MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE SAAD DAHLAB-BLIDA
FACULTE DES SCIENCES
DEPARTEMENT DE MATHEMATIQUES

En vue de l'obtention du diplôme de MASTER
spécialité RECHERCHE OPERATIONNELLE

MEMOIRE
DE FIN D'ETUDES

Thème

Contribution à la résolution du Job shop à 2-machines avec convoyeur

Présenté par : BOUALI Abdenacer

BOUICHE Abdelhak

Devant le jury composé de :

Président du Jury :	O.Tami	USDBlida
Examinateur :	H. El mossaoui	USDBlida
Examinateur :	R.Boudjemaa	UMBBoumerdes
Promoteur :	A. Derbala	USDBlida

Blida, Octobre 2013

Remerciements

Nous tenons à remercier en premier lieu le TOUT PUISSANT DIEU qui nous a donné la volonté nécessaire, la force et la bonne santé, la patience et le courage de mener à bon terme ce mémoire de fin d'études.

Nous exprimons notre gratitude à notre promoteur Monsieur Ali DERBALA qui nous a fait l'honneur de diriger ce travail.

Nous le remercions pour sa disponibilité et ses conseils qui nous ont été d'un apport considérable tant sur l'aspect scientifique que sur la méthode de recherche. Sans son appui et son expérience, rien n'aurait été possible.

Nos vifs remerciements vont également aux membres du jury, pour l'intérêt qu'ils ont porté à notre recherche en acceptant d'examiner notre travail et de l'enrichir par leurs propositions.

Enfin, nous remercions tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail.

Résumé : Dans un atelier manufacturier, le job shop à deux machines et un seul convoyeur est étudié. Le moyen de transport ou convoyeur est chargé de transporter les tâches semi-finies entre les deux machines. Une tâche est dite semi-finie si elle est exécutée sur une des deux machines et les opérations élémentaires qui la constituent ne sont pas toutes exécutées. Deux zones tampons ou aires de stockage appelées zones de stockage d'entrée et de sortie sont en face de chacune des deux machines. Elles sont supposées de capacité illimitée. Les zones de stockages d'entrée et de sortie et la machine qui est en face constituent une station. Le transport d'une tâche entre deux machines peut être décrit comme son chargement de la zone de stockage de sortie d'une machine sur le convoyeur, du transport entre les deux machines et du déchargement de la tâche dans la zone de stockage d'entrée de la seconde machine et vice-versa. L'objectif est la minimisation du temps écoulé depuis le début d'exécution de la première tâche jusqu'à la fin d'exécution de la dernière tâche. Il représente la longueur d'ordonnancement appelé makespan et noté C_{max}. De la littérature, trois règles de priorité « statiques » sont proposées pour la résolution du problème. Une nouvelle règle de priorité « dynamique » appelée OPBM, Ordonnancement par Priorités Bornées en Moyenne, est définie, modifiée et utilisée. Un algorithme OCF-J2(1)C_{max}, une métaheuristique stochastique de type colonie de fourmis, l'OPBM-étendu, la règle de Jackson sont présentés et implémentés par nos soins en utilisant un langage de programmation évolué. Le temps de CPU appelé aussi temps machine, le rapport relatif moyen et la longueur d'ordonnancement C_{max} ont constitué les critères de notre étude comparative. Quel que soit le nombre de tâches à exécuter, la règle de Jackson a fourni un temps CPU le plus court. Sa solution réalisable obtenue n'est pas la meilleure parmi les solutions fournies par les trois méthodes. Pour la fonction C_{max}, l'OPBM-étendu, OCF-J2(1)C_{max} et la règle de Jackson forment un ordre décroissant de performances. Ces dernières sont confirmées par l'exécution de 5x100 instances de problèmes-tests de 10, 50, 100, 200 et 400 tâches. Ces instances sont générées de manière aléatoire d'une distribution uniforme.

Mots-clés : job shop, convoyeur, colonie de fourmis, temps de transport, OPBM, règle de jackson.

ABSTRACT: In a manufacturing workshop, job shop with two machines and one conveyor is studied. The means of transport or conveyor is responsible for transporting semi-finished jobs between the two machines. A job is said to be semi-finite if it is running on one machine and the basic operations are not all executed. Two buffer or storage areas called storage input and output fields are opposite each of the two machines. They are supposed to unlimited capacity. Areas of storage input and output and the machine is in front is said to be a station. Transporting a job between two machines can be described as a loading area for storing output of a machine on the conveyor, the transport between the two machines and unloading of the job in the storage area input the second machine and vice versa. The objective is to minimize the time elapsed since the start of execution of the first job to the end of the last job execution. It represents the length of scheduling called makespan and noted C_{max}. In the literature, three rules of "static" priority are proposed for solving the problem. A new rule of "dynamic" priority called OPBM, Scheduling Priorities by Bounded in Average is defined, modified and used. An algorithm OCF-J2(1) C_{max}, a stochastic metaheuristic ant colony type, OPBM-extended, the rule of Jackson are presented and implemented by us using a programming language. CPU time also called time machine, the average relative ratio and the length of scheduling C_{max} are the criteria for our comparative study. Whatever the number of jobs to be performed, the rule of Jackson gave the shortest CPU time. Its feasible solution is not the best among the solutions provided by the three methods. For C_{max} function, OPBM-extended, OCF-J2(1)C_{max} and the rule of Jackson form a descending order of performance. These are confirmed by the execution of 5x100 problem instances tests of 10, 50, 100, 200 and 400 jobs. These instances are generated randomly from a uniform distribution.

Keywords: job shop, conveyor, ant colony, transporting time, OPBM, rule of Jackson.

ملخص : في ورشة تصنيع , job shopبـآلتين و ناقل وحيد درس . الناقل مخصص لنقل الاعمال نصف منتهية بين الالتين . عمل نصف منتهي هو العمل الذي انجز من طرف الة واحدة و ينتضر اتمام انجازه. منطقتي تخزين بجوار كل الة, نفرض انهما ذو سعة غير منتهية. منطقة حجز الدخول و حجز الخروج و الالة يكونون محطة.
نقل عمل بين آلتين هو تشحين العمل على الناقل و نقله زائد تفريغه.
هدفنا هو تخفيض الوقت الازم لانجاز جميع الاعمال نرمز له بـــ :c_{max} .
من الدراسات الاخرى ثلاث قواعد افضلية مفروضة لحل هذا المشكل.
قاعدة جديدة تسمى :OPBM. خوارزمية $OCF\text{-}J2(1)C_{max}$ من نوع مستعمرة النمل, مغيرة-OPBM,قادعة Jackson عوضو و نفذو من طرفنا باستعمال لغة برمجة متطورة. وقت التنفيذ. التقرير النسبي المتوسط و مدة الانجاز C_{max} هم المقايس المستعملة في المقارنة. مهما يكن عدد الاعمال, قاعدة Jackson تعطينا وقت تنفيذ اقصر لكن مدة الانجاز C_{max} ليست هي الافضل. بالنسبة لمدة الانجاز C_{max}, مغيرة-OPBM.

$OCF\text{-}J2(1)C_{max}$, قاعدة Jackson هو ترتيب الاداء المتناقص. هذا الاخير مؤكد بعد تجريب 500 مثال من 200,100,50,10 و 400 عمل بطريقة عشوائية بتوزيع موحد.
كلمات البحث :Job shop , ناقل , مستعمرة النمل وقت النقل ,OPBM, قاعدة Jackson.

LISTE DES FIGURES ET TABLEAUX

TABLE DES MATIERES

Introduction

Dans le cadre des travaux de ce mémoire de master et pour motiver cette étude faite par nos soins, des questions et des questionnements se posent et à qui on doit répondre.

Pourquoi le Job shop est intéressant à étudier ?

Le Job shop est un outil très utile de gestion des ateliers manufacturiers. Il est utilisé pour modéliser les unités manufacturières où on fabrique un objet ou une pièce ou un produit en plusieurs unités ou en grandes quantités. Les processus types de gestion de ces ateliers sont les lignes d'assemblages telles les chaînes de montage de véhicules, de téléviseurs, de réfrigérateurs, etc.

Supposons que dans un atelier, m machines ($m > 1$) sont disposées en série, disposées l'une derrière l'autre pour exécuter n tâches, $n > 1$. Un Job shop est un ordonnancement particulier dans cet atelier. Une tâche commence son exécution sur la première machine, passe ensuite sur la seconde machine et ainsi de suite jusqu'à ce qu'elle termine son exécution et quitte l'atelier. Une tâche peut revenir pour exécution sur une même machine une seconde fois.

Se donnant une liste de commandes de clients, le gestionnaire d'un atelier doit établir un ordonnancement ou un ordre d'exécution ou de confection de ces produits. L'ordonnancement peut constituer aussi une production interne à l'atelier à réaliser ou une production pour anticiper une éventuelle commande d'articles. L'ordre dans lequel les commandes sont produites peut sérieusement influencer le profit de l'entreprise. La souplesse de la chaîne de montage, sa flexibilité peut aider l'équipe de vente à répondre aux besoins des clients. La production « juste en temps » peut améliorer la qualité des produits en éliminant des crises de production.

Pourquoi le Job shop déterministe ne reflète pas en général la réalité ?

Les modèles d'ordonnancement « déterministes » sont définis dans un environnement où les temps d'exécution des tâches (ou leur durée d'exécution) sur une machine sont supposés être « constants ». Des difficultés surviennent quand le temps d'exécution d'une opération peut varier suite à un environnement d'ordonnancement dynamique. Dans un tel environnement incertain, des modèles stochastiques sont introduits où le temps d'exécution d'une tâche est supposé être une « variable aléatoire » de fonction de distribution connue.

Pourquoi faut-il étudier le Job shop stochastique ?

Pour réaliser de bonnes performances, les décideurs utilisent des modèles qui reflètent la réalité pour prédire le comportement des tâches et pour définir une stratégie d'exécution des tâches. En paramétrisant aléatoirement les données, le résultat prédit est aussi stochastique. Des valeurs stochastiques peuvent être représentées par des distributions, des intervalles et des histogrammes [1]. Dans notre cas d'étude, le stochastique apparait dans le déplacement aléatoire des fourmis par des transitions d'état de phéromone.

Dans beaucoup de secteurs industriels, satisfaire les dates de fin au plus tard est un objectif crucial. D'où un intérêt à bien définir les fonctions objectifs. L'exemple du problème d'ordonnancement de « *n* » tâches sur une machine afin de minimiser la variance des temps de fin d'exécution trouve des applications dans l'organisation des fichiers informatiques, dans le système dit on-line où il est désirable de fournir un « temps de réponse » uniforme pour tous les utilisateurs. Il répond aussi à la philosophie de production « juste à temps » ou « quand la demande se fait sentir ».

Des difficultés dans l'étude du Job shop stochastique

Des difficultés peuvent persister dans certains scénarios. Une collecte suffisante d'information à priori pour caractériser la fonction de distribution du temps d'exécution aléatoire s'avère parfois difficile. Même si la distribution de la variable aléatoire est connue, la distribution de probabilité n'est intéressante à utiliser que pour un nombre assez grand de réalisations dans un environnement d'ordonnancement similaire mais elle n'est d'un cas pratique que pour un petit nombre de réalisations similaires.

Le Job shop à 2-machines avec convoyeur

Dans le job shop classique, le déplacement des taches d'une machine à une autre est considéré comme instantané. Ces déplacements n'étaient donc pas pris en compte lors de la construction des ordonnancements. Or, cette hypothèse est souvent non justifiée en pratique [2]. En effet, le transport des taches doit être intégré aux données du problème. Lee et al. [3] suggèrent que : « la nouvelle tendance dans la théorie de l'ordonnancement est d'étendre les résultats des algorithmes classiques à des modèles plus proches des problèmes réels. Même si beaucoup de résultats ne sont pas immédiatement applicables, ces nouveaux modèles sont au moins motivés par les problèmes industriels et un grand potentiel d'applications ».

Tomkins et White [4] ont montré que la manipulation matérielle peut contribuer jusqu'à 80% du cout total de fabrication.

Dans un atelier, le transport d'une tâche entre deux machines peut être décrit comme son chargement de la zone de stockage de sortie d'une machine sur le convoyeur, du transport entre les deux machines et du déchargement de la tâche dans la zone de stockage d'entrée de la seconde machine et vice-versa. L'opération de transport englobe toutes ces actions ou considère séparément chacune d'elles comme une opération distincte. Notre objectif étant la minimisation du temps d'exécution et des temps de transport de toutes les tâches entre les machines jusqu'à la fin d'exécution de la dernière tâche.
Nous nous sommes intéressées à l'étude d'un job shop à deux machines et un seul convoyeur chargé du transport des tâches semi-finies entre les machines. Une tâche est dite semi-finie si elle est exécutée par au moins une machine, mais dont toutes ses opérations élémentaires ne sont pas encore exécutées. Dans la littérature, ce problème est classé difficile [5].
Il a été déjà étudié par Ruhlmann [6] où elle a comparé trois règles de priorité « statiques » et une métaheuristique « déterministe » de type recherche taboue. Notre étude est une extension au cas de résolution du problème par une règle de priorité « dynamique » et une métaheuristique « stochastique » de type colonie de fourmis.

Dans le premier chapitre, la notion du flow shop est introduite. Le flow shop avec capacité de stockage intermédiaire entre les machines non limitée est défini. La Résolution du flow shop à deux machines dont l'objectif est le makespan est faite par l'algorithme de Johnson. Le job shop ou le flow shop généralisé sur 2-machines, n / 2 / G / C_{max} est indiqué et discuté. Une autre version de l'algorithme de Johnson est donnée. Minimiser le flow-time moyen dans le flow-shop à deux machines (n / 2/ F / $\overline{F}$) constitue un second objectif à atteindre. Le Cas du problème à trois machines noté n / 3 / F / C_{max} est étudié. Le cas particulier dual 2 / m / F / C_{max} où le nombre de tâches est deux et le nombre de machines est « *m* » est solutionné graphiquement par la méthode d'Akers.

Les opérations de transport des tâches par le convoyeur sont définies, les types de convoyeurs sont présentés de façon générale et au cas particulier du job shop. Notre problème est défini en détail en ce second chapitre. Le fonctionnement et la complexité de l'atelier sont indiqués. Neuf propriétés de l'atelier sont établies sans démonstrations. L'influence de l'ajout du convoyeur dans l'atelier est examinée. Une borne inférieure pour le temps total d'exécution est déterminée et sera utilisée dans la suite. Finalement, nos objectifs de recherche sont présentés.

Dans le chapitre suivant, trois méthodes de résolution approchées basées sur des règles de priorité et une métaheuristique de type recherche taboue sont exposées.

Une quatrième règle de priorité dynamique appelée OPBM est définie, modifiée et utilisée pour la résolution de notre problème. Une métaheuristique stochastique de type colonie de fourmis est présentée et un algorithme OCF-J2(1)C_{max} est élaboré pour la résolution du même problème. Ces notions constituent le contenu du chapitre quatre.

Le chapitre final se consacre à une étude comparative entre l'algorithme colonie de fourmis et les deux règles de priorité, l'OPBM-étendu et la règle de Jackson. L'efficacité des trois méthodes de résolution selon trois critères de comparaison, le CPU, le rapport relatif moyen et la longueur d'ordonnancement s'interprètent par des graphiques et des histogrammes.

Une conclusion et des perspectives d'étude ultérieures de ce problème sont établies.

Chapitre 1 : Le Job shop déterministe

Avant d'introduire la notion du job shop, il est indispensable de connaître celle du flow shop. Dans un atelier, un ensemble de machines est dit constituant un « flow-shop » si elles sont disposées en série ou numérotées de telle façon que, pour chaque job considéré, une opération k est exécutée sur une machine de rang supérieur que l'opération j si $j > k$. Le nombre de machines est de deux ou plus de deux. Un exemple d'un tel atelier est la disposition en ligne où les travailleurs ou les stations de travail représentent les machines. N'importe quel groupe de machines servis par un convoyeur unidirectionnellement et non-cycliquement sera considéré comme un flow-shop [7]. Il n'est pas requis que chaque job a une opération sur chaque machine. Dans ce cas, le temps d'exécution du job sur la dite machine est nul. La seule condition est que tout mouvement entre les machines dans l'atelier soit dans une direction uniforme. On suppose que $\{1, 2,..., n\}$ est un ensemble de n jobs à exécuter sur "m" machines ($m \geq 2$) disposées en série ou flow shop. Les machines sont installées dans l'ordre 1, 2,..., m. Les jobs sont supposés être dans l'atelier à l'instant zéro.

Dans un flow shop, il est suffisant de considérer que les ordonnancements de permutation, appelés aussi ordonnancement en liste. Ils sont complètement décrits par une permutation particulière des numéros d'identification de jobs. C'est le cas où il y a une préservation de l'ordre des jobs sur les machines initiales et terminales.

Les résultats évoqués ci-dessous sont cités dans la majorité des livres d'ordonnancement, voir [8], [9], [10], et [11].

1. Le Flow shop avec capacité de stockage intermédiaire entre les machines non limitée

1.1 Quelques résultats

Les contraintes technologiques dans un flow shop ont la forme suivante

Tableau 1: Contraintes technologiques dans un flow shop

Job	Ordre d'exécution				
J_1	M_1	M_2	M_3	...	M_m
J_2	M_1	M_2	M_3	...	M_m
...				....	
J_n	M_1	M_2	M_3	...	M_m

Le premier résultat montré est que le problème d'ordonnancement de longueur minimale dans un flow shop m-machines ($m \geq 3$) est NP-complet. Pour $m = 2$, il existe un algorithme efficace pour trouver de tel ordonnancements. L'algorithme de Johnson nécessite au plus un temps proportionnel à $n \log (n)$. Le second résultat montre que le problème de la détermination d'un flow time pondéré minimale dans un flow shop m-machines est NP-complet pour $m \geq 2$.
Le problème d'un flow shop à 3-machines est NP-complet aussi bien si la longueur des entrées est mesurée par la somme des longueurs de tâches. Il se formule comme un problème de 3-partition.
C'est le problème noté F m // C_{max}. Soit $j_1, j_2, ..., j_n$ une permutation de n jobs sur m machines flow shop. Le temps de fin d'exécution C_{ij} du job j_k sur la machine i peut être facilement calculé par les équations de récurrence:

$$C_{i,j1} = \sum_{l=1}^{i} p_{l,jl}, \qquad i = 1,..., m;$$

$$C_{,ijk} = \sum_{l=1}^{k} p_{l,jl}, \qquad k = 1,..., n$$

$$C_{i,jk} = \text{Max} (C_{i-1,jk}, C_{i,jk-1}) + p_{i,jk}, \quad i = 2,.., m; \quad k = 1, ..., n.$$

Lemme 1 : Dans un flow shop où on désire minimiser le makespan, ordonnancer les jobs selon la permutation $j_1, ..., j_n$ donne le même makespan que celui de la permutation $j_n, ..., j_1$.

Théorème 1 : Pour minimiser n'importe quelle fonction objective, il est suffisant de considérer que les ordonnancements où le même ordre de jobs est prescrit sur les deux premières machines.

Théorème 2 :
Dans le cas du problème n / m / F / C_{max}, il est suffisant de considérer que les ordonnancements où le même ordre d'exécution des jobs est prescrit sur la machine 1 et 2, et le même ordre des jobs est prescrit sur les machines m - 1 et m.

Remarque1 : C'est un résultat non nécessaire. On ne doit pas exclure la possibilité qu'il peut y avoir des ordonnancements optimaux qui ont des ordres différents sur les deux premières

machines. Il peut aussi exister un ordonnancement optimal où l'ordre optimal sur la première et la deuxième machine est différent que celui de la machine m - 1 et m.

Pour toute mesure de performance, et en particulier pour le flow-time pondéré, le maximum flow-time, le flow shop à deux machines est le seul cas où il est suffisant de ne considérer que les ordonnancements de permutation.

Mais un contre-exemple indique aussi que pour le makespan appelé aussi le maximum flow-time, dans les flow shop à quatre ou plus de machines un type général d'ordonnancement doit être considéré.

Contre-exemple 1 :

Soit le problème de deux tâches qui doivent s'exécuter sur quatre machines en série selon les temps d'exécution regroupés dans le tableau ci-dessous.

Tableau 2: Données du contre-exemple 1

	M1	M2	M3	M4
Tâche A	4	1	1	4
Tâche B	1	4	4	1

L'exécution des deux tâches dans un même ordre de passage, l'ordre A-B, nous fournit le diagramme de Gantt suivant :

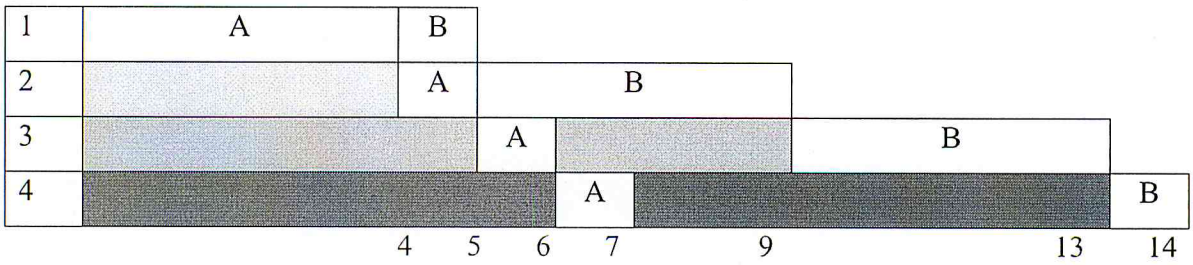


Figure 1: Diagramme de Gantt

Le makespan est de 14 unités de temps.

On peut trouver un ordonnancement meilleur qui ne conserve pas le même ordre de passage sur les machines. Il est de longueur 11.

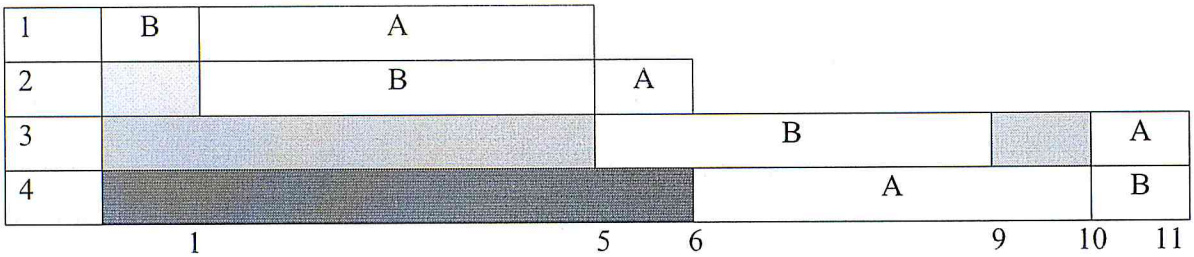


Figure 2: Diagramme de Gantt pour un meilleur ordonnancement

Contre-exemple 2 :

Soit le problème de deux taches qui doivent s'exécuter sur trois machines en série afin de minimiser le « mean flow time ». Leurs temps d'exécution sont donnés par le tableau ci-dessous.

Tableau 3: données du contre-exemple 2

	M1	M2	M3
Tâche A	4	1	1
Tâche B	1	4	4

Les ordres d'exécution (*A, B*) et (*B, A*) sur les machines 1, 2 et 3 nous fournissent un mean flow time de 9.5. Les ordres d'exécution (*B, A*) sur les machines 1 et 2 et (*A, B*) sur la machine 3 nous fournissent un mean flow time de 9.0.

2. Résolution par l'algorithme de Johnson du flow shop à 2 - machines

Chaque job consiste en la donnée d'un couple (A_i, B_i) où :

A_i est le temps d'exécution du job i sur la première machine.

B_i représente le temps d'exécution du job i sur la seconde machine.

Cet ordre sur les machines est le même pour chacun des n jobs, aussi il est permis qu'un A_i ou un B_i soit nul, puisque certain jobs peuvent n'avoir qu'une seule opération. Les contraintes normales d'un atelier simple sont supposés réalisées.

1) Chaque machine j ne peut exécuter qu'un job i à la fois et
2) Chaque job i ne peut être exécuté que par une machine en tout instant. Le job i ne s'exécutera sur la machine 2 que s'il a terminé son exécution sur la machine 1.

Le problème alors est : Soit $2n$ valeurs $A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_n$. Trouver un ordre d'exécution des jobs sur chacune des deux machines tel que ni les contraintes de précédences ou d'occupation ne soient violés et le maximum des $F_i = C_i - r_i$ est rendu aussi petit soit-il.

La préemption et l'oisiveté des machines ne sont pas autorisées.

Le makespan d'une séquence de jobs peut être représenté par le temps de fin d'exécution d'un réseau PERT.

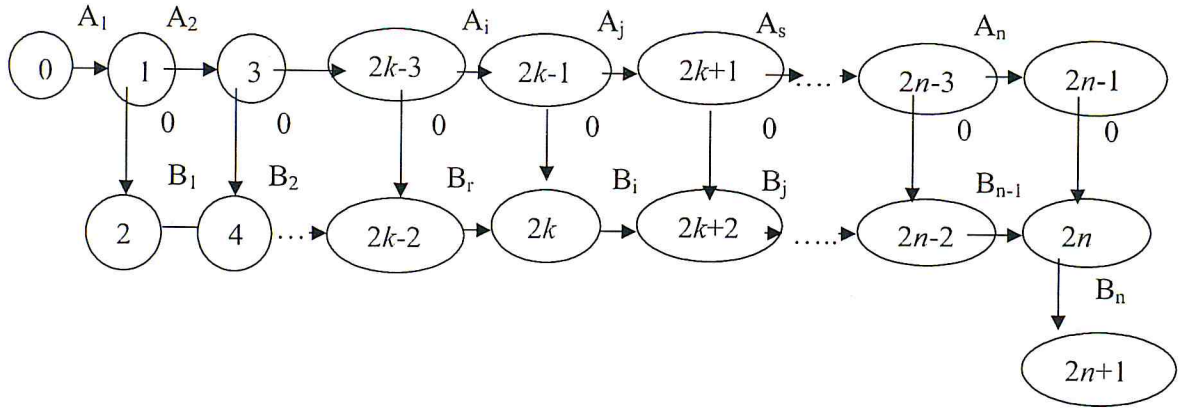


Figure 3: Réseau pour la suite des jobs $(j_1, j_2, \ldots, r, i, j, s, \ldots, j_n)$

Où 0 est le temps de transport de la machine 1 à 2.

Les nœuds début (0) et nœud de fin ($2n$+1) représentent respectivement le début et la fin d'exécution de l'ordonnancement. Les arcs ($2k$ - 3, $2k$ -1), ($2k$, $2k$+2) décrivent l'exécution du job i sur la machine 1 et 2 respectivement.

Les arcs verticaux représentent des activités de durées nulles.

Le makespan est la longueur du plus long chemin du nœud 0 au nœud $2n$ +1.

On notera par :

$A_{[i]}$: le temps d'exécution sur la machine 1 du job qui est à la ième position dans la séquence d'ordonnancement. Le dernier job ne peut terminer son exécution plutôt que le temps requis pour exécuter tous les jobs sur la 1ère machine plus le temps qu'il faut pour l'exécuter sur la machine 2, soit $C_{max} \geq A_{[1]} + \sum_{i=1}^{n} B_{[i]}$. Aussi $C_{max} \geq \sum_{i=1}^{n} A_{[i]} + B_{[n]}$

$\sum_{i=1}^{n} A_{[i]}$ et $\sum_{i=1}^{n} B_{[i]}$ sont directement données par l'information et entièrement indépendant de l'ordre des jobs. Alors pour réduire ses bornes on ne peut influencer que sur $B_{[n]}$ $A_{[1]}$ par le choix de la suite. On choisira la plus petite des $2n$ valeurs, si c'est un A_i on mettra le job i en premier. Si c'est un B_i on le mettra à la fin de l'ordonnancement. On peut répéter

essentiellement le même argument sur les (n-1) jobs restants. Si cette borne est atteinte alors cette construction donne un bon ordonnancement. Cet argument des bornes est précisément la base de la procédure de Johnson et il offre la preuve de son optimalité.

Soit $X_{[i]}$ le temps d'oisiveté sur la machine 2 précédent $B_{[i]}$. Un ordonnancement type ressemble à ce qu'il suit :

Les valeurs de $X_{[i]}$ peuvent être données en fonctions des $A_{[i]}$ et $B_{[i-1]}$ par les relations

$X_{[1]} = A_{[1]}$

$X_{[2]} = max\ (\ A_{[1]} + A_{[2]} - B_{[1]} - X_{[1]},\ 0\ )$

$X_{[3]} = max\ (\ A_{[1]} + A_{[2]} + A_{[3]} - B_{[1]} - B_{[2]} - X_{[1]} - X_{[2]}\ ,\ 0\ )$

et en général on peut écrire

$$X_{[j]} = max\ (\ \sum_{i=1}^{j} A_{[i]} - \sum_{i=1}^{j-1} B_{[i]} - \sum_{i=1}^{j-1} X_{[i]}\ ,\ 0\ ).$$

Les sommes partielles des X peuvent être obtenues des expressions

$X_{[1]} = A_{[1]}$

$X_{[1]} + X_{[2]} = max\ (\ A_{[1]} + A_{[2]} - B_{[1]}\ ,\ A_{[1]}\ )$

$X_{[1]} + X_{[2]} + X_{[3]} = max\ (\ A_{[1]} + A_{[2]} + A_{[3]} - B_{[1]} - B_{[2]}\ ,\ X_{[1]} + X_{[2]}\ ,\ A_{[1]}\ )$

$$= max\ (\ \sum_{i=1}^{3} A_{[i]} - \sum_{i=1}^{2} B_{[i]}\ ,\ \sum_{i=1}^{2} A_{[i]} - B_{[1]},\ A_{[1]}\ ).$$

En général on aura :

$$\sum_{i=1}^{j} X_{[i]} = max(\ \sum_{i=1}^{j} A_{[i]} - \sum_{i=1}^{j-1} B_{[i]}\ ,\ \sum_{i=1}^{j-1} A_{[i]} - \sum_{i=1}^{j-2} B_{[i]}\ ,\ldots,\ \sum_{i=1}^{2} A_{[i]} - B_{(1)}\ ,\ A_{(1)}\ )$$

$$Si\ Y_j = \sum_{i=1}^{j} A_{[i]} - \sum_{i=1}^{j-1} B_{[i]}\ \ alors\ \ \sum_{i=1}^{j} X_{[i]} = max\ (\ Y_1\ ,\ Y_2\ ,\ ..\ ,\ Y_j\ ).$$

Si $F_{max}(S)$ désigne le maximum flow-time d'un ordonnancement particulier S alors

$$C_{max}(S) = F_{max}(S) = \sum_{i=1}^{n} B_{[i]} + \sum_{i=1}^{n} X_{[i]} = \sum_{i=1}^{n} B_{[i]} + max\ (\ Y_1\ ,\ Y_2\ ,\ ..\ ,\ Y_n\ )$$

Puisque la somme des B_i est indépendante de la séquence, le flow-time maximum dépend entièrement de la somme des intervalles des temps d'oisiveté sur la seconde machine et c'est équivalent à max Y_j. Notre but est de trouver S^* tel que $F_{max}(S^*) \leq F_{max}(S)$ pour tout S.

La règle qui minimise le makespan pour le problème de deux machines est celle de Johnson.

Théorème 3 :

Dans un flow shop $n / 2 / F / C_{max}$ où tous les jobs sont disponibles à la même date, le job j précédera le job $j+1$ si $Min\ (A_j, B_{j+1}) < Min\ (A_{j+1}, B_j)$.

Algorithme 1 (de Johnson)

Etape 1. Poser k = 2, l = n

Etape 2. Poser une liste actuelle d'ordonnancement de tâches $\{ J_1, J_2, \ldots, J_n \}$.

Etape 3. Trouver le plus petit temps parmi les a_i et les b_i

Etape 4. Si le plus petit temps est de J_i dans la première machine, i.e.a_i est le plus petit, alors:

i. Placer J_i dans la k^{eme} position.
ii. Supprimer J_i de la liste actuelle.
iii. Incrémenter k à k+1.
iv. Aller à l'Etape 6.

Etape5. Si le plus petit temps est de J_i dans la deuxième machine, i.e.b_i est le plus petit, alors:

i. Placer J_i dans la $1^{ère}$ position.
ii. Supprimer J_i de la liste actuelle.
iii. Réduire l à l-1.
iv. Aller à l'Etape 6.

Etape 6. S'il y a des jobs non ordonnancés, aller à l'Etape 3, sinon arrêter.

A l'Etape 3, s'il existe plus d'un job, alors, choisir J_i arbitrairement.

Exemple 2 : Ordonnancer les tâches dans le problème 7 / 2 / F / C_{max}

Tableau 4 : Donnée de l'exemple 2

Job	M_1	M_2
1	6	3
2	2	9
3	3	3
4	1	8
5	7	1
6	4	5
7	7	6

Job 4 ordonnancé : 4 — — — — — —

Job 5 ordonnancé : 4 — — — — — 5

Job 2 ordonnancé : 4 2 — — — — 5

Job 3 ordonnancé : 4 2 — — — 3 5

Job 1 ordonnancé : 4 2 — — 1 3 5

Job 6 ordonnancé : 4 2 6 — 1 3 5

Job 7 ordonnancé : 4 2 6 7 1 3 5

La procédure décrite auparavant découle directement de cette inégalité. L'ordonnancement optimal résultant est unique s'il y a des inégalités entre chaque couple de jobs, mais il peut y avoir un ensemble d'ordonnancements optimaux s'il y a égalité entre quelques couples de jobs.

Théorème 4 : pour le problème $n / 2 / F / C_{max}$ avec $p_{i1} = a_i$ *et* $p_{i2} = b_i$, $i = 1, \ldots, n$.

i) si $a_k = min \{ a_1, \ldots, a_n, b_1, \ldots, b_n \}$ alors il y a un ordonnancement optimal où le job J_k est placé à la première position de l'ordonnancement.

ii) si $b_k = min \{ a_1, \ldots, a_n, b_1, \ldots, b_n \}$ alors il y a un ordonnancement optimal où le job J_k est placé à la dernière position de l'ordonnancement.

La règle de Johnson peut se formuler d'une autre manière. Divisons les jobs en deux sous-ensembles. L'ensemble 1 contient tous les jobs tel que $p_{1j} < p_{2j}$ et l'ensemble deux contient tous les jobs tel que $p_{1j} > p_{2j}$. S'il y a égalité le job est dans un des deux sous-ensembles. On exécute les jobs du premier ensemble dans l'ordre croissant des p_{1j} (SPT) et suivent les jobs du second ensemble dans l'ordre décroissant des p_{2j} (LPT). L'ordonnancement peut ne pas être unique.

Théorème 5 : Tout ordonnancement SPT(1)-LPT(2) est optimal pour $F2//C_{max}$.

3. Le job shop ou le flow shop généralisé sur 2-machines, n / 2 / G / C_{max}

3.1. Algorithme de Johnson

On a supposé que les tâches doivent s'exécuter sur toutes les machines. Relaxant cette contrainte. Supposons qu'on a un ensemble de jobs $\{J_1, ..., J_n\}$ qu'on peut partitionner en quatre sous-ensembles de jobs.

Type A : les jobs qui ne sont exécutés que sur la machine M_1

Type B : les jobs qui ne sont exécutés que sur la machine M_2

Type C : les jobs qui sont exécutés sur la machine M_1 puis sur M_2

Type D : les jobs qui sont exécutés sur la machine M_2 puis sur M_1

La solution optimale est de la forme :

1) ordonnancer les jobs de type A dans un ordre quelconque pour obtenir une séquence S_A
2) ordonnancer les jobs de type B dans un ordre quelconque pour donner une séquence S_B
3) ordonnancer les tâches de type C selon la règle de Johnson pour donner S_C
4) ordonnancer les jobs de type D selon la règle de Johnson pour donner S_D (en permutant le rôle des machines, la machine 2 devient machine 1' et machine 1 devient 2').

Un ordonnancement optimal est alors

Machine	Ordre d'exécution
M_1	(S_C, S_A, S_D)
M_2	(S_D, S_B, S_C)

Exemple 3 :

Soit *9 / 2 / G / C_{max}* dont les données sont l'ordre de passage sur les machines et les temps d'exécution des tâches.

Tableau 5: Donnée de l'exemple 3

Tâche	Première machine		Seconde machine	
1	M_1	8	M_2	2
2	M_1	7	M_2	5
3	M_1	9	M_2	8
4	M_1	4	M_2	7
5	M_2	6	M_1	4
6	M_2	5	M_1	3
7	M_1	9	-----	
8	M_2	1	-----	
9	M_2	5	-----	

Pour déterminer un ordonnancement optimal.

Les jobs de type *A* : job 7 sur la machine M_1

Les jobs de type *B* : job 8 et 9. Choisissons l'ordre (8, 9).

Les jobs de type *C* : job 1, 2, 3 et 4. La règle de Johnson donne (4, 3, 2, 1).

Les jobs de type *D* : job 5 et 6. La séquence selon Johnson est (5, 6).

L'ordonnancement optimal est:

Machine M_1	(4, 3, 2, 1, 7, 5, 6)
Machine M_2	(5, 6, 8, 9, 4, 3, 2, 1)

Par un diagramme de Gantt, C_{max} vaut 44 unités.

4 Minimiser le flow-time moyen dans le flow-shop à deux machines (n / 2 / F / $\overline{F}$)

Aussi pour le flow shop, la minimisation du flow-time moyen est un problème très difficile. Quoique le théorème de Johnson reste applicable, il est nécessaire de considérer seulement les ordonnancements où la séquence des jobs est la même sur chaque machine. Aucun algorithme constructif comparable à celui de Johnson n'est connu. La procédure de Johnson n'est pas optimale pour le critère et en général elle n'est pas assez bonne. Des exemples existent où SPT est meilleure que celle de Johnson. Une expression générale pour le

flow-time pondéré de n jobs peut être obtenue par un argument qui suit directement la règle de Johnson.

1) Soit $X_{[i]}$ le temps d'oisiveté qui précède l'opération $B_{[i]}$ sur la seconde machine.

2) Soit $Y_j = \sum_{i=1}^{j} A_{[i]} - \sum_{i=1}^{j-1} B_{[i]}$, le flow-time de chaque job est:

$F_{[1]} = A_{[1]} + B_{[1]} = X_{[1]} + B_{[1]}$

$F_{[2]} = X_{[1]} + B_{[1]} + X_{[2]} + B_{[2]}$

$$F_j = \sum_{i=1}^{j} B_{[i]} + \sum_{i=1}^{j} X_{[i]} = \sum_{i=1}^{j} B_{[i]} + max\,(Y_1, Y_2, .., Y_J)$$

$$\overline{F} = 1/n \left(\sum_{j=1}^{n} \sum_{i=1}^{j} B_{[i]} + \sum_{j=1}^{n} max\,(Y_1, Y_2, .., Y_J) \right).$$

On peut énoncer que si deux jobs i et j sont adjacents et si $A_j \geq A_i$ *et* $B_j \geq B_i$ alors le job i précède le job j.

5. Cas de trois machines n / 3 / F / C_{max}

La règle de Johnson peut être étendue au cas de trois machines. On aura besoin des conditions dites de dominance, $\min_i a_i \geq \max_i b_i$ ou $\min_i c_i \geq \max_i b_i$.

Dans ce cas, on regroupe deux machines en une seule et le problème devient celui à deux machines en posant $a'_i = a_i + b_i$ *et* $b'_i = b_i + c_i$.

Exemple 4: Soit 6 / 3/ F / C_{max}. Les valeurs données sont celles des temps d'exécution.

Tableau 6: données de l'exemple 4

Job	M_1	M_2	M_3	1ère machine	2nd machine
1	4	1	3	2	4
2	6	2	9	6	11
3	3	1	2	4	3
4	5	3	7	8	10
5	8	2	6	10	8
6	4	1	1	5	2

$\min_i\{a_i\} = 3$, $\max_i\{b_i\} = 3$ *et* $\min_i\{c_i\} = 1$. D'où les formules de dominance sont vérifiées

M_1	J_2		J_4		J_5		J_1		J_3		J_6	
	6		11		19		23		26		30	
M_2		J_2		J_4			J_5		J_1			J_6
		6	8	14			21		27		31	
M_3				J_2		J_4		J_5		J_1	J_3	J_6
			8		17		24		30		33	35

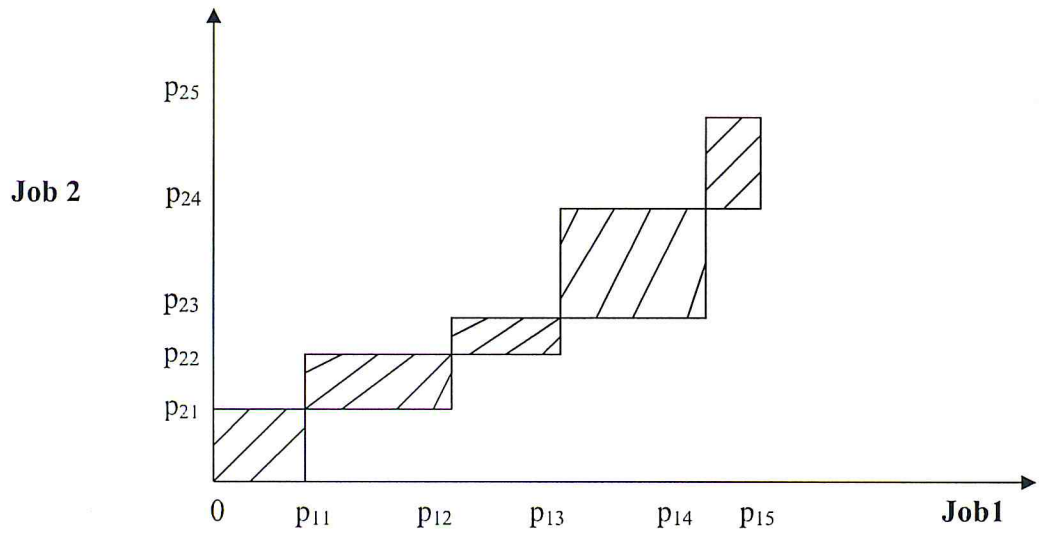
Figure 4: Diagramme de Gantt

(2, 4, 5, 1, 3, 6) est l'ordonnancement optimal.

6. Solution graphique d'Akers pour 2 / m / F / C_{max}

C'est le cas de 2 jobs et " *m* " machines, $m = 5$.

L'axe des abscisses est représenté par le job 1 (p_{11}, p_{12}, p_{13}, p_{14} *et* p_{15}) et des ordonnées par le job 2 (p_{21}, p_{22}, p_{23}, p_{24} *et* p_{25}).



Les zones hachurées est une zone d'interdiction. La même machine ne peut pas exécuter deux tâches en même temps. Un ordonnancement est représenté par le diagramme en traçant une ligne de 0 (0, 0) à ($\sum_{j=1}^{5} p_{1j}$, $\sum_{j=1}^{5} p_{2j}$).

Un ordonnancement peut être composé de segments de droites.

i) horizontalement, représentant l'exécution sur la tâche 1 seulement.

ii) Verticalement, représentant l'exécution sur la tâche 2 seulement

iii) A 45° oblique, représentant l'exécution sur les deux machines.

Le maximum flowtime est : $C_{max} = \sum_{j=1}^{5} p_{1j}$ + la somme des longueurs de segments verticaux

$= \sum_{j=1}^{5} p_{2j}$ + la somme des longueurs de segments horizontaux.

Dans la littérature, des notions de dominance entre les tâches peuvent être définies, la valeur exacte du makespan est parfois établie. Une machine ayant la plus petite vitesse est appelée *"machine goulot" ou "Bottleneck machine"*. Des cas où les capacités intermédiaires de stockage entre les machines sont limitées, des problèmes sont étudiés et des résultats sont obtenus. Le problème Fm // C_{max} est montré fortement NP-difficile (pour tout $m \geq 3$).

Chapitre 2 : Le problème du Job shop avec convoyeur

Dans un atelier, un chariot ou une navette ou un convoyeur désigne le moyen de transport des tâches. Le transport d'une tâche peut être décrit comme son chargement sur le convoyeur, du transport entre deux machines et du déchargement de la tâche. L'opération de transport englobe toutes ses actions ou considère séparément chacune d'elles comme une opération distincte. Les temps de transport associés à ces opérations peuvent être dépendants ou non des tâches transportées. L'introduction d'un convoyeur nécessite une extension de la notation à trois champs de Graham *et al.* [12] à $\alpha(k) / \beta / \gamma$ où k désigne le nombre de convoyeurs dans l'atelier. Aux contraintes « classiques » est ajoutée celle du nombre de types de tâches qui seront exécutées dans l'atelier. Le champ β se note J.

$J2(1) / J > 1 / C_{max}$ désigne un job shop à deux machines et un convoyeur devant exécuter au moins deux sortes de tâches afin de minimiser le makespan.

Une autre extension de la notation est proposée par Lee et Chen [13] où une lettre T est rajoutée à l'environnement machine où ($v = x$ et $c = y$) indique respectivement le nombre de convoyeurs et leur capacité. Selon cette dernière notation, $TJ2 / v = 1, c=2 / C_{max}$ représente un job shop à un seul convoyeur pouvant transporter deux tâches à la fois et dont l'objectif est la minimisation de la longueur d'ordonnancement.

Dans la suite de ce mémoire, le Job shop à deux machines se notera 2- Job shop.

1. Types de convoyeurs dans l'ordonnancement des ateliers

Une tâche est dite semi-finie si elle est exécutée par au moins une machine, mais dont toutes ses opérations élémentaires ne sont pas encore exécutées. Nous considérons les problèmes où les tâches sont semi-finies. Elles sont transportées par un convoyeur entre les machines.

Une fois finie, ces tâches sont délivrées aux clients par d'autres moyens de transport tel des camions, train, etc. Le temps de transport est assimilé au temps de transport par le convoyeur dans l'atelier. Lee *et al.* considèrent trois catégories d'ateliers :

- les cellules robotiques.
- les ateliers possédant un ou plusieurs véhicules guidés et automatisés appelés AVG. Les véhicules guidés et automatisés sont des systèmes de transport de tâches sans conducteur.
- les ateliers possédant une ou plusieurs grues cycliques sujets à des contraintes de temps.

Ces ateliers peuvent être vues comme des cas particuliers de $J_m(K) / J > 1 / \pi_{min}$, à contraintes de fenêtres de temps et où les tâches ne peuvent attendre entre les machines de peur d'être dégradées. Ces contraintes, bien que restrictives, décrivent des ateliers que l'on trouve le plus souvent dans les industries chimiques et de galvanoplasties.

Elles sont représentées par la figure ci-dessous.

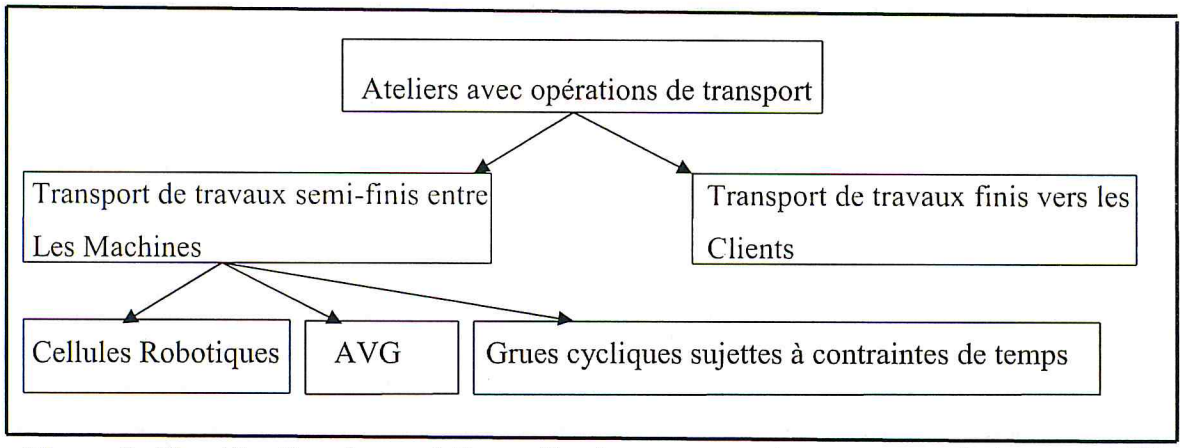


Figure 5: Classification des problèmes d'ordonnancement avec moyen de transport [6]

Les études de ces ateliers sont apparues dans la littérature en 1983 [14]. Les véhicules circulent sur un réseau guidé. La politique d'ordonnancement doit prendre en compte le trafic dans les ateliers et les collisions possibles entre les convoyeurs. Dans la grande distribution, ces véhicules transportent les différents produits à l'intérieur des entrepôts. La galvanoplastie consiste à traiter chimiquement une pièce en l'immergeant dans une solution dans le but d'y déposer une couche de métal. Les pièces déplacées par les grues ne doivent donc pas rester ni trop peu, ni trop longtemps dans la solution.

Ces trois types de moyens de transport dans un atelier sont, eux-mêmes, subdivisés en fonction de la taille des zones de stockage de l'atelier.

Les zones de stockage sont les aires où les tâches sont entreposées dans l'attente d'être exécutées par une machine ou d'être transportées par un convoyeur.

Elles sont de deux types, d'entrée et de sortie. Les zones de stockage d'entrées sont les aires où les tâches sont placées dans l'attente de leurs exécutions sur une machine.

Les zones de stockage de sorties reçoivent les tâches exécutées. Si celles-ci sont semi-finies, elles se trouvent dans une période d'attente d'un convoyeur pour continuer leur gamme d'exécution, ou elles ont terminées leur exécution. Une zone de stockage, d'entrée ou de sortie, peut être réservée à une machine particulière ou servir à plusieurs machines. Si de tels

espaces n'existent pas ou sont de tailles limitées alors les machines peuvent avoir à attendre une tâche ou bien devoir garder une tâche exécutée en l'absence de convoyeur.
Dans ces conditions, ces machines seront bloquées. De même, les convoyeurs peuvent se retrouver bloqués. Ces cas se produisent lorsque le convoyeur se trouve devant une machine qui n'a pas fini l'exécution de sa tâche ou si l'espace de stockage de la machine sur laquelle il arrive est plein, ce qui l'empêche alors de déposer la tâche qu'il transporte.

2. Le job shop avec contraintes de transport

L'introduction des contraintes de transport dans un job shop implique que les convoyeurs doivent être pris en compte dans l'ordonnancement de l'atelier. Deux choix sont possibles : soit les tâches sont ordonnancées puis les convoyeurs sont agencés en fonction de l'ordonnancement des tâches [15]. Soit, une approche globale est effectuée où les tâches et les convoyeurs sont ordonnancés en même temps.
Dans ce mémoire nous nous limitons à étudier le job shop à un seul convoyeur.
Bécart et *al.* [16] modélisent ce problème comme celui d'un programme linéaire mixte et le résolvent par une méthode de séparation et d'évaluation progressive. Hurink et Knust [17] appliquent une recherche taboue à ce type d'atelier. Ils résolvent le problème de job shop à m machines possédant des espaces de stockage de taille illimitée et un seul convoyeur suivant deux approches. La première considère une approche globale des machines et du convoyeur. La seconde traite les machines et le convoyeur séparément. Brucker et Knust [18] proposent des bornes inférieures. Elles sont basées sur la seconde approche proposée par Hunrik et Knust [17].

3. Présentation du problème

Selon Lee et *al.* [2], le problème de 2- Job shop à un convoyeur pour minimiser le makespan se note $J2(l) / J > 1 / C_{max}$. Ce problème est constitué de deux stations identiques nommées respectivement station A et station B.
Comme le montre la Figure 6, une station est composée d'une machine et de deux aires de stockages, zone de stockage d'entrée et zone de stockage de sortie. La machine exécute en premier lieu, les tâches se trouvant dans la zone de stockage d'entrée. Une fois l'exécution de la tâche achevée, celle-ci est placée sur la zone de stockage de sortie.

Une zone de stockage est le lieu où les tâches en attente sont laissées temporairement en dépôt. Elle fonctionne comme une liste ou file d'attente de type FIFO, premier arrivé, premier sorti. La zone de stockage et stock seront synonymes.

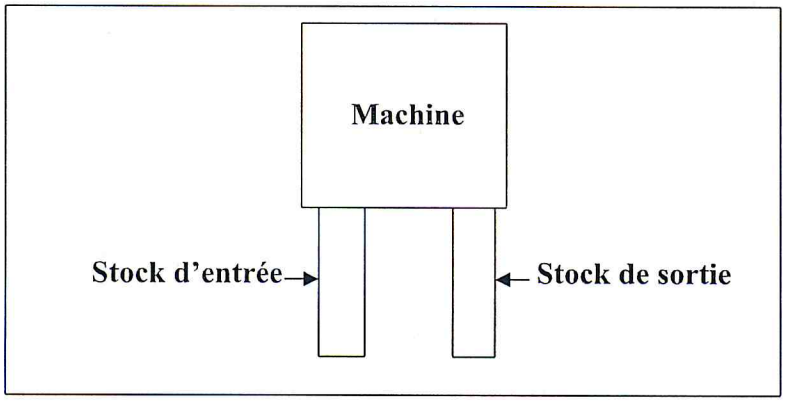


Figure 6: Une Station dans un atelier [6]

On doit exécuter un ensemble de n tâches, $N = \{1, 2, ..., n\}$.

Chaque tâche $j \in N$, consiste en deux opérations: O_{ij}, $j \in N$ et $i \in \{A, B\}$.

O_{Aj} et O_{Bj} sont exécutés respectivement sur les machines A et B avec respectivement des temps d'exécutions t_{Aj} et t_{Bj}.

Un temps d'exécution t_{ij} est la somme des temps nécessaires à une tâche pour passer de la zone de stockage d'entrée à la machine, noté t_E, à son exécution sur la machine, noté t_M et son placement sur la zone de stockage de sortie, noté t_S.

Une équation linéaire est établie et est donnée par la formule :

$$\forall j \in \mathbb{N}, \forall i \in \{A, B\}, t_{ij} = t_{Eij} + t_{Mij} + t_{Sij} \text{ tel que } \exists\, (t_{ij}, t_{Eij}, t_{Mij}, t_{Sij}) \in \mathbb{N}^4$$

Une tâche ayant un temps d'exécution nul sur l'une des deux machines n'aura donc pas besoin d'être transportée d'une machine à une autre. Elle sera considérée comme achevée à la fin de son exécution sur la machine où elle s'exécute.

Le temps d'exécution d'une tâche est la somme des trois temps définis auparavant par t_E, t_M et t_S.

Dans la suite, tous les temps considérés sont à valeur entière.

Si une tâche j a un temps d'exécution identique sur les deux machines alors ce temps sera noté simplement par t_j. t_i représente aussi un temps d'exécution identique sur une station i pour toutes les tâches.

Une tâche qui s'est exécutée sur au moins une machine et qui a terminé son exécution est dite « tâche réalisée ». Elle ne reste pas dans la zone de stockage de sortie de la dernières machine qui l'a exécuté, elle sort directement du système.

A tout instant *t*, une tâche est au plus exécutée par une machine et une machine n'exécute pas plus d'une tâche à la fois.

La préemption n'est pas autorisée.

Les temps de transport surviennent lorsqu'une tâche, semi-finie, se trouve sur la zone de stockage de sortie de la station qui l'a exécuté, et que sa gamme d'exécution indique son passage sur l'autre machine. Il correspond au temps que met le convoyeur pour transporter une tâche *j* d'une station à une autre. Il est la somme du temps de chargement de la tâche *j* sur le convoyeur, de son temps de transport et du temps de son déchargement sur la zone de stockage d'entrée de la station d'arrivée.

De façon similaire aux temps d'exécution, les temps de transport sont considérés globalement. Ils ne seront donc pas séparés en trois temps différents.

Si *j* est l'indice de la tâche transportée, *O* est la station de départ du convoyeur, *D* sa station d'arrivée où *O* ne peut être la même station que *D*, un temps de transport τ_{ODj} est défini pour tout, $j \in N$, $(O,D) \in \{A,B\}$ et $O \neq D$.

τ_{ABj3} représente le temps que met la tâche J_3 pour être transportée de la station *A* vers la station *B*.

τ_{BA} indique que tous les temps de transport pour aller de la station *B* vers la station *A* sont identiques.

Le convoyeur ne peut transporter qu'une tâche à la fois. Initialement, il est placé devant la sortie d'une zone des stockages de sorties de l'une des stations *A* ou *B*.

4. Fonctionnement de l'atelier

Les deux stations sont considérées disponibles et prêtes à fonctionner.

Les tâches doivent, tout d'abord, être placées sur les zones de stockages d'entrées des deux stations. La façon dont les tâches sont placées dans les zones de stockages d'entrées est appelée « politique de placement ».

Le temps de placement des tâches, n'est pas pris en compte dans le makespan.

Une politique du convoyeur est supposée définie.

Ces deux politiques de placement, des tâches sur les machines et celle du convoyeur sont illustrées sur l'exemple ci-dessous.

Soit $N = \{j_1, j_2, j_3, j_4, j_5\}$ un ensemble de cinq tâches, leur placement est décrit par la Figure 7

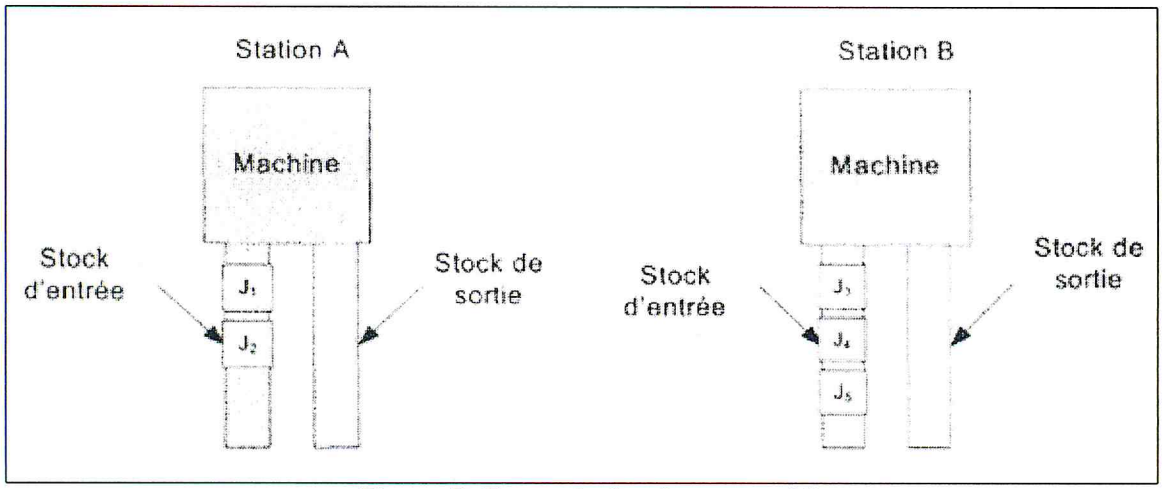


Figure 7: Exemple de placement [6]

Les tâches ayant une gamme d'exécution partant de la station A et terminant sur la station B forment le sous-ensemble N_{AB}, les tâches qui ont une gamme d'exécution de la station B à la station A constituent le sous-ensemble N_{BA}.

De notre exemple ci-dessus, $N_{AB} = \{j_1, j_2\}$ et $N_{BA} = \{j_3, j_4, j_5\}$.

Une fois le placement des tâches opéré sur les stations, les premières tâches se trouvant sur chaque zone de stockage d'entrée, j_1 et j_3 sont placées et exécutées respectivement sur les machines A et B. A la fin d'exécution de ces deux tâches, les machines exécutent immédiatement toutes les tâches disponibles dans leurs zones de stockages d'entrée.

Eu égard à la politique imposée au convoyeur, ce dernier transporte les tâches semi-finies d'une machine vers l'autre. La Figure 8 présente ce fonctionnement.

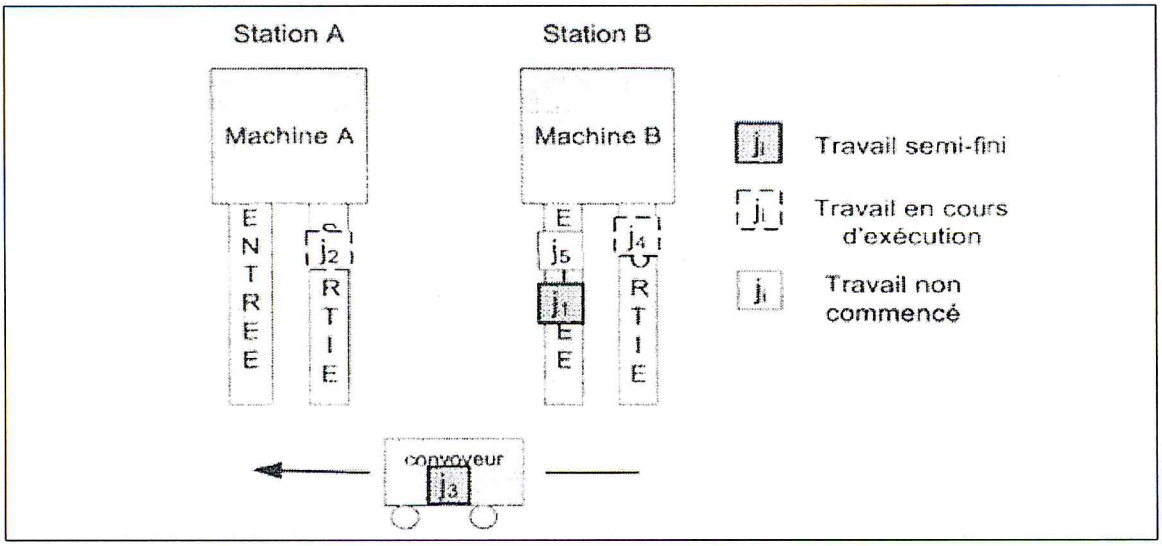


Figure 8: L'atelier en cours de fonctionnement [6]

j_1 étant exécutée sur la machine A est semi-finie. A la suite de son transport par le convoyeur, elle se trouve en attente dans la zone de stockage d'entrée de la station B.

j_3 est la première tâche exécutée sur la machine B. Elle est véhiculée vers la zone de stockage d'entrée de la station A.

Les tâches j_2 *et* j_4 sont en cours d'exécution respectivement sur leurs machines A et B.

j_5 n'a pas encore commencé son exécution. Elle est donc en attente d'exécution.

Tout travail dans l'atelier s'arrête quand toutes les tâches sont réalisées ou toutes leurs opérations élémentaires sont exécutées. Le makespan est le temps écoulé depuis le début de l'ordonnancement jusqu'à l'arrêt des travaux dans l'atelier.

Il est associé au convoyeur un temps de transport correspondant à la durée de tous les temps de transport des tâches dans l'atelier, jusqu'à déchargement de la dernière tâche

5. Complexité du problème

Hunrik et Knust [5] ont montré que le problème de job shop à deux machines et un convoyeur avec des espaces de stockages illimités est NP-difficile, même si les temps de transport sont unitaires et que le temps de retour à vide du convoyeur est nul.

Il est un cas particulier de notre problème étudié, en prenant $N_{BA}= \emptyset$.

6. Quelques propriétés de l'atelier

Des propriétés particulières de l'ordonnancement du 2- Job shop avec convoyeur sont établies par Ruhlmann [6]. Elles sont données ici sans démonstration.

Propriété 1 : Le makespan du convoyeur est toujours inférieur à celui de l'atelier.

Propriété 2 : Le makespan de l'atelier correspond au maximum des temps d'accomplissement totaux des deux stations.

Propriété 3 : II n'y a pas de blocages possibles des machines ou du convoyeur.

Propriété 4 : Les premières opérations des tâches sont ordonnancées de manière continue.

Propriété 5 : Si les temps d'exécutions des tâches sont tous égaux et que les temps de transport sont quelconques alors ordonner les tâches dans un sens croissant ou décroissant de leur temps de transport n'est pas nécessairement optimal.

Propriété 6 : Si pour toutes les tâches, les temps de transport (y compris les temps de transport à vide éventuels) sont strictement inférieurs aux temps d'exécution alors n'importe quelle permutation de tâches est optimale.
Le makespan est de valeur $C_{max} = \max \{\sum_{i=1}^{n} t_{Ai}, \sum_{i=1}^{n} t_{Bi}\}$.

Propriété 7 : Pour minimiser le temps d'utilisation du convoyeur, il doit être positionné sur la station possédant le plus grand nombre de tâches dans sa zone de stockage d'entrée.

Supposons que les temps de transport des tâches sont strictement supérieurs à leur temps d'exécution.
Propriété 8 : Sur la zone de stockage contenant le plus grand nombre de tâches, en face de la machine, placer en première position la tâche qui a le temps d'exécution minimal sur cette machine, et à la dernière position la tâche qui a le temps d'exécution minimal sur l'autre machine.
Un ordonnancement similaire se fait sur la station qui a le moins de tâches.
Cette stratégie n'est pas optimale.

Le makespan est dépendant du comportement du convoyeur et de l'ordonnancement des tâches. Il est intéressant d'étudier le seul problème du convoyeur.

7. Étude du convoyeur

Lorsque le convoyeur arrive chargé d'une tâche semi-finie à une station, il la dépose sur sa zone de stockage d'entrée. Trois cas peuvent être envisagés.

Soit une tâche semi-finie est disponible dans la zone de stockage de sortie, il se chargera de la transporter à l'autre station pour être exécutée.

Si après déchargement de la tâche semi-finie dans la zone de stockage d'entrée, aucune tâche n'est disponible sur la zone de stockage de sortie, deux alternatives sont possibles.

Le convoyeur peut attendre une tâche qui s'exécute sur la machine pour la charger ou repartir à vide, sans charger de tâche.

Il est à rappeler qu'une autre décision à prendre pour le convoyeur est le choix de son placement au départ de fonctionnement de l'atelier (propriété 7). Le convoyeur doit être placé à la sortie d'une des zones de stockages de sorties d'une des deux stations.

Le critère de choix entre l'attente ou le retour à vide du convoyeur est discuté.

Supposons qu'un convoyeur est en face d'une station. Une tâche est en cours d'exécution sur la dite-machine et qu'aucune tâche n'est disponible dans la zone de stockage de sortie de la dite-station. Le convoyeur ne partira à vide que s'il y a au moins une tâche dans la station opposée, en exécution de temps restant pour fin d'exécution $t_{execRestantStationOpposée}$ ou en attente dans la zone de sortie et que le temps de transport $t_{transport}$ de cette tâche plus le temps de retour à vide $t_{retourVide}$ est inférieur au temps qui reste $t_{attente}$ pour achever l'exécution de la tâche en cours d'exécution.

Une inégalité s'obtient : $t_{retourVide} + t_{execRestantstationOpposée} + t_{tranport} < t_{attente}$

Le tableau ci-dessous résume les différents comportements du convoyeur en fonction des situations possibles.

Tableau 7: Comportements du convoyeur en fonction des différentes situations [6]

Comportement du convoyeur	Situations
Chargement de la tâche et transport vers la station opposée.	Une tâche est en attente dans le stock de sortie de la machine.
Attente	Tâche en attente dans le stock de sortie ou tâche en cours d'exécution sur la station opposée et : $t_{\text{retourVide}}+t_{\text{execRestantstationOpposée}}+t_{\text{tranport}} \geq t_{\text{attente}}$
Retour à vide	Tâche en attente dans le stock de sortie ou tâche en cours d'exécution sur la station opposée et : $t_{\text{retourVide}} + t_{\text{execRestantstationOpposée}}+t_{\text{tranport}} < t_{\text{attente}}$

8. Calcul d'une borne inférieure

Pour pouvoir juger la qualité des solutions obtenues, il faut un élément de comparaison. Le mieux étant bien sûr de prendre comme élément comparateur la solution optimale. Or, aucune solution optimale ne peut être déterminée analytiquement. Un autre critère est à prendre en considération. C'est celui de la détermination d'une « borne inférieure ».

La charge de travail d'une station i correspond à la somme des temps d'exécution des tâches que la station doit exécuter, elle est notée $w_i(Q)$ avec Q un sous ensemble de N.

Pour les deux stations A et B, la charge est définie par l'expression

$$W_A(Q) = \sum_{j \in Q} t_{\text{Aj}} \quad et \; W_B(Q) = \sum_{j \in Q} t_{\text{Bj}}$$

Par convention $w_A(\emptyset) = w_B(\emptyset) = 0$.

Si une station n'a pas de tâches à exécuter alors sa charge de travail est nulle.

Si $Q = N$, alors $W_A(N)$ représente la charge totale de travail de la station A.

Une borne inférieure, notée LB est définie dans Strasevitch [19] par :

$$LB = \max\left\{\sum_j \tau_{\text{j}}, W_A(N), W_A(N), \max\{t_{\text{Aj}} + \tau_{\text{j}} + t_{\text{Bj}}\} + \min\{\tau_{\text{k}}\} | j, k \in \mathbb{N}\right\}$$

La relation $LB \leq C_{max}$ est ainsi toujours vérifiée. En effet, la charge totale des machines ne prenant en compte que les temps d'exécution des tâches, le makespan de l'atelier ne peut que

leur être supérieur ou égal. Le cas d'égalité se produit dans les cas où les machines n'ont aucun temps mort durant la totalité de fonctionnement de l'atelier.

Comme les temps d'attente ou de retour à vide ne sont pas considérés, la somme des temps de transport effectués est inférieure au temps de travail du convoyeur.

Or, il a été prouvé que le temps de travail du convoyeur était inférieur au makespan. Par conséquent, l'inégalité $\sum_j \tau_j < C_{max}$ en découle.

II est prouvé que le makespan de l'atelier sera toujours supérieur à la somme constituée par la tâche la plus longue et le temps de transport le plus court [6].

La tâche la plus longue est la tâche dont le temps constitué par ses temps d'exécution sur les deux stations et son temps de transport est le plus long.

Propriété 9 : $\max\{t_{Aj} + \tau_j + t_{Bj}\} + \min\{\tau_k\} \leq C_{max}$

9. Objectifs de notre travail

Le job shop présente un intérêt scientifique et pratique. Celui à deux machines est connu pour son abondance dans la littérature. L'ajout d'un convoyeur dans ce type d'atelier rend son étude plus réaliste d'un point de vue industriel. Notre objectif était de proposer des algorithmes efficaces pour le résoudre. Or, ce problème d'atelier est classé NP-difficile [5]. Des résolutions sont proposées. Elles se résument en :

1. De la littérature, proposer et utiliser trois règles de priorité statiques et dans un environnement stochastique une règle de priorité dynamique appelée OPBM
2. Appliquer une métaheuristique de type colonie de fourmis, évaluer les solutions engendrées et les comparer à celles fournies par les règles de priorité.

Chapitre 3 : Trois règles de priorité pour la résolution du $J2(1) / J \geq 1 / C_{max}$

Pour résoudre un problème d'ordonnancement NP-difficile, deux classes de méthodes peuvent être utilisées, les méthodes exactes et les méthodes approchées. Les méthodes exactes ne sont efficaces que si le nombre de tâches est réduit. De notre expérience, le nombre de tâches ne doit pas dépasser 8 par station. Si le nombre de tâches est assez grand, la détermination d'une solution optimale par une méthode exacte peut prendre un temps démesuré. L'énumération de toutes les solutions possibles est combinatoire, d'un ordre factoriel, exponentiel etc. Dans ce cas, et si on se contente d'une solution approchée, une heuristique est à prescrire.

Pour la résolution de ce problème et de la littérature, une tentative a été faite par Ruhlmann [6] où trois règles de priorité statiques sont proposées et comparées à la métaheuristique recherche taboue. Deux règles sont basées sur des couples de tâches et la troisième règle n'est que l'algorithme de Jackson. Cette auteure affirme que la règle de Jackson est la meilleure des trois règles. A ce problème, elle a modifié et appliqué une méthode de recherche avec taboue. Cette application n'a pas été très performante car elle n'a pas permis de traiter un grand nombre de tâches.

De ce fait, cette remarque nous a motivé à faire un autre travail comparatif dans le chapitre ultérieur. Nous proposons et comparons une règle de priorité dynamique attribuée aux tâches appelée OPBM, Ordonnancement par Priorités Bornées en Moyenne et une métaheuristique stochastique de type colonie de fourmis.

1. Descriptions des règles de priorité [6]

L'ensemble N_{LJ} contient les tâches ayant une gamme d'exécution partant de la station L et terminant sur la station J. Les deux premières règles sont définies à partir des ensembles N_{AB} et N_{BA}.

La première règle R1 forme des couples de tâches (k, j) ayant leurs temps d'exécution les plus courts où $k \in \{min\ t_{AK}\}$ et $j \in \{min\ t_{Bj}\}$.

t_{AK} est le temps d'exécution de la tâche k sur la machine A, $k \in N_{AB}$.

t_{Bj} le temps d'exécution de la tâche j sur la machine B, $j \in N_{BA}$.

La règle *R2* forme des couples *(k, j)* ayant leurs temps d'exécution les plus courts sur la machine *A* et les plus grands sur *B* où $k \in \{min\ t_{AK}\}$ et $j \in \{max\ t_{Bj}\}$.
Le choix de $k \in \{min\ t_{Bk}\}$ et $j \in \{max\ t_{Aj}\}$ est aussi possible.

Pour chaque règle *R1* ou *R2*, une fois les couples de tâches déterminés, l'algorithme de Johnson s'applique sur la station qui a le moins de tâches dans sa zone de stockage d'entrée. Sur l'autre station, les tâches sont rangées de façon à retrouver les couples déterminés précédemment.

Le choix de la station ayant le moins de tâches dans sa zone de stockage d'entrée pour l'application de l'algorithme de Johnson s'explique par le fait que l'on désire garder le couplage. En effet, sur la station ayant le plus de tâches dans sa zone de stockage d'entrée, le couplage implique que certaines tâches seront « célibataires ». Or, si l'algorithme de Johnson est appliqué sur cette station, un mélange de tâches couplées et non couplées sera créé. Le rangement, pour rétablir le couplage entre les tâches sur la station ayant le moins de tâches dans sa zone de stockage d'entrée, impliquera la création de « vide » pour garder le couplage ; ce qui n'est pas souhaitable.

Par exemple, si $N_{AB} = \{k_1, k_2, k_3, k_4\}$ et $N_{BA} = \{j_1\ j_2\}$ et que le couplage « min-min », suivant *R1*, ait généré les couples (k_1, j_2) et (k_4, j_1). Deux tâches sont alors « célibataires » k_2 et k_3. L'application de l'algorithme de Johnson sur la station qui a le plus de tâches dans sa zone de stockage d'entrée, c'est-à-dire *A*, génère la séquence k_3, k_1, k_2, k_4.
Comme le montre la Figure 9, le rangement sur la station *B*, permettant de retrouver le couplage, entraîne la création de deux tâches vides v tels que v,j_2, v,j_1.

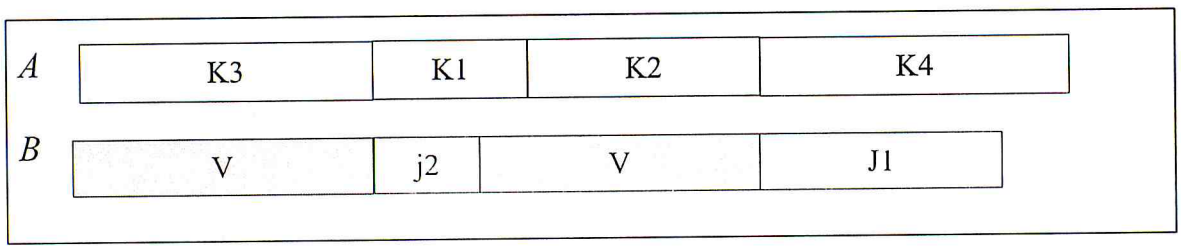


Figure 9: Création de tâches vides

II est clair que cet ajout de tâches vides va nuire à la qualité des solutions.

La troisième règle est celle de Jackson, elle consiste d'appliquer l'algorithme de Johnson [Joh54] indiqué au chapitre 1 sur la station contenant les plus grand nombre de tâches et ordonner les tâches de l'autre station suivant un ordre quelconque.
Les algorithmes des deux premières règles de priorité sont comme suit.

3.1. Algorithme min-min

---

1. Sélectionner parmi $\{N_{AB}, N_{BA}\}$ l'ensemble qui contient le moins de tâches. Supposons N_{AB} cet ensemble
2. Tant qu'il y a des tâches dans N_{AB}, faire
   1.4. Créer un couple (KN_{AB}, jN_{BA}) tel que :
      1.4.1. KN_{AB} <- tâche qui a le temps d'exécution t_{AK} minimum de N_{AB}
      1.4.2. jN_{BA} <- tâche qui a le temps d'exécution t_{BJ} minimum de N_{BA}
   1.5. Garder en mémoire ce couple
   1.6. Placer les tâches KN_{AB} et jN_{BA} dans les zones de stockages d'entrées de leurs stations respectives
3. Fin Tant Que
4. Appliquer l'algorithme de Johnson sur la zone de stockage d'entrée de A
5. Réordonner les taches contenues dans le stock de B de façon à ce que les tâches se trouvent à la même position dans le stock que celle de la tâche avec laquelle elles ont été précédemment couplées

---

3.2. Algorithme min-max

---

1. Sélectionner parmi $\{N_{AB}, N_{BA}\}$ l'ensemble qui contient le moins de tâches. Supposons N_{AB} cet ensemble
2. Tant qu'il y a des tâches dans N_{AB} faire
   1.7. Créer un couple (KN_{AB}, jN_{BA}) tel que :
      1.7.1. Si A est la station où l'on souhaite sélectionner
         Les tâches qui ont les temps d'exécution les plus long alors
         1.7.1.1. KN_{AB} <- tâche qui a le temps d'exécution t_{AK} maximum de N_{AB}
         1.7.1.2. jN_{BA} <- tâche qui a le temps d'exécution t_{Bj} minimum de N_{BA}
      1.7.2. Sinon
         1.7.2.1. KN_{AB} <- tâche qui a le temps d'exécution maximum de N_{AB}

1.7.2.2. jN_{BA} <- tâche qui a le temps d'exécution minimum de N_{BA}

1.7.3. fin Si

1.8. Garder en mémoire ce couple

1.9. Placer les tâches KN_{AB} et jN_{BA} dans les zones de stockages d'entrées de leurs stations respectives

3. Fin Tant Que

4. Appliquer l'algorithme de Johnson sur la zone de stockage d'entrée de *A*

5. Réordonner les taches contenues dans le stock de B de façon à ce que les tâches se trouvent à la même position dans le stock que celle de la tâche avec laquelle elles ont été précédemment couplées.

Dans ces deux algorithmes, c'est l'ensemble N_{AB} qui est supposé contenir le moins de tâches. Or, si ce n'est pas le cas, alors il suffit de permuter les ensembles et les stations dans les deux algorithmes.

La troisième règle, consiste à appliquer l'algorithme de Jackson. Ce choix de cette règle est motivé par le fait que cet algorithme est performant lorsque les opérations de transport sont considéré instantanée, il est donc intéressant de l'étudier quand cela n'est plus le cas.

Notons que ces trois règles de priorité ont une complexité temporelle en $O(n \log n)$.

3.3. Application des trois règles

Illustrons maintenant par un exemple ces trois règles. Par la suite, seuls les ordonnancements sur les deux stations au départ du fonctionnement de l'atelier seront présentés. Soient les tâches décrites par le Tableau 8 suivant.

Tableau 8: données de l'exemple

Identification de la tâche	Temps d'exécution sur A	Temps d'exécution sur B	Station de départ
K_1	4	5	A
K_2	3	4	A
K_3	9	2	A
K_4	7	4	A
K_5	5	3	A
j_1	1	8	B
j_2	7	6	B
j_3	4	7	B
j_4	3	2	B

Les ensembles N_{AB} et N_{BA} sont les suivants : $N_{AB} = \{k_1, k_2, k_3, k_4, k_5\}$ et $N_{BA} = \{ j_1, j_2, j_3, j_4\}$. La station qui a le moins de tâches est la station *B*. Sur celle-ci l'algorithme de Johnson sera appliqué pour les deux premières règles. Cet algorithme fournit la séquence suivante : j_1, j_3, j_2, j_4.

Commençons par la règle *R1*. La tâche qui a le temps d'exécution le plus court sur *A* est k_2 et celle sur *B est* j_4, ce qui donne le couple (k_2, j_4). De la même façon, les couples (k_1, j_2), (k_5, j_3) et (k_4, j_1) sont obtenus. La tâche k_3 est célibataire. Suite à l'application de l'algorithme de Johnson, les tâches de la station *A* sont rangées pour retrouver les couples déjà crées, ce qui donne la séquence : k_4, k_5, k_1, k_2, k_3. La Figure 10 décrit l'ordonnancement obtenu par la règle *R1*

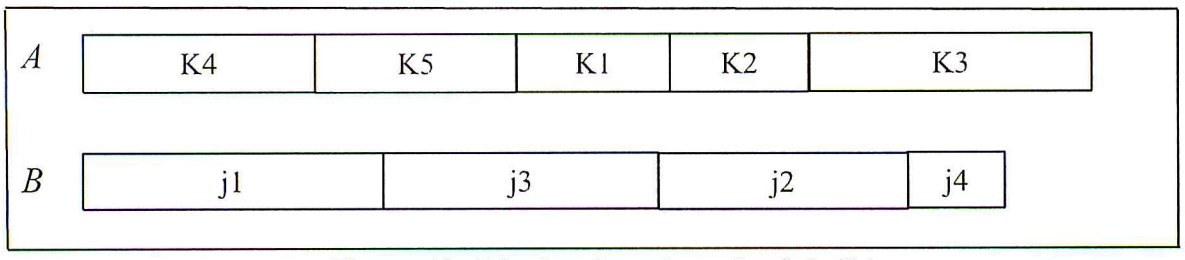


Figure 10: Résultat fourni par la règle R1

Pour la règle *R2*, commençons par choisir la station *A* pour sélectionner les tâches ayant les temps d'exécution les plus courts et la station *B* pour le choix des tâches ayant les temps d'exécution les plus longs. La tâche qui a le temps d'exécution le plus court sur *A* est k_2. La tâche de temps d'exécution le plus long sur *B* est j_1, ceci forme le couple (k_2, j_1). De la même façon, les couples $(k_1\ j_3)$, (k_5, j_2) et (k_4, j_4) sont formés. k_3 est célibataire. Après application de l'algorithme de Johnson sur la station *B*, le rangement des tâches sur la station *A* fournit l'ordonnancement : $k_1, k_2\ k_5, k_4, k_3$.

La règle *R2* construit l'ordonnancement décrit par la Figure 11 suivante.

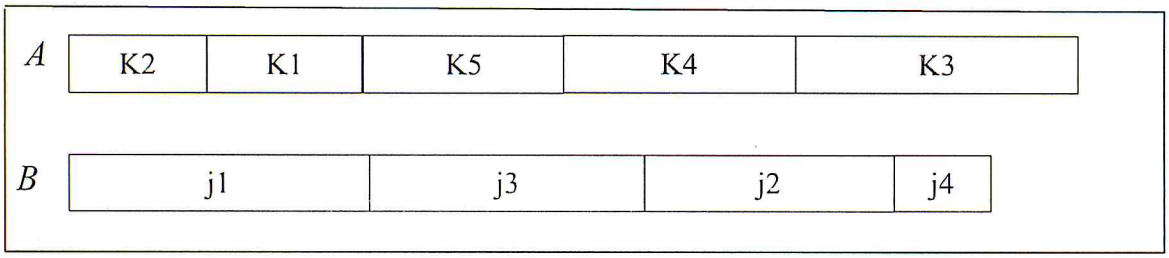


Figure 11 : Résultat fourni par la règle R2 avec les temps d'exécution les plus courts sur la station A

Appliquons une nouvelle fois, la règle *R2*, mais cette fois-ci avec *B* la station sur laquelle les tâches ayant le temps d'exécution les plus courts sont sélectionnées et *A* les temps d'exécution les plus longs. Les couples suivants (j_4, k_3), (j_2, k_4), (j_3, k_5), (j_1, k_1) sont formés. La tâche k_2 est célibataire. Le rangement sur *A* après application de l'algorithme de Johnson donne l'ordonnancement suivant: $k_1\ k_5, k_4, k_3, k_2$.

La Figure 12 montre l'ordonnancement sur les deux stations.

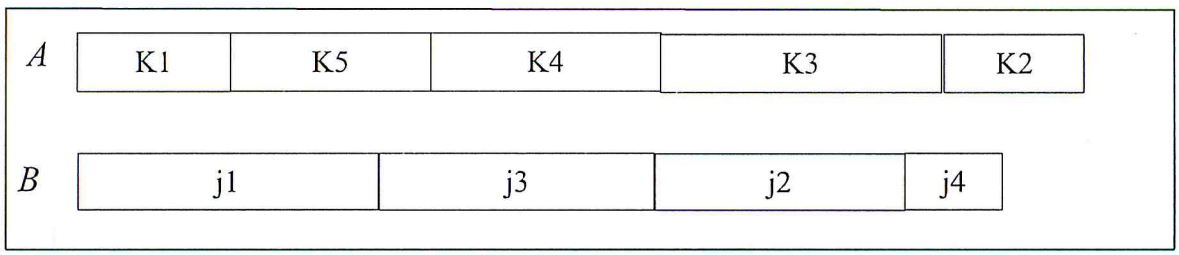


Figure 12: Résultat fourni par la règle R2 avec les temps d'exécution les plus courts sur la station *B*

La troisième règle est celle de Jackson. L'application de cette règle, comme le montre la Figure 13, donne l'ordonnancement k_2, k_1, k_4, k_5, k_3 sur la station A et sur B les tâches sont ordonnancées de façon quelconque, par exemple dans l'ordre naturel : j_1, j_2. j_3. j_4.

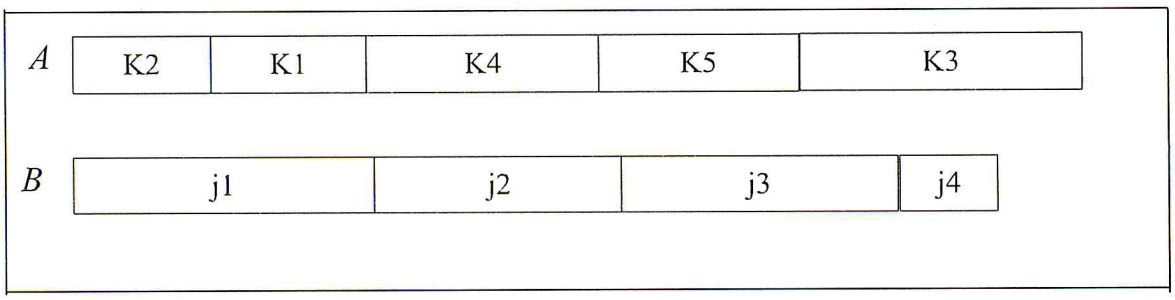


Figure 13: Résultat fourni par la règle de Jackson

2. Description et Implémentation d'une Recherche Taboue (RT)

Les recherches taboues ont été inventées par Glover en 1985. Elles sont de conception plus récente que le recuit simulé, n'ont aucun caractère stochastique et paraissent meilleures à temps d'exécution égal. Elles sont caractérisées par trois points :

- A chaque itération, on examine complètement le voisinage *V(s)* de la solution actuelle *s*, et on va sur la meilleure solution *s'*, même si le cout remonte.
- On s'interdit de revenir sur une solution visitée dans un passé proche grâce à une liste taboue *T* stockant de manière compacte la trajectoire parcourue. On cherche donc *s'* dans *V(s)-T*.
- On conserve la meilleure solution trouvée, contrairement au recuit, c'est rarement la dernière.

On stoppe après un nombre maximal d'itérations sans améliorer la meilleure solution ou quand *V(s)-T= {}*. Il ne se produit que sur de très petits problèmes pour lequel le voisinage tout entier peut se retrouver enferme dans *T*.

Une méthode taboue échappe aux minima locaux, même si *s* est un minimum local, l'heuristique va s'échapper de la région *V(s)* en empruntant un « *col* ».

En début de calcul, la méthode trouve une suite de solutions améliorées, comme une recherche locale. On voit ensuite le cout osciller puis redescendre vers un meilleur minimum local. Les améliorations deviennent de plus en plus rares au cours des itérations.

2.1. La liste taboue

Le point délicat est la capacité *NT* de la liste taboue *T*. Glover montre que *NT* de 7 à *20* suffit pour empêcher les cyclages, quelle que soit la taille du problème. *T* fonctionne comme une « mémoire à court terme ».

A chaque itération, la *NT* ième transformation de *T*, la plus ancienne, est écrasée par la dernière effectuée. En pratique, *T* se gère simplement avec une structure de données de type « file ». En théorie, il faudrait stocker les *NT* dernières solutions sur lesquelles l'algorithme est passé. En ordonnancement, il faudrait conserver les *NT* dernières permutations des *n* taches. Si la solution actuelle est *s*, la prochaine solution sur laquelle on va se déplacer doit être dans *V(s)-T*. Il faut vérifier que la permutation générée n'est pas déjà dans *T*.

Les voisinages considérés étant souvent grands, il est évident que le test répétitif de présence dans *T* est très couteux, sans parler de la mémoire nécessaire pour coder le détail de *NT* solutions. Le stockage explicite des solutions n'est donc jamais pratique.

On recommande les critères moins couteux pour éviter le cyclage. Ces critères sont aussi plus restrictifs : ils peuvent interdire certains mouvements qui ne conduiraient pas à un cyclage. Les plus utilisés de ces critères consiste à stocker dans *T* les transformations ayant permis de changer de solution, et d'interdire de faire les transformations inverses pendant *NT* itérations, on stockerait les deux taches permutées à chaque itération, et on s'interdirait de les remettre dans une permutation pendant *NT* itérations. Un critère plus simple est d'interdire d'utiliser les *NT* dernières valeurs de la fonction objective. Il suffit de stocker un nombre par itération.

2.2. Algorithme Général de la Recherche Taboue

```
- Construire une solution initiale quelconque s
  z* = f(s)=z {meilleur cout obtenu}
  s* =s  {meilleur solution connue}
- Initialiser MaxIter {nombre maximum d'itérations}
  T = {} {la liste taboue est vide}
- NIter :=0
- Répéter
  NIter = NIter + 1
  z'' = + ∞
  Pour toute solution s' de V(s)
        Si s' n'est pas dans T alors
              Si f(s') < z'' alors  { mise à jour du
     voisin « le moins pire »}
                        S''=s' , z'' = f(s')
                  Stocker la transformation dans u
               FS
     FS
  FP
  Si  z''<+ ∞ alors {si un voisin non tabou a été
      trouvé}
     s = s'' ,  z = z''
  Enlever la transformation en tête de T {la plus
      ancienne}
  Ajouter  u en fin de T
  Si  z < z'' alors {mise à jour de la meilleure
      solution}
    s*= s , z*=z
  FS
  FS
  Jusqu'à (NbIter = MaxIter) ou (z'' = + ∞)
```

Chapitre 4 : Contribution à la résolution du J2(1) / J ≥ 1 / C_{max} par l'OPBM et une métaheuristique de type colonie de fourmis

La règle de priorité dynamique appelée OPBM, Ordonnancement par Priorités Bornées en Moyenne, est proposée, étudiée et utilisée. Elle a été définie par Haro et Proust [20] dans un système d'exploitation d'ordinateur. Sa caractéristique est qu'elle est asymptotiquement équitable entre les tâches. Les résultats fournis par son application seront comparés à ceux donnés par la métaheuristique stochastique de type colonie de fourmis. Dans la suite, elles seront introduites et définies en détail.

1. O.P.B.M : ordonnancement par priorités bornées en moyenne

Soit « *n* tâches » à exécuter sur une seule machine.

A chaque tâche « *i* » on associe une priorité instantanée : $P_i(t) = \frac{M_i.t_i^{(a)}(t)+m_i.t_i^{(e)}(t)}{t_i^{(a)}(t)+t_i^{(e)}(t)}$

Pour $t = 1, 2, \ldots$ et $P_i(0) \in]\, m_i, M_i\, [$ m_i et M_i sont des valeurs réelles données. $t^{(e)}(t)$ et $t^{(a)}(t)$ sont respectivement les temps d'exécutions qu'a reçu la tâche i et son temps d'attente pour s'exécuter durant l'intervalle [0, t]. En chaque date de décision, on exécute la tâche avec la plus grande priorité. En cas de conflit ou d'égalité entre les priorités, la règle Tourniquet ou Round Robin est utilisée pour départager les tâches. En générale une priorité est interprétée comme une urgence dans l'exécution d'une tâche. Haro et Proust [20], concepteurs, ont défini cette priorité particulière et l'ont implémenté dans le noyau multitâche d'un système d'exploitation d'ordinateur. Elle est définie pour améliorer les performances du système informatique. Dans notre étude, elle est utilisée pour ordonnancer les tâches dynamiquement ou en tout instantes.

Une tâche T_i, $\forall i\,, 1 \leq i \leq n$ quelconque commence à s'exécuter à l'instant zéro avec une priorité initiale $P_i(0)$. A une date $t > 0$ quelconque, elle s'exécute avec une priorité $P_i(t)$ donnée par : $\forall i, 1 \leq i \leq n, \forall\, t > 0 :\ P_i(t) = \frac{M_i.t_i^{(a)}(t)+m_i.t_i^{(e)}(t)}{t_i^{(a)}(t)+t_i^{(e)}(t)}$

C'est la moyenne des bornes M_i et m_i pondérée par le temps d'exécution et d'attente de tâche, sur un intervalle du temps] 0, t[

Cette politique favorise les tâches dont les temps d'attentes sont élevés, ainsi celles faisant beaucoup d'Entrées et de Sorties, au détriment de celles qui utilisent excessivement la machine.

D'autre part on a : $\forall i, i = 1 \ldots n, \forall t > 0 : t_i^{(a)}(t) + t_i^{(e)}(t) = t \Leftrightarrow t_i^{(a)}(t) = t - t_i^{(e)}(t)$

En posant $\forall i, i = 1 \ldots n : \Delta P_i = M_i - m_i$

La formule de la priorité devient : $P_i(t) = \frac{M_i\left(t - t_i^{(e)}(t)\right) + m_i . t_i^{(e)}(t)}{\left(t - t_i^{(e)}(t)\right) + t_i^{(e)}(t)} = \frac{M_i . t - t_i^{(e)}(t)(M_i - m_i)}{t}$

1.1. Définition

On appelle ordonnancement par priorités bornées en moyenne noté O.P.B.M, un ordonnancement où la priorité d'une tâche est déterminée par la nouvelle formule condensée $P_i(t) = M_i - \Delta P_i \frac{t_i^{(e)}(t)}{t}$.

1.2. Equité de l'OPBM

Une propriété de cette priorité est qu'elle est asymptotiquement équitable. Quand les tâches ont la même priorité, un ordonnancement est équitable si l'attente pour exécution est identique pour toutes les tâches. Elles reçoivent en moyenne le même traitement. Si les tâches ont des priorités différentes, l'équité s'exprime par la qualité de service qui est une fonction croissante de la priorité. Cette qualité peut être mesurée par exemple par l'inverse du temps d'attente moyen ou par une autre expression.

Les processus de E sont repartis en « r » classes disjointes de priorités :

$\forall k > 0, 1 \leq k \leq \boldsymbol{r}$ et $\forall h, 1 \leq h \leq \boldsymbol{r} : [m_k, M_k] \cap [m_h, M_h] = \emptyset$

Considérons l'ensemble des n processus d'une même classe E = $\{ \tau_i \ / \ i_1 \leq i \leq i_n \}$

Avec $1 \leq i_1, i_n \leq L$ de E/R, de bornes m, M

Pour simplification de notation posons $E = \{ \tau_i \ / \ 1 \leq i \leq n \}$

Dans la suite $t_i^e(t)$ se notera $t_i(t)$.

1.3. Le Taux D'occupation

Le taux d'occupation de la machine par les tâches est le réel ρ défini par :

$\forall t > 0 : \quad \rho(t) = \frac{\sum_{i=1}^{n} t_i(t)}{t}$

A un instant $t > 0$ quelconque, T_i a été exécuté pendant $t_i(t)$.

Le rapport de $t_i(t)$ sur $t > 0$ représente le taux d'occupation de la machine par T_i .

1.4. Hypothèses de travail

Nous supposons que:

1. les tâches T_i de E sont présentes dans l'atelier
2. Elles sont toujours prêtes pour exécution et de temps d'exécution supposé illimité.
3. On suppose que la priorité varie dans un intervalle non vide et ne se réduit pas à un point c.à.d. $m \neq M$.

Des définitions sont énoncées.

- le taux d'occupation est le réel ρ : $\forall$ t > 0 : $\rho(t) = \frac{\sum_{i=1}^{n} t_i(t)}{t}$
- l'ordonnancement est équitable pour l'ensemble E = $\{ \tau_i \ / \ 1 \leq i \leq n \}$ lorsque :

$\forall i, i = 1..n, \forall t > 0 : \rho_i(t) = \frac{\rho(t)}{n}$

- l'ordonnancement est asymptotiquement équitable pour E = $\{ \tau_i \ / \ 1 \leq i \leq n \}$

Lorsque : $\forall i, i = 1..n$:

$$\lim_{t \to \infty} \rho_i(t) = \frac{\rho(t)}{n}$$

L'implémentation selon Haro et Proust a été réalisée par nos soins. Des détails sont donnés dans la partie Annexe-A de ce mémoire. Dans un problème d'ordonnancement préemptif, l'exécution des tâches se fait par morceaux ou en plusieurs étapes. En tout instant, les tâches les plus prioritaires seront élues pour exécution. La priorité d'une tâche peut être fixe ou dynamique.

2. O.P.B.M appliqué au 2-Job Shop avec convoyeur

Si N_G et N_p représentent respectivement la station contenant respectivement le plus grand nombre de tâches et le plus petit nombre de tâches, des modifications sont opérées sur les formules de calculs des priorités. A l'instant initial zéro :

$$P_i(0) = \begin{cases} \tau_i + max\{t_{Ai}, t_{Bi}\} \ si \ i \in N_G \\ \tau_i + min\{t_{Ai}, t_{Bi}\} \ si \ i \in N_P \end{cases}$$

Avec : $\tau_i, 1 \leq i \leq n$ est le temps de transport de la tâche i.

A une date t, $t > 0$, la priorité $P_i(t)$ d'une tâche i est donnée par :

$$\forall i, i = 1 \ldots n, \forall t > 0 : \ P_i(t) = \frac{\max\{P_i(0), i=1..n\}.t_i^{(a)}(t) + \min\{P_i(0), i=1..n\}.t_i^{(e)}(t)}{t_i^{(a)}(t) + t_i^{(e)}(t)}$$

$t_i^{(a)}(t)$ et $t_i^{(e)}(t)$ représentent respectivement le temps d'attente et d'exécution de la tâche i à une date t.

La tâche qui a la plus grande priorité s'exécutera sur la dite-machine. Si plusieurs tâches ont la même plus grande priorité, la règle Round Robin sera utilisée. Chaque tâche est autorisée à s'exécuter pendant une durée fixée Q appelée Quantum. Si une tâche se termine avant la fin du Quantum, la machine est immédiatement réallouée à la tâche suivante.

Cette extension de la formule n'a été proposée qu'à la suite de déroulement de plus d'une centaine d'exemples où il fallait intégrer le temps de transport de tâches. Cette addition de temps d'exécution d'une tâche et de son temps de transport d'une machine à une autre a fourni un comportement d'exécution ou le makespan obtenu est plus court.

L'application de l'OPBM à un problème d'ordonnancement non préemptif n'est autre que la règle SPT, règle très usuelle. Dans ce cas d'ordonnancement et pour éviter l'utilisation de cette dernière règle, nous proposons une extension de la définition de l'OPBM.

3. O.P.B.M-étendu pour les problèmes d'ordonnancements non préemptifs

Dans un problème d'ordonnancement non préemptif, l'exécution des tâches ne peut être interrompue qu'en des instants de fin d'exécution. Dans ce cas, l'OPBM peut être une politique non réalisable. Une extension de sa définition s'impose tout en conservant son comportement évolutif de la priorité. Une séquence d'ordonnancement obtenue dépend de l'évolution des priorités des tâches pendant leur exécution. L'évolution de la priorité d'une tâche dépend de tout son historique d'exécution.

Chaque tâche a un historique d'exécution égal à son temps d'exécution. Refaire l'exécution d'une séquence de tâches est dite « exécution virtuelle ». Si on refait virtuellement l'exécution des tâches, les priorités seront fixes ou dynamiques.

Il y a lieu de rappeler qu'avant chaque exécution réelle des tâches, nous supposons qu'une machine virtuelle est disponible pour déterminer, pour chaque séquence de tâches, la longueur de son ordonnancement. La meilleure solution obtenue sera conservée.

OPBM est étendu de la façon suivante :

Exécuter les tâches sur la machine virtuelle selon la priorité suivante :

$$\forall\, i, i = 1 \ldots n, \forall\, t > 0 \;\; \mathrm{P_i(t)} = M_i - \Delta P_i \frac{t_i^{(e)}(t)}{\mathbf{t}}$$

Il existera un instant t_0 où les tâches auront terminé leur exécution. A cet instant, on détermine la valeur du C_{max}. L'exécution virtuelle des tâches est refaite sur la machine virtuelle mais avec un nouvel historique attribué à chaque tâche.

Pour fixer les idées, soient :

- Cinq tâches de temps d'exécution $t_1 = 13$, $t_2 = 4$, $t_3 = 8$, $t_4 = 29$ et $t_5 = 17$.
- $U_1 = 0$, $U_2 = 0$, $U_3 = 0$, $U_4 = 0$, $U_5 = 0$, représentent l'historique d'exécution des tâches. Avant toutes exécutions, ils sont initialisés par la valeur zéro.

A un instant de fin d'exécution de l'ordonnancement t_0, $t_0 > 0$, les priorités sont supposées être égales à : $P_1(t) = 1$; $P_2(t) = 2$; $P_3(t) = 3$; $P_4(t) = 4$; $P_5(t) = 5$; et l'historique d'exécution de chaque tâches égal à la valeur de son temps d'exécution. $U_1 = 13$, $U_2 = 4$, $U_3 = 8$, $U_4 = 29$, $U_5 = 17$.

Formons les couples de tâches selon la procédure suivante. Le premier couple de tâches constitué sera formé par la tâche qui a la plus grande priorité et la tâche ayant la plus petite priorité, le second couple de tâches sera formé par la tâche qui a la seconde plus grande priorité et la tâche ayant l'avant-dernière priorité …ainsi de suite. Le résultat obtenu est : (tâche5, tâche1), (tâche4, tâche2). La tâche3 sera dite célibataire, au sens où elle n'est couplée à aucune tâche, et son temps d'exécution ne changera pas. Pour chaque couple, on fait l'échange de tout le passé de temps d'exécution entre les tâches. D'où la nouvelle attribution du passé de temps d'exécution est $U_1 = 17$, $U_2 = 29$, $U_3 = 8$, $U_4 = 4$, $U_5 = 13$.

Une exécution est refaite sur la nouvelle gamme des tâches …etc. La meilleure séquence de tâches obtenue s'utilise pour une exécution sur les machines de l'atelier.

Cet échange de temps d'exécution permet au processus de changer l'état d'activité pendant un intervalle de temps relevant de temps nouvellement attribué. Il peut aussi susciter une nouvelle allure des priorités qui n'assure pas la stabilité ou la convergence des valeurs des priorités à la valeur d'équité $M - \left(\frac{\Delta P}{n}\right)$.

Quand les temps d'exécution des tâches forment une suite arithmétique, un cas particulier, l'équité de l'OPBM-étendu est établie analytiquement par nos soins dans l'Annexe-B.

4. Résolution du J2(1) / J ≥ 1 / C_{max} par une métaheuristique de type colonie de fourmis

L'algorithme de colonie de fourmis [21] s'inspire d'une expérience menée en 1989 par Goss *et al.* [22]. Des fourmis avaient été mises en contact avec une source de nourriture reliée à la fourmilière par un « bâton de pèlerin » ayant deux branches de longueurs différentes, tel qu'illustré à la Figure 14. Elles parvenaient après quelque temps à emprunter toujours le chemin le plus court entre la fourmilière et la nourriture. Il a été observé que la probabilité que toutes les fourmis empruntent le chemin le plus court augmentait avec la différence de longueur entre les deux branches du bâton.

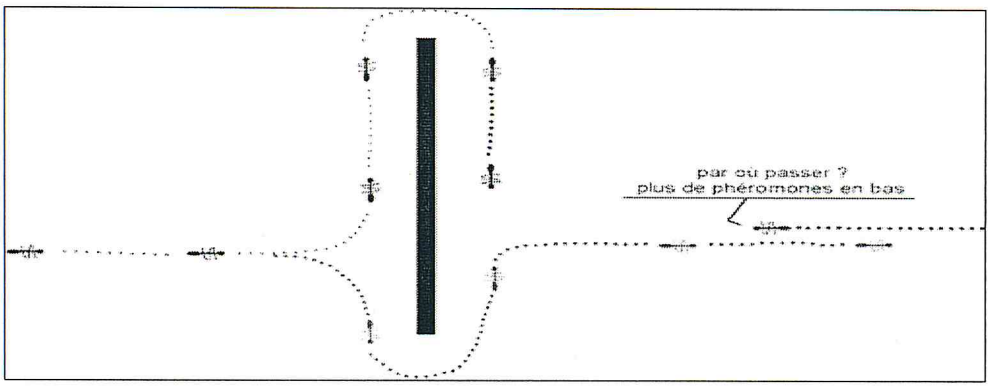


Figure 14: Disposition de l'expérience de Goss *et al.* [24]

Goss *et al.* [22] ont établi que les fourmis partageaient des informations sur leurs expériences d'exploration à l'aide d'une substance chimique appelée *phéromone.* Elles déposent par terre une certaine quantité de cette substance lorsqu'elles se déplacent et en même temps, scrutent le sol pour « lire » les informations laissées par leurs prédécesseurs. Lorsqu'elles ont à faire un choix quant à la direction à prendre, les fourmis sélectionnent un chemin de façon probabiliste en fonction de la quantité de phéromone présente sur les différentes voies possibles. Le comportement observé par Goss s'explique par le fait que les fourmis qui empruntent le chemin le plus court parviennent à la source de nourriture et reviennent à la fourmilière plus vite que celles qui ont pris le plus long chemin. En laissant toujours une trace de phéromone, elles rendent le chemin emprunté plus attirant pour les fourmis suivantes. Comme la phéromone s'évapore avec le temps et le vent, le chemin le plus long et le plus venteux est de moins en moins emprunté et sa trace disparaît presque complètement. Le temps nécessaire à la stabilisation du système est imputable au fait qu'au

départ aucune phéromone n'est présente sur aucune route. Les fourmis ont alors autant de chances de choisir le plus court que le plus long chemin.

Dans ce qui suit, pour résoudre le problème de Job Shop à deux machines et un convoyeur, un algorithme s'inspirant de ce comportement des fourmis est élaboré, justifié et est montré fini.

4.1. Algorithme OCF-J2(1)C_{max}

Nous modélisons notre problème comme celui de la recherche d'un meilleur chemin dans un réseau particulier ($G < T, L$) où T est l'ensemble des nœuds, des tâches à ordonnancer et L, l'ensemble des arrêtes entre les nœuds.

Le couple *(Ti, Tj)* ($i \neq j$) désigne l'ordonnancement de la tâche *Ti* puis de la tâche *Tj*.

On suppose qu'on dispose de « m » fourmis empruntant un chemin dans un réseau.

Chaque fourmi est positionnée aléatoirement sur un nœud de départ dans le réseau.

La trace de phéromone sur une arrête (i, j) notée τ_{ij} et la visibilité d'une fourmi pour l'arrête *(i, j)* notée η_{ij} sont deux facteurs influents sur le choix du prochain nœud à visiter ou la prochaine tâche à ordonnancer. Au risque de se répéter, nœud et tâche sont synonymes.

Une matrice nxn de trace de phéromone peut-être ainsi définie dont ses éléments sont les τ_{ij}.

De même, une seconde matrice peut-être définie, elle est appelée la matrice de l'information heuristique. Elle est constituée d'éléments visibilités η_{ij}

Plus cette trace sur l'arrête est grande, plus la probabilité de l'emprunter à nouveau est élevée.

Chaque fourmi k possède une forme de mémoire enregistrée dans une liste taboue lui rappelant la séquence de tâches déjà ordonnancées.

Au début de l'application de l'algorithme, l'intensité de la trace de phéromone τ_{ij}, se voit accorder une valeur faible et positive notée τ_0.

La visibilité d'une fourmi pour une arrête indique les tâches les plus aptes à ordonnancer.

Une probabilité de transition $p_{ij}^{k}(t)$ effectue ainsi un compromis entre l'intensité de la trace de phéromone et la visibilité pour déterminer la prochaine destination ou la prochaine tâche à exécuter.

M et n représentent respectivement le nombre de fourmis utilisées et le nombre de tâches à exécuter.

Un cycle est réalisé lorsqu'une fourmi emprunte tous les n nœuds du réseau.

Une itération de l'algorithme n'est que la réalisation de m cycles par les m fourmis.

Une mise à jour de la matrice de phéromone est effectuée. Elle est le remplacement d'un élément τ_{ij} par un autre élément obtenu par des formules établies ci-dessous.

A la fin de chaque cycle, la fourmi ayant la meilleure solution, un ordonnancement de longueur minimum, mettra à jour la matrice de trace de phéromone.

Une évaporation de la trace de phéromone est également réalisée sur toutes les arrêtes ne faisant pas partie de ce meilleur chemin qui représente la meilleure séquence obtenue dans cette itération.

Les autres fourmis tenteront d'emprunter les arrêtes en inspirant la trace de phéromone la plus forte.

L'algorithme se termine lorsqu'un nombre donné de cycles fixé par l'utilisateur est réalisé. D'habitude, ce nombre est fixé à 10. Le nombre de fourmis est en général égal au nombre de tâches.

Le schéma général de l'algorithme se résume aux cinq instructions élémentaires.

Début
NC (Nombre de Cycle) = 0 ;
Initialiser la matrice de trace de phéromone pour chaque paire de tâche *ij*;
POUR *NC de 1 à* NC_{max} FAIRE
POUR *I de 1 à n* FAIRE
POUR *k de 1 à m* FAIRE
Sélectionner la tâche à être ajoutée à la séquence selon p_{ij}^{k} *(t)* ;
Effectuer la mise à jour locale de la trace de phéromone pour *(i,j)* ;
POUR chaque fourmi i FAIRE évaluer la solution *k*
Effectuer la mise à jour globale de la trace par la meilleure solution du cycle *f*
Fin.

Figure 15: Algorithme OCF-J2(1)C_{max}

4.2. Schéma général de l'algorithme OCF-J2(1)C_{max}

Dans la suite τ_0, ρ, α, q_0, β, *P(nxn)* et *H(nxn)* désigneront respectivement cinq valeurs et deux matrices à initialiser par l'utilisateur. *n* désignera le nombre de tâches à exécuter par les deux machines.

(1) Initialisation

Il y a lieu d'indiquer :

- La valeur initiale de phéromones τ_0 ;
- L'information heuristique liée aux différentes tâches ;
- Le nombre de fourmis utilisées ;
- ρ et α sont deux paramètres à indiquer par l'utilisateur dans la règle de mise à jour de la phéromone, paramètres indiqués dans une formule ci-dessous ;
- q_0 est une constante conditionnelle. Elle peut être confirmative ou infirmative. β est un second paramètre utilisé dans la formule de la probabilité de transition ;
- le critère d'arrêt de l'algorithme donné par le nombre de cycles ou de générations à engendrer.

Deux autres paramètres sont encore à initialiser:

i- L'information phéromone.

Elle est représentée par une matrice *P(nxn)*. L'élément τ_{ij} de la $i^{éme}$ ligne et $j^{éme}$ colonne de cette matrice est initialisé à τ_0, une valeur réelle faible fixée par l'utilisateur et modifiée au cours du traitement par les fourmis.

Les modifications sont reportées sur les éléments de cette matrice par une mise à jour, étape 4. τ_{ij} représente la quantité de phéromones posé sur l'arrête (i, j) où i indique une tâche déjà ordonnancée et j une tâche à ordonnancer.

ii- L'information heuristique

La prochaine tâche à ordonnancer dépend aussi de l'information heuristique. Cette information est de deux types, statique et dynamique. La statique n'est que l'inverse de la somme des temps d'exécution translaté. L'information dynamique est une différence entre l'unité et une somme pondérée. Leurs formules (4) et (5) sont ci-dessous indiquées.

□ *Information heuristique statique* $S\eta_{ij}$ *:* Elle est fournie par une matrice *H(n×n)*. L'information heuristique est dite statique car elle reste inchangée au cours du traitement et se calcule à l'initialisation de l'algorithme. Pour augmenter la visibilité, il y a lieu de considérer dans la formule de son calcul l'inverse d'une expression positive non nulle et tendant vers zéro. Une formule proposée pour calculer η_{ij} est, $S\eta_{ij} = \frac{1}{1+\sum_r P_{jr}}$ ... (4).

p_{jr} représente le temps d'exécution de la tâche j sur la machine r, $r = \{1, 2\}$.

□ *Information heuristique dynamique* $D\eta_{ij}$. Les valeurs de l'information sont recalculées à chaque itération de l'algorithme. La valeur heuristique d'une tâche pour une

fourmi *k* est formée par les temps d'exécutions des tâches qui appartiennent à l'ensemble des sommets voisins à celui où la fourmi est positionnée. Elle est donnée par la formule suivante :

$$D\eta_{ij} = 1 - \frac{\sum P_{jr}}{\sum_{l \in N_k(S)} \sum_r P_{jr}} \dots (5)$$

Où *i* désigne une tâche en exécution. Elle est dite tâche courante.

r étant le nombre machine et *l* une tâche du voisinage $N_k(S)$, ensemble des sommets voisins du sommet où la fourmi *k* est positionnée.

(2) Positionnement des fourmis

Une fourmi peut prendre son départ à partir de n'importe quel sommet du réseau. Elle est dite « positionnée » sur la tâche associée à ce sommet. Si une fourmi *k* prend son départ d'un sommet S_{k0}, ce sommet est inséré dans une liste taboue T_k. Cette liste taboue sera formée de tous les sommets visités antérieurement par la fourmi.

(3) Construction d'un ordonnancement réalisable

Une fourmi en tournée passe dans un ordre indiqué par tous les *n* sommets du réseau. Cette permutation des *n* sommets constitue un ordonnancement qui est aussi appelé « solution » du problème. L'ordre de passage de la fourmi est constitué des étapes suivantes :

i- le choix de la prochaine tâche à ordonnancer se fait par l'application de la règle de transition d'états. Cette règle indique comment une fourmi *k* positionnée sur la tâche *i* choisit la tâche *j* à l'instant t avec une probabilité $P_{ij}^k(t)$.

Si $q \leq q_0$

$$P_{ij}^k(t) = \begin{cases} 1 \quad si\ j = \arg\operatorname{Max}\left([\eta_{ij}(t)]^\beta [\tau_{ij}(t)]^\alpha\right) \\ 0 \quad sinon \end{cases}$$

Sinon

$$P_{ij}^k(t) = \frac{[\eta_{ij}(t)]^\beta [\tau_{ij}(t)]^\alpha}{\sum_{i \in N_i^k} [\eta_{ij}(t)]^\beta [\tau_{ij}(t)]^\alpha}$$

ii- La tâche j ainsi choisie par une des deux formules dans l'étape précédente est insérée dans la liste taboue T_k de la fourmi k. Cette liste est un vecteur de taille n qui interdit à une fourmi d'ordonnancer une tâche déjà ordonnancée.

iii- Une fourmi qui transite de la tâche i vers la tâche j met à jour la valeur de l'information phéromone τ_{ij} sur l'arrête (i, j) par des formules indiquées ci-dessous.

iv- Si l'ensemble des tâches à ordonnancer par la fourmi k, où le voisinage du sommet où la fourmi k est positionnée n'est pas vide, alors le processus de déplacement de la fourmi dans le réseau est réitéré à partir de l'étape (iii).

A la fin de la construction d'une séquence de sommets visités par chaque fourmi, les listes Taboues seront réinitialisées en un ensemble vide, ensemble ne contenant aucune tâche.

(4) Mise à jour de la meilleure solution

A chaque génération, quand toutes les k-fourmis ont fournies chacune une tournée sur l'ensemble des n sommets, un ordonnancement réalisable des n tâches, une mise à jour de phéromone est effectuée sur le chemin optimal. La valeur du makespan se détermine par la longueur du chemin minimum. La matrice « information phéromone » notée P sera mise à jour en effectuant un changement de valeurs de ses éléments correspondant aux indices des tâches formant le chemin optimal. Cette mise à jour est faite selon la formule suivante:

$$\tau_{ij}(t+1) = (1-\rho)\,\tau_{ij}(t) + \rho\,\Delta\tau_{ij}(t) \qquad \text{Avec : } \Delta\tau_{ij}(t) = 1 / C_{max}$$

(5) Critère d'arrêt

A chaque itération, l'algorithme recommence son exécution depuis son étape (2). On l'arrête quand le nombre total d'itérations atteint une valeur fixée.

4.3. Choix des paramètres [23]

Dans la littérature, les auteurs conseillent d'utiliser la trace de phéromone initiale $\tau_0 = 0{,}1$. Pour une longueur d'ordonnancement minimum et un temps d'exécution court, le nombre de fourmis doit être un paramètre important. Des expériences antérieures ce nombre de fourmis ne doit pas dépasser 10. Il peut être aussi fixé expérimentalement [24, 25]. Dans notre implémentation, il est considéré de valeur égale au nombre de tâches. Ce choix du nombre de fourmis ne nous a causé aucun désagrément au blocage de l'ordinateur sur lequel déroulent les programmes.

Pour de grandes valeurs de α combinées à des valeurs de β qui ne sont pas suffisamment grandes, l'algorithme entre rapidement dans un état de *chemin-unique,* où les fourmis empruntent toujours les mêmes arrêtes sans trouver de bonne solution. Si on ne donne pas une valeur suffisamment grande à α ou trop grande à β, alors l'algorithme ne peut produire que des solutions de qualité moyenne. Dorigo et al. [26] proposent que α doit être autour de 1, et β entre 2 et 5. Dans notre algorithme ils sont fixés respectivement à 1et 2. D'où les valeurs données à α et β ont un impact important sur la qualité des solutions obtenues. La valeur de q_0 est fixée à 0,8 et ρ doit être de valeur proche de 0,5.

Chapitre 5 : Etude comparative entre l'algorithme de colonie de fourmis, l'OPBM-étendu et la règle de Jackson

Nos expérimentations consistent en le déroulement de programmes informatiques élaborés et exécutés sur un PC équipé d'un processeur Intel(R) Pentium(R) 4 CPU 2.93GHz avec une mémoire de 2038 MB de RAM. Ce processeur fonctionne sous Windows XP. Cet environnement informatique est considéré comme accessible, non couteux, fiable et souple. Les règles de priorité et la métaheuristique colonie de fourmis utilisées dans notre étude sont implémentées en utilisant un langage de programmation évolué. Les programmes importent les données d'entrée d'un fichier contenant les paramètres associées aux tâches à exécuter par les machines dans l'atelier. Pour des raisons de comparaisons, les données des problèmes exécutés tels les longueurs d'intervalle de distribution uniforme et le temps de retour à vide du convoyeur sont identiques à celles utilisées par Ruhlmann [6]. Pour obtenir une distribution uniforme d'un vecteur d'éléments, une fonction random_shuffle(vecteur.begin(), vecteur.end()) est utilisée.

1. Mise en œuvre informatique de l'OPBM-étendu

L'OPBM-étendu, est bien définie auparavant dans le paragraphe 3 du chapitre 4. Des priorités sont associées dynamiquement aux tâches à exécuter dans l'atelier. En tout instant, on exécute la tâche qui a la plus grande priorité. Si plusieurs tâches ont la même plus grande priorité, nous exécuterons arbitrairement l'une de ces tâches selon la règle Round-Robin appelée dans la littérature « le tourniquet ». Chacune des tâches en conflit reçoit, à tour de rôle, un quantum Q de temps d'exécution qui ne peut s'interrompre qu'à la fin d'exécution de la tâche. Le quantum sera le temps d'exécution de la tâche en conflit. Très souvent et de nos expériences réalisées, ce type d'ordonnancement fournit de meilleurs résultats en longueur d'ordonnancement. On rappelle que cette règle OPBM originale a déjà été implémentée par un binôme d'ingénieurs au sein de notre département [27] et par nos soins (voir Annexe-A). A l'origine, cette règle a été implémentée dans un système d'exploitation multitâches monoprocesseur par Haro et Proust [20]. Ces derniers dans leur implémentation supposent que pendant une période du temps Ts la machine exécute toutes les tâches de l'atelier. Une différence primordiale existe avec l'implémentation faite par nos soins. Nous proposons que la machine exécute une seule tâche à chaque période de temps Ts.

2. Déroulement de l'algorithme au cas du convoyeur

Les identificateurs sont enregistrés dans des fichiers textes qui contiennent :

- $P_i(0)$: la priorité initiale de la tâche *i*, sont calculées comme suite :

$$P_i(0) = \begin{cases} \tau_i + \max\{t_{Ai}, t_{Bi}\} \text{ si } i \in \text{station contenant le plus de tâches} \\ \tau_i + \min\{t_{Ai}, t_{Bi}\} \text{ si } i \in \text{station contenant le moins de tâches} \end{cases}$$

où : τ_i , $1 \leq i \leq n$ est le temps de transport de la tâche i entre les machines.

- *m*, *M* sont les paramètres qui apparaissent dans la définition de la priorité. Ils sont choisis respectivement de valeur $\max\{P_i(0), 1 \leq i \leq n\}$ et $\min\{P_i(0), 1 \leq i \leq n\}$
- $\Delta P = M - m$, la différence des bornes de la priorité.
- *n* : nombre de tâches.
- *j* : numéro de la période d'ordonnancement.
- Ts : la durée d'une période d'ordonnancement.
- $P_i(j)$: la priorité de la tâche *i* durant la période *j*.
- $t_i^e(j)$: le temps d'exécution de la tâche *i* durant la période *j*.
- $s(k)$: ordonnancement provisoire (pas forcément optimal) des tâches obtenu à l'itération *k*.

Pour éviter une lourdeur et pour une bonne visibilité du texte, les étapes nécessaires pour réaliser l'implémentation de l'OPBM-étendu sont indiquées dans l'Annexe-C.

3. Instances numériques utilisées

Un exemple de données d'un problème est appelé « une instance ». Un ensemble d'instances aléatoires variées a été élaboré. 5x100 instances de problèmes-tests consistant en des temps d'exécutions et de transports des tâches ont été générées de manière aléatoire sur l'intervalle]0, 50[. Le nombre de tâches considéré dans ces instances est 10, 50, 100, 200 et 400. Ce nombre augmente en palier pour confirmer l'efficacité de notre implémentation. A un instant initial, début de fonctionnement de l'atelier, le convoyeur est supposé placé en face de la station possédant le plus grand nombre de tâches dans sa zone de stockage d'entrée. Le temps de retour à vide du convoyeur est fixé à 10 unités de temps.

4. Résultats et critères de validité des tests

Une étude statistique des résultats générés est effectuée pour déterminer la performance des trois méthodes de résolution utilisées, OPBM-étendu, règle de Jackson et l'algorithme colonie de fourmis. Trois critères sont pris en considération.

1. Pour chacune des trois méthodes utilisées, un temps d'exécution ou temps CPU est

déterminé. Il est représenté dans un repère orthonormé dont l'abscisse est le nombre de tâches et l'ordonnée est le temps d'exécution. Si le nombre de tâches est inférieur à 50, les trois méthodes utilisées ont un temps d'exécution identique. Aucune préférence ne peut être constatée. On peut utiliser arbitrairement n'importe quelle de ces trois règles. Si le nombre de tâche est situé entre 50 et 100, la résolution par l'algorithme OCF-J2(1)C_{max} nécessite un temps plus grand que les deux autres méthodes qui ont un temps d'exécution presque identiques. Par contre, si le nombre de tâche dépasse 100, les deux règles OPBM-étendu et Jackson ont un temps d'exécution très court par rapport à celui donné par l'algorithme OCF-J2(1)C_{max}. La résolution d'un problème de 400 tâches a nécessité un temps d'exécution presque de 6 minutes. A la limite de ce nombre de 400 tâches, il est remarquable que la règle de Jackson soit la mieux indiquée et préférée. Les résultats de nos différents essais, et les comportements des trois méthodes de résolution sont représentés dans les quatre graphes, figures 16-20, obtenus et représentés ci-dessous. Ces représentations sont obtenues on utilisant le grapheur Excel.

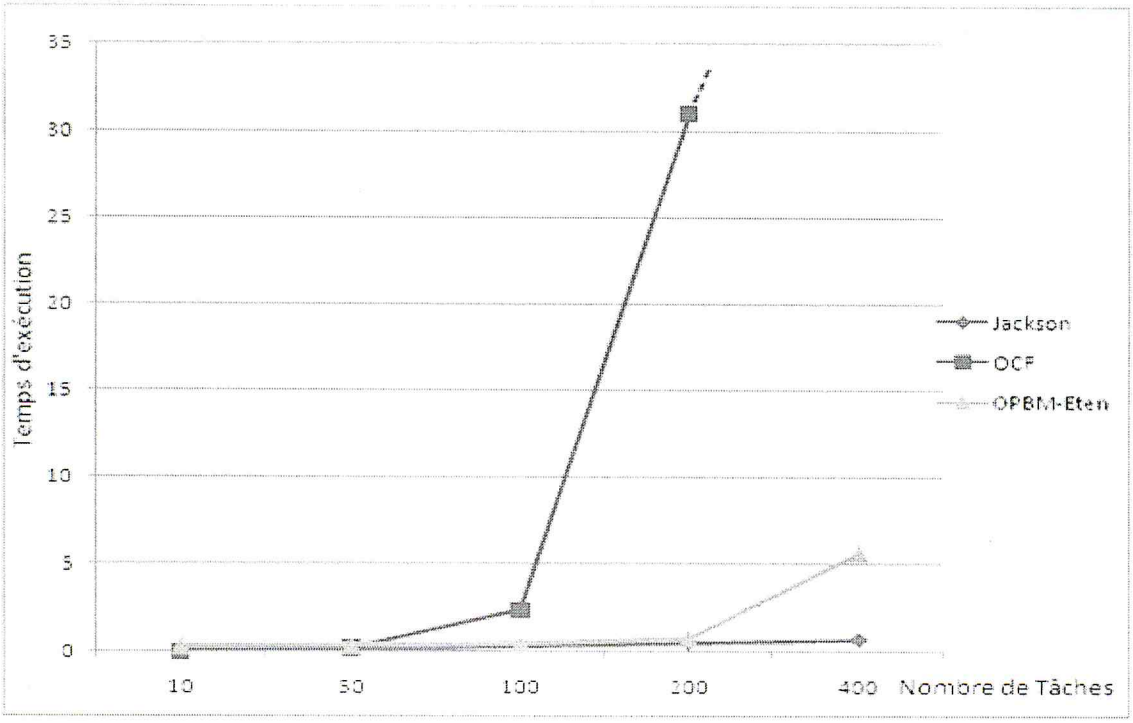


Figure 16: Graphe comparatif des méthodes en temps d'exécution, fonction du nombre de tâches

2. le deuxième critère est le Rapport Relatif Moyen, représentant l'estimation de la déviation de la valeur de la solution obtenue par rapport à la valeur de la borne inférieure proposée dans le chapitre 2. Il est défini par l'expression $\left(\frac{Cmax}{\text{Borne inférieure}}\right)$. Il doit prendre une valeur toujours supérieure ou égale à 1.

100 exemples de problèmes d'ordonnancement à 10 tâches ont été déroulés. L'OPBM-étendu s'avère amplement le meilleur. De même, avec ce même nombre d'exemples de problèmes d'ordonnancement et pour un nombre de tâches dépassant la centaine, nous confirmons encore que l'OPBM-étendu persiste toujours dans sa dominance mais l'écart entre les rapports donnés par les trois méthodes s'amoindri et se rétréci.

Ces résultats sont regroupés dans le graphique qui confirme la variation de ce rapport.

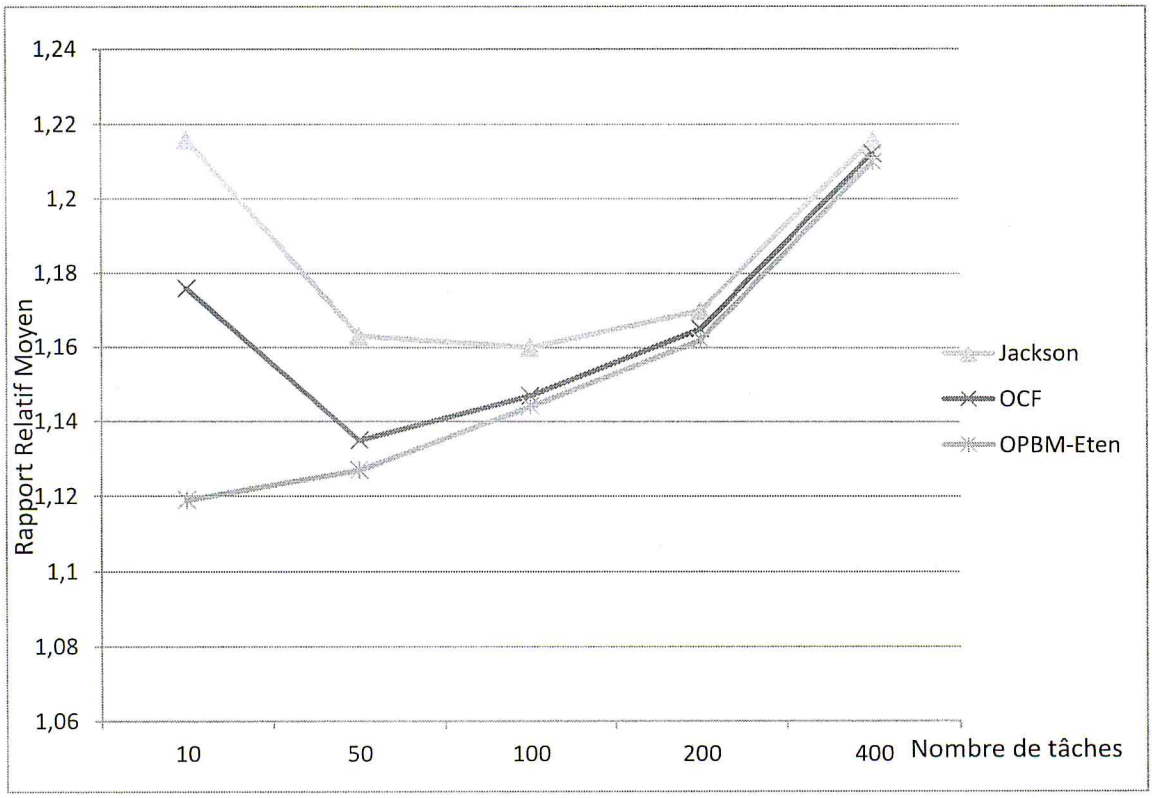


Figure 17: Courbe comparative du rapport relatif moyen, fonction du nombre de tâches

A partir du nombre de 200 tâches, la différence entre les valeurs des rapports relatifs moyens n'apparaissent pas clairement sur le graphique représenté ci-dessus. Ceci nous a incité à récapituler et à représenter nos résultats sous forme d'histogramme à bâtonnets.

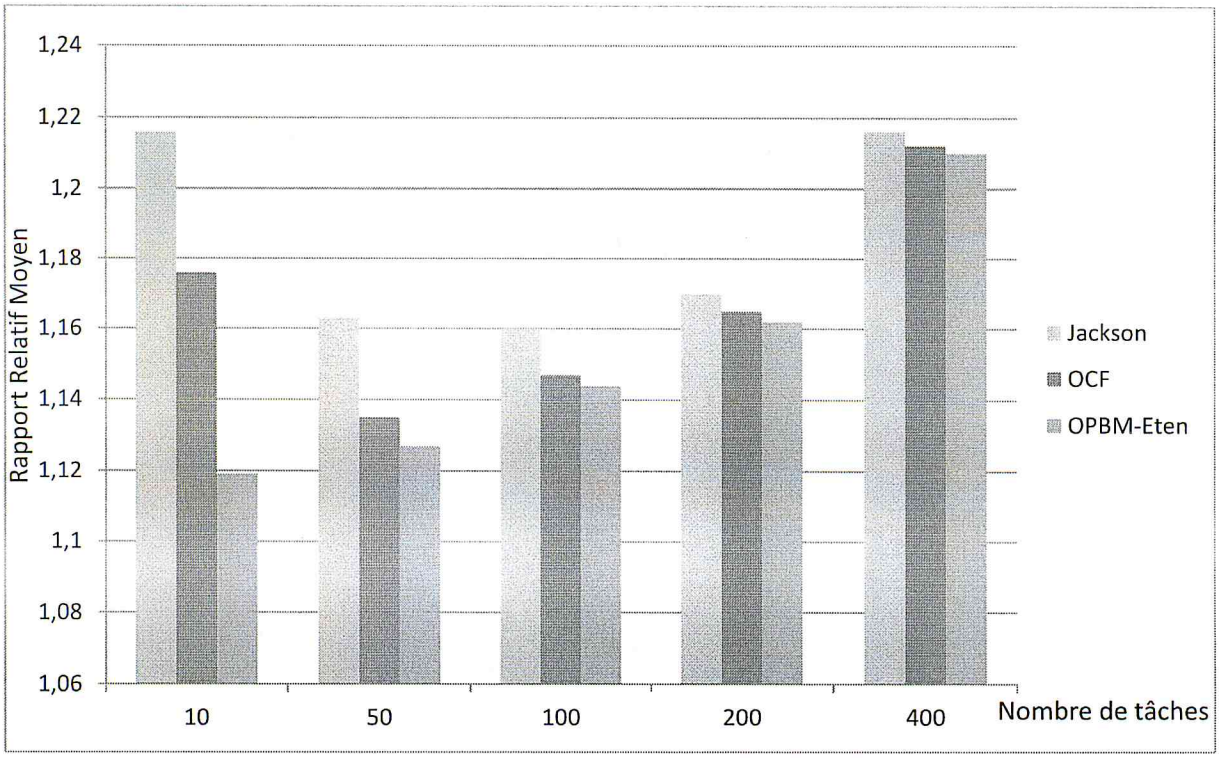


Figure 18: Histogramme comparatif du rapport relatif moyen, fonction du nombre de tâches

3. Le dernier critère est la longueur d'ordonnancement. Le graphique ci-dessous représente le comportement de la variation du C_{max} par rapport au nombre de tâches.

Comme on était animé d'un esprit de comparaison de nos résultats à ceux obtenus par Ruhlmann [6], le choix des données du problème était impérativement fait dans l'intervalle] 0, 50[. Techniquement parlant, on prenant pour unité sur l'axe des ordonnées la valeur de 2000, la représentation de la valeur du makespan pour ces trois fonctions n'indique pas une différence visible à l'œil nu. Pour obtenir une bonne visualisation des différences il fallait choisir les données des tâches et du convoyeur appartenant à l'intervalle] 0, 10[et d'imposer un nombre de tâches ne dépassant pas la centaine.

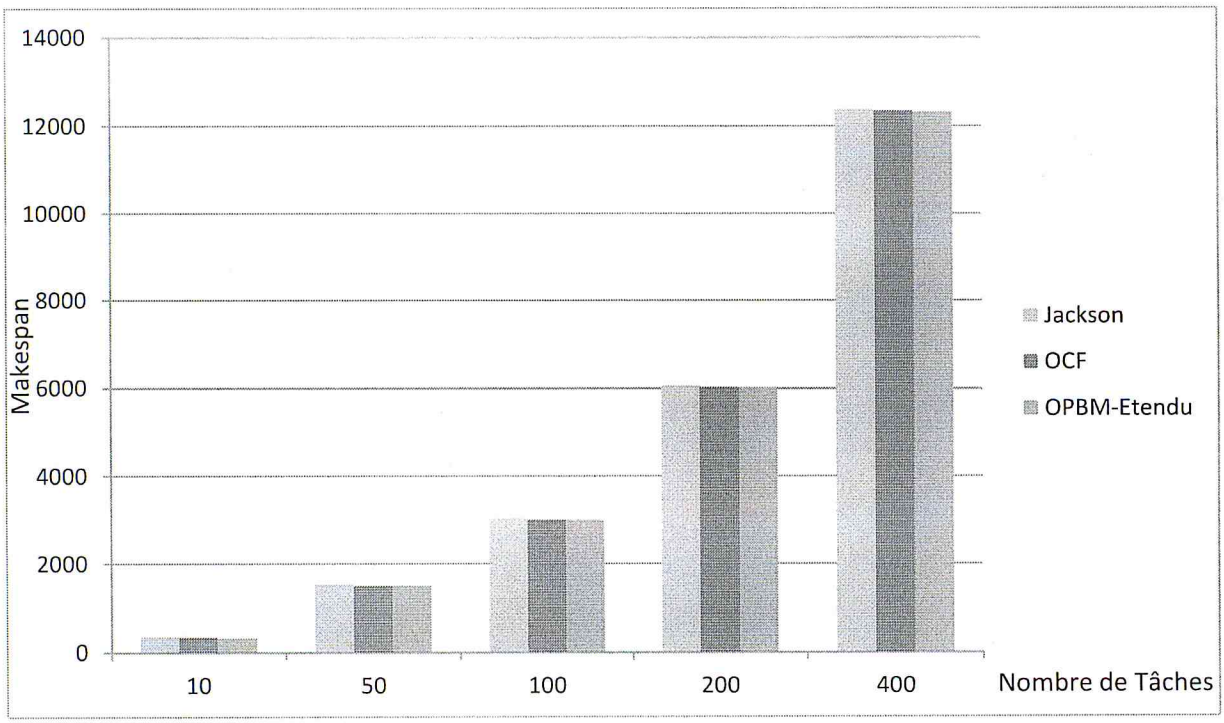


Figure 19: Histogramme comparatif du makespan, fonction du nombre de tâches

5. Conclusion et perspectives

Notre travail a porté sur l'étude de la résolution du problème d'ordonnancement dans les ateliers à deux machines disposées en série de type Job Shop avec convoyeur dont l'objectif est la minimisation du C_{max}, le temps d'achèvement total des travaux à ordonnancer appelé aussi la longueur d'ordonnancement. Deux règles de priorités, OPBM-étendu et Jackson, une métaheuristique stochastique de type colonie de fourmis sont utilisées dans la réalisation d'une étude comparative. Quel que soit le nombre de tâches à exécuter dans l'atelier, la règle de Jackson nous a fourni un temps CPU le plus court mais la solution n'est pas en générale optimale. Pour la longueur d'ordonnancement, la meilleure solution est donnée par l'application de la règle OPBM-étendu. Il s'est avéré que l'algorithme colonie de fourmis OCF-J2(1)C_{max} est moins performant que l'OPBM-étendu. Des problèmes de ce type d'atelier sont encore sans réponse ou ouverts. C'est le cas où les aires de stockages n'existent pas ou sont de tailles limitées alors les machines peuvent avoir à attendre une tâche ou bien devoir garder une tâche terminée en l'absence de convoyeur. Dans ces conditions, ces machines seront bloquées. De même, les convoyeurs peuvent se retrouver eux-mêmes bloqués. La taille de l'entreposage peut aussi poser problème si l'atelier n'accepte pas les temps

d'attente. Un autre problème est celui du transport des tâches finies entre les ateliers et les clients. Ils constitueront surement des sujets de réflexion et de recherches ultérieures.

References

1. Schuurman, P and Woeginger, J. Polynomial time approximation algorithms for machine scheduling : Ten open problems. Journal of scheduling, 2 , pp. 203-213, 1999.

2. Kunst S., (1999): Shop-scheduling problems with transportation, Osnabruck: Osnabrack. p. 137.

3. Lee C.-Y., Lei L., and Pinedo M., (1997): Current trends in deterministic scheduling, Annals of Operations Research, 70(0): pp. 1-41.

4. Tomkins J.A. and White J.A., (1984): Facilities planning. New York: Wisley.

5. Hurink J. and Kunst S., (1998): Flow-shop problems with transportation times and a single robot, Universitât Osnabrack.

6. Ruhlmann Carine. Etude du problème de job shop avec convoyeur, thèse de maîtrise en informatique, l'université du QUEBEC A CHICOUTIMI, 2 mai 2007.

7. Pirlot M., (1996): General local search methods, European Journal of Operational Research, 92(3): pp.493-511.

8. Baker, K.R. Introduction to sequencing and scheduling. John Wiley & sons, Inc. 1974.

9. Coffman, E.G Jr. . Scheduling in computer and job shop systems. J. Wiley, 1976.

10. Blazewicz, J ; Ecker, K.H ; Schmidt, G and Weglarz, J. Scheduling in computer and manufacturing systems. Second revised Edition. Springer-verlag, 1994.

11. Pinedo, M. Scheduling. Theory, Algorithms and systems. Prentice Hall, 1995

12. Graham R., Lawler E., Lenstra J.K., and Rinnooy Kan A., (1979): Optimization and approximation in deterministic sequenecing and scheduling theory: a survey, Annals of discrete mathematics, 5: pp.2 87-326.

13. Lee C.-Y. and Chen Z.-L., (2001): Machine scheduling with transportation considerations, Journal of Scheduling, 4(1): pp.3-24.

14. Müller T., (1983): Automated guided véhicules. IFS Ltd Springer-verlag

15. Hurink J. and Knust S., (2002): A tabu search algorithm for scheduling a single robot in a job-shop environment, Discrete Applied Mathematics, 119(1 -2): pp. 181 -203.

16. Bécart M., Lacomme P., Moukrim A., and Tchernev N. (2004), Proposition de résolution exacte d'un MILP pour les systèmes flexibles de production du type job-shop avec transport, in 5e conference francophone de Modélisation et simulation MOSIM, Nantes.

17. Hurink J. and Knust S., (2005): Tabu search algorithms for job-shop problems with a single transport robot, European Journal of Operational Research, 162(1); pp.99- 111.

18. Brucker P. and Knust S., (2002): Lower bounds for scheduling a single robot in a job-shop environment, Annals of Operations Research, 115(1): pp.147.

19. Strusevich V.A., (1999): A heuristic for the two-machine open-shop scheduling problem with transportation times, Discrete Applied Mathematics, 93(2-3): pp.287-304.

20. Haro C, Proust C, Un ordonnancement Equitable par Priorité Bornées. In Actes du colloque international, méthodes et outils d'aide à la décision, MOAD'92. Algeria : Béjaia, 1992 p. 46-9.

21. Dréo .J, A.Pétrowski, P.Siarry et E.Taillard. Métaheuristiques pour l'optimisation difficile. EYROLLES, 2003

22. S. Goss, S. Aron, J. L. Deneubourg et J. M. Pasteels (1989). Selforganized shortcuts in the Argentine ant. Naturwissenschaften, Vol. 76, pp. 579-581.

23. Messaoudi-Ouchen Mohamed. Résolution du problème difficile P/prec/C_{max} par un algorithmes fourmis. Thèse de Magister, Département de mathématiques, USDBlida, **29 Janvier 2013.**

24. M. Dorigo et G. D. Caro (1999a). Chapter 2 : The Ant Colony Optimization Meta-Heuristic. Dans New Ideas in Optimization, D. Corne, M. Dorigo et F. Glover, Editors: London, p. 11-32.

25. M. Dorigo, G. D. Caro et L. M. Gambardella (1999b). Ant Algorithms for Discrete Optimization. Artificial Life, Vol. 5 (2), p. 137-172.

26. M. Dorigo, V. Maniezzo et A. Colorni (1996b). The Ant System : Optimization by a colony of cooperating agents. Cybernetics-Part B, Vol. 26(1), p. 1-13.

27. Rahal Nordine et Soukhal Ameur. Implementations de l'OPBM, Ordonnancement par priorité bornée en moyenne. Mémoire d'ingénieurs. Département de mathématiques, USDBlida **Septembre 1996**.

ANNEXE-A

I. Implémentation selon Haro & Proust :

Cette implémentation se base sur deux hypothèses fondamentales. La première est que le taux d'occupation du processeur par les processus d'une classe reste constant, et la deuxième est que sur un intervalle du temps de longueur Δt chaque processus s'exécute en moyenne pendant une durée proportionnelle a sa priorité, ce qui se traduit par :

Hypothèse 1 :

$\forall t > 0$, $\rho(t)$ constant.

Les processus de E utilisent toujours la même fraction du temps d'unité centrale.

Hypothèse 2 :

Pendant un intervalle du temps Δt chaque processus s'exécute en moyenne pendant une durée proportionnelle à sa priorité.

$$\forall i, 1 \leq i \leq n , \forall t > 0 : \ t_i(\Delta t) = \frac{P_i(t)}{P} . \rho . \Delta t$$

Tel que : $P = \sum_{i=1}^{n} P_i(t)$

Lemme :

La somme des priorités des n processus de E en tout instant $t > 0$ est constante. Elle est donnée par :

$$\forall t > 0 : \sum_{i=1}^{n} P_i(t) = P = n . M - \rho . \Delta P$$

Preuve :

$$\forall t > 0 : \sum_{i=1}^{n} P_i(t) = \sum_{i=1}^{n} (M - \Delta P . \frac{t_i(t)}{t}) \qquad \text{-Définition de } P_i(t)$$

$$= n . M - \frac{\Delta P}{t} . \sum_{i=1}^{n} t_i(t) \qquad \text{-M et } \Delta P \text{ constants}$$

$$= n . M - \frac{\Delta P}{t} . \rho . t \qquad \text{-Définition de } \rho$$

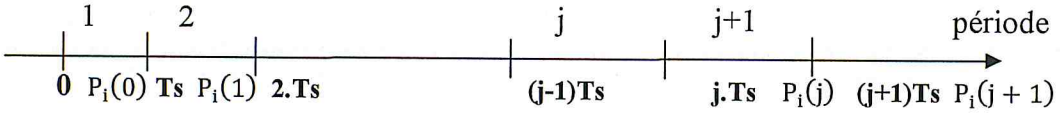
$$= n . M - \rho . \Delta P$$

1. Cas d'une seule classe :

Considérons l'ensemble de n processus d'une même classe E = $\{ \tau_i \ / \ 1 \leq i \leq n\}$ de E/R, de bornes m et M.

L'ordonnanceur est un processus système τ' périodique, de période Ts > 0. Il est de priorité fixe et n'appartient pas à E.

Tout les Ts unités de temps, il réévalue la priorité de tous les processus de E et réorganise en conséquence la queue d'exploitation.



$\forall i, 1 \leq i \leq n, \forall t \in [(j-1)Ts, j.Ts[: P_i(t) = P_i(j-1) =$ Constante.

Donc durant la période (j-1) , τ_i a la priorité $P_i(j)$,$1 \leq i \leq n$.

Cette implémentation ne tient compte que du passé récent d'un processus pour recalculer sa priorité. C.à.d. la priorité relevant du comportement du processus lors de la dernière pour une durée $t_i(j)$, et il attend pendant $t_i^{(a)}(j)$.

Donc $\forall i, 1 \leq i \leq n, \forall j > 0 : P_i(j) = M - \Delta P \frac{t_i(j-1)}{Ts}$

Et le rapport $\frac{t_i(j-1)}{Ts}$ est le taux d'occupation $\rho_i(j-1)$de l'unité centrale par τ_i durant la période j-1.

Donc le taux d'occupation est donnée par : $\rho = \frac{\sum_{i=1}^{n} t_i(j-1)}{Ts}$

D'autre part : $\forall i, 1 \leq i \leq n, \forall j > 0 : \rho_i(j) = \frac{P_i(j)}{P} . \rho$ car durant Ts ,le processeur exécutera au moins τ' un processus système qui n'appartient pas à E. on a : $0 < \rho < 1$

2. Equité :

L'ordonnancement est équitable pour E si $\forall i, 1 \leq i \leq n, \forall j > 0 : t_i(j) = \frac{\rho.Ts}{n}$

Et il est asymptotiquement équitable si : $\forall i, 1 \leq i \leq n : \lim_{t \to \infty} t_i(t) = \frac{\rho.Ts}{n}$

Durant la période j+1, un processus $\tau_i$quelconque de E s'exécute pendant un temps $t_i(t)$ moyen donnée par : $\forall i, 1 \leq i \leq n, \forall j > 0 :$ $t_i(j) = \frac{P_i(j)}{P} . \rho . Ts$

Algorithme :

Début.

Lire : m , M , n , Ts , Durée ;

Pour i=1 à n

Faire :

lire : $P_i(0)$;

Fait.

Fin pour

$\Delta P = M - m$;

$P = n.M - \Delta P$;

$j = 1$;

Répéter

Pour i=1 à n

Faire :

$te_i(j) = \frac{P_i(j-1)}{P} . Ts$:

Fait.

Fin pour

j++ ;

Pour i=1 à n

Faire :

$P_i(j) = M - \Delta P \frac{te_i(j-1)}{Ts}$;

Fait.

Fin pour

Ecrire ($P_i(j)$ ***et*** $te_i(j-1)$)

TantQue($i.Ts \leq$ Durée) ;

Fin.

Exemple 1 :

Le système contient 2 Jobs :

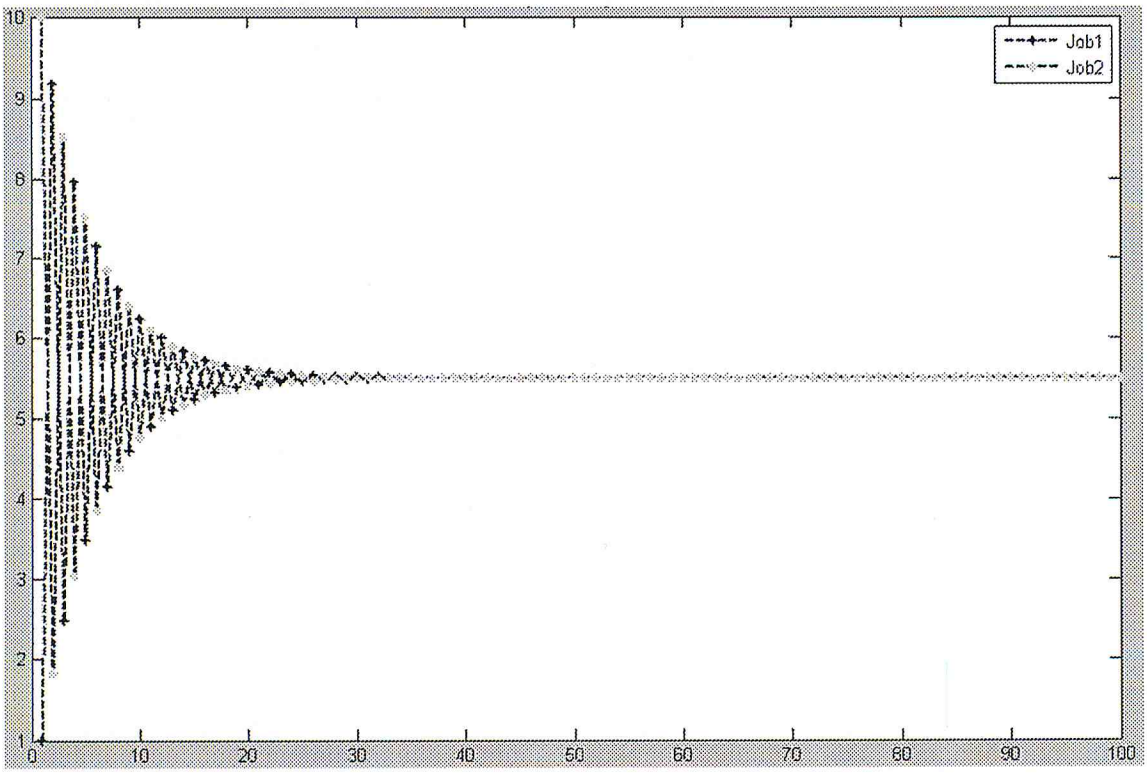
- Durée de la période Ts = 1 ;
- Les bornes de la classe : m = 1 , M = 10
- $P_1(0) = 1, P_2(0) = 10$.
- Durée d'ordonnancement Durée = 100 (en réalité cette durée est infini).

Exemple 2 :

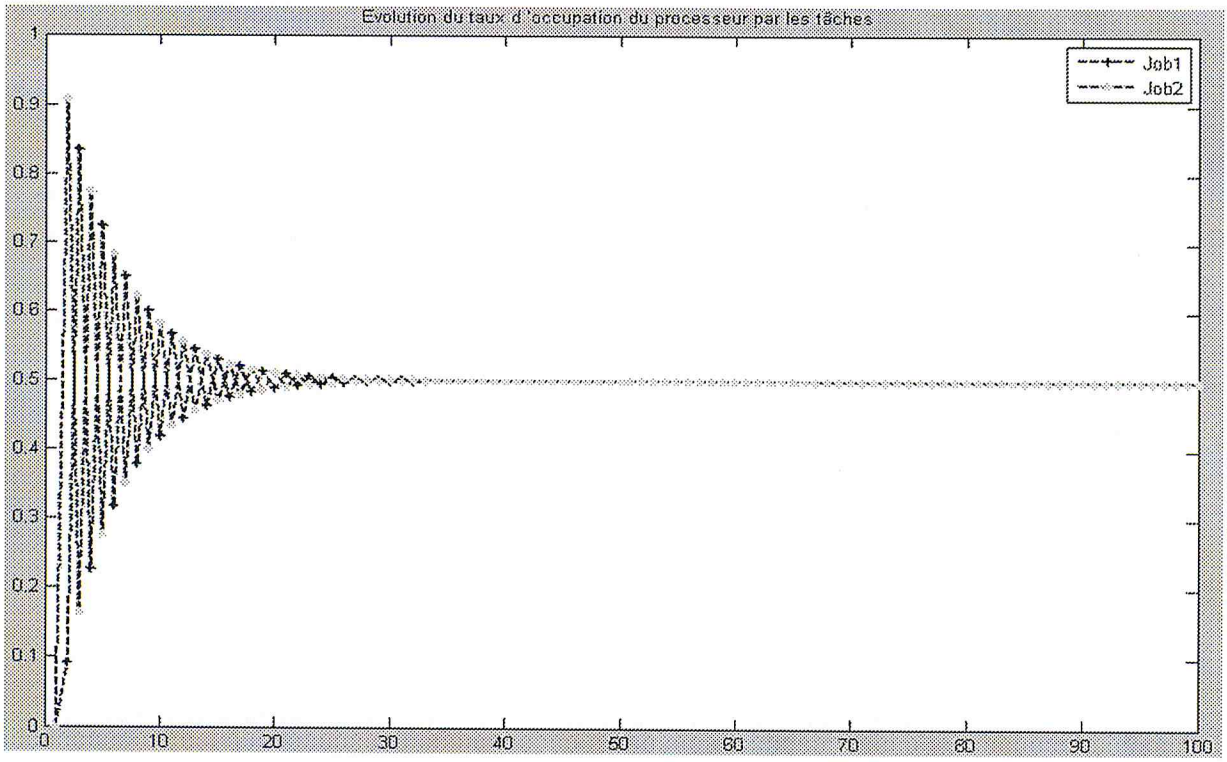
Le système contient 5 Jobs :

- Durée de la période Ts = 1 ;
- Les bornes de la classe : m = 1, M = 20
- $P_1(0) = 1, P_2(0) = 5, P_3(0) = 11, P_4(0) = 15, P_5(0) = 20$
- Durée d'ordonnancement Durée = 10.

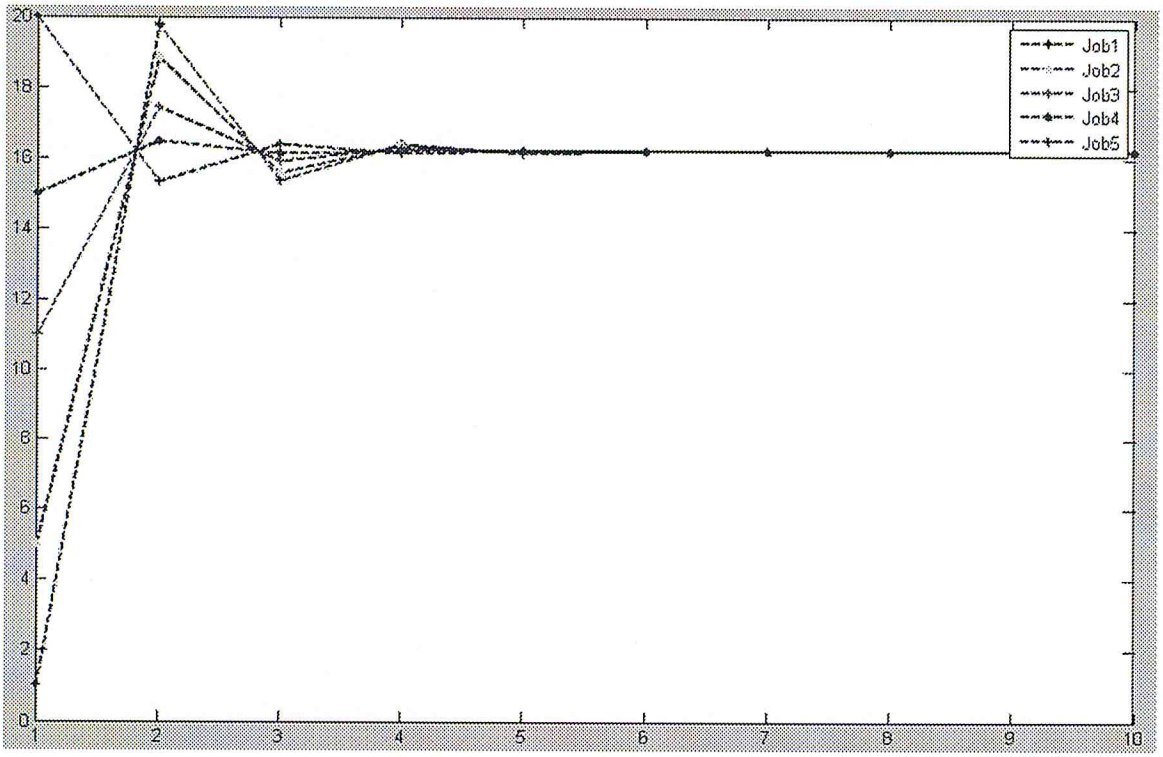
Graphique I.1 : Evolution des priorités des tâches.



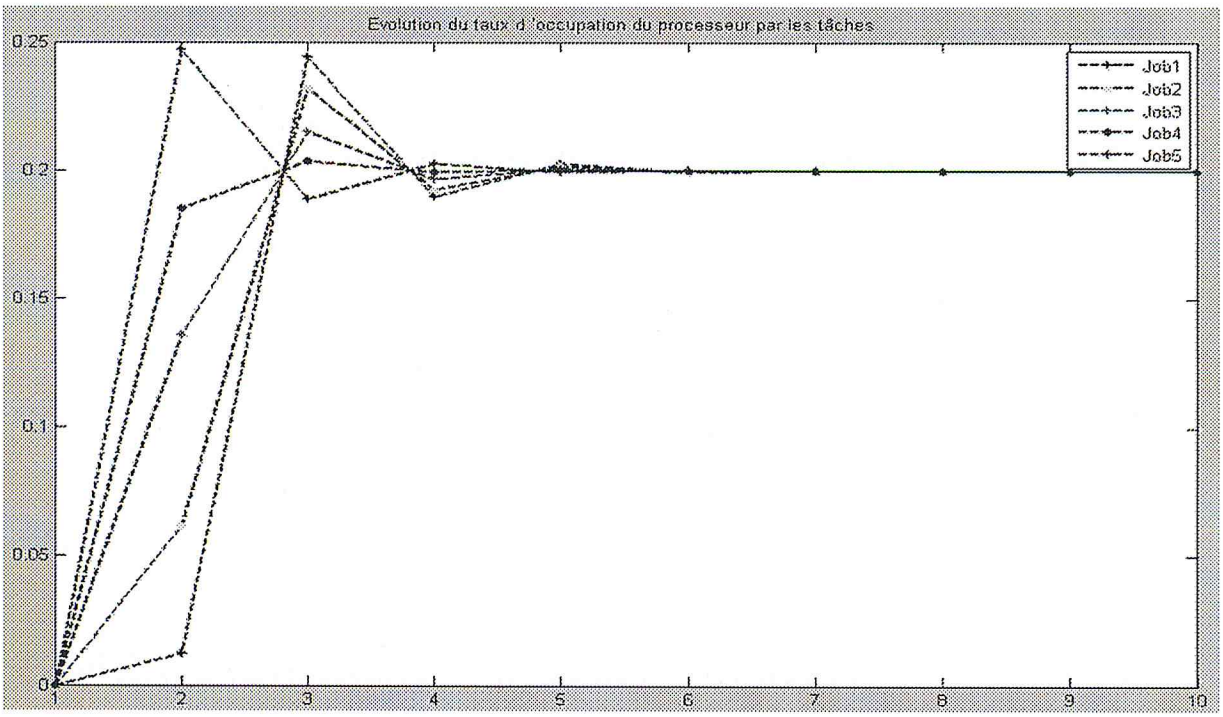
Graphique I.2 : Evolution du taux d'occupation du processeur par les Tâches.



Graphique I.3 : Evolution des priorités des tâches.

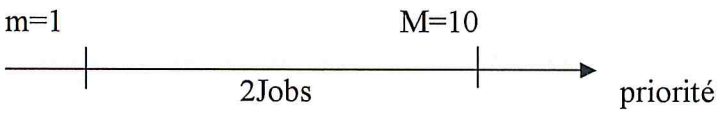


Graphique I.4 : Evolution du taux d'occupation du processeur par les Tâches.



3. Vérification de l'équité :

Pour l'exemple 1 :



Si l'ordonnancement est équitable alors les priorités tendent vers la valeur $\frac{P}{n} = \frac{11}{2} = 5.5$

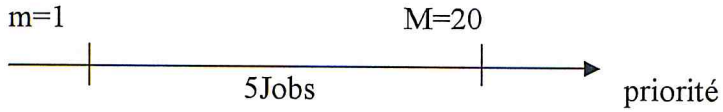
Le graphique 1. Montre la convergence de priorités vers la valeur 5.5, on peut constater qu'elle est rapidement atteinte.

Equité en termes de taux d'occupation :

Un ordonnancement équitable en termes de taux d'occupation devra accorder à chaque tâche la moitié du temps total d'exécution.

Le graphique 2. Montre la convergence des taux d'occupation des deux tâches vers la valeur 0.5.

Pour l'exemple 2 :



L'ordonnancement sera équitable si les priorités des jobs tendent vers la valeur :

$$\frac{P}{n} = \frac{(5.20-19)}{5} = \frac{81}{5} = 16.2$$

L'évolution des priorités est représentée par le graphique 3. Et on peut voir l'équité de priorités à 16.2.

Equité en termes de taux d'occupation :

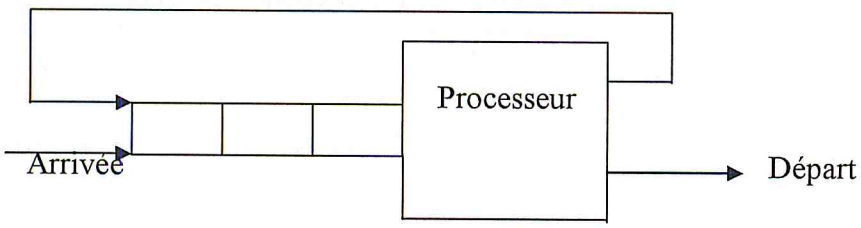
Le graphique 4. Montre la convergence des taux d'occupation des cinq jobs vers la valeur 0.2.

II. Implémentation proposée

L'un des algorithmes le plus répandu et le plus fiable est le Round Robin il s'appelle aussi Tourniquet ou l'ordonnancement circulaire.

Chaque processus est autorisé à s'exécuter pendant une durée fixe Q appelée Quantum.

Si un processus se termine ou se bloque avant la fin du Quantum, le processeur est immédiatement réallouer au suivant, cette politique est mise en œuvre en ordonnant les processus dans une file circulaire (figure 1.)



- **Figure 1** -

Cette implémentation tient compte de tout le passé d'un processus pour recalculer sa priorité. Ce calcule s'effectue chaque Ts unité de temps par la formule.

$\forall i, 1 \leq i \leq n, \forall j > 0 : \quad P_i(j.Ts) = M - \frac{\Delta P}{i.Ts} . te_i((j-1).Ts)$

Les temps d'exécutions à l'instant (j.Ts) de chaque job sont calculés par :

$\forall i, 1 \leq i \leq n, \forall j > 0 :$

$$te_i(j.Ts) = \begin{cases} te_i((j-1).Ts) + Ts \ /si\ seulement\ le\ job\ i\ a\ une\ priorité\ maximale. \\ te_i((j-1).Ts) + Q \ /si\ le\ job\ i\ est\ l'un\ des\ taches\ qui\ sont\ en\ conflit \\ te_i((j-1).Ts) \ / sinon \end{cases}$$

Le taux d'occupation du processeur par les processus est donné par :

$\forall i, 1 \leq i \leq n, \forall j > 0 : \ \rho_i(j.Ts) = \frac{te_i((j-1).Ts)}{j.Ts}$

Algorithme

Début.

Lire : m, M, n, Ts, Q, Durée.

Pour i=1 à n

Faire :

lire : $P_i(0)$;

Fait.

Fin pour

$\Delta P = M - m$;

$j = 1$; $t = 0$;

Pour i=1 à n

Faire :

$te_i(0) = 0$;

Fait.

Fin pour

$Max_{conflit} = \{i / P_i(0) \geq P_1(0) \forall i \neq 1 \}$

Répéter

Si card(Max)≥1

i = 1 ;

Répéter

Si i ∈ Max

$te_i = te_i + Q$;

t = t +Q ;

Sinon

i = i +1 ;

TantQue (i≤n) ;

Sinon

t = t +Ts ;

élu = i / i ∈ Max ;

$te_{élu} = te_{élu} + Ts$;

j = j+1 ;

Pour i=1 à n

Faire :

$P_i(t) = M - \Delta P * \frac{te_i}{t}$;

Fait.

Fin pour

Ecrire ($P_i(t)$) ;

$Max_{conflit} = \{i / P_i(t) \geq P_1(t) \forall\, i \neq 1\}$;

TantQue (t≤Duree) ;

Fin.

Remarque :

Si on veut tenir compte de tout le passé d'un processus, le calcul de sa priorité s'effectuera alors en fonction de son temps d'exécution cumulé.

Exemple 1 :

Le système contient 2 Jobs :

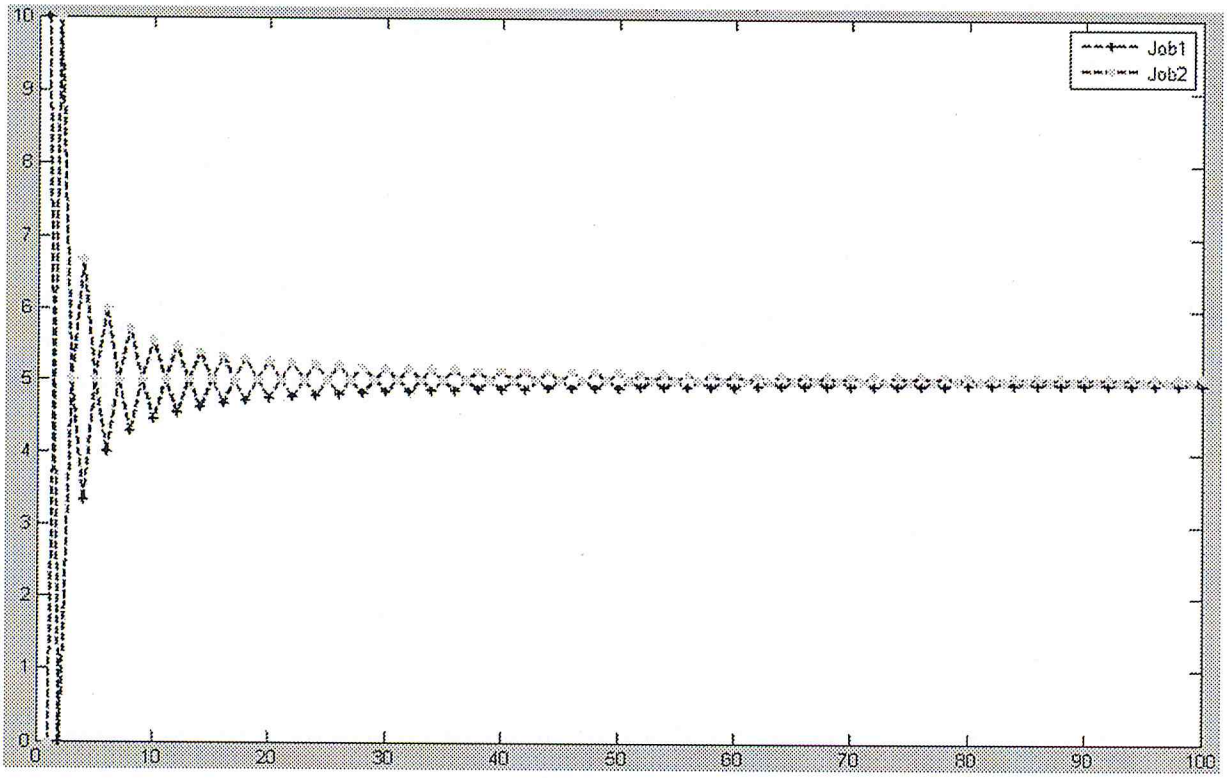
- Durée de la période Ts = 1, Q = 1 ;
- Les bornes de la classe : m = 0 , M = 10 ;
- Les priorités initiales : $P_1(0) = 0, P_2(0) = 10;$
- Durée d'ordonnancement Durée = 100;

Exemple 2 :

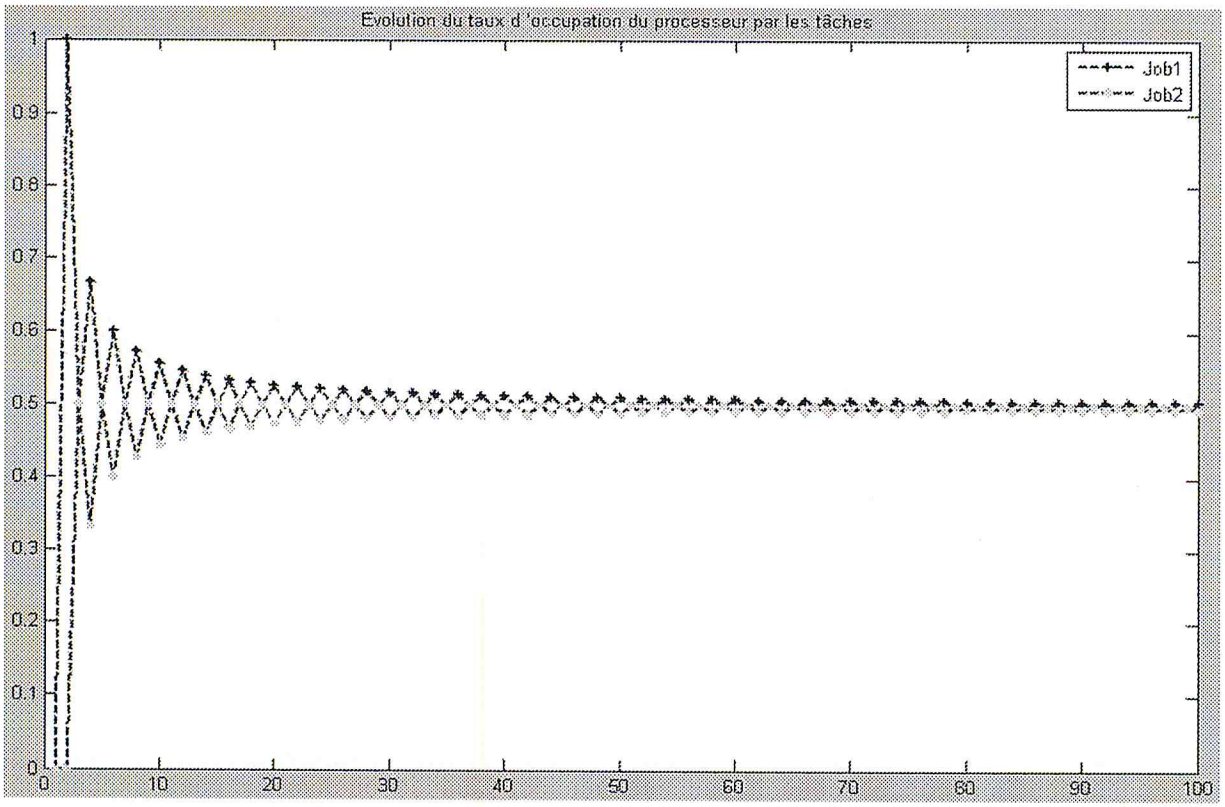
Le système contient 4 Jobs :

- Durée de la période Ts = 1,Q = 1 ;
- Les bornes de la classe : m = 0 , M = 20
- $P_1(0) = 0, P_2(0) = 5, P_3(0) = 15, P_4(0) = 20$
- Durée d'ordonnancement Durée = 100.

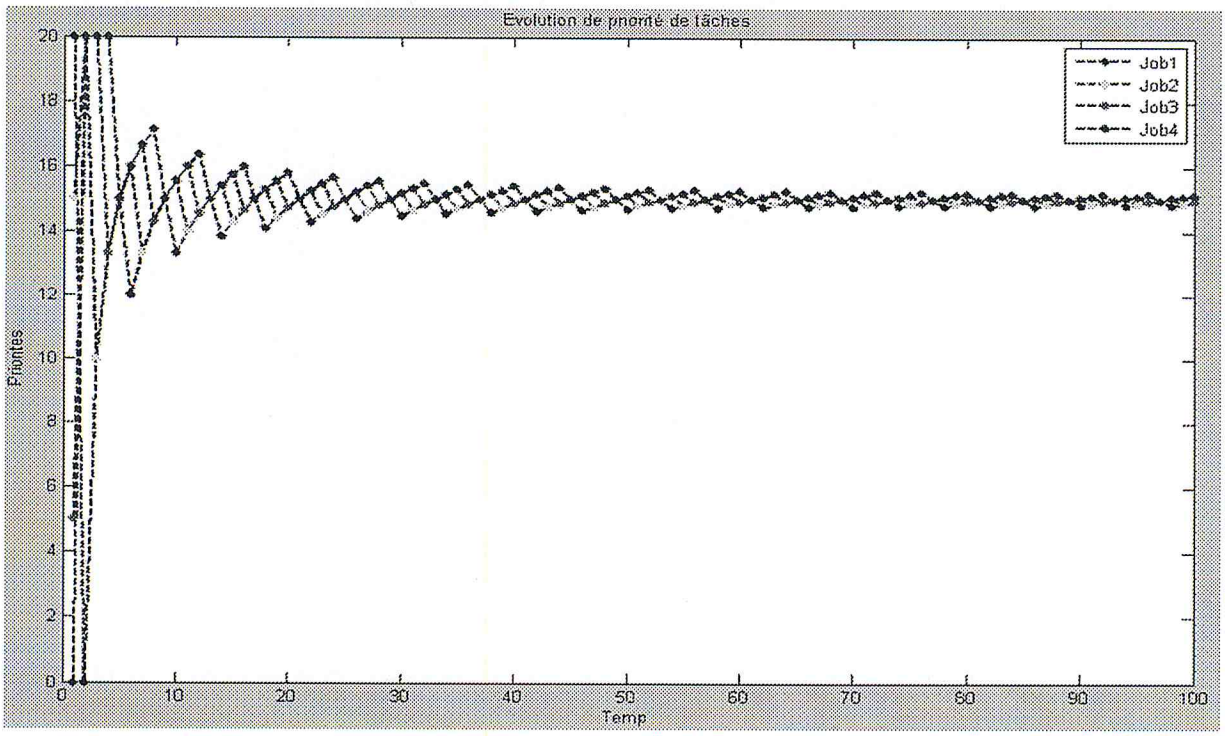
Graphique III.1 : Evolution des priorités des tâches.



Graphique III.2 : Evolution du taux d'occupation du processeur par les Tâches.



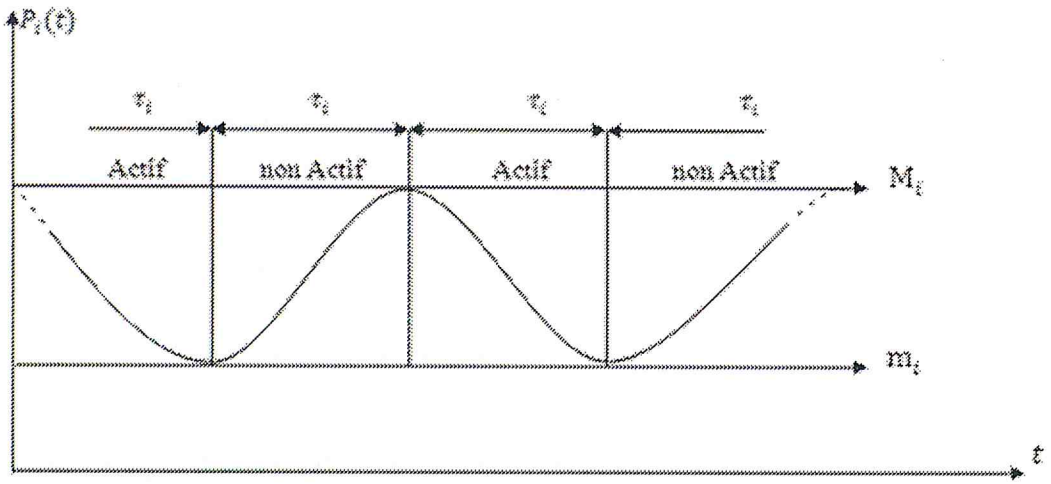
Graphique III.3 : Evolution des priorités des tâches.



ANNEXE-B

1. Caractéristique du processus :

Si les processus de même classe E = $\{ \tau_i / i_1 \leq i \leq i_k / k \leq n\}$ sont actifs, alors la priorité de chaque processus continue à diminuer.



En général, les processus ont comme caractéristiques :

1. Si un processusτ_i, est actif ⇒ le passé d'exécution $t_i(t)$ croit ⇒ la valeur de priorité $P_i(t)$ décroit.
2. Si un processusτ_i, est inactif ⇒ le passé d'exécution $t_i(t)$ constant ⇒ la valeur de priorité $P_i(t)$ croit.

Il y a lieu de rappeler que les processus ont comme changement d'état d'activité

Actif ⇆ Inactif

2. Système contenant deux processus

Soit un système contenant deux processus$\tau_i$etτ_j, utilisent le processeur pendant les intervalles finis du temps t_i et t_j avec $t_i \geq t_j > 0$.

L'exécution de processus ne peut s'interrompre qu'à la fin de son temps d'exécution, donc à chaque itération, le temps de calcul t augmente par la quantité $(t_i + t_j)$.

D'où : t prendra à chaque itération k les valeurs $(t_i + t_j), 2(t_i + t_j) \ldots k\,(t_i + t_j). \forall\, k > 0$

Le passé d'exécution de chaque processus, après la première exécution des processus, augmente par sa valeur de temps d'exécution. Une nouvelle évolution de priorité s'est produite :

$P_i \leq P_j \Leftrightarrow M - \Delta P \frac{t_i}{t} \leq M - \Delta P \frac{t_j}{t}$ Avec un temps de calcul t valant $t_i + t_j$

Le système possède deux processus, (τ_i, τ_j) est le seul couple d'échange de passé d'exécution entre les deux processus. A la 1ère itération l'échange de temps d'exécution est de permuter t_i et t_j

$P_i \geq P_j \Leftrightarrow M - \Delta P \frac{t_j}{t} \geq M - \Delta P \frac{t_i}{t}$ Avec $t = (t_i + t_j)$

Refait l'exécution des processus avec le passé nouvellement attribué fait augmenter le passé d'exécution de chaque processus par sa valeur de temps d'exécution.

Le passé d'exécution du processus τ_i augmente par la valeur t_i et de τ_j augmente par t_j. Des nouvelles valeurs des priorités sont présentées

$P_i = P_j \Leftrightarrow M - \Delta P \left(\frac{t_i + t_j}{t}\right) = M - \Delta P \left(\frac{t_j + t_i}{t}\right)$ Avec un temps de calcul t valant $2(t_i + t_j)$

Les priorités sont égales, le passé d'exécution de chaque processus reste inchangeable.

A la 2^{eme} itération on aura :

$P_i = P_j \Leftrightarrow M - \Delta P \left(\frac{t_i + t_j}{t}\right) = M - \Delta P \left(\frac{t_j + t_i}{t}\right)$ Avec $t = 2(t_i + t_j)$

Une troisième exécution des processus retourne une nouvelle variation de priorités :

$P_i \leq P_j \Leftrightarrow M - \Delta P \left(\frac{2t_i + t_j}{t}\right) \leq M - \Delta P \left(\frac{t_i + 2t_j}{t}\right)$ Avec un temps de calcul t valant $3(t_i + t_j)$

Le couple d'échange de passé d'exécution est toujours (τ_i, τ_j)

A la 3^{eme} itération on aura :

$P_i \geq P_j \Leftrightarrow M - \Delta P \left(\frac{t_i+2t_j}{t}\right) \geq M - \Delta P \left(\frac{t_j+2t_i}{t}\right)$ Avec t =3($t_i + t_j$)

A la 4^{eme}itération : le passé d'exécution de chaque processus augmente par sa valeur de temps d'exécution

$P_i = P_j \Leftrightarrow M - \Delta P \left(\frac{2(t_i+t_j)}{t}\right) = M - \Delta P \left(\frac{2(t_i+t_j)}{t}\right)$ Avec t = 4($t_i + t_j$)

A la 5^{eme} itération:$P_i \geq P_j \Leftrightarrow M - \Delta P \left(\frac{2t_i+3t_j}{t}\right) \geq M - \Delta P \left(\frac{2t_j+3t_i}{t}\right)$ Avec t =5($t_i + t_j$)

Et on continue de façon fréquentative.

A la k^{eme} itération :

Si k est pair $P_i = P_j \Leftrightarrow M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(t_i+t_j)}{t}\right) = M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(t_i+t_j)}{t}\right) /t = k(t_i + t_j)$ … **(1)**

Si k impair

$$P_i \geq P_j \Leftrightarrow M - \Delta P\left(\frac{\left(\frac{k+1}{2}\right)t_j+\left(\frac{k-1}{2}\right)t_i}{t}\right) \geq M - \Delta P\left(\frac{\left(\frac{k-1}{2}\right)t_j+\left(\frac{k+1}{2}\right)t_i}{t}\right) /t = k(t_i + t_j) \ldots . \textbf{(2)}$$

La vérification de l'équité s'effectue lorsque le nombre d'itération k tend vers l'infini

(1) $\Rightarrow P_i = P_j \Leftrightarrow M - \Delta P \left(\frac{\left(\frac{k}{2}\right)(t_i+t_j)}{k(t_i+t_j)}\right) = M - \Delta P \left(\frac{\left(\frac{k}{2}\right)(t_i+t_j)}{k(t_i+t_j)}\right)$

$\Leftrightarrow M - \Delta P \left(\frac{k}{k2}\right) = M - \Delta P \left(\frac{k}{k2}\right) \Rightarrow P_i = P_j = M - \left(\frac{\Delta P}{2}\right)$ C'est un nœud … **(3)**

(2) $\Rightarrow P_i \geq P_j \Leftrightarrow M - \Delta P \left(\frac{\left(\frac{k+1}{2}\right)t_j+\left(\frac{k-1}{2}\right)t_i}{k(t_i+t_j)}\right) \geq M - \Delta P \left(\frac{\left(\frac{k-1}{2}\right)t_j+\left(\frac{k+1}{2}\right)t_i}{k(t_i+t_j)}\right) \Leftrightarrow$

$M - \Delta P\left(\frac{\left(\frac{k}{2}\right)}{k} - \frac{\left(\frac{1}{2}\right)(t_i-t_j)}{(k)(t_i+t_j)}\right) \geq M - \Delta P\left(\frac{\left(\frac{k}{2}\right)}{k} - \frac{\left(\frac{1}{2}\right)(t_j-t_i)}{(k)(t_i+t_j)}\right)$ $\Leftrightarrow$ Lorsque k tend vers l'infini on aura

$$P_i = \lim_{k\to\infty}\left(M - \Delta P\left(\frac{\left(\frac{k}{2}\right)}{k} - \frac{\left(\frac{1}{2}\right)(t_i-t_j)}{(k)(t_i+t_j)}\right)\right) = M - \Delta P\ \frac{1}{2}$$

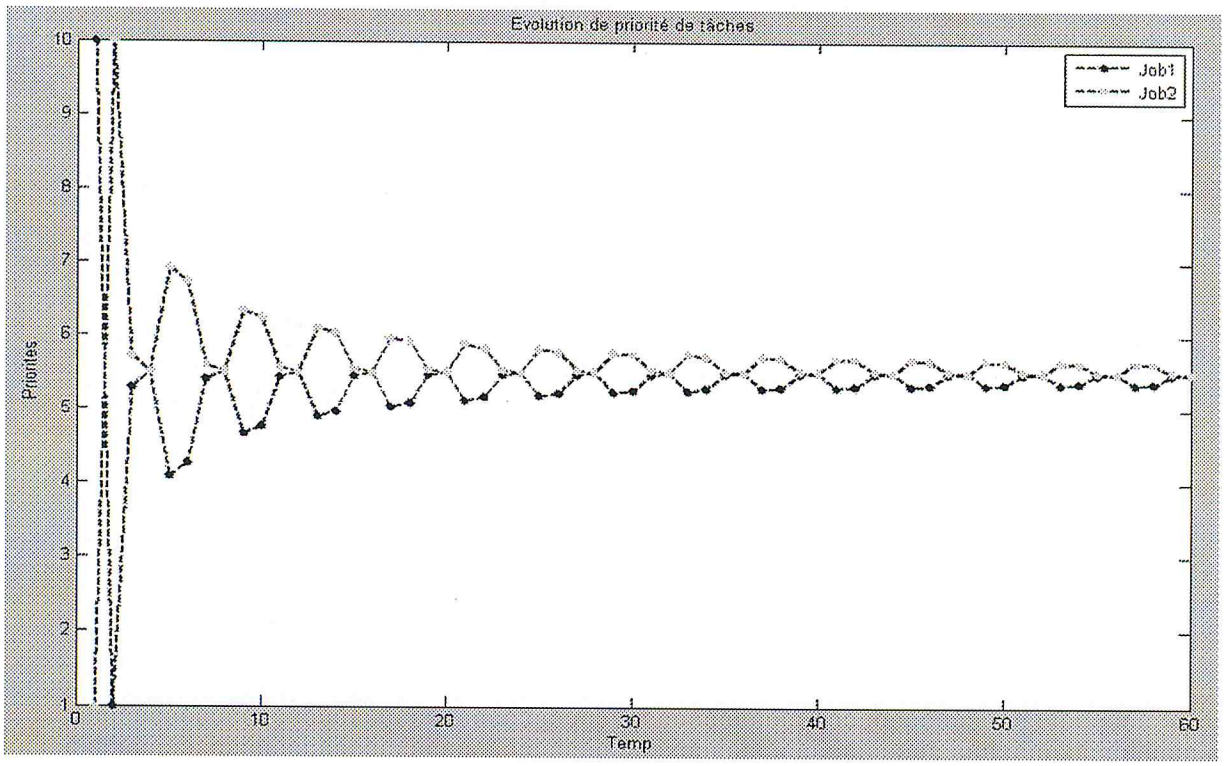
$$P_j = \lim_{k\to\infty}\left(M - \Delta P\left(\frac{\left(\frac{k}{2}\right)}{k} - \frac{\left(\frac{1}{2}\right)(t_j-t_i)}{(k)(t_i+t_j)}\right)\right) = M - \Delta P\ \frac{1}{2}$$

⇒ Les priorités tendent vers la valeur d'équité $(M - \frac{\Delta P}{2})$ **… (4)**

Alors, de **(3)** et **(4)** les priorités sont équitables.

Le graphique ci-dessous présente l'évolution de priorité de l'exemple suivant :

Un système contenant deux processus τ_1, τ_2, utilisent le processeur pendant les durées t_1=10, t_2=1. Supposons M = 10 et m = 1. $P_i(0) = 10$ et $P_j(0) = 1$.



Graphique B.1: Evolution de priorité des tâches

Remarque :

Le cas ou les temps d'exécution sont identiques valant l'invariant C

$$\textbf{(1)} \Leftrightarrow P_i = P_j \Leftrightarrow M - \Delta P \left(\frac{\left(\frac{k}{2}\right)(t_i+t_j)}{k(t_i+t_j)}\right) = M - \Delta P \left(\frac{\left(\frac{k}{2}\right)(t_i+t_j)}{k(t_i+t_j)}\right)$$

$$\Leftrightarrow M - \Delta P \left(\frac{\left(\frac{k}{2}\right)(2C)}{k.2C}\right) = M - \Delta P \left(\frac{\left(\frac{k}{2}\right)(2C)}{k.2C}\right) \Rightarrow P_i = M - \frac{\Delta P}{2}$$ C'est un nœud

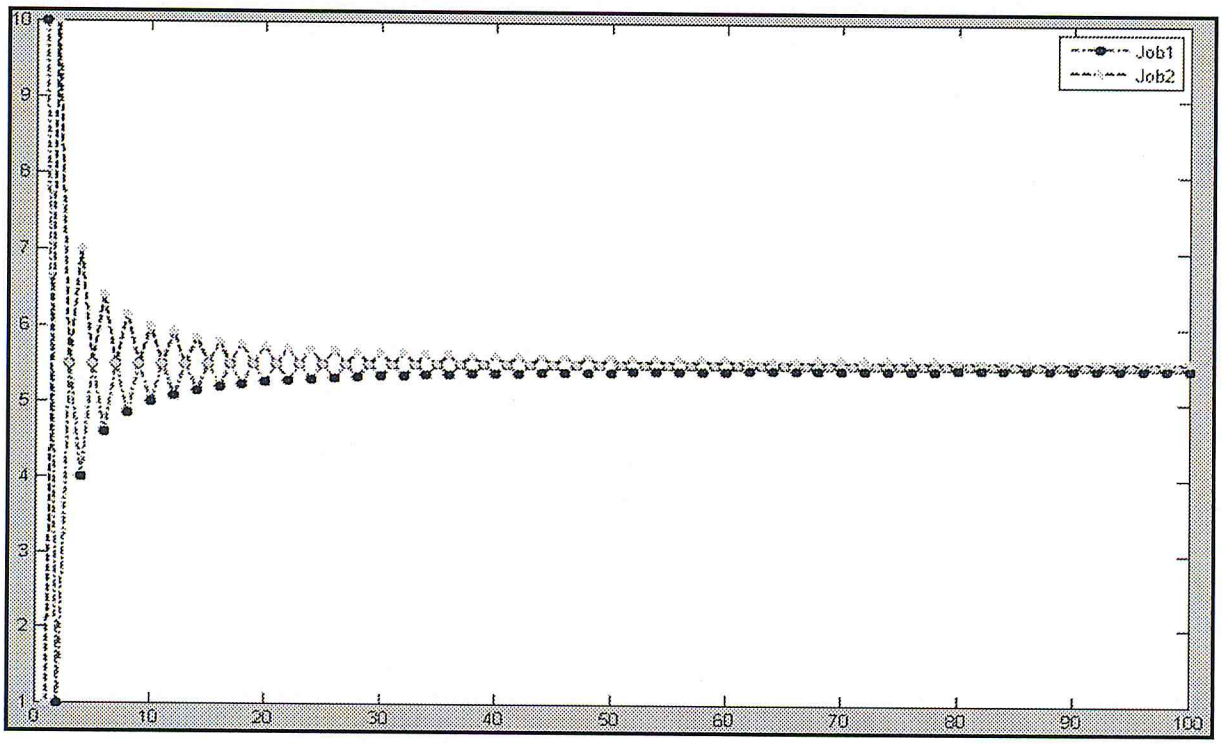
$$\textbf{(2)} \Leftrightarrow P_i \geq P_j \Leftrightarrow M - \Delta P \left(\frac{\left(\frac{k+1}{2}\right)t_j + \left(\frac{k-1}{2}\right)t_i}{k(t_i+t_j)}\right) \geq M - \Delta P \left(\frac{\left(\frac{k-1}{2}\right)t_j + \left(\frac{k+1}{2}\right)t_i}{k(t_i+t_j)}\right) \text{ avec } t = k(t_i + t_j)$$

$$\Leftrightarrow M - \Delta P \left(\frac{\left(\frac{k+1}{2}\right)C + \left(\frac{k-1}{2}\right)C}{k(2C)}\right) \geq M - \Delta P \left(\frac{\left(\frac{k-1}{2}\right)C + \left(\frac{k+1}{2}\right)C}{k(2C)}\right) \text{ Avec } t = 2kC$$

$$\Leftrightarrow M - \Delta P \left(\frac{kC}{k(2C)}\right) \geq M - \Delta P \left(\frac{kC}{k(2C)}\right) \Rightarrow P_i = M - \frac{\Delta P}{2} \text{ C'est un nœud}$$

Le graphique ci-dessous montre que la modification de l'OPBM ne change rien si les temps de service des processus sont unitaires. Et que l'OPBM n'est qu'un cas particulier de l'OPBM modifiée.

On constate que le graphique obtenu est celui de l'OPBM appliqué sur un problème préemptif. On attribue ce comportement à le fait que, si les t_i^e sont unitaires donc les deux processus ont le même passé à chaque itération donc il n'y aura aucun échange.



Graphique B.2 : Evolution de priorité des processus.

3. Système contenant plus de deux processus :

On avait vu le comportement de l'OPBM modifiée pour un système contenant deux processus, et on a pu montrer l'équité de priorité, et on avait vu que si on prend les temps d'exécution identiques on aboutit à l'OPBM ordinaire, la question qui se pose

Est-ce que les priorités vont garder les caractéristiques vues auparavant ?

Soit un système contenant n processus, chaque processus utilise le processeur pendant les intervalles finis du temps valant $t_1 \geq t_2 \geq \ldots\ldots \geq t_{n-1} \geq t_n$ où tous les $t_i > 0, \forall i = 1 \ldots n$.

Il est indispensable d'ordonner les temps d'exécution d'ordre décroissant car la cardinalité de l'ensemble de tâches prioritaires en conflit $\text{Max}_{\text{conflit}} = \{i/P_i(0) \geq P_1(0) \forall\, i \neq 1\}$ relevant de la valeur de priorité du premier processus, il aurait pu être croissant, mais l'ensemble $\text{Max}_{\text{conflit}}$ devient $\text{Max}_{\text{conflit}} = \{i/P_i(0) \geq P_n(0) \forall\, i \neq n\}$.

L'exécution de processus ne peut s'interrompre qu'à la fin de son temps d'exécution, donc à chaque itération, le temps de calcul t augmente par la quantité $\sum_{i=1}^{n} t_i$. D'où, t prendra à chaque itération k les valeurs suivantes :$\sum_{i=1}^{n} t_i$, $2(\sum_{i=1}^{n} t_i)$, $3(\sum_{i=1}^{n} t_i)$, …, k $(\sum_{i=1}^{n} t_i)$ $\forall k > 0$.

Après la première exécution des processus, des nouvelles valeurs de priorités sont produites :

$P_1 \leq P_2 \leq \ldots \leq P_{n-1} \leq P_n \Leftrightarrow$

$$M - \Delta P\left(\frac{t_1}{t}\right) \leq M - \Delta P\left(\frac{t_2}{t}\right) \leq \cdots \leq M - \Delta P\left(\frac{t_{n-1}}{t}\right) \leq M - \Delta P\left(\frac{t_n}{t}\right)$$

A la $1^{ère}$ itération on fait l'échange entre les termes, car on posé $t_1 \geq t_2 \geq \ldots\ldots \geq t_n$.

Les couples d'échange sont : (τ_1, τ_n), (τ_2, τ_{n-1}), (τ_3, τ_{n-2}),….

$P_1 \geq P_2 \geq \ldots\ldots \geq P_{n-1} \geq P_n \Leftrightarrow$

$$M - \Delta P\left(\frac{t_n}{t}\right) \geq M - \Delta P\left(\frac{t_{(n-1)}}{t}\right) \geq \ldots \geq M - \Delta P\left(\frac{t_2}{t}\right) \geq M - \Delta P\left(\frac{t_1}{t}\right)$$

Après la deuxième exécution des processus, le passé d'exécution de chaque processus augmente par sa valeur de temps d'exécution.

Nouvelles valeurs de priorités se produisent, mais d'ordre inconnu:

$P_1 \leq\geq P_2 \leq\geq \ldots\ldots \leq\geq P_{n-1} \leq\geq P_n \Leftrightarrow$

$M - \Delta P\left(\frac{t_n + t_1}{t}\right) \leq\geq M - \Delta P\left(\frac{t_{(n-1)} + t_2}{t}\right) \leq\geq \cdots \leq\geq M - \Delta P\left(\frac{t_2 + t_{(n-1)}}{t}\right) \leq\geq M - \Delta P\left(\frac{t_1 + t_n}{t}\right)$

Avec un temps de calcul $t = 2\sum_{i=1}^{n} t_i$

Dès maintenant les couples d'échange seront inconnus car on ne sait pas l'ordre de priorités, sauf si les t_i forment une suite arithmétique.

Si les t_i forment une suite arithmétique de raison t_n, une nouvelle allure se présente

On a posé $t_1 \geq t_2 \geq \ldots \geq t_{n-1} \geq t_n$, donc les temps d'exécutions sont $t_1 = n\, t_n$, $t_2 = (n-1)\, t_n \ldots t_{n-1} = 2\, t_n$, $t_n = t_n$

Nous nous intéressons au passé d'exécution U_i $1 \leq i \leq n$. Avant toutes exécutions des processus sont initialisés par la valeur zéro.

Après la première exécution des processus le passé d'exécution de chaque processus a augmenté par sa valeur de temps d'exécution

$U_1 \geq U_2 \geq \ldots \geq U_{n-1} \geq U_n \Leftrightarrow t_1 \geq t_2 \geq \ldots \geq t_{n-1} \geq t_n$ le remplacement de valeur de temps d'exécution de chaque processus nous a donné

$U_1 \geq U_2 \geq \ldots \geq U_{n-1} \geq U_n \Leftrightarrow n\, t_n \geq (n-1)\, t_n \geq \cdots \geq 2\, t_n \geq t_n$.

Il est clair que les couples de premier échange entre les processus sont : (τ_1, τ_n), (τ_2, τ_{n-1}), (τ_3, τ_{n-2}),....

De premier échange on a obtenu :

$U_1 \leq U_2 \leq \ldots \leq U_{n-1} \leq U_n \Leftrightarrow t_n \leq 2\, t_n \leq \cdots \leq (n-1)\, t_n \leq n\, t_n$ ..... **(5)**

Après la deuxième exécution des processus, le passé d'exécution de chaque processus a augmenté par sa valeur de temps d'exécution.

Une nouvelle variation de passé d'exécution s'est produite:

$U_1 \leq U_2 \leq \ldots \leq U_{n-1} \leq U_n \Leftrightarrow t_n + t_1 \leq 2\, t_n + t_2 \leq \cdots (n-1)\, t_n + t_{n-1} \leq n\, t_n + t_n$

Remplaçons la valeur de chaque t_i :

$U_1 \leq U_2 \leq \ldots \leq U_{n-1} \leq U_n \Leftrightarrow$

$t_n + n\, t_n \leq 2\, t_n + (n-1)\, t_n \leq \cdots \leq (n-1)\, t_n + 2\, t_n \leq t_n + n\, t_n$

$\Leftrightarrow (n+1)\,t_n = (n+1)\,t_n = \cdots = (n+1)\,t_n = (n+1)t_n$

Les processus ont le même passé d'exécution, il n'y aura aucun échange de passé d'exécution.

À l'itération suivante on obtient :

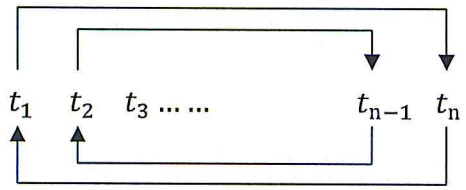
$U_1 \geq U_2 \geq \ldots \geq U_{n-1} \geq U_n \Leftrightarrow$

$(n+1)\,t_n + t_1 \geq (n+1)\,t_n + t_2 \geq \cdots \geq (n+1)\,t_n + t_{n-1} \geq (n+1)t_n + t_n$

On voit clairement qu'on a abouti à l'expression (5) plus une constate dépend de t_n et nombre de processus valant $(n+1)\,t_n$

Si les t_i forment une suite arithmétique de raison t_n, on peut déterminer à chaque itération l'ordre de priorité ainsi les couples d'échange de temps d'exécution.

L'échange continue de la façon suivante :



Ou bien les couples d'échange seront : (τ_1, τ_n), (τ_2, τ_{n-1}), (τ_3, τ_{n-2}),....

A la 1^{er} itération on a fait l'échange entre les éléments des couples (τ_1, τ_n), (τ_2, τ_{n-1}), (τ_3, τ_{n-2}),....

On a obtenu :$P_1 \geq P_2 \geq \ldots\ldots \geq P_{n-1} \geq P_n \Leftrightarrow$

$M - \Delta P\left(\frac{t_n}{t}\right) \geq M - \Delta P\left(\frac{t_{(n-1)}}{t}\right) \geq \ldots \geq M - \Delta P\left(\frac{t_2}{t}\right) \geq M - \Delta P\left(\frac{t_1}{t}\right)$

Remplaçons les valeurs des t_i :

$M - \Delta P\left(\frac{t_n}{t}\right) \geq M - \Delta P\left(\frac{2t_n}{t}\right) \geq \ldots \geq M - \Delta P\left(\frac{(n-1)t_n}{t}\right) \geq M - \Delta P\left(\frac{nt_n}{t}\right)$

Puis, des nouvelles valeurs de priorités se produisent, maintenant d'ordre connu:

$P_1 = P_2 = \ldots\ldots = P_{n-1} = P_n \Leftrightarrow$

$M - \Delta P\left(\frac{t_n+t_1}{t}\right) = M - \Delta P\left(\frac{t_{(n-1)}+t_2}{t}\right) = \cdots = M - \Delta P\left(\frac{t_2+t_{(n-1)}}{t}\right) = M - \Delta P\left(\frac{t_1+t_n}{t}\right)$

L'égalité de tous les termes est observée après avoir remplacé chaque t_i par sa valeur

$$M - \Delta P\left(\frac{t_n + nt_n}{t}\right) = M - \Delta P\left(\frac{2t_n + (n-1)t_n}{t}\right) = \cdots = M - \Delta P\left(\frac{(n-1)t_n + 2t_n}{t}\right) = M - \Delta P\left(\frac{nt_n + t_n}{t}\right)$$

$\Leftrightarrow$

$$M - \Delta P\left(\frac{t_n + nt_n}{t}\right) = M - \Delta P\left(\frac{2t_n + (n-1)t_n}{t}\right) = \cdots = M - \Delta P\left(\frac{(n-1)t_n + 2t_n}{t}\right) = M - \Delta P\left(\frac{nt_n + t_n}{t}\right)$$

$\Leftrightarrow$

$$M - \Delta P\left(\frac{(n+1)t_n}{t}\right) = M - \Delta P\left(\frac{(n+1)t_n}{t}\right) = \cdots = M - \Delta P\left(\frac{(n+1)t_n}{t}\right) = M - \Delta P\left(\frac{(n+1)t_n}{t}\right)$$

Les priorités sont égales, le passé d'exécution de chaque processus reste inchangeable.

A la 2^{eme} itération on a : $t = 2\left(\sum_{i=1}^{n} t_i\right)$

$P_1 = P_2 = P_3 = \ldots\ldots = P_n \Leftrightarrow$

$$M - \Delta P\left(\frac{(n+1)t_n}{t}\right) = M - \Delta P\left(\frac{(n+1)t_n}{t}\right) = \cdots = M - \Delta P\left(\frac{(n+1)t_n}{t}\right) = M - \Delta P\left(\frac{(n+1)t_n}{t}\right)$$

Après l'exécution des processus on aura une augmentation au passé d'exécution de chaque processus par sa valeur d'exécution

$P_1 \leq P_2 \leq P_3 \leq \ldots\ldots \leq P_n \Leftrightarrow$

$$M - \Delta P\left(\frac{t_n + 2t_1}{t}\right) \leq M - \Delta P\left(\frac{t_{(n-1)} + 2t_2}{t}\right) \leq \cdots \leq M - \Delta P\left(\frac{t_2 + 2t_{(n-1)}}{t}\right) \leq M - \Delta P\left(\frac{t_1 + 2t_n}{t}\right)$$

L'échange du passé de temps d'exécution sera entre les éléments des couples (τ_1, τ_n), (τ_2, τ_{n-1}), (τ_3, τ_{n-2}),....

Donc à la 3^{eme} itération on obtient un temps de calcul t valant 3 $\left(\sum_{i=1}^{n} t_i\right)$ est une nouvelle évolution de priorités

$P_1 \geq P_2 \geq P_3 \geq \ldots\ldots \geq P_n \Leftrightarrow$

$$M - \Delta P\left(\frac{t_1 + 2t_n}{t}\right) \geq M - \Delta P\left(\frac{t_2 + 2t_{(n-1)}}{t}\right) \geq \cdots \geq M - \Delta P\left(\frac{t_{(n-1)} + 2t_2}{t}\right) \geq M - \Delta P\left(\frac{t_n + 2t_1}{t}\right)$$

Après l'exécution de tâches on obtiendra :

$P_1 = P_2 = P_3 = \ldots\ldots = P_n \Leftrightarrow$

$$M - \Delta P\left(\frac{2t_n+2t_1}{t}\right) = M - \Delta P\left(\frac{2t_{(n-1)}+2t_2}{t}\right) = \cdots = M - \Delta P\left(\frac{2t_2+2t_{(n-1)}}{t}\right) = M - \Delta P\left(\frac{2t_1+2t_n}{t}\right)$$

Donc à la 4^{eme} itération on aura : $t = 4\ (\sum_{i=1}^{n} t_i)$

$P_1 = P_2 = P_3 = \ldots\ldots = P_n \Leftrightarrow$

$$M - \Delta P\left(\frac{2t_n+2t_1}{t}\right) = M - \Delta P\left(\frac{2t_{(n-1)}+2t_2}{t}\right) = \cdots = M - \Delta P\left(\frac{2t_2+2t_{(n-1)}}{t}\right) = M - \Delta P\left(\frac{2t_1+2t_n}{t}\right)$$

A la k^{eme}itération : On aura un temps de calcul vaut $t = k\ (\sum_{i=1}^{n} t_i) = k\left(\frac{n}{2}\right).(t_1 + t_n)$, et une évolution de priorités classée en deux cas :

<u>Si k est pair</u> $P_1 = P_2 = P_3 = \ldots\ldots = P_n \Leftrightarrow$

$$M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(t_1+t_n)}{t}\right) = M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(t_2+t_{(n-1)})}{t}\right) = \cdots = M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(t_{(n-1)}+t_2)}{t}\right)$$

$$= M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(t_1+t_n)}{t}\right) \qquad \ldots\ldots (6)$$

Remplaçons les différents t_i et le temps de calcul t, on obtient :

$$P_1 = P_2 = \ldots = P_{n-1} = P_n \Leftrightarrow M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(n\ t_n+t_n)}{k\left(\frac{n}{2}\right).(t_1+t_n)}\right) = M - \Delta P\left(\frac{\left(\frac{k}{2}\right)((n-1)\ t_n+2\ t_n)}{k\left(\frac{n}{2}\right).(t_1+t_n)}\right) = \cdots$$

$$\ldots = M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(2\ t_n+(n-1)\ t_n)}{k\left(\frac{n}{2}\right).(t_1+t_n)}\right) = M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(n\ t_n+t_n)}{k\left(\frac{n}{2}\right).(t_1+t_n)}\right)$$

$\Leftrightarrow$

$$M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(n+1)t_n}{k\left(\frac{n}{2}\right).(n+1)t_n}\right) = M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(n+1)t_n}{k\left(\frac{n}{2}\right)(n+1)t_n}\right) = \cdots$$

$$\ldots = M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(n+1)t_n}{k\left(\frac{n}{2}\right)(n+1)t_n}\right) = M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(n+1)t_n}{k\left(\frac{n}{2}\right)(n+1)t_n}\right)$$

$\Rightarrow$ Tout terme égal à la valeur :

$$M - \Delta P\left(\frac{\left(\frac{k}{2}\right)}{n\left(\frac{k}{2}\right)}\right) = M - \Delta P\left(\frac{1}{n}\right) = M - \left(\frac{\Delta P}{n}\right)$$ C'est un nœud.

<u>Si k est impair</u>

$P_1 \geq P_2 \geq \ldots\ldots \geq P_{n-1} \geq P_n \Leftrightarrow$

$$M - \Delta P\left(\frac{\left(\frac{k+1}{2}\right)t_n + \left(\frac{k-1}{2}\right)t_1}{t}\right) \geq M - \Delta P\left(\frac{\left(\frac{k+1}{2}\right)t_{(n-1)} + \left(\frac{k-1}{2}\right)t_2}{t}\right) \geq$$

$$\ldots. \geq M - \Delta P\left(\frac{\left(\frac{k+1}{2}\right)t_2 + \left(\frac{k-1}{2}\right)t_{(n-1)}}{t}\right) \geq M - \Delta P\left(\frac{\left(\frac{k+1}{2}\right)t_1 + \left(\frac{k-1}{2}\right)t_n}{t}\right) \ldots\ldots\ldots (7)$$

Remplaçons les différents t_i : $P_1 \geq P_2 \geq \ldots \geq P_{n-1} \geq P_n \Leftrightarrow$

$$\Leftrightarrow M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(n+1)t_n}{k\left(\frac{n}{2}\right)(n+1)t_n} + \frac{\frac{1}{2}(1-n)t_n}{k\left(\frac{n}{2}\right)(n+1)t_n}\right) \geq M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(n+1)t_n}{k\left(\frac{n}{2}\right)(n+1)t_n} + \frac{\frac{1}{2}(3-n)t_n}{k\left(\frac{n}{2}\right)(n+1)t_n}\right) \geq$$

$$\ldots \geq M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(n+1)t_n}{k\left(\frac{n}{2}\right)(n+1)t_n} + \frac{\frac{1}{2}(n-3)t_n}{k\left(\frac{n}{2}\right)(n+1)t_n}\right) \geq M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(n+1)t_n}{k\left(\frac{n}{2}\right)(n+1)t_n} + \frac{\frac{1}{2}(n-1)t_n}{k\left(\frac{n}{2}\right)(n+1)t_n}\right)$$

$$\Leftrightarrow P_1 \geq P_2 \geq P_3 \geq P_4 \ldots\ldots P_{n-3} \geq P_{n-2} \geq P_{n-1} \geq P_n \Leftrightarrow$$

$$M - \Delta P\left(\frac{1}{n} + \frac{(1-n)}{k\,n\,(n+1)}\right) \geq M - \Delta P\left(\frac{1}{n} + \frac{(3-n)}{k\,n(n+1)}\right) \geq M - \Delta P\left(\frac{1}{n} + \frac{(5-n)}{k\,n(n+1)}\right)$$

$$\geq M - \Delta P\left(\frac{1}{n} + \frac{(7-n)}{k\,n(n+1)}\right) \geq \cdots$$

$$\ldots \geq M - \Delta P\left(\frac{1}{n} + \frac{(n-7)}{k\,n(n+1)}\right) \geq M - \Delta P\left(\frac{1}{n} + \frac{(n-5)}{k\,n(n+1)}\right) \geq M - \Delta P\left(\frac{1}{n} + \frac{(n-3)}{k\,n(n+1)}\right)$$

$$\geq M - \Delta P\left(\frac{1}{n} + \frac{(n-1)}{k\,n(n+1)}\right) \qquad \ldots\ldots (8)$$

D'après **(6)** et **(8)** on peut calculer tout les termes à chaque itérations, et lorsque k tend vers l'infini l'équité est satisfaite puisque l'équité de la formule **(6)** a bien été vérifiée et pour la **(8)** on voit que toutes les P_i restreignent entre le premier terme et le dernier donc :

$$M - \Delta P\left(\frac{1}{n} + \frac{(1-n)}{k\,n\,(n+1)}\right) \geq P_i \geq M - \Delta P\left(\frac{1}{n} - \frac{(n-1)}{k\,n(n+1)}\right), \forall i = 1, n$$

Alors lorsque k tend vers l'infini on aura :

$$\lim_{k\to\infty}\left(M - \Delta P\left(\frac{1}{n} + \frac{(1-n)}{k\,n\,(n+1)}\right)\right) \geq \lim_{k\to\infty} P_i \geq \lim_{k\to\infty}\left(M - \Delta P\left(\frac{1}{n} - \frac{(n-1)}{k\,n(n+1)}\right)\right)$$

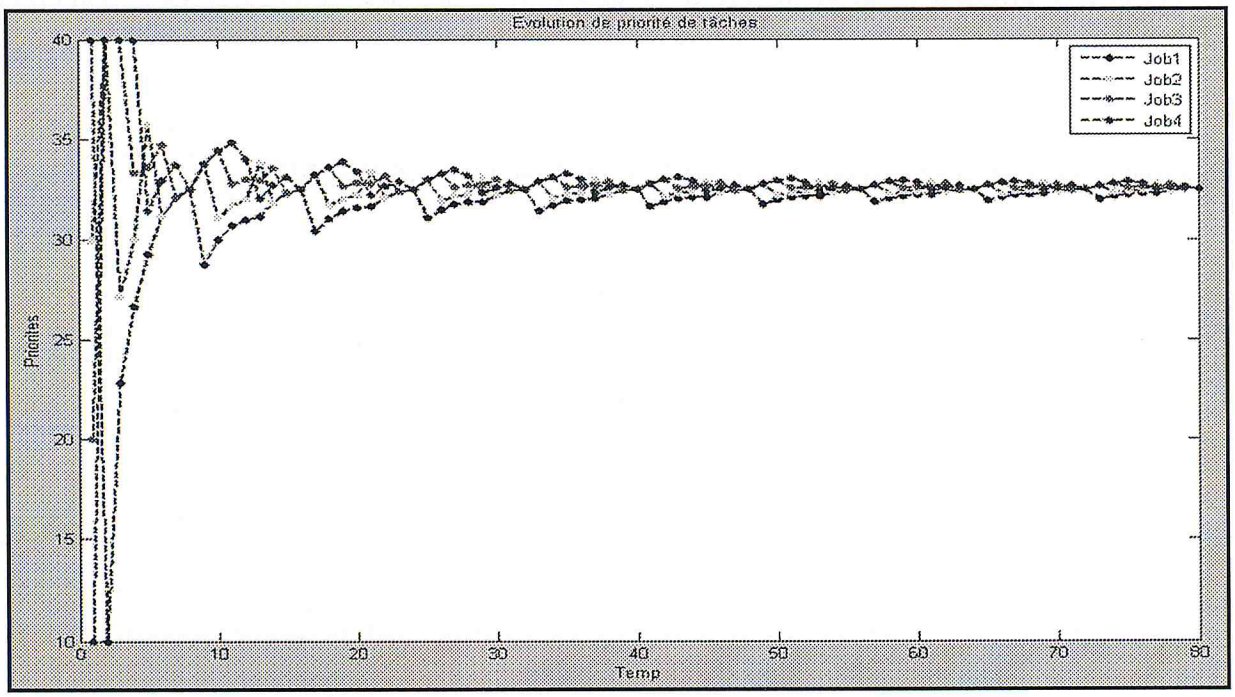
$$M - \frac{\Delta P}{n} \geq \lim_{k\to\infty} P_i \geq M - \frac{\Delta P}{n}$$

Donc : $\lim_{k\to\infty}(P_i) = M - \frac{\Delta P}{n}$

D'où: si les t_i forment une suite arithmétique l'équité est bien vérifiée.

Par exemple : Nombre de tâches est 4 et M = 40, m = 10 ,et les temps d'exécutions formant une suite arithmétique sont : $t_1 = 40, t_2 = 30, t_3 = 20, t_4 = 10.$

La valeur d'équité est $\mathrm{M} - \frac{\Delta \mathrm{P}}{\mathrm{n}} = 40 - \frac{30}{4} = 32{,}5$



Graphique B.3 : Evolution de priorité des processus.

On voit d'après **(8)** que si les t_i^e forment une suite arithmétique de raison t_n l'évolution de priorité des processus à chaque itération est indépendante de temps d'exécution des processus, Donc on peut extraire une formule générale de valeurs des priorités à chaque itération $k > 0$:

$$\forall k > 0, \forall i, 1 \leq i \leq n : P_i(kt) = \begin{cases} \mathrm{M} - \Delta \mathrm{P} \left(\frac{1}{\mathrm{n}} + \frac{2\mathrm{i} - (\mathrm{n}+1)}{k\, n(n+1)}\right) si\ k\ impair \\ \mathrm{M} - \left(\frac{\Delta \mathrm{P}}{\mathrm{n}}\right) si\ k\ pair \end{cases}$$

Avec $t = \sum_{i=1}^{n} t_i^e$

Remarque :

Si les temps d'exécutions des processus sont unitaires valant une constante c alors on aura :

(6) $\Leftrightarrow P_1 = P_2 = P_3 = \ldots\ldots = P_\mathrm{n} \Leftrightarrow$

$$M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(2c)}{t}\right) = \cdots = M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(2c)}{t}\right) \quad \text{Avec } t = k.n.c$$

$$\Leftrightarrow M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(2c)}{k.n.c}\right) = \cdots = M - \Delta P\left(\frac{\left(\frac{k}{2}\right)(2c)}{k.n.c}\right) \Rightarrow P_i = M - \left(\frac{\Delta P}{n}\right) \forall\, i = 1..n$$

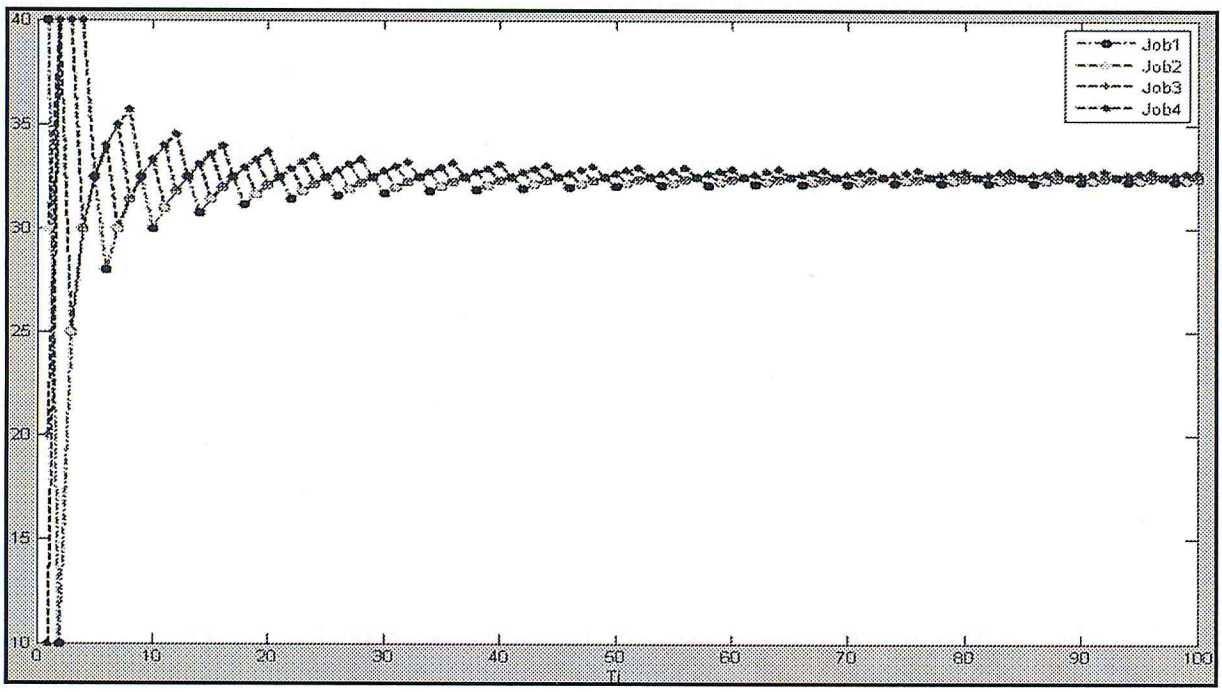
$$(7) \Leftrightarrow P_1 \geq \ \ldots \geq P_{\text{milieu}} \geq \ldots \geq P_n \Leftrightarrow$$

$$M - \Delta P\left(\frac{\left(\frac{k+1}{2}\right)c + \left(\frac{k-1}{2}\right)c}{k\left(\frac{n}{2}\right).(2c)}\right) \geq \cdots \geq M - \Delta P\left(\frac{k.c}{k\left(\frac{n}{2}\right).(2c)}\right) \geq \cdots \geq \ M - \Delta P\left(\frac{\left(\frac{k+1}{2}\right)c + \left(\frac{k+1}{2}\right)c}{k\left(\frac{n}{2}\right).(2c)}\right)$$

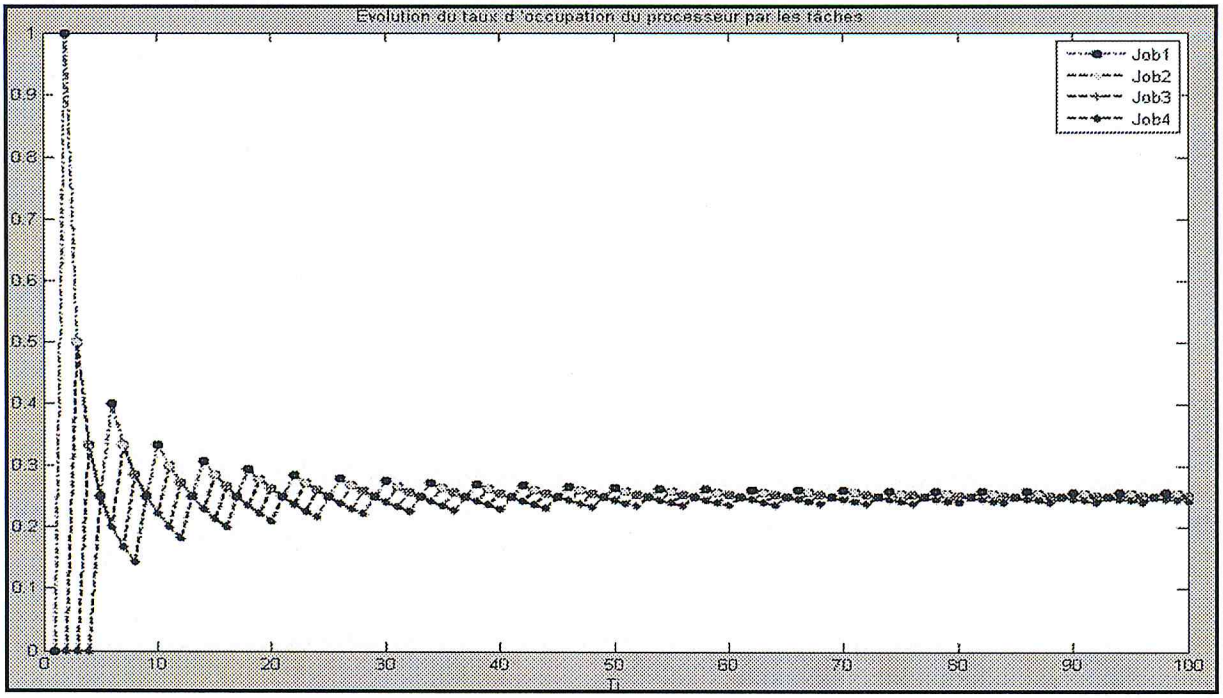
$$\Leftrightarrow M - \Delta P\left(\frac{k.c}{k\left(\frac{n}{2}\right).(2c)}\right) \geq \cdots \geq M - \Delta P\left(\frac{k.c}{k\left(\frac{n}{2}\right).(2c)}\right) \geq \cdots \geq \ M - \Delta P\left(\frac{k.c}{k\left(\frac{n}{2}\right).(2c)}\right).$$

$$\Leftrightarrow M - \Delta P\left(\frac{1}{n}\right) \geq \cdots \geq M - \Delta P\left(\frac{1}{n}\right) \Rightarrow \forall\, i = 1..n\ P_i = M - \left(\frac{\Delta P}{n}\right)$$

Si les temps d'exécution des processus sont unitaires on aura un nœud à chaque itération. On attribue ce comportement à le fait que les processus avaient le même passé à chaque itération donc il n'y aura aucun échange. Les graphiques ci-dessous montrent l'évolution de priorité ainsi le taux d'occupation lorsqu'on prend les temps d'exécutions unitaires.



Graphique B.4 : Evolution de priorité lorsqu'on a pris les temps d'exécutions unitaires.

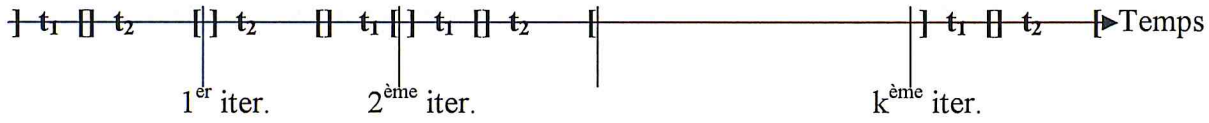


Graphique B.5 : Evolution du taux d'occupation du processeur par les processus.

4. Analogie entre l'OPBM et la version non préemptive modifiée :

L'ordonnancement OPBM-étendu s'interprète comme un problème d'ordonnancement préemptif avec un temps d'exécution de processus reste inchangeable à chaque période d'ordonnancement.

t_1, t_2 les temps d'exécution de deux processus τ_1, τ_2, s'exécutent avec les mêmes durées à chaque itération, mais d'ordre différent. Cet ordre devient négligeable quand les temps d'exécutions sont unitaires.



Attribuer le même concept d'OPBM-étendu à l'OPBM dans les cas préemptif nous amène à accepter qu'un processus s'exécute avec l'unité du temps à chaque itération, et que le temps de calcul augmente à chaque itération nettement par la quantité $\sum t_i$ et cela veut dire que tous les processus sont exécutés entre l'itération k et (k+1) une fois seulement.

ANNEXE-C

Implémentation de l'OPBM-étendu

Les étapes nécessaires pour réaliser l'implémentation de l'OPBM-étendu sont indiquées ci-dessous. Une déclaration de :

- $P_i(0)$, un vecteur de taille n contenant les priorités initiales
- t_i^e, un vecteur de taille n contenant les temps d'exécution des tâches.
- $U_i^e(0)$, un vecteurd'historique de temps d'exécution où tous les éléments sont initialisés à la valeur zéro.
- Un vecteur $\text{Max}_{\text{conflit}} = \{i/P_i(t) \geq P_1(t), \forall i \neq 1\}$ contenant les tâches prioritaires en conflit à un instant t.

Pour déterminer les tâches prioritaires en conflit, la technique consiste à toujours comparer les priorités des (n-1) tâches restantes à la priorité de la première tâche. Les valeurs des priorités initiales $P_i(0)$sont à ordonner dans un sens décroissant en conservant les positions des tâches dans l'ordonnancement actuel. Les indices de positions des tâches formeront un vecteur.

1. initialiser à zéro le nombre de périodes d'ordonnancement j.
2. Déterminer la cardinalité de l'ensemble des tâches prioritaires en conflit noté $\text{Max}_{\text{conflit}} = \{i/P_i(t) \geq P_1(t), \forall i \neq 1\}$
3. Si $\text{card}(\text{Max}_{\text{conflit}}) \geq 1$ alors, appliquer la règle Round-Robin ci-dessous.

```
Soit un entier i = 0 ;
Si i ∈ Max_conflit
Q = t_i^e ;
U_i += Q ;
t += Q ;

Sinon i += 1;
```

4. Au cas où $P_1(t)$ est la plus grande, la tâche une est la prioritaire et $\text{card}(\text{Max}_{\text{conflit}}) = 0$. Aucun conflit ne surviendra entre les tâches. L'historique d'exécution de la première tâche $U_1(j)$ augmente d'une valeur égale à Ts.

$Ts = t_{élu}^e$;

$U_{élu} += Ts$;

$t \mathrel{+}= Ts$;

$\text{Tâche}_{\text{élu}}$ c'est la tâche correspond à la période d'ordonnancement.

5. Calculer les nouvelles priorités à $t+$ par :$P_i(t) = M - \Delta P * \frac{U_i}{t}$
6. Placer la tâche prioritaire T_p à la première position de la séquences(k).
7. Déterminer la période d'ordonnancement suivante $j = \text{Max}\ \{P_i(t), \forall i = 1 \ldots n\}$
8. Déterminer la cardinalité de l'ensemble des tâches prioritaires en conflit $\text{Max}_{\text{conflit}} = \{i/P_i(t) \geq P_{T_p}(t), \forall i \neq p\}$. Allez à l'étape3.
9. Quand la taille de $s(k)$ devient n et pour faire l'échange, les priorités actuelles doivent être gardées dans un vecteur copie.
10. Une méthode pour faire l'échange des temps d'exécution de tâches, échange défini auparavant au paragraphe 3 du chapitre 4, est définie par le code source suivant :

```
vector<int>TposTabou; //pour éviter les position déjà choisies
vector<int> TposEchange1;//premier terme du couple
vector<int> TposEchange2;//deuxième terme du couple
vector<double>Priorite; //contient une copie de priorités
                        //obtenue à la fin d'exécution de tâches
Do{
double _Min = 99999; // pour trouver le minimum
double _Max = 0;     // pour trouver le maximum
int _PosMin = 0;        // pour trouver la position de minimum
int _PosMax = 0;        // pour trouver la position de maximum

for(inti = 0 ; i<nombre de tâches ; i++){
if((_Min >Priorite [i]) && (Priorite [i] >= 0)){_Min = Priorite
[i],_PosMin = i;}}

for(inti = 0 ; i<nombre de tâches ; i++){
if((_Max <Priorite [i]) && (Priorite [i] >= 0)){_Max = Priorite
[i],_PosMax = i;}}

TposEchange1.insert(TposEchange1.end(),_PosMin);
TposEchange2.insert(TposEchange2.end(),_PosMax);

TposTabou.insert(TposTabou.end(),_PosMin);
TposTabou.insert(TposTabou.end(),_PosMax);

for(inti = 0 ; i<TposTabou.size() ; i++){
Priorite[TposTabou[i]]=-1;              }

}while(TposTabou.size()<nombre de tâches);

//Texe[i] : vecteur contenant l'historique de temps
//d'exécution de la tâche i.
// Variable : variable pour faire l'échange.

for(inti = 0 ; i< TposEchange1.size() ; i++){

variable = Texe[TposEchange1[i]];
```

```
Texe[TposEchange1[i]] = Texe[TposEchange2[i]];
Texe[TposEchange2[i]] =  variable;}
```

---

11. Réordonner la séquence obtenue selon l'ordre initial des priorités pour calculer le C_{max} par application du code suivant.

```
//sequence1:la séquence obtenue
//sequence2:la séquence issue de l'ordre initiale de priorité
//Initiale avant l'ordonner d'ordre croissant.
//séquence: la séquence finale
for(inti=0 ; i< n; i++ ){
Sequence.insert(Sequence.end(),sequence2[sequence1[i]]); }
```

---

12. Garder le C_{max} minimal à chaque itération. Allez à l'étape 1.
13. Le fonctionnement de l'algorithme s'arrête quand le nombre d'itérations maximum ou un temps de calcul *t* sont atteints ou lorsque les priorités ont les mêmes valeurs.

Pour garantir le fonctionnement voulu de l'OPBM-étendu, l'évolution de priorité doit être adéquate avec les formules de priorités. Afin d'éviter d'alourdir ce texte, ces formules de priorités sont indiquées en détails dans l'Annexe-B jointe à ce mémoire.

5.4.2 Exemple

Soit à exécuter neuf tâches sur deux machines à un seul convoyeur. Les données de ce problème sont récapitulées dans ce tableau.

JOB	M1	M2	TempsTransp.
1	M1 3	M2 6	2
2	M1 5	M2 2	8
3	M1 1	M2 2	3
4	M1 6	M2 6	6
5	M1 7	M2 5	8
6	M2 4	M1 2	1
7	M2 5	-- -	2
8	M2 4	M1 3	5
9	M2 2	M1 7	6

le temps de retour a vide : 10

Tableau 9 : données de l'exemple

Les priorités initiales de tâches de la première station sont : 8, 13, 5, 12, 15.

La valeur d'équité des priorités pour la première machine est : $15 - \frac{(15-5)}{5} = 13$

Les priorités initiales de tâches de la deuxième station sont : 3, 2, 8, 8.

La valeur d'équité des priorités pour la seconde machine est : $8 - \frac{(8-2)}{4} = 6.5$

Au bout de huit itérations, l'ordonnancement optimal s'obtient et est de $C_{max} = 55$ unités.

Sur la machine A : T_4, T_2, T_5, T_1, T_3

Sur la machine B : T_9, T_8, T_7, T_6

Le graphe ci-dessous représente l'évolution des priorités. A chaque station, une asymptotique convergence des priorités est constatée.

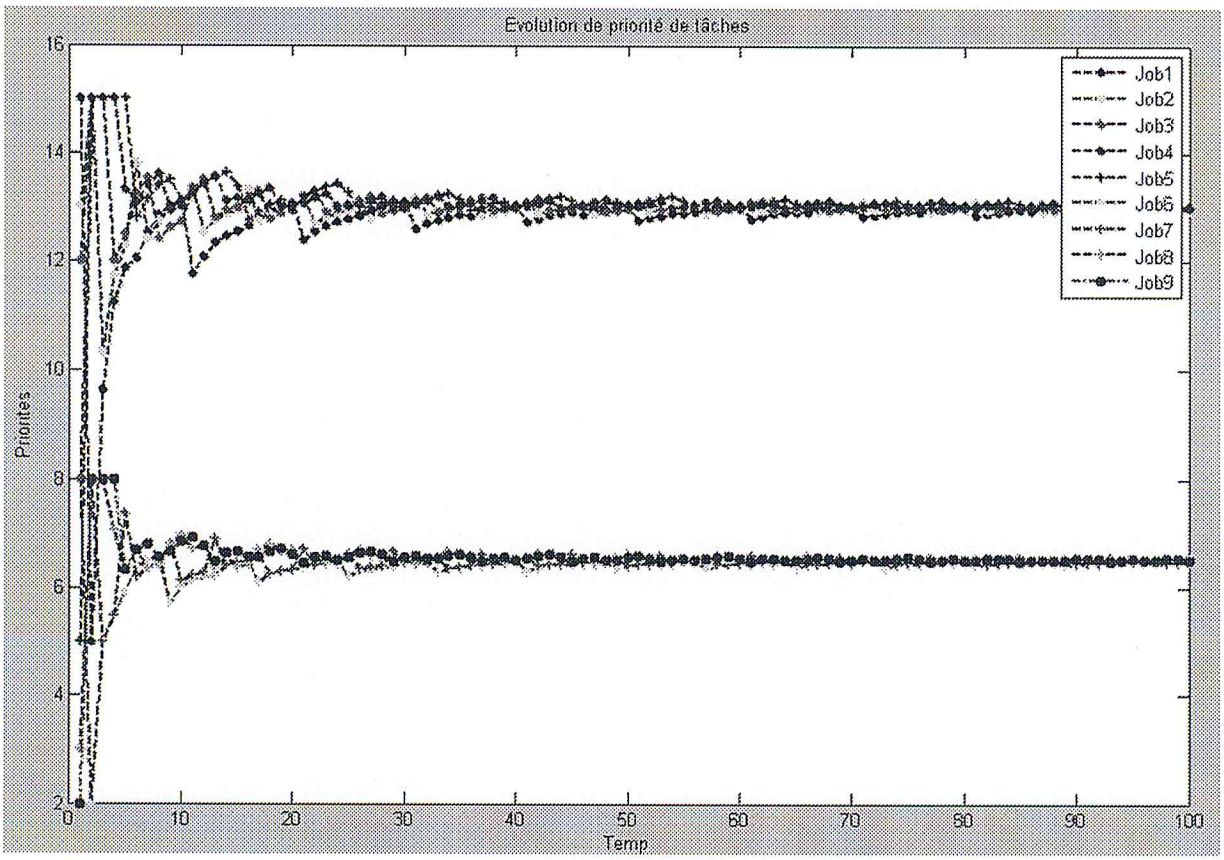


Figure 19: Evolution de priorités des tâches des deux stations