1,2,3,3,3 4,3,2,3,1

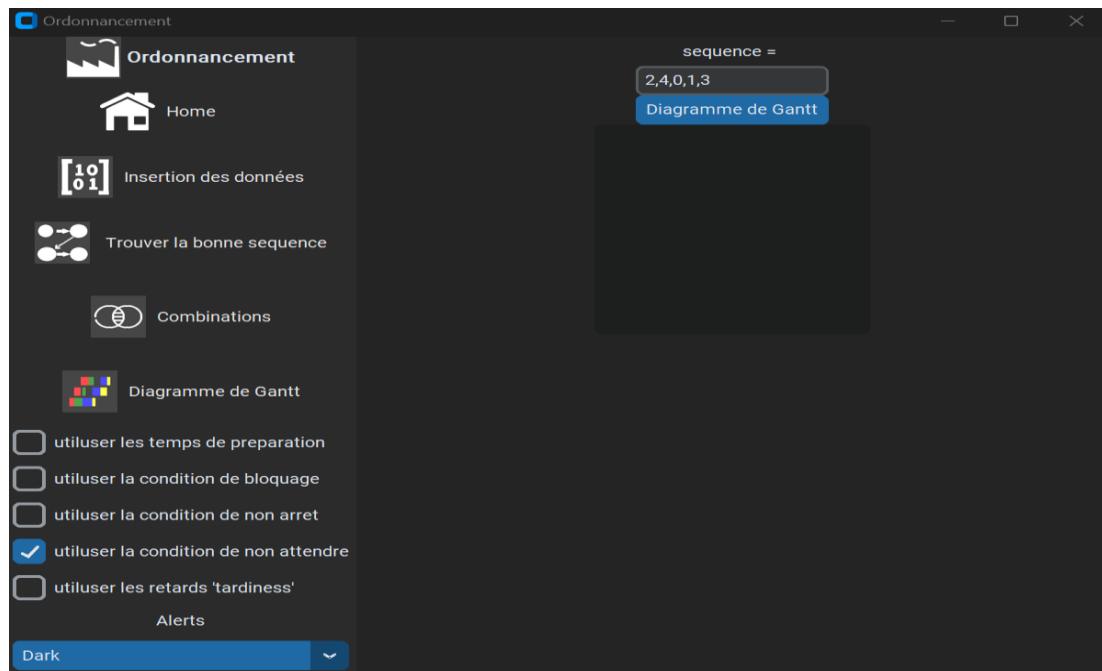
utiliser les temps de préparation  
utiliser la condition de blocage  
utiliser la condition de non arrêt  
 utiliser la condition de non attendre  
utiliser les retards 'tardiness'

enregistrer les temps de processus  
enregistrer les temps de préparation

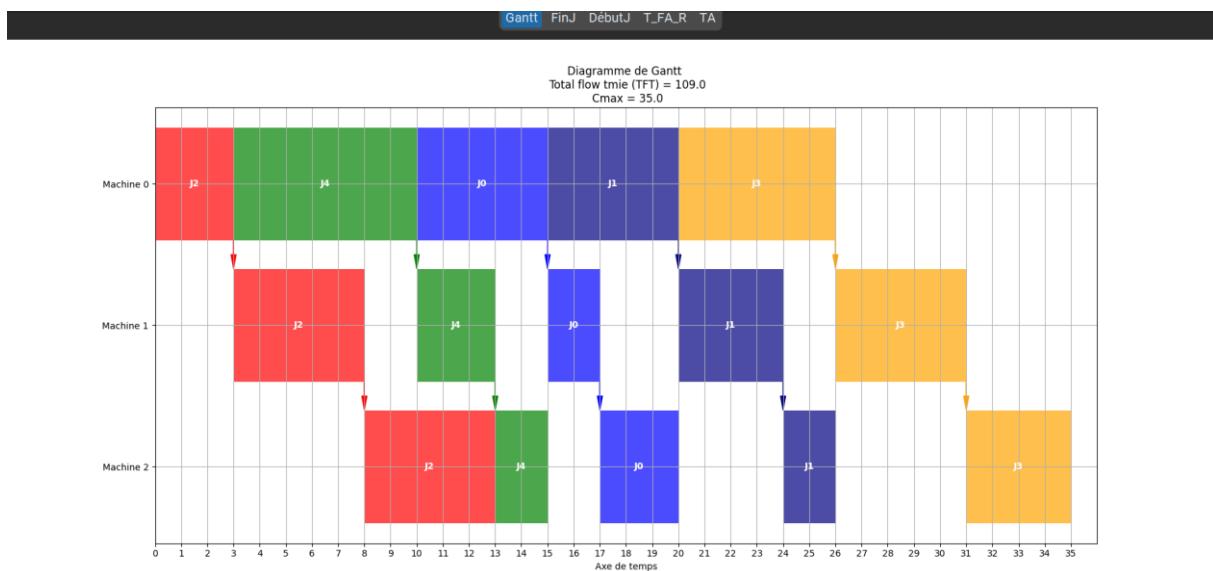
Diagramme de Gantt

92

On choisit la séquence qui suit :



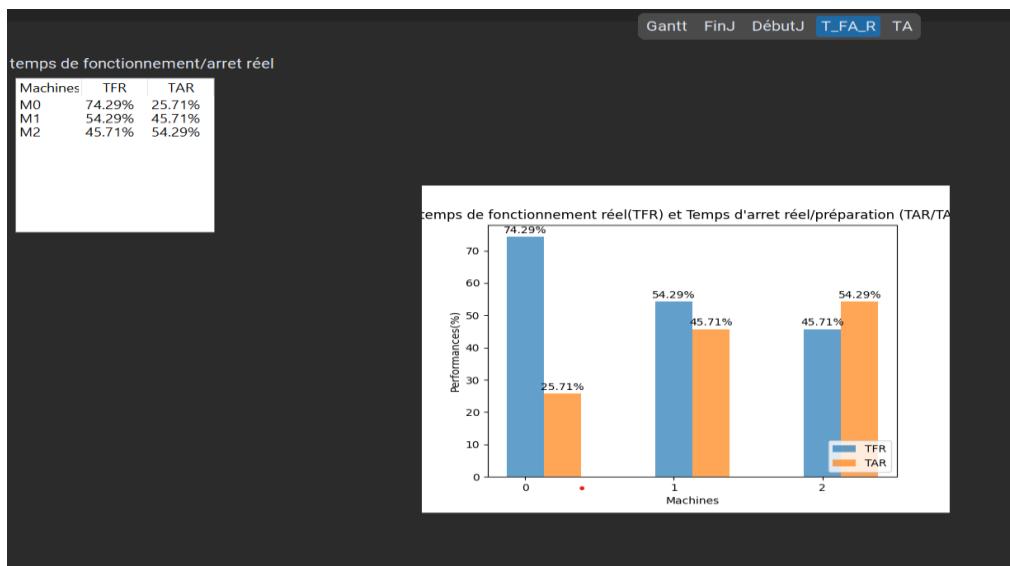
Extraction des données à partir du diagramme de GANTT



## Temps d'attente de chaque job

Gantt	FinJ	DébutJ	T_FA_R	TA
Temps d'attente des Jobs				
Machines Job2/4   Job4/0   Job0/1   Job1/3				
M0	0.0	0.0	0.0	0.0
M1	2.0	2.0	3.0	2.0
M2	0.0	2.0	4.0	5.0
somme	2.0	4.0	7.0	7.0

## Taux de fonctionnement et taux d'arrêts des machines



## CONCLUSION

La réalisation de cette application basée sur Python pour résoudre le problème d'ordonnancement flow shop, avec la prise en compte des contraintes de blocage et de préparation, représente une avancée significative dans la résolution de ce défi complexe. À travers le développement d'algorithmes dédiés, nous avons pu aborder différentes facettes de ce problème, offrant ainsi une solution complète et adaptable à divers contextes.

L'implémentation d'algorithmes pour le flow shop sans contrainte a permis de mettre en évidence la puissance de notre application dans la gestion efficace des flux de travail. En intégrant ensuite les contraintes de blocage, nous avons relevé le défi supplémentaire de l'optimisation tout en prenant en compte des réalités opérationnelles, renforçant ainsi la pertinence de notre solution dans des environnements réels.

La gestion des contraintes de préparation, qui souvent complique davantage le problème d'ordonnancement, a été abordée avec succès grâce à notre approche algorithmique. Cela témoigne de la flexibilité et de l'adaptabilité de notre application pour s'adapter à des scénarios plus proches de la réalité industrielle.

En résumé, notre application Python offre une solution complète et robuste pour le problème d'ordonnancement flow shop, couvrant les aspects sans contrainte, avec contrainte de blocage et avec contrainte de préparation. Les résultats obtenus démontrent son efficacité et sa capacité à optimiser les processus d'ordonnancement dans des situations variées. Ce travail représente une contribution significative à la résolution pratique des défis rencontrés dans le domaine de l'ordonnancement flow shop.

