République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la
Recherche Scientifique

Ecole Nationale Polytechnique

Département de Génie Industriel

Mémoire du Projet de Fin d'Etudes d'Ingénieur

Thème

Développement d'une approche heuristique pour la résolution d'un problème d'ordonnancement en temps-réel sur des ressources identiques en parallèle

Réalisé par:

M[lle] ARBIA Amina

Dirigé par :

M. BAKALEM

Promotion : juin 2010

Dédicaces

A mes parents,

A mon frère et à ma sœur,

A tous ceux qui me sont chers et à qui je suis chère.

Remerciements :

Je remercie avant tout Dieu, le Tout Puissant, pour m'avoir donné, le courage, la patience, la volonté et la force nécessaire, pour affronter toutes les difficultés et les obstacles qui se sont hissées au travers de mon chemin, tout au long de ce travail.

Je tiens à adresser mes remerciements à l'ensemble des enseignants qui m'ont assistée pour que ce projet de fin d'études soit fructueux et profitable.

C'est avec une attention particulière que je remercie Mr BAKALEM pour bien avoir voulu encadrer mon projet, pour son aide et les renseignements précieux qu'il m'a fourni.

Merci à Melle BOUGCHICHE pour ses conseils précieux et ses encouragements.

Je tiens à exprimer ma profonde affection à ma mère pour sa patience et son grand soutien, durant toutes ces années d'études.

Je voudrais également remercier toutes les personnes qui par leur aide ou simplement par leur gentillesse ont contribué d'une manière ou d'une autre à la bonne réalisation de ce projet.

ملخص

هذا العمل يهتم بدراسة ا شكا لية جدولة الأشغال في زمن واقعي على وسيلتين متماثلتين على التوازي. الغاية هي تخفيض مدة البرمجة الكلية و زيادة عدد الاشغال الغير متوقعة في البرمجة .

نقترح في هذه الدراسة طريقة تعتمد علي التوزيع اين كل وسيلة تسير جدولتها الخاصة (جدولة بالة واحدة) الجدولة المحلية تتمتع بمرونة في المقاطع ,هذه المرونة تسمح بمواجهة وصول اعمال جديدة في الزمن الواقعي .هذه المرونة متولدة عن نظرية الهيمنة ,و هي نظرية اللاهرامات.

نظرا للصعوبة الشديدة للمشكلة, اخترنا مقاربة تجريبية محضة مستعملين تقنيات التصنع لتقييم الفعالية.

المفاتيح:
جدولة بالتوزيع, زمن واقعي, وسائل متماثلة على التوازي, المرونة في المقاطع, نظرية الأهرام, تقنيات التصنع, مقاربة

Résumé :

Ce travail s'intéresse au problème d'ordonnancement en temps réel sur deux ressources identiques en parallèles. L'objectif est de minimiser la durée totale d'ordonnancement, Makespan, ainsi que la maximisation du nombre de tâches aléatoires exécutées.

Nous proposons dans ce travail une approche de résolution distribuée ou Chaque ressource gère son propre ordonnancement local (ordonnancement à une machine). Les ordonnancements locaux sont des ordonnancements incorporant de la flexibilité séquentielle, cette flexibilité permet de faire face à l'arrivée aléatoire en temps réel d'autres taches. Cette flexibilité séquentielle est engendrée par un théorème de dominance, qu'est le théorème des pyramides.

Du fait du caractère NP-difficile du problème, nous avons opté pour une approche empirique moyennant la simulation afin d'évaluer la performance.

Mots clés : *Ordonnancement distribué, Ordonnancement temps réel, Ressources Identiques en Parallèle, flexibilité séquentielle, théorème des pyramides, Makespan, Simulation.*

Abstract:

In this study we focused on the scheduling problem in real time on two identical parallel resources. The purpose is to minimize the scheduling total duration "Makespan", and to maximize the executed random tasks number.

We propose in this study a distributed solving approach where each machine manages its own local scheduling. The local schedulings includes à sequential flexibility, this last will permit to schedule jobs that acquires in real time. This sequential flexibility is generated by a theorem of dominance, which is the theorem of the pyramids.

Because of the NP-hard feature problem, we opted for an empirical approach through the simulation to assess performance.

Keywords: *Distribute Scheduling, Simulation, collaborative approach, multi agent systems, Communication Protocol, On-line Scheduling, Parallel Identical Resources, Makespan.*

Table des matières :

Chapitre IV : Simulation et analyses des résultats.

Liste des figures :

Chapitre I :

Chapitre II :

Chapitre III :

Chapitre IV :

Liste des tableaux :

Chapitre II :

Chapitre III :

Chapitre IV :

Liste des abréviations:

SPT: Shortest Processing Time

LPT: Longest Processing Time

FIFO: First in First out

EDD: Earliest deadline

PSE : Procédures par Séparation et Évaluation

PL : La Programmation Linéaire

PLNE : La Programmation Linéaire en nombres entiers

PD : La Programmation Dynamique

NP: Non polynomial

IAD : Intelligence Artificielle Distribuée

SMA : Systèmes Multi-Agents

SdP: Système de production

VBA: Visual Basic for Application

BS: Borne supérieure

FATCI : File d'attente des tâches candidates à l'insertion

PERT : Program Evaluation and Revue Technique

CPM : Critical Path Method

MPM : Méthode des Potentiel Métra

Liste des symboles :

Symboles	Description
r_i	La date de disponibilité de la tâche i
p_i	La durée opératoire de la tâche i
d_i	La date échue de la tâche i
ta_i	Le début au plus tôt de la tâche i
tb_i	Le début au plus tard de la tâche i
$C_{\max}$	La durée totale de l'ordonnancement
$F_{\max}$	La plus grande durée de séjour
$L_{\max}$	Le plus grand retard algébrique
$T_{\max}$	Le plus grand retard vrai
$\overline{F}$	La durée moyenne de séjour
$\overline{T}$	Le retard moyen
S_{dom}	L'ensemble des séquences dominantes
S_α	Un sommet
P_α	Une pyramide
$NQ(i)$	L'état de la file d'attente numéro i
$Pmtn$	les opérations peuvent être interrompues
$\Pr ec$	il existe des contraintes de précédence
T_{NOW}	Run current time

Introduction générale et problématique :

On rencontre les problèmes d'ordonnancement dans des domaines aussi variés que l'organisation opérationnelle du travail ; dans les usines, dans la planification des grands projets, dans l'organisation d'activités de service et dans l'allocation dynamique des ressources dans les systèmes d'exploitation d'ordinateurs. L'étude des problèmes d'ordonnancement est également d'un intérêt théorique toujours renouvelé pour les chercheurs, du fait de leur caractère combinatoire leur attribuant le plus souvent un statut de problèmes NP-difficile.

Les méthodes d'ordonnancement se différencient par la nature du problème considéré (nombre de ressources, structure particulière du problème), la nature des contraintes prises en compte, les objectifs à satisfaire (minimisation des coûts, de la durée totale de mise en œuvre) et la nature de l'approche de résolution adoptée (heuristiques, méthodes exactes).

La plupart de ces méthodes d'ordonnancement opèrent sous l'hypothèse classique que les données du problème d'ordonnancement sont parfaitement connues : l'ensemble des tâches à ordonnancer est déterminé a priori, les durées opératoires sont fixes et stables et les capacités des ressources sont connues à tout instant. Cette hypothèse permet de traiter un problème de façon déterministe, hors, les ateliers de production sont soumis à un certain nombre d'incertitudes, qui dégradent les performances prédites par les méthodes déterministes : les arrivées aléatoires de nouvelles tâches, les modifications de dates de disponibilité et de dates échues, les indisponibilités des ressources, etc.

Compte tenu du caractère incertain de l'environnement d'une entreprise, la probabilité qu'un ordonnancement prévisionnel soit exécuté comme prévu est faible **[ESS 03].** Les notions de flexibilité et de robustesse sont alors définies pour caractériser un système d'ordonnancement capable de faire face aux incertitudes. La flexibilité fait référence à la liberté décisionnelle dont on dispose durant la phase d'exploitation d'un ordonnancement prévisionnel. La notion de robustesse est étroitement liée à celle de flexibilité, en effet, de la flexibilité est souvent injectée dans une solution déterministe afin de la rendre robuste.

L'anticipation des perturbations consiste à déterminer un ensemble flexible de solutions. La flexibilité est séquentielle, c'est à dire que l'ordre des tâches sur chaque ressource est différent d'une solution à l'autre de l'ensemble. Ce type de flexibilité est intéressant car il permet, lorsqu'une ressource se libère, de disposer de plusieurs choix quant à la prochaine tâche à traiter.

Le champ d'application de l'ordonnancement considéré, dans le cadre de ce travail, est celui des systèmes d'ordonnancement en temps-réel sur ressources identiques en parallèle, noté $P_m \,||\, C_{\max}$, Il consiste à ordonnancer un ensemble de travaux sur des ressources identiques en parallèles, partant d'un nombre précis de tâches à exécuter sur ces ressources (les tâches programmables) selon l'ordre de leurs arrivées. Au cours de cette exécution, d'autres tâches apparaissent aléatoirement (les tâches aléatoires). L'exécution de ces tâches est possible si elles n'induisent pas de retard sur le traitement des tâches programmables.

L'objectif est de minimiser la durée totale d'ordonnancement, Makespan, ainsi que la maximisation du nombre de tâches aléatoires exécutées, sous contraintes de dates de disponibilité des tâches, contraintes liées aux ressources, aux intervalles temporels et dates échues qu'il est impératif de respecter, ainsi que des durées opératoires.

La communauté scientifique considère le problème d'ordonnancement sur ressources identiques en parallèle, NP-Difficile. La résolution de ce problème consiste à développer et utiliser des approches heuristiques.

Un travail a été effectué pour le traitement du problème d'ordonnancement temps-réel dans le cas d'une seule ressource [Our 01]. Les résultats obtenus ont été adaptés au cas de deux ressources identiques en parallèle dans un autre travail [Bou 07], ces résultats ont été validés dans [BL 09], ces derniers ont développé une approche collaborative basée sur les systèmes multi-agents (SMA) pour la résolution de ce problème.

Notre approche traite du même problème, afin d'améliorer la performance des résultats obtenus et dans le but de maximiser le nombre de tâches aléatoires traitées, notre approche consiste à décomposer le problème global ($P_m \,||\, C_{\max}$) en plusieurs sous problèmes à une machine ou chaque machine gère son propre ordonnancement local. Les ordonnancements locaux sont dans ce cas des ordonnancements incorporant de la flexibilité séquentielle, cette flexibilité permet de faire face aux incertitudes liées à la mise en œuvre. Cette flexibilité séquentielle est engendrée par le théorème des pyramides paru dans les années quatre-vingt et initié par Ershler et al [EFM 85].

Ce travail est organisé en quatre chapitres. Le premier chapitre est consacré à la présentation de la problématique, des notions générales liées à la fonction ordonnancement ainsi que les méthodes de résolution proposées dans la littérature. Dans le deuxième chapitre, il est question de décrire le théorème de dominance qui permet de définir un ensemble flexible de solutions pour le problème à une machine. Dans le troisième chapitre, on présentera l'approche de résolution développée dans ce travail, nous présenterons également la méthode retenue pour l'évaluation des performances, à savoir, la simulation, aussi, on présentera le modèle ARENA qui reprend son principe et pour finir, dans le quatrième chapitre, les résultats de notre approche sont Présentés afin de faire une comparaison entre notre méthode et les méthodes proposées dans [BOU 07] et [BL 09].

Chapitre I :

Généralités sur l'ordonnancement.

I. Introduction :

Ce chapitre a pour objet de présenter toutes les notions élémentaires essentielles à notre étude afin de l'entamer sans aucune ambiguïté. La première section est consacrée à l'ordonnancement dans le processus de pilotage de la production en respectant la hiérarchie de pilotage de la production.
La deuxième section a pour but de présenter tous les éléments fondamentaux d'un problème d'ordonnancement ainsi que les différentes méthodes de résolution proposées dans la littérature.
La troisième et quatrième section feront l'objet d'une présentation des définitions et des notions de base de l'ordonnancement temps-réel et de l'ordonnancement parallèle. Dans la dernière section, nous décrirons la problématique de ce travail, puis, les différentes méthodes de résolutions existantes pour le problème d'ordonnancement temps-réel sur des ressources identiques en parallèles.

II. Positionnement de l'ordonnancement dans la gestion de production :

La gestion de la production était, à l'image de l'organisation de l'entreprise tout entière, structurée par fonctions et services, spécialisée et liée hiérarchiquement. Cette hiérarchie obéit à un modèle de résolution de problème, décomposant les décisions en trois niveaux, stratégique (portefeuille d'activités, investissement), tactique (planification) et opérationnel (pilotage et suivi quotidien des flux de matières, du travail, de la main d'œuvre). [ESQ 99]

Pour Giard [GIA 88] « La gestion de la production a pour objet la recherche d'une organisation efficace de la production de biens et de services. Elle s'appuie sur un ensemble d'outils d'analyse et de résolution des problèmes qui visent à limiter les ressources nécessaires à l'obtention d'une production dont les caractéristiques technico-commerciales sont connues ».

La place de la fonction ordonnancement varie entre le niveau tactique (ordonnancement moyen terme) et le niveau opérationnel (ordonnancement quotidien ou ré-ordonnancement). Le rôle de cette fonction est de piloter un ensemble donné de ressources (ateliers, îlots, machines) et de régler les conflits d'utilisation que pose la réalisation d'un ensemble de produits sur un horizon de temps donné.

En général, on effectue :

- La planification stratégique (à long terme « stratégique ») : elle se traduit par la formulation de la politique à long terme de l'entreprise (plus de deux-ans).
- La planification à moyen terme « tactique », qui s'effectue sur plusieurs mois (6 à 18 mois). Elle prévoit la charge de l'atelier et constitue l'entrée du calcul des besoins. Elle construit à moyen terme un plan de production qui définit les quantités à produire pour chaque ressource sur un ensemble de périodes élémentaires.
- La planification à court terme « opérationnelle »: elle assure la flexibilité quotidienne nécessaire pour faire face à la demande, d'autre part, Elle génère les ordres de fabrications et d'approvisionnements. En effet, Sur la base du plan de production, la fonction ordonnancement définit un plan de fabrication détaillé à court terme précisant les dates de début et de fin de chaque activité. La fonction pilotage exécute en temps réel le plan de fabrication en réagissant aux aléas inhérents à ce type de production (retards, pannes de ressources, ruptures d'approvisionnement...) et en maîtrisant la performance.

Le terme « planification » est utilisé à tous les niveaux, cependant, on parle d'ordonnancement au niveau de la planification à court terme.

III. Généralités sur l'ordonnancement d'atelier :

Cette section est consacrée à la fonction ordonnancement d'un système de production.

Nous aborderons respectivement : la définition de l'ordonnancement, ses éléments et les méthodes de résolution des problèmes d'ordonnancement.

1. Définition :

Plusieurs définitions d'un problème d'ordonnancement sont données dans la littérature :
« Le problème d'ordonnancement consiste à organiser dans le temps la réalisation de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchaînement, ...) et de contraintes portant sur l'utilisation et la disponibilité de ressources requises » [ESQ 99]
« Ordonnancer c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution ». [CC 88]

« Ordonnancer un ensemble de tâches, c'est programmer leur exécution en leur allouant les ressources requises et en fixant leurs dates de début ». [GOT93].

Le problème d'ordonnancement d'atelier est un cas particulier du problème d'ordonnancement en général. Dans le domaine de la gestion de la production, la fonction ordonnancement a pour rôle d'organiser dans le temps l'exécution des opérations sur les machines et de réagir à tout changement non prévu tel que, panne des machines, manque de matières, ordres de fabrication non prévus, et ceci afin de réaliser le plan de production.[JDE 2007]
En résumé, un problème d'ordonnancement d'atelier consiste à :

- déterminer les dates de début de traitement des produits,
- trouver un ordre de traitement admissible (le problème peut être sur-contraint),
- déterminer une planification robuste aux événements aléatoires.

2. Eléments de base d'un problème d'ordonnancement :

La définition d'un problème d'ordonnancement se base sur des variables que sont les tâches, les ressources, les contraintes et les objectifs ;

- **La tâche** :

Une tâche est une entité élémentaire localisée dans le temps par une date de début (r_i : release date) et de fin (C_i : date d'achèvement), dont la réalisation nécessite un certain temps appelé durée opératoire (p_i : process time).

- **Les ressources :**

La ressource est un moyen technique ou humain destiné à être utilisée pour la réalisation d'une tâche. Plusieurs types de ressources sont à distinguer :

- Disjonctive (non partageable) : la machine ne peut traiter qu'une tâche à la fois. De même une tâche ne peut être exécutée que sur une seule machine à la fois.

- Cumulative (partageable) : la machine peut traiter plusieurs tâches à la fois en fonction de sa capacité.

o Les contraintes :

La notion de contrainte est relative à un ensemble de variables de décision. Elle exprime une restriction sur les domaines de valeur de ces variables. Il ya trois types de contraintes :

- Les contraintes temporelles : respect d'un début au plus tôt, d'une fin au plus tard, d'une date échue…;
- Les contraintes d'antériorité : respect de la gamme ou de la cohérence technologique (durées minimales ou maximales entre tâches) ;
- Les contraintes de ressource : on distingue ;
    - Les contraintes cumulatives sont liées à l'utilisation simultanée d'une même ressource par plusieurs tâches et indiquent que leur capacité est limitée ;
    - Les contraintes disjonctives spécifient que le traitement d'un ensemble de tâches sur la ressource disjonctive ne peut se faire simultanément au même instant ;

o Les objectifs et les critères d'un problème d'ordonnancement :

Un problème d'ordonnancement est constitué de contraintes et d'un objectif. Chaque objectif d'un ordonnancement, a un ensemble de critères, permettant son atteinte. Le but est alors de trouver l'ordonnancement réalisable de meilleure qualité possible, c'est-à-dire minimisant ou maximisant l'objectif. Les critères les plus souvent rencontrés dans la littérature pour l'évaluation d'une solution d'un problème d'ordonnancement sont :

- La durée totale de l'ordonnancement $C\max = \max(Ci)$ (makespan) ;
- Le plus grand retard algébrique $L\max = \max(Li)$;
- La plus grande durée de séjour $F\max = \max(Fi)$;
- La durée moyenne de séjour $\overline{F} = \frac{1}{n}\sum_{1}^{n} Fi$;
- Le plus grand retard vrai $T\max = \max(Ti)$;
- Le retard moyen $\overline{T} = \frac{1}{n}\sum_{1}^{n} Ti$;
- Le nombre de tâches en retard $N_T = \sum Ui$;

Dans certains cas, un objectif unique n'est pas suffisant : effectuer des compromis vis-à-vis de plusieurs objectifs est nécessaire. Par exemple, on peut vouloir limiter les retards, tout en utilisant au maximum les ressources. Un compromis entre ces deux objectifs est alors nécessaire, et on doit aborder le problème sous forme multi-objectif.

3. Classification des problèmes d'ordonnancements :

La classification permet de décrire un problème d'ordonnancement. Grâce à cette classification, il est facile d'identifier un problème d'ordonnancement. La notation utilisée fut d'abord proposée par Garey et Johnson en 1979 [GAR 79] puis étendue par Brucker [BRU 98] Cette notation se décompose en trois champs α β et δ. Un problème se décrit par ces trois champs de la manière suivante $\alpha|\beta|\delta$.

- Le champ α :

Le champ α décrit le système à ordonnancer et se décompose généralement en deux sous champs $\alpha 1$ et $\alpha 2$, le second ($\alpha 2$) précise le nombre de machine, tandis que le premier ($\alpha 1$) indique le parcours des produits sur les différentes machines (la gamme opératoire), une gamme opératoire impose un ordre de passage des produits sur les machines, et donc un ordre des opérations associées à chaque travail ;

Mac Carthy et Liu décrivent dans [ML93] une typologie de problèmes d'ordonnancement qui permet de les classifier en fonction de la nature des ressources et des gammes des travaux à réaliser suivant sept types :

On distingue principalement:

- Les ateliers à cheminement unique ou « Flow shop »; Chaque travail est constitué de m opérations et l'ordre de passage sur les différentes ressources est le même pour tous les travaux (contraintes de précédences identiques pour tout travail).
- Les ateliers à cheminement multiple ou « Job shop »; Le nombre d'opérations n'est pas forcément le même pour tous les travaux et chaque travail a son propre ordre de passage sur les machines (contraintes de précédences propres à chaque travail).
- Les ateliers à cheminement libre ou « Open shop » ; Le nombre d'opérations n'est pas forcément le même pour tous les travaux et l'ordre de passage sur les ressources est totalement libre (pas de contrainte de précédences).
- Les ateliers à une seule ressource; chaque travail est assimilé à une opération unique, exécutée par une seule et même ressource m.
- Les ateliers à ressources parallèles; chaque travail est assimilé à une opération unique, exécutée par une ressource à sélectionner dans un ensemble ;
- Le problème job shop à ressources dupliquées ; chaque travail a sa gamme opératoire propre et chaque opération est réalisée par une ressource à sélectionner dans un ensemble.

- Le champ β :

Le champ β décrit les types de contraintes prises en compte, Il est constitué d'une suite de sous champs séparés par des virgules. Les principaux sous champs sont décrits dans l'annexe 1.

- Le champ δ :

Le champ δ décrit l'objectif à optimiser ($C_{\max}, T_{\max}, L_{\max}$,…….et)

Exemples :

A titre illustratif citons le problème $J||C_{\max}$ qui se traduit par « problème de job shop à m machines avec comme objectif la minimisation du makespan ».

Le one machine problem, noté $1|r_i|L_{\max}$ est le problème à une ressource, avec des dates de début au plus tôt dont l'objectif est de minimiser le retard algébrique.

Le flow shop à deux machines, se note $J2||C_{\max}$.

4. Les méthodes de résolution des problèmes d'ordonnancement :

Plusieurs auteurs partitionnent les méthodes de résolution en deux classes : les méthodes exactes et les méthodes approchées :

a) Les méthodes exactes :

Le terme « méthodes exactes » recouvre un ensemble de méthodes d'ordonnancement issues de la recherche opérationnelle, ces méthodes sont dites exactes du fait qu'elles convergent vers une solution optimale du problème considéré. Pour cela, elle se base sur des calculs mathématiques complexes et coûteux rendant difficiles leur mise en œuvre et leur utilisation, ce qui explique qu'elles ne sont utilisables que sur des problèmes de moyenne ou de petite tailles. Ces méthodes présentent un grand risque de convergence vers un optimum local si nous choisissons mal les conditions initiales. Parmi ces méthodes, on peut citer [1]:

- Les procédures par séparation et évaluation (PES) qui énumèrent par une recherche arborescente un ensemble de solutions, en éliminant les branches de l'arbre de recherche non optimales. Cette méthode est basée sur une énumération implicite et intelligente de l'ensemble des solutions réalisables. Pour cela, la séparation consiste à décomposer le problème initial en plusieurs sous problèmes qui sont à leurs tours décomposables.

1 : nous invitons le lecteur à consulter [GRA 05] pour plus de détails concernant les différentes méthodes de résolution.

- Les méthodes basées sur la programmation linéaire (PL) modélisant les critères et les contraintes comme des fonctions linéaires des variables, et utilisant pour la résolution généralement la méthode du Simplexe.
- Les méthodes basées sur la programmation dynamique (PD) : consistent en une décomposition du problème principal de dimension n, en sous problèmes de dimension n-1, que l'on résout en tenant compte à chaque étape des informations issues de la résolution du sous problème précédent.

b) Les méthodes approchées (heuristiques):

Les méthodes heuristiques sont souvent utilisées pour traiter des problèmes que les méthodes optimales sont incapables de résoudre en un temps acceptable. Elles produisent généralement une solution faisable de bonne qualité en un temps relativement court. Ces heuristiques sont classiquement classifiées en cinq catégories [GOT 93] :

- Les algorithmes par construction progressive (gloutons) :

Cette catégorie de méthodes procède à la construction d'une solution globale à partir d'une solution partielle complétée au fur et à mesure de la résolution.

- Les méthodes par voisinage (tabou, recuit simulé, algorithmes génétiques) :

Ces méthodes consistent à explorer l'espace des solutions d'ordonnancement en partant d'une solution initiale complète de laquelle on extrait une solution voisine, si cette solution est meilleure, elle devient la nouvelle solution de référence.

Le calcul continue jusqu'à ce qu'une solution satisfaisant au critère de recherche soit obtenue.

- Les méthodes par décomposition :

Les méthodes par décomposition procèdent à la décomposition du problème d'ordonnancement afin de faciliter sa résolution. Cette approche permet de simplifier les calculs, la décomposition peut être :

  - Hiérarchique : décomposition du problème en niveau d'abstractions différents et décroissants, chaque niveau coordonne les unités de pilotage du niveau inférieur, chaque niveau à des relations de dépendance vis à vis du niveau supérieur et de dominance vis à vis du niveau inférieur.
  - Décomposition de l'ensemble des solutions du problème : l'espace des solutions est décomposé en catégories, chacune étant évaluée séparément des autres.
  - Décomposition temporelle : cette décomposition est spécifique aux problèmes d'ordonnancement dynamique, elle consiste à décomposer le problème en intervalle de temps.

- Les méthodes par relaxation des contraintes : Cette méthode consiste à modifier le modèle que nous avons à résoudre, pour ce faire, on a recours à une relaxation de certaines contraintes du problème.

IV. L'ordonnancement sur des ressources en parallèle :

1. Définition :

On parlera de parallélisme lorsque plusieurs ressources peuvent réaliser la même tâche et qu'une fois une tâche est traitée par une ressource, elle ne pourra plus être traitée par une autre. Les problèmes à ressources parallèles généralisent naturellement les problèmes à une ressource, ils sont intensivement étudiés depuis le milieu des années soixante. Ces problèmes s'imposent d'un point de vue pratique dès que, dans un atelier, plusieurs ressources peuvent effectuer la même tâche. C'est un problème fréquemment rencontré dans des applications réelles, en particulier dans le cadre de l'ordonnancement de traitement informatique [ESQ 01].Dans ce dernier cas, les opérations correspondent au traitement de programmes sur des processeurs ou les programmes correspondent aux tâches et les processeurs aux ressources.

Selon la classification des problèmes d'ordonnancement, ces problèmes sont de type R ||δ, Il existe de nombreux ouvrages et articles abordant les problèmes d'ordonnancement en général et l'ordonnancement de ressources parallèles en particulier, citons Esquirol [ESQ 01], Yalaoui [YAL 00], Hall [HAL 00], etc. …. Ce type de problème est généralement désigné dans la littérature par PMS (Parallel Machines Scheduling).

Les travaux les plus couramment rencontrés dans la littérature traitent classiquement l'optimisation de critères tels que $C_{\max}$, $L_{\max}$, $T_{\max}$ ...etc. Pour les systèmes industriels à machines parallèles, le problème le plus complexe est celui des ressources différentes [SID 97].

Généralement, dans les ateliers de production, on distingue trois types de problèmes à ressources parallèles :

a. le parallélisme identique : noté P m| β|δ, chaque ressource va mettre autant de temps qu'une autre pour traiter un job donné.
b. Le parallélisme uniforme : noté Q m|β|δ, dans ce cas de figure, une vitesse est associée à chaque ressource. Les vitesses relatives sont indépendantes des taches, ainsi si une ressource est deux fois plus rapide qu'une autre, elle effectuera toutes les tâches deux fois plus vite que sa concurrente.
c. Le parallélisme différent : noté R m|β|δ, dans ce cas il n'existe aucun lien entre les ressources (i.e. la durée opératoire est totalement variables entre les différentes

machines), une durée opératoire est associée à chaque couple tâche-ressource, traduisant le fait qu'une ressource peut être performante pour certaines tâches et ne pas l'être pour d'autres.

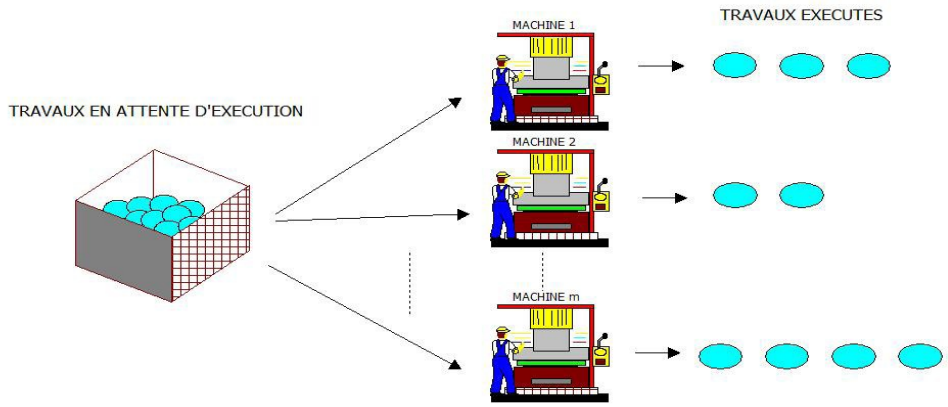


Figure I.1 : représentation d'un atelier d'assemblage à ressources parallèles [BL 09].

2. L'ordonnancement en parallèle avec minimisation du C max :

L'optimisation de critère comme la minimisation de $C_{\max}$ (makespan) à m ressources parallèles est le point sur lequel Chen et Vestjens [CV 97] ont consacré leur étude en adoptant l'heuristique de Jackson. Dans le cas d'opérations interruptibles (P_m |pmtn| $C_{\max}$), l'algorithme de Mac Naughton fournit un ordonnancement de durée minimale. Cet algorithme optimise le problème.

Pour le problème (P_m |prec| $C_{\max}$) i.e. problème ressources parallèles avec contraintes de précédence et dans le cas où le nombre de ressources est au moins égal au nombre de travaux (on le note P∞|prec| $C_{\max}$), on a recours à des méthodes à chemins critiques (PERT, CPM et MPM). En revanche, le problème (P_m |prec| $C_{\max}$) avec m<n est NP-difficile au sens fort. [EL 99].

Lorsque le graphe de précédence se présente comme une arborescence et la durée des tâche pi est constante (P_m |tree, pi = cst| $C_{\max}$), l'algorithme de HU donne une solution optimale pour le problème [HU 61].

3. L'ordonnancement sur ressources identiques parallèle avec minimisation du $C_{\max}$:

Le problème consistant a ordonnancer un ensemble de tâches sur un ensemble de ressources parallèles identiques, tout en cherchant a minimiser la durée totale de production, est noté $P_m \| C_{\max}$ dans la classification proposée par Graham et al dans [GRA 79], ce problème est NP-difficile au sens fort selon Garey et Johnson [GAR 79], c'est l'un des problèmes les plus étudiés dans l'optimisation combinatoire.

Le problème $P_m \| C_{\max}$ est largement étudié. Parmi les travaux qui ont étudié ce problème on peut citer, Garey et Johnson [GAR 78] qui ont démontré que le problème $P_m \| C_{\max}$ est fortement *NP*-difficile et McNaughton [MCN 59] qui a prouvé que la préemption rend ce problème facilement solvable. Grâce à cette relaxation, McNaughton a proposé une borne inférieure pour ce problème qui est :

$$C^*_{max} = max\left(\frac{1}{m}\sum_{i=1}^{n} p_i , max(p_1, \dots, p_n)\right)$$

m étant le nombre de machines, *pi* la durée opératoire de la tâche *i* et *n* le nombre des tâches. Pour le même problème, l'heuristique basée sur la règle *LPT* (Longest Processing Time) est une heuristique simple et efficace. Sa performance [2] a été évaluée par Graham dans [GRA 69] égale à : $\frac{C_{\max}(LPT)}{C^*_{\max}} \leq \frac{4}{3} - \frac{1}{3m}$

Avec $C^*_{\max}$ la valeur de la solution optimale, $C_{\max}(LPT)$ la solution obtenue avec l'heuristique *LPT* et *m* le nombre de machines.

2 : Performance d'une heuristique : $R_h = \frac{Z^h}{Z^*}$ où Z_h est la valeur de la solution donnée par l'heuristique et Z^* est la valeur pour une solution optimale.

V. L'Ordonnancement temps-réel :

Les systèmes temps-réel ont pris une plus grande ampleur avec l'évolution de l'informatique. L'informatique temps-réel est largement utilisée et prend une importance de plus en plus grande dans la vie quotidienne. Plusieurs domaines d'application sont concernés, notamment, l'automobile, l'avionique, la téléphonie mobile, le multimédia, l'énergie, ...etc. Les systèmes temps-réel concernent des applications informatiques ayant un rôle de suivi ou de contrôle de procédé dans un environnement qui évolue dynamiquement. L'application est réactive, en effet, elle doit réagir au changement d'état du système contrôlé dans des contraintes temporelles précises. Certains systèmes sont classés critiques lorsque le non respect des contraintes temporelles peut provoquer des conséquences catastrophiques (perte de vies humaines, destruction de matériel, ...). Les systèmes de contrôle de vol, systèmes de contrôle de centrale nucléaire en sont des exemples.

1. Description des systèmes temps-réel :

On appelle classiquement une application temps-réel un système de traitement de l'information ayant pour mission de commander un environnement, en respectant les contraintes de temps. Ordinairement on utilise la notion de systèmes temps-réel en informatique.

En effet, selon l'aspect abordé dans les systèmes temps-réel, on choisit une définition qui s'approche le plus de la problématique traitée. Ainsi, on assimile, selon le cas, un système temps-réel à :

- un système rapide
- un système réactif
- un système qui ne fournit pas de réponse en différé
- un système avec un comportement prédictible
- un système robuste.

Les applications temps-réel sont caractérisées par la nécessité absolue de respecter des contraintes temporelles strictes.

On qualifie de temps-réel tout système informatique dont la correction du fonctionnement ne dépend pas seulement de l'exactitude logique des résultats qu'il fournit, mais surtout dépend de la date à laquelle ces résultats sont produits. Ainsi, le système ne traite plus uniquement des valeurs, mais des couples valeur et temps. [DES 82] En effet, les applications temps-réel doivent réagir en tenant compte de l'écoulement du temps. Cette caractéristique fondamentale distingue globalement les applications temps-réel des autres types d'applications informatiques.

Une définition des systèmes réactifs décrivant les systèmes temps-réel est donnée dans [GKS 03] :

« Un système réactif est un système qui réagit continument avec son environnement à un rythme imposé par cet environnement, Il reçoit, par l'intermédiaire de capteurs, des entrées provenant de l'environnement, appelées stimuli, réagit à tous ces stimuli en effectuant un certain nombre d'opérations et produit, grâce à des actionneurs, des sorties utilisables par l'environnement, appelées réactions ou commandes ».

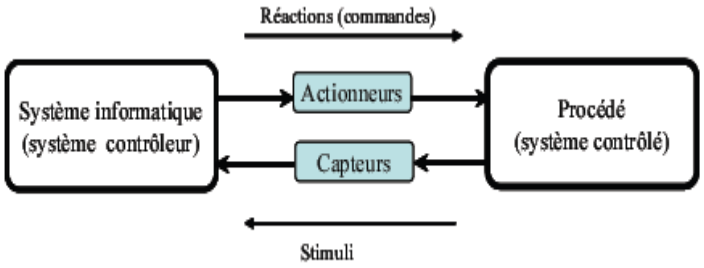


Figure I.2 : Système temps-réel [GKS 03]

2. Problème d'ordonnancement temps-réel :

Un mécanisme d'ordonnancement temps-réel a pour rôle de mettre en œuvre l'exécution d'un ensemble de tâches sur un ensemble de ressources. Dans un modèle dynamique, ceci s'accompagne d'une gestion dans le temps où le mécanisme d'ordonnancement a pour charge de suivre cette exécution et de la modifier en fonction des précisions obtenues sur certains paramètres et des nouvelles tâches créées.

La théorie de l'ordonnancement des systèmes temps-réel se focalise principalement sur deux problèmes [BL 09] :

- Le problème de faisabilité : étant donné les spécifications des tâches et les contraintes sur l'environnement d'ordonnancement (par exemple si les priorités sont fixes/dynamiques au niveau des tâches, si les préemptions sont admises, etc.), il s'agit de déterminer l'existence d'un ordonnancement qui satisfasse toutes les échéances ;
- Le problème d'ordonnançabilité en ligne : étant donné les spécifications des tâches (et des contraintes de l'environnement) supposées être faisables, il s'agit de déterminer un algorithme d'ordonnancement qui construit un ordonnancement qui satisfasse toutes les échéances.

Pour les problèmes en environnement parallèle, les tâches à exécuter sont distribuées sur les différentes ressources ; il faut alors, donner un ordre d'exécution des tâches (i.e. : résoudre des problèmes d'ordonnancement). Le respect des contraintes de temps, généralement des contraintes des dates échues, est imposé par l'environnement. Le problème de l'ordonnancement temps-réel est donc de donner un ordre aux tâches qui respectent ces contraintes de temps.

VI. Ordonnancement temps-réel sur ressources identiques parallèles :

Notre problématique traite de l'ordonnancement temps-réel sur deux ressources identiques en parallèle, ainsi, cette section est consacrée à sa description et aux méthodes proposées dans la littérature.

1. Présentation du problème :

Le problème posé consiste à ordonnancer sur plusieurs ressources identiques parallèles un certain nombre de tâches arrivant aléatoirement « tâches aléatoires », sachant que le système dispose d'un certain nombre de tâches programmables. L'objectif est de minimiser la durée totale de l'ordonnancement (Makespan) tout en maximisant le nombre de tâches aléatoires traitées. Ce problème est soumis à des contraintes liées aux ressources, aux intervalles temporelles et aux tâches elles-mêmes.

Ourari, [Our 01], a traité le problème d'ordonnancement temps-réel dans le cas d'une seule ressource. Elle a proposé deux algorithmes d'ordonnancement basés sur les règles de priorité ainsi qu'une approche basée sur l'insertion de la tâche aléatoire dès son arrivée.
Bougchiche dans [BOU 07] a adapté ces résultats sur le cas de deux ressources identiques parallèles. Ensuite, Behiri et Latrache dans [BL 09] ont validé les approches proposées par Bougchiche et ont développé une approche collaborative, en vu de permettre l'ordonnancement d'un plus grand nombre de tâches aléatoires en respectant les délais d'exécution de toutes les tâches.

On s'intéresse dans cette section, au problème décrit dans [Bou 07] et dans [BL 09]. Notre travail, aussi, traitera du même problème. Ainsi, nous présentons les caractéristiques et les paramètres de ce problème, mais aussi, les différentes approches de résolution qui ont été proposées.

- Les Hypothèses :

- Il n y'a que deux types de tâche : programmables et aléatoires.
- Les tâches programmables sont toutes traitées dans leurs intervalles temporels.
- Les temps inter opératoires sont nuls.
- La préemption est interdite.
- Les temps morts sont minimisés.
- Toute tâche aléatoire induisant, lors de son exécution, un retard sur une tâche programmable est rejetée.
- Une tâche aléatoire est rejetée s'il n'est pas possible de l'exécuter dans son délai.

- Chaque tâche est exécutée sur une ressource seulement.
- Toutes les ressources sont identiques.
- Il n'y a pas d'interdépendance entre les tâches.
- Toutes les ressources sont disponibles à l'instant zéro.
- Le temps de transport entre les ressources est négligeable.
- Les temps de préparation des ressources et des tâches sont négligeables.
- Les capacités de stockage des files d'attente sont illimitées.
- Les tâches aléatoires ne sont pas nécessairement exécutées dés leurs apparitions.

o Caractéristiques des tâches programmables :

Toutes les données suivantes correspondant à chaque tâche programmable (i) sont supposées connues à l'instant zéro :

- r_j : la date de disponibilité,
- p_j : la durée opératoire,
- d_j : la date échue,

Toutes les tâches sont supposées exécutables dans leurs intervalle d'exécution [r_j d_j]

o Caractéristiques des tâches aléatoires :

Chaque tâche aléatoire j est caractérisée par les éléments suivants :

- r_j : la date de disponibilité,
- p_j : la durée opératoire,
- d_j : la date échue.

Contrairement aux tâches aléatoires, ces informations ne sont connues qu'au moment de l'apparition de la tâche aléatoire (i.e. : au moment r_j pour la tâche j).

Les différentes distributions régissant les dates de disponibilités, les durées opératoires et les dates échues sont identifiées comme suit :

- Le processus d'arrivée des tâches aléatoires suit une loi de poisson, cette variable aléatoire est notée X.
- Les durées opératoires suivent une loi triangulaire (Min, Mode, Max).
- Les dates échues sont données par la variable aléatoire D = X+ Ct (ou Ct un paramètre du problème à déterminer).

Une tâche aléatoire est rejetée s'il n'est pas possible de l'exécuter dans son délai.

2. Les différentes approches proposées dans [BOU 07] et dans [BL 09]

Rappelons que Le problème $P_m // C_{\max}$ est un problème NP-Difficile. Il n'est pas facile à résoudre puisque même le simple cas d'ordonnancement sur deux ressources est NP-Difficile [Kar 72], [Pin 95] et [Mok 04].

Ourari, [Our 01], a traité le problème d'ordonnancement temps-réel dans le cas d'une seule ressource. Elle a proposé deux algorithmes d'ordonnancement basés sur les règles de priorité ainsi qu'une approche basée sur l'insertion de la tâche aléatoire dès son arrivée.

Bougchiche dans [BOU 2007] a adapté ces résultats sur le cas de deux ressources identiques parallèles, le but est de minimiser le makespan tout en maximisant le nombre de tâches aléatoires traitées.

L'ordonnancement s'effectue en deux étapes : l'ordonnancement statique, correspond à l'ordonnancement des tâches prévisionnelles et l'ordonnancement dynamique qui consiste à introduire les tâches aléatoires dans l'ordonnancement obtenu précédemment.

Etape 1 : L'ordonnancement statique : ou il s'agit de répartir les tâches prévisionnelles sur les deux ressources selon une règle d'ordonnancement permettant d'exécuter les tâches dans leurs intervalles temporels à l'intérieur d'une plage de temps la plus faible possible. La limite supérieure de cette plage représente la borne supérieure qui doit être minimisée. Minimiser cette borne permet de minimiser la durée totale de l'ordonnancement. La règle d'ordonnancement choisie est basée sur l'ordre croissant des dates de disponibilité des tâches. Une fois la répartition achevée, une séquence d'exécution des tâches est choisie sur chaque ressource. De là, considérer une ressource et trouver la séquence admissible pour l'exécution des tâches prévisionnelles qui ne sera plus modifiée (ultérieurement). La même procédure est à appliquer à l'autre ressource. Toute tâche prévisionnelle disponible et non exécutée par une ressource est en attente d'exécution et stockée dans une file d'attente.

Etape 2 : L'ordonnancement dynamique : cette étape consiste à utiliser le programme d'exécution de l'ordonnancement statique en insérant les tâches aléatoires parmi les tâches programmables en respectant les différentes contraintes (chaque tâche aléatoire doit être exécutée dans son intervalle temporel, l'ordonnancement d'une tâche aléatoire sur une ressource ne doit pas remettre en cause l'exécution de la tâche programmable, …).Principalement, deux approches de résolutions ont été proposées par Bougchiche dans[Bou 07]. La première approche, consiste à stocker les tâches aléatoires dans une file d'attente commune aux deux ressources, pour cette approche, deux algorithmes ont été développés, à savoir, l'algorithme d'ordonnancement temps-réel au plus tard et d'ordonnancement temps-réel au plus tôt. La deuxième approche, propose deux files d'attente pour

les tâches aléatoires (pour chaque ressource, une file d'attente).Pour cette approche, aussi, deux algorithmes ont été développés, à savoir, l'algorithme d'ordonnancement temps réel au plus tard et l'algorithme de calcul de la marge disponible.

a)Algorithme d'ordonnancement temps-réel au plus tard dans le cas d'une file d'attente :

Cet algorithme donne la priorité à la tâche aléatoire dans l'utilisation des marges disponibles. La tâche aléatoire doit vérifier certaines conditions pour être ordonnancée et qui sont :

- L'intervalle temporel de la tâche aléatoire : une tâche dont la date échue n'est pas respectée (i.e. : tnow + pj ≤ dj) est rejetée;
- L'intervalle d'exécution des tâches programmables : l'ordonnancement d'une tâche aléatoire induisant un retard sur l'exécution d'une tâche programmable, ordonnancée au plus tard (i.e. : tnow + pj ≤ tbi) est rejetée.

b) Algorithme d'ordonnancement temps-réel au plus tôt dans le cas d'une file d'attente :

Cet algorithme donne la priorité à la tâche programmable dans l'utilisation des marges disponibles, ainsi, la tâche programmable est exécutée dès que l'instant courant est égal à la date de début au plus tôt de cette tâche tai ≥ tnow.

Pour être exécutée, une tâche aléatoire doit vérifier les mêmes conditions que dans le premier cas, à savoir, l'intervalle temporel de son traitement et l'intervalle d'exécution des tâches programmables.

c) Algorithme d'ordonnancement temps-réel au plus tard dans le cas de deux files d'attente :

Cet algorithme permet d'affecter les tâches aléatoires aux files d'attente des ressources en les insérant directement dans la position d'exécution. Par conséquent, à chaque apparition d'une tâche aléatoire, elle doit satisfaire des conditions pour pouvoir l'exécuter. Dès son affectation, elle a une position entre les tâches programmables. Cet algorithme donne la priorité à la tâche aléatoire à chaque fois qu'un choix se présente (i.e. : lors de l'ordonnancement dès son arrivée ou chaque fois qu'une ressource est libre).

Pour être exécutée, une tâche aléatoire doit vérifier les mêmes conditions que dans les deux premiers cas, à savoir, l'intervalle temporel et l'intervalle d'exécution des tâches programmables.

d) Algorithme de calcul de la marge disponible dans le cas de deux files d'attente :

La démarche de résolution est constituée principalement de deux étapes :

La première étape détermine le premier espace libre entre deux tâches programmables consécutives, pour insérer la tâche aléatoire intervenant Aj. D'où, une première position provisoire pour Aj est obtenue ;

La seconde étape définit la position d'insertion de la tâche qui permet de maximiser sa chance d'acceptation et qui préserve au mieux l'avenir. L'idée, dans ce cas, consiste à avancer éventuellement certaines tâches programmées vers la droite pour insérer Aj plus tôt que sa position provisoire.

Cet algorithme, à l'instant courant, vérifie les contraintes sur l'intervalle temporel de la tâche aléatoire (i.e. : tnow + pj ≤ dj) et sur la marge libre permettant d'insérer cette tâche qui doit être supérieure ou égale à sa durée opératoire.

e) L'approche collaborative :

D'autre part, Behiri et Latrach dans [BL 09] ont validé les approches proposées par Bougchiche et ont développé une approche collaborative. Cette approche est basée sur les systèmes multi-agents. Les systèmes multi-agents appartiennent à la branche de l'intelligence artificielle distribuée. Plutôt que de modéliser la résolution d'un problème sous forme centralisée, il s'agit de créer une communauté d'entités autonomes (les agents) permettant de résoudre des sous-problèmes du problème générique. La résolution du problème global doit donc émerger de la résolution des problèmes élémentaires. Il s'agit donc d'une forme de résolution de problème par décomposition, les agents représentant les différentes machines, Le processus de négociation est enclenché lorsqu'il y a des tâches à tester. Les calculs sont répartis sur les machines. Chaque machine a son propre code de calcul.

Cette approche a permis d'alléger la complexité du problème original, en distribuant l'ordonnancement sur les différents agents (machines) constituant l'atelier de production. Cette approche peut être généralisée au cas de plusieurs machines en parallèle sans augmenter, pour autant, la difficulté du problème initial, ce qui n'était pas le cas avec les approches déjà existantes, qui voient la complexité de leur application augmenter en fonction du nombre de machines constituant l'atelier, jusqu'à rendre la solution inapplicable.

VII. Conclusion :

Dans ce chapitre, la problématique générale de ce travail a été posée : on s'intéresse à la résolution d'un problème d'ordonnancement temps-réel sur deux machines identiques en parallèle, nous avons commencé par définir les concepts généraux liés à notre problématique, nous avons démontré la complexité liée à ce problème, ainsi que les approches de résolution qui ont été proposées.

Par ailleurs, l'ordonnancement dans ce chapitre, est présenté comme un problème déterministe : l'ensemble des tâches à ordonnancer est déterminé a priori, les durées opératoires sont fixes et stables et les capacités des ressources sont connues à tout instant.
En pratique, l'ordonnancement d'atelier n'est pas aussi déterministe : des livraisons sont en retard, des opérations prennent plus de temps que prévu ou un client a besoin de son produit plus tôt que ce qu'il avait demandé à l'origine.

Pour pallier ces différentes incertitudes, des travaux de recherches tentent de les prendre en compte. Le chapitre suivant sera consacré à la présentation des généralités sur l'ordonnancement en milieu incertain.

Chapitre II :

L'ordonnancement en milieu incertain et généralités sur le théorème des pyramides.

I. Introduction :

Ce chapitre est consacré à la présentation des généralités sur l'ordonnancement en milieu incertain.
La première et deuxième sections seront consacrés à la définition des notions relatives à l'ordonnancement en environnement incertain et aux méthodes d'ordonnancement sous-incertitudes citées dans la littérature.
La troisième section sera consacrée aux approches d'ordonnancement distribuée et à l'approche coopérative. Dans la quatrième section, nous allons proposer une approche robuste pour la résolution du problème d'ordonnancement à deux machines identiques en parallèles, cette approche est basée sur un théorème de dominance qu'est le théorème des pyramides. Le théorème des pyramides permet de définir un ensemble flexible de solutions pour le problème à une machine, c'est pourquoi, nous allons décomposer tout d'abord le problème global en plusieurs sous problèmes à une machine.

II. Les notions d'incertitude et de robustesse en ordonnancement :

Que ce soit en gestion de production ou en gestion de projet, une caractéristique importante de l'environnement dans lequel s'applique un ordonnancement réside dans son caractère incertain. L'ordonnancement dépend de paramètres internes, propres au système (capacité des ressources, durées opératoires, . . .), et de paramètres externes, correspondant aux informations données par les fournisseurs et les sous-traitants participant indirectement à sa mise en œuvre (délais, volumes de production, . . .). Lors de la réalisation de l'ordonnancement, ces paramètres sont mal connus et susceptibles de varier dans le temps.

Cette section est consacrée à la présentation des outils développés dans la littérature pour résoudre les problèmes d'ordonnancement en milieu incertain, nous commençons par définir les notions d'incertitude, de robustesse et de flexibilité.

1. Les incertitudes en ordonnancement :

Les incertitudes sont la principale difficulté dans l'utilisation de l'ordonnancement en pratique. Nous prendrons comme définition celle que propose Esswein dans [ESS 03] :

« On parle d'incertitudes pour désigner les modifications potentielles des données d'un problème d'ordonnancement qui peuvent intervenir entre le calcul d'un ordonnancement et la fin de sa mise en œuvre réelle dans l'atelier. »

En pratique, dans le cas de l'ordonnancement d'atelier, les incertitudes généralement étudiées sont :

- Le changement de la durée opératoire d'un travail.
- L'insertion ou la suppression d'une opération. Cela arrive fréquemment, généralement sous la forme d'une commande urgente à traiter.
- La panne machine.
- La différence entre le modèle et la réalité. Par exemple, la non-présence des temps de transport entre ressources dans le modèle alors que dans la réalité, ce temps n'est pas négligeable.

Les aléas sont des types particuliers d'incertitudes qui concernent les données structurelles du problème (l'arrivée d'un nouveau travail, le changement du nombre des machines disponibles) et non pas les données chiffrées du problème (une durée opératoire plus longue que prévu).

Ces incertitudes ont une influence importante dans l'ordonnancement d'atelier. Il est donc utile de les prendre en compte. La flexibilité et la robustesse ont pour objectifs de mesurer l'influence des incertitudes et de tenter de les limiter.

2. Flexibilité et robustesse en ordonnancement :

La flexibilité :

Avant de voir les différentes formes de flexibilité, commençons par définir ce que nous entendons par flexibilité, selon Essewein dans [ESS 03] :

« La flexibilité est la liberté dont on dispose durant la phase de mise en œuvre d'un ordonnancement ».

Erscheler et Terssac dans [ET 88] définisse la flexibilité comme étant une propriété qui recouvre généralement deux aspects complémentaires mais distincts : Un aspect interne lié à une capacité de changement, de déformation, à une variété d'états possibles. Et un aspect externe lié à une capacité d'adaptation à des modifications de l'environnement, à des perturbations.

La flexibilité dans un ordonnancement peut prendre plusieurs formes :

- La flexibilité temporelle : elle concerne les dates de début d'exécution des tâches, elle permet à ces dernière une dérive dans le temps.
- La flexibilité séquentielle : elle permet de modifier l'ordre des séquences des opérations sur les machines, elle est parfois appelée flexibilité sur les ordres. L'introduction de flexibilité séquentielle impose de manipuler, non pas un seul ordonnancement, mais une famille d'ordonnancements ;
- La flexibilité sur les affectations : elle permet de déplacer une opération d'une machine à une autre, cette flexibilité est très pratique en cas de panne de machine.
- La flexibilité sur les modes d'exécution, i.e. sur la modification du modèle du problème, citons par exemple le changement de gamme ou la modification du nombre de ressources nécessaires pour effectuer une opération.

La robustesse :

La robustesse est un terme utilisé dans beaucoup de domaines, on rencontre plusieurs définitions dans la littérature. En règle générale, on peut définir un ordonnancement robuste comme un ordonnancement peu sensible aux incertitudes [BMS 04].

Il existe un lien certain entre flexibilité et robustesse. La flexibilité est un outil pour réaliser un ordonnancement robuste : l'utilisation de la flexibilité pendant la réalisation d'un ordonnancement donnera un résultat plus robuste que si la flexibilité n'est pas utilisée. De plus, un ordonnancement peut être robuste sans être flexible.

3. Les Phases de l'ordonnancement :

En prenant en compte l'environnement dynamique de mise en œuvre de l'ordonnancement, le problème globale d'ordonnancement peut se décomposer en deux phases [ESS 03] :

- la phase prédictive (statique, hors ligne) : cette phase a lieu avant l'exécution de l'ordonnancement dans l'atelier, l'exécution de l'ordonnancement est « prédit ».Cette phase repose intégralement sur un modèle : elle n'est pas réalisée sur l'atelier, mais hors ligne.
- la phase réactive (dynamique, en ligne) : qui a lieu pendant l'exécution de l'ordonnancement, le but de cette phase est de réagir en temps réel aux événements de l'atelier. les décisions à réaliser par le système doivent être prises rapidement, avec généralement un temps de réaction imposé.

Proposer une méthode d'ordonnancement revient donc à proposer, idéalement, à la fois un ordonnancement statique et un ordonnancement dynamique. Pourtant plusieurs travaux ne portent que sur la phase hors ligne, et négligent l'aspect dynamique de l'ordonnancement

[ESS 03].Ces deux phases sont des notions primordiales pour l'ordonnancement sous incertitudes, il faut prendre en compte les incertitudes et les aléas pour répondre de manière efficace à l'aspect indiscutablement dynamique de l'environnement de l'application. Cette prise en compte se fait en développant des méthodes d'ordonnancement robustes.

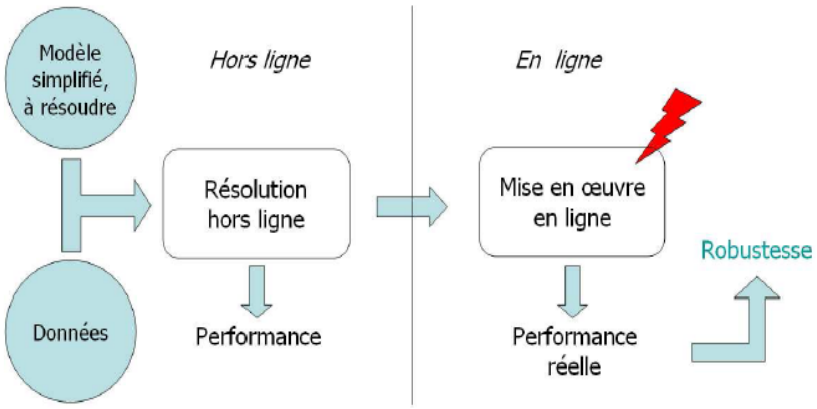


Figure II.1 : les phases d'un ordonnancement [ESS 03]

III. Les méthodes d'ordonnancement sous-incertitudes :

[DB00] propose une classification des méthodes d'ordonnancement sous incertitudes.

Les phases de l'ordonnancement pendant lesquelles les incertitudes sont prises en compte permettent de classer ces méthodes. Il en ressort trois catégories principales : les méthodes proactives, les méthodes réactives et les méthodes proactives-réactives.

1. Les méthodes proactives :

Les approches proactives tentent de prendre en compte l'incertain lors de la phase d'ordonnancement hors-ligne uniquement. Il s'agit d'anticiper les incertitudes, en jouant sur la flexibilité, de sorte à produire un ordonnancement, ou une famille d'ordonnancements, relativement insensible aux incertitudes. [TRU 05]

2. Les méthodes réactives :

Contrairement aux approches proactives qui se proposent de prendre en considération les incertitudes lors de la phase statique du processus d'ordonnancement global, les méthodes réactives procèdent de manière symétrique, en ne prenant en compte les incertitudes que lors de la phase dynamique du processus de résolution. [JEL 07]

On ne cherche donc pas à anticiper les incertitudes, mais plutôt à réagir en temps réel, de façon opportune, lorsque des aléas surviennent. Le temps d'élaboration des décisions d'ordonnancement,

nécessaire à la prise en compte d'un aléa, doit être suffisamment court relativement à la dynamique du système d'activité considéré.

3. Les méthodes proactive-réactive :

Une approche proactive-réactive tente de combiner les méthodes proactives et réactives. Dans un premier temps, un ou plusieurs ordonnancements sont générés (au niveau statique). Ensuite, la phase dynamique utilise et adapte ces ordonnancements en temps réel.
Selon [DB00], un système qui est capable de prendre en compte les incertitudes en ordonnancement est un système qui prend en compte les incertitudes durant la phase de génération de la solution statique, et qui peut réagir, durant l'exécution, à l'arrivée d'événements inattendus.

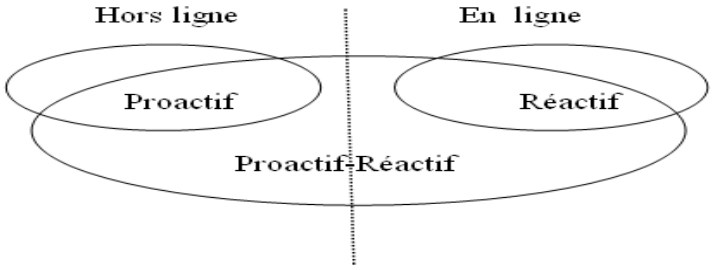


Figure II.2: Positionnement des différentes approches d'ordonnancement en présence d'incertitudes dans les différentes phases du processus global d'ordonnancement [Ess03].

IV. Choix de la méthode d'ordonnancement

Pour la suite de notre étude, nous avons besoin d'une méthode d'ordonnancement flexible et robuste permettant la résolution de notre problème, L'approche choisie est la classe proactive-réactive. Différents avantages de cette méthode nous ont conduits à ce choix.

Tout d'abord, les méthodes proactive-réactive sont très bien adaptées pour faire face aux incertitudes, Ces méthodes permettent de réagir en ligne aux événements imprévus qui apparaissent durant l'exécution, en autorisant le passage d'une solution devenue obsolète à une autre sans remise en compte de calculs déjà effectués. Ensuite, Ce type d'approche pallie d'une part, les inconvénients des approches proactives qui sont basées sur une simple modélisation des incertitudes et ne prennent pas en compte les évènements réels du système étudié et d'autre part, les approches réactives qui ne fournissent que de faibles performances comparativement à l'ordonnancement optimal.

De façon classique, la résolution en- ligne ou hors-ligne des problèmes d'ordonnancement avec prise en compte des incertitudes est classiquement assimilée à un problème de décision global car la fonction ordonnancement gère l'organisation de la totalité des ressources, chacune devant respecter la

solution établie [OBB 07]. Il est donc nécessaire d'adopter un fonctionnement coopératif dans lequel toute décision locale doit faire l'objet d'un processus de décision collectif.

Partant du fait que les ressources possèdent une certaine autonomie décisionnelle, et que la négociation est décrite comme étant une coopération dont l'objectif commun est l'obtention d'un accord, nous avons opté dans notre travail pour une démarche coopérative entre les différents acteurs représentant les ressources.

Dans ce qui suit, nous allons procéder à la distribution de la fonction ordonnancement, le problème d'ordonnancement sur deux machines identiques en parallèle sera décomposer en deux sous problèmes à une machine ou chaque machine gère son propre ordonnancement local, l'ordonnancement global résultant d'une coopération entre les diverses ressources.
On suppose que les ordonnancements locaux sont des ordonnancements incorporant de la flexibilité séquentielle, cette flexibilité permettant à chaque ressource, de faire face aux aléas liées à la mise en œuvre.
La flexibilité séquentielle est engendrée par un théorème de dominance, qu'est le théorème des pyramides.
Nous commencerons par définir les notions relatives à l'ordonnancement distribué et à la coopération, ensuite nous allons présenter le théorème des pyramides et expliquer comment engendrer un ensemble flexible de solutions.

V. Les systèmes d'ordonnancement distribués :

Nous aborderons dans cette section les concepts relatifs à l'ordonnancement distribué.

1. Vers un système d'ordonnancement distribué :

L'accroissement des besoins industriels d'agilité et de traçabilité, ainsi que l'apparition de nouvelles possibilités techniques, conduisent à la proposition de systèmes de pilotage distribués, qui s'opposent à des méthodes de gestion plus classiques, centralisées.

La principale limite de l'approche centralisée est la désynchronisation entre les flux de matière de l'atelier et les flux d'informations, en conséquence, une perte de réactivité. En effet, la différence entre les périodes de chaque niveau hiérarchique conduit à des délais pouvant être importants entre une prise de décision et sa mise en œuvre sur le terrain. Ce délai de propagation de l'information entre niveaux hiérarchiques conduit donc à une perte de synchronisation entre les plans.

Pendant longtemps, les systèmes d'ordonnancement issus de l'Intelligence Artificielle (IA) se sont appuyés sur une conception centralisée de la résolution du problème d'ordonnancement. Et ce n'est qu'au cours des années 80, que les progrès technologiques ont permis d'envisager la distribution du calcul d'ordonnancement [TRA01]. Ainsi, L'utilisation de techniques d'intelligence artificielle dans l'approche distribuée lui procure des qualités d'adaptabilité à des conditions changeantes et incertaines.

Le changement de paradigme (distribué versus centralisé) consiste à aborder un problème d'ordonnancement selon plusieurs points de vue (partiels ou complets) permettant ainsi d'améliorer la qualité de sa modélisation comme l'illustre la figure II. 3

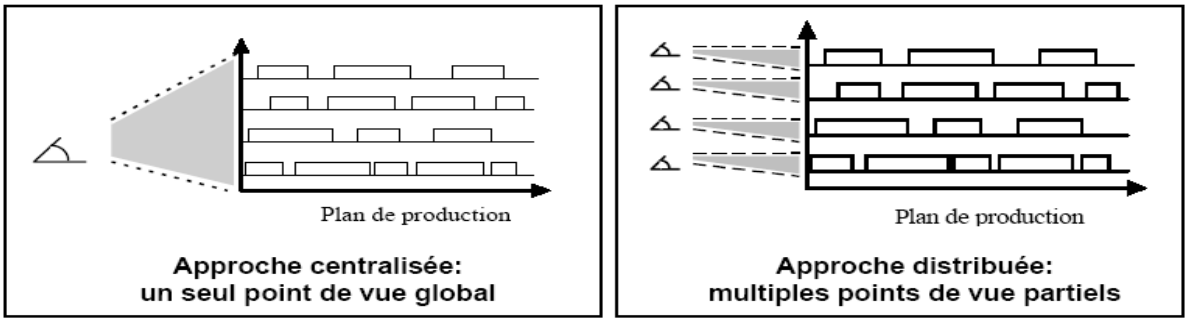


Figure II.3 : Différentes perceptions d'un problème d'ordonnancement [TRA01]

D'autre part, Kalel et al. dans [KPB85], considèrent que l'organisation purement hiérarchique des systèmes de production est un frein à leur rapidité de réaction face aux aléas. Ils lui opposent une approche décentralisée dans laquelle des centres de décision autonomes coopèrent pour gérer des conflits de ressource sur des tâches de production. De la même manière, ce constat a été suivi par l'approche de résolution coopérative de problèmes d'ordonnancement, et ce, pour remettre en cause l'organisation des unités de résolutions jusqu'alors fortement hiérarchisée.

2. Nécessité d'une approche distribuée :

Distribuer c'est avant tout décomposer, c'est à dire identifier dans un problème global un ensemble de sous problèmes partiels. Le processus d'identification est fortement induit par la modélisation initiale du problème global, en conséquence, il ya autant de mode de distribution qu'il y a de possibilités de décomposer un problème d'ordonnancement et d'interpréter cette décomposition [TRA01]. Ainsi, l'objet de la distribution peut être le problème lui-même ou le système réel qu'il modélise. Il convient ensuite de définir les unités de résolution auxquelles vont être allouées les sous problèmes identifiés (agents artificiels et/ou humain), déterminer leur organisation sociale (hiérarchique ou décentralisée)

et leurs modes d'interaction dans la construction d'une solution d'ordonnancement finale (mode de communication, coordination ou coopération, etc.).

Initialement, le domaine de l'Intelligence Artificielle (IA) cherchait à décrire et à résoudre des problèmes complexes identifiés par des experts. Dans ce domaine, il est possible de construire des programmes informatiques capables d'exécuter un nombre important de tâches en centralisant « l'intelligence » au sein d'un système unique. Cependant, l'IA a été confrontée à de nombreux problèmes théoriques, en particulier celui lié à l'explosion combinatoire.

L'apport de l'Intelligence Artificielle Distribuée (IAD) permet d'éviter de manipuler de nombreuses connaissances dans une seule entité en répartissant ces dernières sur plusieurs entités intelligentes. Comme le soulignent Bond et Gasser, l'intelligence Artificielle Distribuée présente les avantages suivants : [BAR03]

• L'IAD est bien adaptée à la distribution de problèmes spatiaux, logiques…etc.

• Les processus distribués entre différents ordinateurs augmentent la vitesse de calcul et de raisonnement.

• Dans certains cas, les systèmes confèrent aux agents individuels des ressources limitées pour résoudre les problèmes, la coopération et la coordination sont essentielles à la résolution de ces problèmes.

Une des branches issue de ce domaine, les Systèmes Multi-Agents (SMA), permet d'introduire dans un système, un ensemble d'individus (ou agents) dotés de connaissances, d'intentions et de capacités d'évolution différentes, ces agents sont capables d'interagir entre eux.

VI. L'approche coopérative :

Nous nous proposons dans ce travail d'étudier une résolution de ce problème en faisant appel à la coopération.

La coopération est une façon de parvenir à une solution d'ordonnancement globale en respectant l'autonomie décisionnelle de chaque ressource. Elle s'appuie sur la capacité des ressources à coopérer pour aboutir à une solution harmonieuse.

Dans ce qui suit, nous allons détailler les notions relatives à la coopération.

1. Définition :

La coopération est habituellement définie comme une action collective organisée autour d'un ensemble d'acteurs partageant un but commun. En ce sens, elle est souvent considérée comme un moyen de dépasser les frontières de l'action individuelle.

Dans la littérature, on trouve plusieurs définitions du mot « coopération », la définition parue dans [ERS 97] répond bien à notre problématique :

« Coopérer signifie que l'on considère que les décisions ne peuvent être prises ou mises en œuvre sans interactions avec autrui ; la coopération indique que l'action nécessite des échanges, des confrontations et des négociations entre différents centres de décision »

C'est donc la coordination entre les centres de décision qui assurera la cohérence des objectifs propres aux centres de décision avec l'objectif global du système coopératif.

2. Les formes de la coopération :

Selon les travaux de [SPO 00] et [CAM 00], il existe trois formes de coopération :

- La coordination : visant à synchroniser les activités individuelles nécessaires à la réalisation d'une activité ou d'une décision globale,
- La collaboration : qui amène plusieurs acteurs à travailler ensemble à l'élaboration d'une activité, Le terme collaboration s'utilise à la place de coopération lorsque les actions individuelles ne sont pas différentiables.
- La codécision : signifie la collaboration de plusieurs acteurs en vue de prendre des décisions. Cette codécision peut être le résultat de mécanismes de négociation ou de renégociation.

3. Le processus de coopération :

Avant de définir le processus de coopération, commençons par définir ce qu'est la négociation ; on reprend la définition donnée par Bussman et Muller dans [BM92]

« Tous les chercheurs s'accordent sur la finalité de la négociation, à savoir l'aboutissement à un accord commun satisfaisant. Mais la négociation est elle-même définie comme un processus. Toute la diversité des recherches en négociation provient de ce mot : processus. La négociation peut donc être vue comme une boîte noire ayant en entrée un conflit et en sortie un accord, dans le meilleur des cas. La recherche sur la négociation consiste donc à étudier les mécanismes de cette boîte noire, pour la rendre transparente. »

Ces auteurs affirme également que si les différentes entités négociantes peuvent seulement accepter ou refuser les propositions, alors la négociation peut être gourmande en temps et inefficace. Pour améliorer l'efficacité du processus de négociation, le destinataire de l'offre doit être capable de fournir un feedback plus utile sur les propositions qu'il reçoit. Ce feedback peut prendre la forme d'une critique ou d'une contre-proposition. Grâce à de tels feedbacks, l'initiateur devrait être en position de générer une proposition qui est plus à même de conduire à un accord.

Le déroulement du processus de négociation est appelé protocole de négociation. [VER04].

Lors de ce processus, trois phases peuvent être distingués :

- la phase de négociation au cours de laquelle le couple de ressources doit s'entendre pour la première fois pour rendre cohérent leur ordonnancement local;
- la phase de coordination au cours de laquelle le couple de ressource s'échange, au fur et à mesure de la réalisation de leur ordonnancement ;
- la phase de renégociation où il s'agit de remettre en cause les décisions prises lors de la phase de négociation.

Un processus de négociation est initié lorsqu'un centre de décision émet une proposition : par exemple une ressource amont veut transférer un produit sur une autre ressource avale. La ressource avale peut refuser ou accepter une proposition ou bien émettre une contre-proposition. De façon générale, une conversation correspond à une suite de propositions/ contre-propositions et s'achève soit par une acceptation, soit par un refus.

Nous distinguons des propositions de deux natures : une proposition pour engagement qui, si elle est acceptée, induit un engagement ferme, et une proposition pour évaluation, qui induit une réponse concernant la faisabilité de la demande souhaitée (sans aucune notion d'engagement). Dans le premier cas, le processus de négociation se termine soit par un engagement (acceptation), soit par un refus. Dans le deuxième cas, le processus de négociation conclut soit à la faisabilité (acceptation), soit à l'infaisabilité (refus). Conclure à la faisabilité d'une proposition n'engage aucun des partenaires, une négociation pour engagement sera tout de même nécessaire. De plus, cette dernière n'aboutira pas nécessairement au même cadre de décision que celui déterminé lors de l'analyse de faisabilité.

La coordination consiste en l'envoi de messages. Ceux-ci peuvent être générés automatiquement. Quatre possibilités s'offrent alors au décideur : envoyer, différer, modifier ou annuler l'envoi d'un message.

Un processus de renégociation est initié lorsqu'une des ressources souhaite modifier un cadre de décision. Pour entamer la renégociation, la ressource émet une proposition.

Le destinataire de cette demande de renégociation peut accepter ou refuser cette proposition ou bien envoyer une contre proposition.

VII. Démarche de résolution de la problématique:

Rappelons notre problématique qui traite de l'ordonnancement en temps-réel sur deux ressources identiques en parallèle. Sachant qu'il existe un nombre précis de tâches à exécuter sur ces ressources (les tâches programmables) selon l'ordre de leurs arrivées. Au cours de cette exécution, d'autres tâches apparaissent aléatoirement (les tâches aléatoires).

L'objectif est de minimiser la durée totale d'ordonnancement, Makespan, ainsi que la maximisation du nombre de tâches aléatoires exécutées.

Ce problème est NP-Difficile au sens fort [Bou 07]. La résolution de ce problème consiste donc à développer et utiliser des approches heuristiques.

Dans notre approche, nous supposons que chaque ressource est assimilée à un centre de décision, qu'elle gère son propre ordonnancement local, et qu'elle dispose donc de sa propre flexibilité décisionnelle. Il s'agit d'une flexibilité séquentielle exprimée par le nombre de séquences dominantes que le théorème des pyramides permet de caractériser sur chaque machine.

Donc notre approche se base sur la décomposition du problème Pm || $C_{\max}$, en deux sous problèmes $1|ri|C_{\max}$. Un ordonnancement local est géré sur chaque ressource disjonctive (ordonnancement à une machine) et l'ordonnancement global résulte d'une coopération entre les diverses ressources.

Pour chaque problème à une machine, l'application du théorème des pyramides permet de déterminer un ensemble de séquences dominantes. La dominance est alors, relative au problème à une machine et non au problème global, Le but principal est alors, d'établir un protocole de coopération qui permet à la fois de préserver l'autonomie décisionnelle de chaque ressource afin qu'elle puisse maximiser le nombre de tâches aléatoires traitées, sans imposer une perpétuelle remise en cause des décisions déjà prises, et d'assurer une performance globale satisfaisante relativement à la minimisation du $C_{\max}$.

1. Décomposition du problème global en plusieurs sous problèmes à une machine :

Comme paru précédemment, la fonction ordonnancement doit être distribuée afin de parvenir à une solution globale résultant d'une négociation entre plusieurs acteurs tout en conciliant les objectifs collectifs et individuels. [MON 05]. Ceci justifie notre choix pour une organisation distribuée basée sur le concept de coopération entre centres de décisions.

La coopération est une façon de parvenir à une solution d'ordonnancement globale en respectant l'autonomie décisionnelle de chaque ressource. D'une part, Il s'agit de trouver un compromis accordant à chaque ressource suffisamment de flexibilité pour pouvoir maximiser le nombre de tâches aléatoires traitées, sans remettre en cause les décisions d'ordonnancements, d'autre part, le compromis trouvé devra assurer une performance globale satisfaisante relative à la durée totale de réalisation ($C_{\max}$) considéré.

Donc dans ce qui suit, le problème d'ordonnancement global est décomposé en plusieurs sous problèmes à une machine ou chaque machine gère son propre ordonnancement local, la méthode

choisie est basée sur le théorème de dominance paru dans les années quatre-vingt et initié par Ershler et al. Nous nous proposons de décrire ce théorème ainsi que les notions relatives à ce théorème.

2. Une approche d'ordonnancement robuste pour l'ordonnancement à une machine :

Dans cette partie, nous nous intéressons aux travaux d'Eschler et al, qui concerne le théorème de dominance qu'est le théorème des pyramides, tout d'abord, le théorème des pyramides a été formulé, en dehors de tout contexte de recherche de robustesse, dans le but premier de limiter la complexité algorithmique liée à tout problème de séquencement.

En 1983, Erschler et al mettent en évidence une nouvelle condition de dominance pour la recherche de solutions admissibles. Un ensemble V= {1,…, n} de n travaux est considéré, chaque travail $i \in V$ étant caractérisé par une date de disponibilité r_i, une date échue d_i, et une durée opératoire p_i, les parties suivantes sont les principaux résultats issus de ces recherches.

3. Notions de bases concernant le théorème des pyramides :

a) Définition d'un ensemble dominant :

D'après la définition d'Esquirol et Lopez, un sous-ensemble de solutions est dit dominant pour l'optimisation d'un critère donné, s'il contient au moins un optimum pour ce critère.

En ordonnancement, le concept de dominance d'un sous ensemble de solutions est précieux pour limiter la complexité algorithmique liée à la recherche d'une solution optimale au sein d'un ensemble de solutions. Un sous-ensemble dominant est tel qu'il garantit l'existence d'au moins une solution optimale, vis-à-vis d'un critère donné, parmi les solutions qu'il contient.

Un sous ensemble de solutions dominantes est caractérisé à l'aide de conditions de dominance. Mettre en évidence des conditions de dominance efficaces, vis-à-vis de la résolution d'un problème d'optimisation, conduit généralement à la conception de méthodes de résolution elles mêmes très efficaces.

Pour caractériser l'ensemble de séquences dominantes, les notions de structure d'intervalles, de sommet et de bases sont utilisées.

Le corps d'hypothèses que les auteurs ont étudié en vue de la dominance prend en compte uniquement l'ordre relatif des dates de début au plus tôt ri et de fin au plus tard di des travaux à ordonnancer. Les durées opératoires pi ainsi que les valeurs explicites des ri et di ne sont donc pas considérées. Les résultats suivants restent donc valides quelles que soient les valeurs r_i et d_i dans le respect de l'ordre relatif décrit dans le corps d'hypothèses.

Dans ce qui suit, on définit les notions de structure d'intervalles, de sommet et de pyramide.

b) Définition de structures d'intervalles :

Une structure d'intervalles est définie par un couple <I, C> où $I = i_1, i_2 ..... i_n$ est un ensemble d'intervalles et C est un ensemble de contraintes sur $I * I$. Chaque intervalle i_j est définie par un couple de points x_j et y_j tel qu'une relation d'ordre $x_j \prec y_j$ quelconque soit vérifiée.

L'utilisation de structure d'intervalles permet une représentation intéressante des caractéristiques d'un problème. En effet, les sommets, bases et pyramides d'une structure pouvant être définis sans avoir connaissance des valeurs explicites des paramètres d'un problème.

c) Définitions de sommets et de pyramides :

Définition 1 :

Un travail s est dit sommet d'une structure d'intervalle <I, C> si et seulement si

$\forall i \in I$ tel que $r_i < r_s \wedge d_i > d_s$.

Les sommets sont indicés dans l'ordre croissant de leur date de début au plus tôt, ou en cas d'égalité, dans l'ordre croissant de leur date de fin au plus tard. Ainsi, si s_α et s_β sont deux sommets tels que $\alpha<\beta$ alors $r_{s_\alpha} \leq r_{s_b}$ et $d_{s_\alpha} \leq d_{s_b}$.

Si deux sommets possèdent des dates de début au plus tôt et de fin au plus tard identiques, alors ils peuvent être indicés de façon quelconque.

Définition 2 :

Un travail b est dit base d'une structure d'intervalle <I, C> si et seulement si

$\forall i \in I$ tel que $r_i > r_s \wedge d_i < d_s$

Définition 3 :

Une pyramide p_α associée au sommet s_α est le sous-ensemble de travaux $i \in I$ tel que : $r_i \prec r_{s_\alpha}$ et $d_i \succ d_{s_\alpha}$

Les définitions de 1 à 3 sont illustrées sur la structure d'intervalles représentée sur la figure II.4.

On identifie les sommets {C, D, E} qui caractérisent respectivement les pyramides P_c= {A, B, G}, P_D = {G}, P_E= {F, G}.

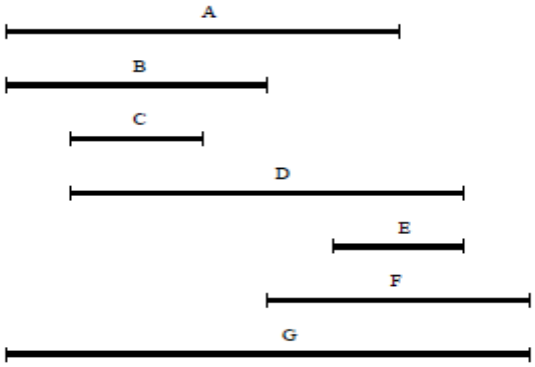


Figure II.4 : Exemple d'une structure d'intervalles.

Un travail non sommet peut appartenir à plusieurs pyramides. Le théorème suivant, appelé Théorème des pyramides, définit alors un ensemble de séquences dominantes, au sens de la recherche de séquences admissibles, pour le problème à une machine avec fenêtres d'exécution.

d) Enoncé du théorème [JEL 05] :

Un ensemble dominant de séquences peut être constitué par les séquences telles que :

- les sommets sont ordonnés dans l'ordre de leur indice ;
- avant le premier sommet, seuls sont placés les travaux appartenant à la première pyramide rangés dans l'ordre croissant de leur date de disponibilité ou, en cas d'égalité, dans un ordre arbitraire ;
- après le dernier sommet, seuls sont placés les travaux appartenant à la dernière pyramide rangés dans l'ordre croissant de leur date échue ou, en cas d'égalité, dans un ordre arbitraire ; entre deux sommets s_k et s_{k+1}, sont placés en premier les travaux appartenant à la pyramide P_k et n'appartenant pas à P_{k+1} dans l'ordre croissant de leur date échue (dans un ordre arbitraire en cas d'égalité), puis les travaux communs aux pyramides P_k et P_{k+1} dans un ordre arbitraire, et enfin les travaux appartenant à la pyramide P_{k+1} et n'appartenant pas à P_k dans l'ordre croissant de leur date de disponibilité (dans un ordre arbitraire en cas d'égalité).

e) Cardinal de l'ensemble dominant :

Une propriété intéressante du théorème des pyramides est qu'il permet d'évaluer la cardinalité de l'ensemble dominant S_{dom} grâce à la formule :

$$Card(S_{dom}) = \prod_{q=1}^{N} (q+1)^{n_q}$$ Où

n_q désigne le nombre d'intervalles appartenant exactement à q pyramides et N le nombre total de pyramides.

Exemple :

Dans le but d'illustrer le théorème des pyramides, nous utilisons l'exemple de Carlier cité dans [TRU 05].

Considérant le problème d'ordonnancement à une machine avec fenêtres d'exécution,

où V= {1,…, n} est un ensemble de n travaux, chaque travail i ∈ V étant caractérisé par une date de disponibilité r_i, une date échue d_i, et une durée opératoire p_i, les dates de début, les dates de fin et les durées opératoires sont indiquées dans le tableau qui suit.

Travail	r_i	d_i	p_i
1	10	44	5
2	13	25	6
3	11	27	7
4	20	30	4
5	30	43	3
6	0	34	6
7	30	51	2

Tableau II.1 : Données du problème.

Le diagramme des intervalles associé à cet exemple est donné sur la figure, ainsi que la décomposition en pyramides correspondante. Les durées opératoires sont indiquées par des rectangles, les sommets sont foncés.

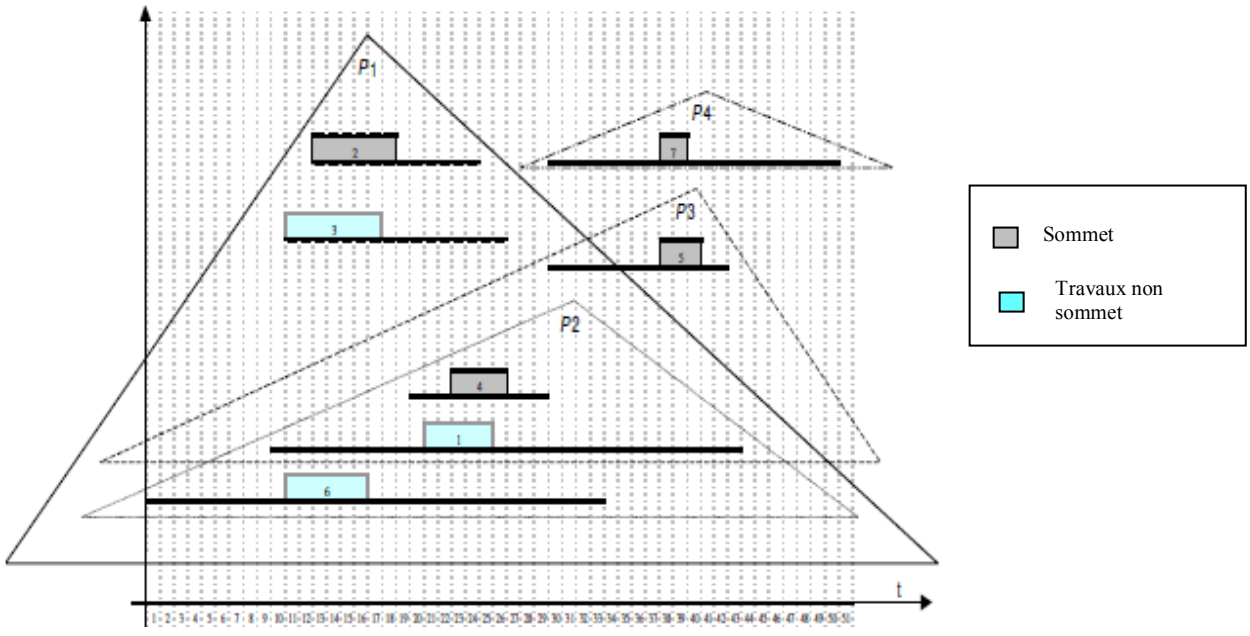


Figure II.5 : Diagramme des intervalles, sommets et pyramides [TRU 05].

On distingue quatre sommets : s1 = 2, s2 = 4, s3 = 5 et s4 = 7, caractérisant les quatre pyramides : P1 = (1; 3; 6 ; 2) P2 = (1; 6 ; 4) P3 = (1 ; 5) et P4 = (7) ;

La cardinalité de l'ensemble des séquences dominantes est calculée par la relation $Card(S_{dom}) = \prod_{q=1}^{N} (q+1)^{n_q}$ et elle est égale à :

$\text{Card(Sdom)} = (1+1)^1 * (2+1)^1 * (3+1)^1 = 24$

Notons qu'il y a au plus n! Séquences à examiner pour trouver une solution admissible, le nombre de séquences possibles est de 7!= 5040.

Les vingt-quatre séquences dominantes sont :

(6, 1, 3, 2, 4, 5, 7) ; (1, 3 , 2, 6, 4, 5, 7) ; (1, 3, 2, 4, 6, 5, 7) ;
(6, 1, 2, 3, 4, 5, 7) ; (1, 2 , 3, 6, 4, 5, 7) ; (1, 2, 3, 4, 6 , 5, 7) ;
(6, 3, 2, 1, 4, 5, 7) ; (3, 2, 6, 1, 4, 5, 7) ; (3, 2, 1, 4, 6 , 5, 7) ;
(6, 2, 3, 1, 4, 5, 7) ; (2, 3, 6, 1, 4, 5, 7) ; (2, 3, 1, 4, 6, 5, 7) ;
(6, 3, 2, 4, 1, 5, 7) ; (3, 2, 6, 4, 1, 5, 7) ; (3, 2, 4, 6, 1, 5, 7) ;
(6, 2, 3, 4, 1, 5, 7) ; (2, 3, 6, 4, 1, 5, 7) ; (2, 3, 4, 6, 1, 5, 7) ;
(6, 3, 2, 4, 5, 1, 7) ; (3, 2, 6, 4, 5, 1, 7) ; (3 , 2, 4, 6, 5, 1, 7) ;
(6, 2, 3, 4, 5, 1, 7) ; (2, 3, 6, 4, 5, 1, 7) ; (2, 3, 4, 6, 5, 1, 7) ;

Cet ensemble est schématisé par la figure suivante :

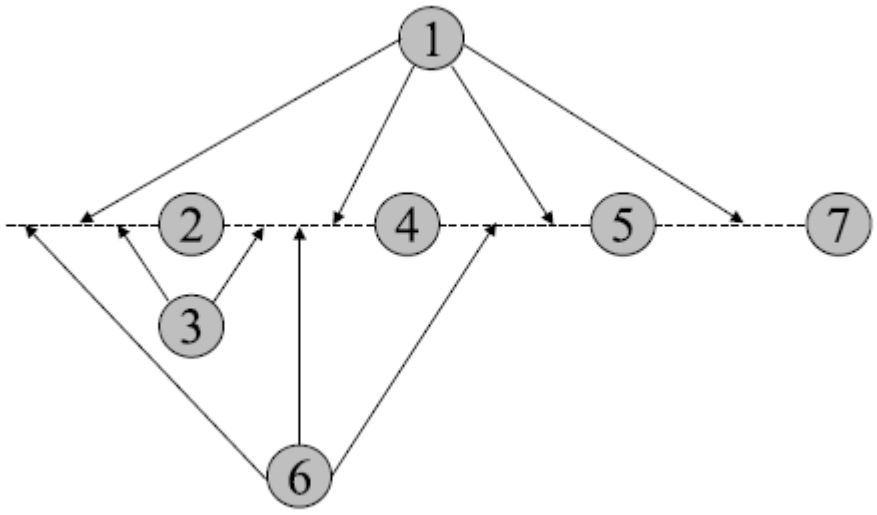


Figure II.6: Séquences engendrées par le théorème des pyramides [TRU 05]

4. Théorème des pyramides et minimisation du retard algébrique (L max)

Dans ce qui suit, nous allons démontrer l'intérêt du théorème des pyramides vis-à-vis du retard algébrique $L_{\max}$.

Il est montré dans [EFM 85] que le théorème des pyramides garantit la dominance de l'ensemble des séquences vis-à-vis de l'admissibilité. La dominance est aussi garantie vis-à-vis du plus grand retard vrai Tmax. Cette propriété a été étendue au retard algébrique maximal $L_{\max}$ dans [EBH 03]. Il existe donc toujours une séquence optimale vis-à-vis du retard $L_{\max}$ dans l'ensemble dominant de séquences caractérisé grâce au théorème des pyramides.

Il a été montré que le théorème des pyramides permet d'évaluer la performance d'un ensemble dominant vis-à-vis du retard algébrique L max ;

H.Trung dans [TRU 05] a développé une méthode qui permet de calculer des valeurs L_j^{min} et L_j^{max} correspondant respectivement au pire et au meilleur retard de j parmi toutes les séquences de Sdom.

Le calcul de ces valeurs est basé sur l'évaluation, pour chaque travail, de la séquence la plus favorable et de la plus défavorable vis-à-vis du retard algébrique.

Donc, il possible d'associer à chaque travail i de S_{dom} un intervalle de retard $[L_i^{min}, L_i^{max}]$ où, L_i^{min} (resp L_i^{max}) désigne le pire (resp le meilleur) retard de i parmi toutes les séquence de S_{dom}. Le retard algébrique optimal L_{max} est alors tel que :

$$\underset{i\in V}{Max}\left(L_i^{\min}\right) \le L_{\max} \le \underset{i\in V}{Max}\left(L_i^{\max}\right)$$

Il est donc possible de juger si un ensemble dominant de séquences est acceptable ou non, relativement à la performance au pire. D'autre part, il est montré dans [ETB 07] comment éliminer de l'ensemble dominant les solutions les moins bonnes, de sorte à améliorer les performances au pire.

Les valeurs de L_i^{min} et L_i^{max} permettent de déduire les valeurs des dates de début au mieux et au pire s_i^{min} et s_i^{max} de chaque tâche :

$$s_i^{min} = L_i^{min} + d_i - p_i \quad \text{et}$$

$$s_i^{max} = L_i^{max} + d_i - p_i$$

Les valeurs de L_i^{min} et L_i^{max} peuvent être représenté dans un diagramme qu'on appelle le diagramme des retards, en effet, il est intéressant de disposer d'une représentation visuelle permettant de mettre en évidence la flexibilité et les performances de l'ensemble dominant.

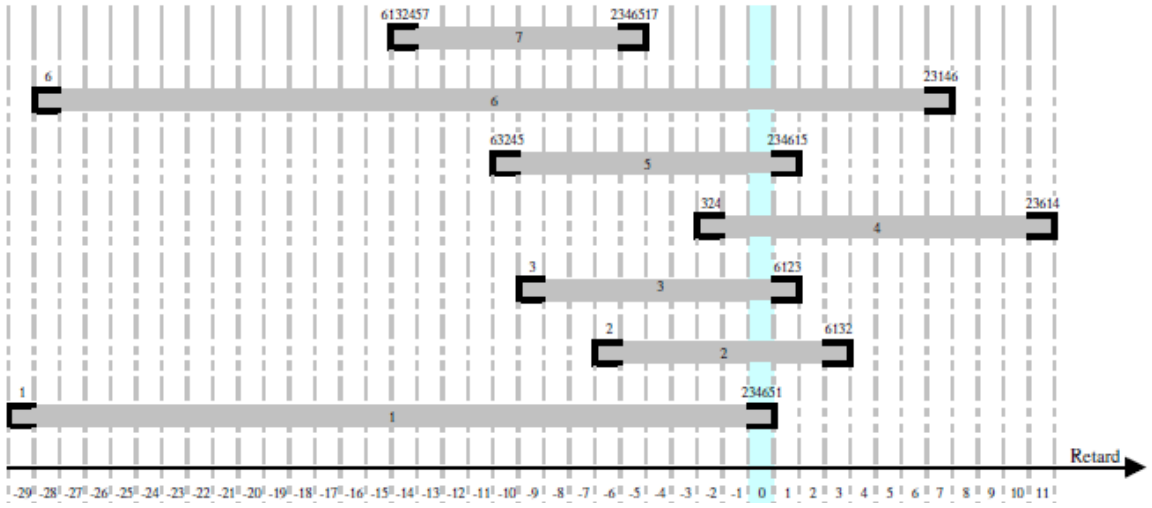


Figure II.7 : diagramme des retards.

Les séquences partielles ayant permis le calcul de chaque valeur min(Li) et max (Li) sont indiqués au dessus de chaque borne. Le diagramme indique que, parmi toutes les séquences caractérisées, le retard algébrique $L_{\max}$ est tel que $\underset{i\in V}{Max}\left(L_i^{\min}\right) \le L_{\max} \le \underset{i\in V}{Max}\left(L_i^{\max}\right)$. On remarque de plus que, quelle que soit la séquence de S_{dom} considérée, les travaux 1 et 7 ne seront jamais en retard.

Ce diagramme permet à un décideur de visualiser les performances associées à l'ensemble dominant de séquences S_{dom} d'un problème, du point de vue du retard algébrique, ainsi que celui de la flexibilité, d'autant plus grande que l'amplitude des intervalles de retard est importante. Dans l'hypothèse ou les dates de fin des travaux correspondent à des délais de livraison (cas de la production à la commande), le décideur pourrait juger si un retard "au pire" est acceptable ou non. Dans la négative, une interaction serait possible, via cette représentation graphique, dans laquelle le décideur demanderait une réduction des retards qui ne lui conviendraient pas. Une procédure d'ordonnancement aurait alors pour objectif d'éliminer, au détriment de la flexibilité, les séquences de l'ensemble dominant qui produisent les retards non acceptables. [EBH 03].

VIII. Conclusion :

Il est intéressant de considérer chaque machine comme une unité autonome capable de résoudre un problème. En effet, ceci réduit considérablement la complexité du problème d'ordonnancement posé.

Nous avons vu l'apport des systèmes distribués et comment la coopération conduit à la caractérisation d'une famille de solutions permettant de faire face aux aléas,

Dans le chapitre suivant, il sera question d'exploiter la flexibilité séquentielle apportée par le théorème des pyramides pour résoudre le problème d'ordonnancement temps-réel sur deux ressources identiques en parallèles de manière distribuée, en faisant coopérer les ressources entre elles. Nous allons voir aussi, comment insérer dynamiquement les tâches arrivant en temps réel « tâches aléatoires ».

Le problème étudié est NP-difficile au sens fort [Bou 07], ainsi, nous avons opté pour une approche empirique moyennant la simulation afin d'évaluer la performance de l'approche proposée. Donc le prochain chapitre, est consacré à l'approche empirique qu'est la simulation et qui sera utilisée, par la suite, pour évaluer la performance de l'approche proposée dans ce travail.

Chapitre III :

Proposition d'une approche heuristique pour la résolution du problème Pm||Cmax.

I. Introduction :

Ce chapitre est consacré à la modélisation de l'approche de résolution proposée dans ce travail.

Ainsi, ce chapitre se décompose comme suit : la première section a pour but de présenter l'approche retenue pour la résolution du problème d'ordonnancement temps-réel sur deux ressources identiques en parallèle, nous illustrons cette dernière par un exemple. Dans la deuxième section, la méthode retenue pour l'évaluation des performances sera présentée, à savoir, la simulation, cette section débutera par, une description des généralités sur la simulation et la modélisation des systèmes de production et s'achèvera par la présentation du langage de simulation retenu, à savoir, ARENA 10. Dans la troisième section, on présentera le modèle ARENA qui reprend le principe de l'approche développée dans ce travail.

II. Proposition d'une approche heuristique basée sur le théorème des pyramides pour la résolution du problème Pm || C max :

Le problème posé consiste à ordonnancer sur plusieurs ressources identiques en parallèle un certain nombre de tâches arrivant aléatoirement « tâches aléatoires », sachant que le système dispose d'un certain nombre de tâches programmables.

Les paramètres qui concernent les tâches programmables (date de disponibilité, durées opératoires et dates échues) sont connus à l'avance, par conséquent, leur ordonnancement est planifié à l'avance. Contrairement aux taches aléatoires qui arrivent aléatoirement dans le temps, leur ordonnancement se fait en les plaçant dans les marges disponibles de l'ordonnancement des tâches programmables en exploitant la flexibilité séquentielle.

L'ordonnancement étant alors adapté de façon réactive, l'approche d'ordonnancement proposée est distribuée, Chaque ressource gère son propre ordonnancement local (ordonnancement à une machine), l'ordonnancement global résultant d'une coopération entre les deux ressources. Les

ordonnancements locaux sont des ordonnancements incorporant de la flexibilité séquentielle grâce au théorème des pyramides, cette flexibilité permet de maximiser le nombre de tâches aléatoires traitées. D'autre part, la fonction ordonnancement doit être distribuée afin de parvenir à une solution globale résultant d'une négociation entre plusieurs acteurs tout en conciliant les objectifs collectifs et individuels [OBB07], donc, notre approche consiste en la décomposition du problème global (Pm|| C_{max}) en deux sous problèmes à une machine (1|ri| C_{max}) ou chaque ressource gère on propre ordonnancement local.

Arrivé à ce stade, deux problèmes d'ordonnancement à une machine sont à résoudre (1|ri| C_{max}). La résolution d'un tel problème s'effectue en deux étapes.
La première étape consiste à résoudre le problème concernant les tâches programmables. Il est connu sous le nom d'ordonnancement statique (la phase Hors-ligne). La seconde, se base sur l'ordonnancement obtenu au niveau de l'étape précédente en incluant les nouvelles tâches imprévues. C'est l'ordonnancement dynamique (la phase en ligne).

L'idée est de générer un ensemble d'ordonnancements calculé durant la phase statique de l'ordonnancement, puis de l'utiliser durant la partie réactive (dynamique), au lieu de proposer une opération sur une machine à un moment donné, il est proposé un ensemble d'opérations possibles à un moment sur une machine. Ainsi, un grand nombre d'ordonnancements peut être proposé. Cette méthode permet de réagir en ligne aux événements imprévus qui apparaissent durant l'exécution, en autorisant le passage d'une solution à une autre, sans remise en cause des calculs déjà effectués.

A l'arrivée des n tâches programmables, on effectue une répartition de ces n tâches programmables sur les deux ressources selon une règle d'ordonnancement, La règle d'ordonnancement choisie est basée sur l'ordre croissant des dates de disponibilité des tâches en respectant les délais, les tâches présentant les mêmes dates de disponibilité sont triées selon leurs dates échues (de la plus petite à la plus grande).
La plus grande date de fin des tâches représente la borne supérieure (BS). Cette borne supérieure BS = C_{max} doit être minimisée.
Une fois la répartition achevée, deux problèmes d'ordonnancement à une ressource sont à résoudre. Sur chaque ressource, il est question d'un ordonnancement statique et d'un ordonnancement dynamique.

1. L'ordonnancement statique (la phase hors-ligne):

L'ordonnancement statique concerne le séquencement des tâches connues a priori sur les ressources. Dans cette étape nous allons faire appelle au théorème des pyramides, tout d'abord rappelons notre approche qui se base sur la décomposition du problème Pm $||C_{max}$, en deux sous problèmes à une machine ($1|ri|C_{max}$). Pour chaque problème à une machine, l'application du théorème des pyramides permet de déterminer un ensemble de séquences dominantes. En effet, pour chaque machine, grâce au théorème des pyramides, on génère un ensemble dominant de séquences, cet ensemble contient l'ensemble de toutes les séquences admissibles, parmi ces séquences on sélectionne les séquences ayant la borne supérieure optimale, c.à.d. les séquences minimisant le Makespan.

Afin d'illustrer l'étape d'ordonnancement statique, considérons l'exemple suivant :

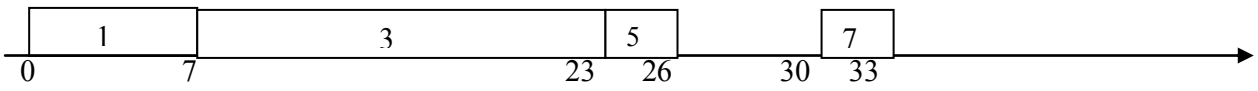
- Exemple :

L'exemple comporte sept taches à exécuter sur deux machines. Les données relatives aux durées opératoires p_i et aux machines M_j utilisées pour exécuter les tâches sont données dans le tableau suivant :

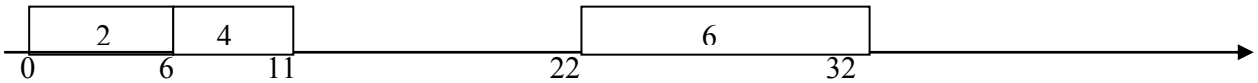
Tâches (i)	1	2	3	4	5	6	7
Ri	0	0	3	0	6	22	30
Pi	7	6	15	5	3	10	3
Di	20	32	35	30	36	35	33

Tableau III.1 : Caractéristiques des tâches programmables.

a) Affecter les tâches aux ressources selon leur Ri croissant :



(a) La ressource 1



(b) La ressource 2

Figure III.1 : (a) et (b) ; Affectation des tâches aux ressources selon Ri croissant.

Suivant cette répartition le C_{max} obtenu pour la première ressource est de 33, tandis que le C_{max} obtenu pour la seconde ressource est de 32, donc la borne supérieure (BS) est égale à 33.

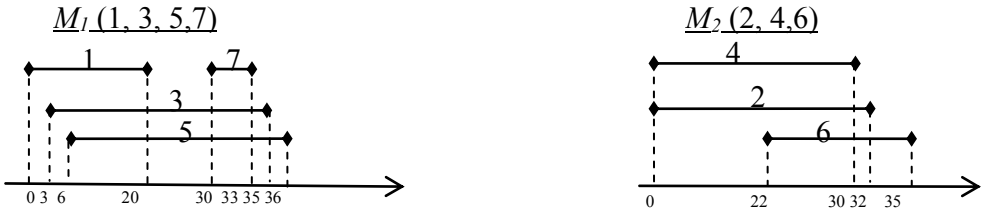
Calcul des dates début au plus tôt et dates début au plus tard [3]:

I	1	2	3	4	5	6	7
Tai	0	0	7	6	22	22	30
Tbi	5	14	12	20	27	25	30

Tableau III.2 : dates début au plus tôt (Tai) et dates début au plus tard (Tbi)

Nous allons maintenant appliquer le théorème des pyramides sur chaque ressource afin de définir l'ensemble de séquences dominantes, par suite nous allons choisir la séquence la plus favorable vis-à-vis du C_{max}, cette séquence nous permettra de commencer l'ordonnancement statique.

b) Application du théorème des pyramides sur chaque ressource pour définir un ensemble de séquence dominantes et choisir la séquence optimale pour l'ordonnancement statique.



(a) La ressource 1 (b) La ressource 2

Figure III.2 : (a) et (b) structures d'intervalles sur les deux ressources.

Nous remarquons que seule la machine M_1 possède une flexibilité séquentielle, puisque la structure d'intervalles qui lui est associée permet de caractériser deux pyramides P1 et P2 :

P1= {3,7} le sommet est {7}

P2 = {5, 7} le sommet est {7}

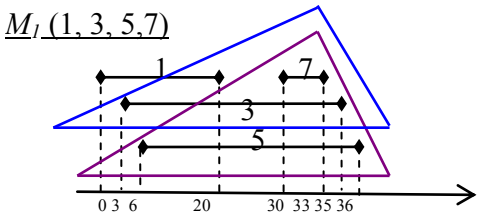


Figure III.3 : deux pyramides P1 et P2.

3 : se référer au travail de [BOU 07] afin de voir la méthode de calcul des dates début au plus tard et au plus tôt

Pour chaque problème à une machine M_k, l'utilisation du théorème des pyramides permet de déterminer l'ensemble des séquences dominantes S_{dom}. L'utilisation de ce théorème sur notre exemple donne les résultats illustrés par le tableau suivant :

M_k	M_1	M_2
S_{dom}	(1, 3, 5,7) ;(1, 3, 7,5) ; (1, 7, 3,5) ; et (1, 5, 7,3)	(2, 4,6)

Tableau III.3 : Séquences dominantes.

En calculant $C_{\max}$ pour chaque séquence, on obtient les durées correspondant à chaque séquence :

Pour la première séquence (1, 3, 5,7) $C_{\max} = 33$

Pour la deuxième séquence (1, 3, 7,5) $C_{\max} = 36$

Pour la troisième séquence (1, 7, 3,5) $C_{\max} = 51$

Pour la quatrième séquence (1, 5, 7,3) $C_{\max} = 48$

Nous déduisons que $33 \leq C_{\max} \leq 51$. On note que l'ordonnancement optimal possède un makespan de $C^*_{\max} = 33$, qui correspond à la première séquence (1, 3, 5,7).

Donc la séquence admissible sur la première ressource est P1= {1, 3, 5,7}

La deuxième machine ne possède pas de flexibilité séquentielle car il n y a pas de pyramides :

La séquence admissible est P2 = {2, 4,6} et son $C_{\max}$ correspondant est égale à 32.

Ces deux séquences ne sont pas définitives, elles seront modifiées ultérieurement, au cours de l'ordonnancement dynamique.

Donc, pour cet exemple la borne BS= $C_{\max}$ = {33}

En résumé, La première étape consiste à générer un ensemble de séquences dominantes, de calculer pour chaque séquence la borne supérieure et de sélectionner l'ensemble de séquence qui minimise le $C_{\max}$, parmi cet ensemble, une séquence est choisie pour le début de l'ordonnancement, dans la deuxième étape, il s'agit d'insérer les tâches aléatoires parmi les tâches programmables en exploitant la flexibilité séquentielle que le théorème des pyramides propose afin de maximiser le nombre de tâches aléatoires traitées.

2. l'ordonnancement dynamique (la phase en ligne) :

Le principe est d'utiliser le programme d'exécution de l'ordonnancement statique et d'y insérer les tâches aléatoires parmi les tâches programmables tout en respectant l'objectif de production.

Partant du séquencement statique, un nouvel ordonnancement est généré à chaque fois, en effet, les tâches aléatoires arrivant dans le temps sont incluses parmi les tâches programmables en temps réel si elles vérifient certaines conditions.
La démarche proposée donne la priorité à la tâche aléatoire dans l'utilisation des marges disponibles.
La tâche aléatoire doit vérifier certaines conditions pour être ordonnancée :

- Une contrainte qui concerne son intervalle temporel : $t_{now} + p_j \leq d_j$ (une tâche dont la date échue n'est pas respectée est rejetée)
- Une contrainte liée à l'intervalle d'exécution des tâches programmables : $t_{now} + p_j \leq tb_i$ (l'ordonnancement d'une tâche aléatoire induisant un retard sur l'exécution d'une tâche programmable, ordonnancée au plus tard, est rejetée).

Si toutefois les conditions d'insertion de la tâche aléatoire ne sont pas vérifiées, on permute entre deux tâches programmables consécutives à chaque fois que cela est possible (i.e. si on peut exploiter la flexibilité séquentielle que propose le théorème des pyramides).

Si c'est possible de permuter entre deux tâches programmables, dans ce cas de figure, on définie la nouvelle séquence obtenue et on calcule les nouvelles dates débuts au plus tard de chaque tâche programmable. Pour être exécutée, la tâche aléatoire doit vérifiée les conditions d'insertion vis-à-vis de la nouvelle séquence obtenue.

Si les conditions d'insertion ne sont toujours pas vérifiées, alors on passe à l'autre étape qui consiste à appliquer le théorème des pyramides sur la tâche aléatoire et les tâches programmables restantes, ceci engendre un ensemble de séquences. Parmi ces séquences, on choisit la séquence admissible la plus favorables vis-à-vis du $C_{\max}$, s'il n'existe aucune séquence admissible minimisant le $C_{\max}$, alors, la tâche aléatoire est rejetée (i.e. la tâche ne peut être ordonnancée dans ce cas).
Il est nécessaire d'augmenter la valeur de la borne supérieure (BS) dès l'exécution d'une tâche tâches aléatoires.
La valeur de BS ($C_{\max}$) est augmentée en lui affectant la nouvelle valeur de $C_{\max}$ (celle obtenue sur une ressource en ayant ordonnancé une tâche aléatoire). De ce fait, à partir de cette itération, considérer à chaque fois le nouveau $C_{\max}$ en incluant les tâches aléatoires.

Notre méthode se base d'une part, sur les principales règles de l'algorithme d'ordonnancement temps réel au plus tard dans le cas d'une file d'attente , d'autres part, on s'intéresse à la flexibilité

séquentielle proposé par le théorème des pyramides pour maximiser le nombre de tâches aléatoires et rendre le modèle d'ordonnancement temps réel au plus tard plus performant.

Pour illustrer cette étape, on considère l'exemple précédent ;
On considère l'ensemble des tâches aléatoires dans le tableau suivant :

Tâches aléatoires (j)	1	2	3
Rj	0	6	23
Pj	1	6	7
Dj	5	12	35

Tableau III.4 : Caractéristique des tâches aléatoires.

A tnow=0 la tâche aléatoire A1 intervient
Les deux ressources sont disponibles et les tâches programmables P1 et P2 sont disponibles
On vérifie les conditions d'insertion sur les deux ressources :

$$t_{now} + p_j \le d_j \text{ Et } t_{now} + p_j \le tb_i$$

Pour la Ressource 1 :

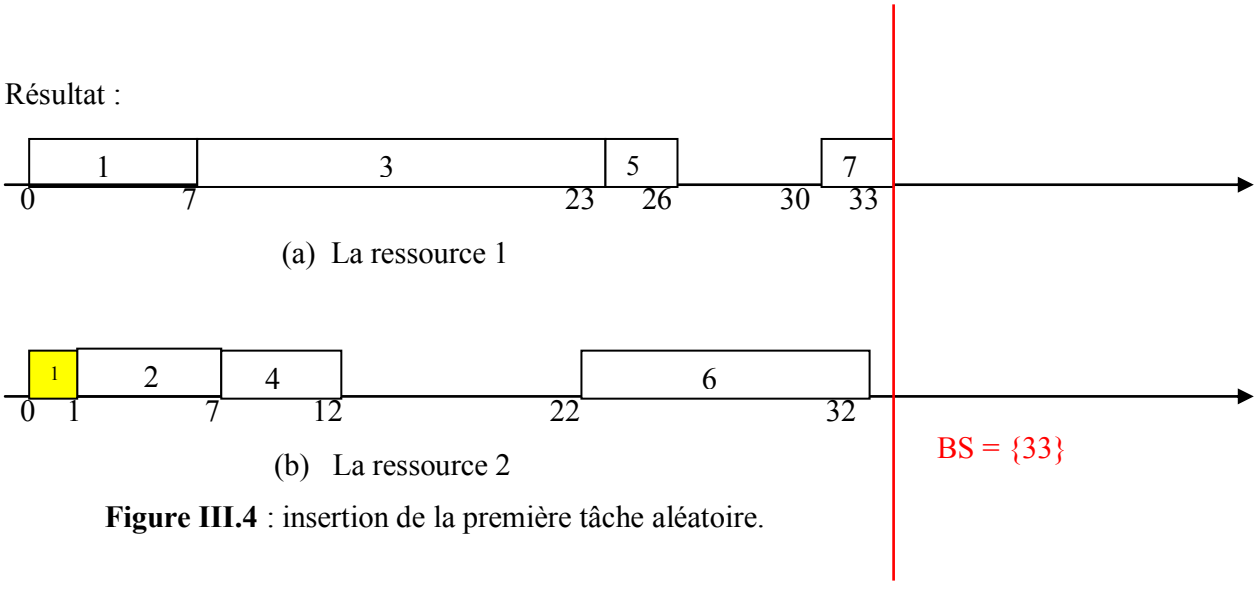
$$\begin{cases} 0 + 1 \le 5 \\ 0 + 1 \le 5 \end{cases}$$

Pour la ressource 2 :

$$\begin{cases} 0 + 1 \le 14 \\ 0 + 1 \le 5 \end{cases}$$

Les conditions d'insertion sont vérifiées pour les deux ressources, par conséquent la tâche A1 peut être exécutée sur les deux ressources, toutefois, on préfère ordonnancer cette tâche sur la ressource 2, on aura un décalage d'une unité sur deux tâches programmables tandis que si on l'ordonnance sur le première ressource, on aura un décalage d'une unité sur trois tâches programmables.

Résultat :



(a) La ressource 1

(b) La ressource 2

Figure III.4 : insertion de la première tâche aléatoire.

A tnow = 6 ; la tâche A2 intervient, les deux ressources sont occupées, dans ce cas de figure, la tâche aléatoire est en attente dans la file d'attente tâche aléatoire.

A tnow = 7 : les deux ressources se libèrent, on vérifie les conditions pour les deux ressources :

Pour la Ressource 1 :

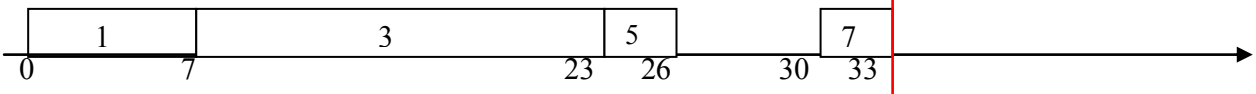
$$\begin{cases} 7 + 6 \le 12 \\ 7 + 6 \le 12 \end{cases}$$

Pour la ressource 2 :

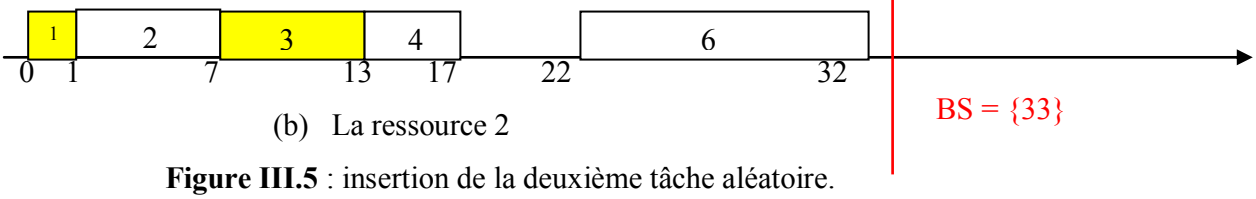
$$\begin{cases} 7 + 6 \le 20 \\ 7 + 6 \le 12 \end{cases}$$

Les conditions ne sont pas vérifiées pour la première ressource, donc la tâche aléatoire A2 est ordonnancée sur la deuxième ressource,

Résultat :



(a) La ressource 1



(b) La ressource 2

Figure III.5 : insertion de la deuxième tâche aléatoire.

A tnow = 23 la tâche aléatoire A3 intervient ;

La ressource 2 est occupée, la ressource 1 est libre, donc on vérifie les conditions d'insertion sur la ressource 1 uniquement : $\begin{cases} 23 + 7 \le 25 \\ 23 + 7 \le 35 \end{cases}$

Les conditions d'insertion ne sont pas vérifiées, la tâche aléatoire A3 ne peut être ordonnancée dans ce cas.

Rappelons qu'on dispose d'une flexibilité séquentielle entre les deux tâches programmables P5 et P7, en conséquence, nous pouvons permuter entre ces deux tâches.

La nouvelle séquence admissible est {1 3 7 5} et son C_{max} est égal à 36 ;

Nous recalculons les dates début au plus tôt et au plus tard sur la ressource 1

I	1	3	5	7
Tai	0	7	33	30
Tbi	10	17	33	30

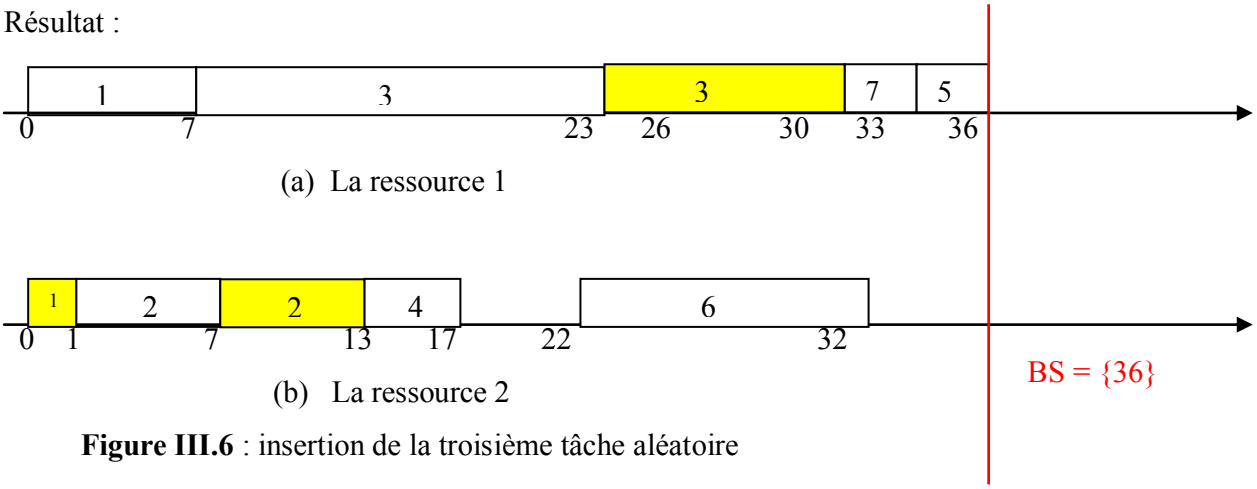
Tableau III.5 : les nouvelles valeurs des tai et tbi

On revérifie les conditions d'insertion de la tâche sur la nouvelle séquence ;

$$\begin{cases} 23 + 7 \leq 30 \\ 23 + 7 \leq 35 \end{cases}$$

Les conditions sont vérifiées sur la nouvelle séquence admissible ; donc on va ordonnancer la tâche aléatoire sur la ressource 1 et selon la nouvelle séquence :

Résultat :



(a) La ressource 1

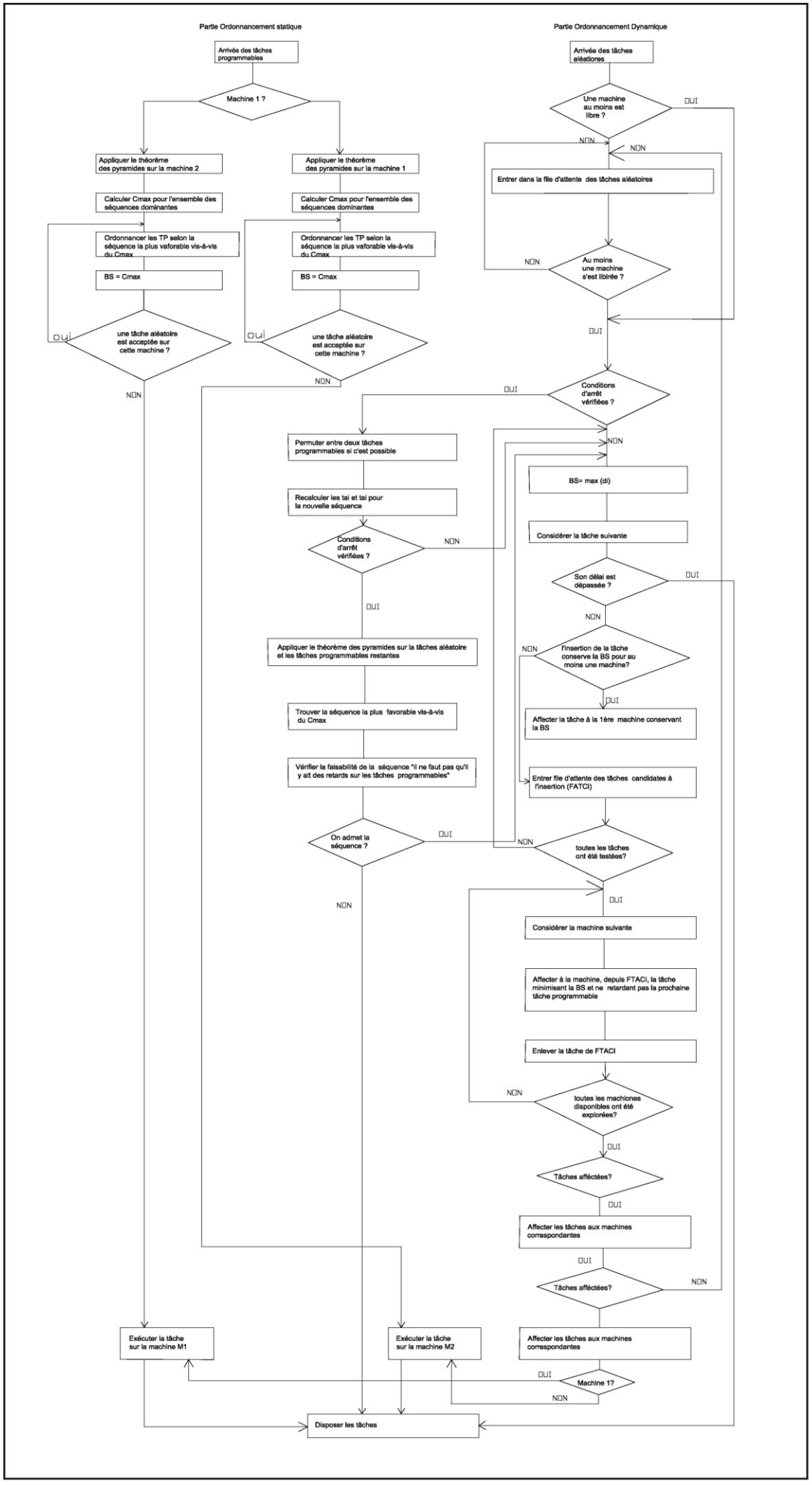
(b) La ressource 2

Figure III.6 : insertion de la troisième tâche aléatoire

Donc les trois tâches ont été insérées et la borne supérieure est égale à 36 et les marges disponibles ont été bien exploitées.

Remarque : dans le cas ou les conditions d'insertion de la tâche aléatoire n'étaient pas vérifiée après le changement de la séquence admissible ou bien, si on ne disposait pas de flexibilité séquentielle, à ce moment là, on pourra passer à l'étape suivante qui consiste à insérer la tâche aléatoire parmi les tâches programmables restantes et d'appliquer le théorème des pyramides sur ces nouvelles tâches afin d'avoir toutes les séquences admissibles et maximiser le nombre de tâches aléatoires.

Les étapes de cette démarche sont schématisées comme suit :


Partie Ordonnancement statique
Partie Ordonnancement Dynamique
Arrivée des tâches programmables
Machine 1 ?
Appliquer le théorème des pyramides sur la machine 2
Appliquer le théorème des pyramides sur la machine 1
Calculer Cmax pour l'ensemble des séquences dominantes
Calculer Cmax pour l'ensemble des séquences dominantes
Ordonnancer les TP selon la séquence la plus vaforable vis-à-vis du Cmax
Ordonnancer les TP selon la séquence la plus vaforable vis-à-vis du Cmax
BS = Cmax
BS = Cmax
oui
une tâche aléatoire est acceptée sur cette machine ?
oui
une tâche aléatoire est acceptée sur cette machine ?
NON
NON
Arrivée des tâches aléatiores
Une machine au moins est libre ?
OUI
NON
NON
Entrer dans la file d'attente des tâches aléatoires
NON
Au moins une machine s'est libirée ?
OUI
OUI
Conditions d'arrêt vérifiées ?
Permuter entre deux tâches programmables si c'est possible
NON
BS= max (di)
Recalculer les tai et tai pour la nouvelle séquence
Considérer la tâche suivante
Conditions d'arrêt vérifiées ?
NON
Son délai est dépassée ?
OUI
OUI
NON
Appliquer le théorème des pyramides sur la tâches aléatoire et les tâches programmables restantes
NON
l'insertion de la tâche conserve la BS pour au moins une machine?
OUI
Trouver la séquence la plus favorable vis-à-vis du Cmax
Affecter la tâche à la 1ère machine conservant la BS
Vérifier la faisabilité de la séquence "il ne faut pas qu'il y ait des retards sur les tâches programmables"
Entrer file d'attente des tâches candidates à l'insertion (FATCI)
On admet la séquence ?
OUI
NON
toutes les tâches ont été testées?
NON
OUI
Considérer la machine suivante
Affecter à la machine, depuis FTACI, la tâche minimisant la BS et ne retardant pas la prochaine tâche programmable
Enlever la tâche de FTACI
NON
toutes les machiones disponibles ont été explorées?
OUI
Tâches affectées?
OUI
Affecter les tâches aux machines correspondantes
OUI
Tâches affectées?
NON
Exécuter la tâche sur la machine M1
Exécuter la tâche sur la machine M2
Affecter les tâches aux machines correspondantes
OUI
Machine 1?
NON
Disposer les tâches

III. Modélisation et simulation de l'approche proposée :

Nous avons vu que le problème étudié est NP-difficile au sens fort (dans [Bou 07]), ainsi, nous avons opté pour une approche empirique moyennant la simulation afin d'évaluer la performance de l'approche proposée. Cette partie est consacrée à des généralités sur la simulation et à la présentation du langage de simulation retenu, ARENA 10.

1. La simulation :

La simulation est l'un des plus puissants outils d'analyse des systèmes complexes.
Aujourd'hui, elle est devenue indispensable pour résoudre les problèmes d'optimisation des flux physiques ou des flux d'informations dans les systèmes de production manufacturiers.
La simulation permet de reproduire le fonctionnement du système réel sur ordinateur et de comparer des scénarios d'exploitation. On peut de cette manière, tester l'impact d'un investissement, d'une modification de paramètre, du lancement d'un nouveau produit sur une ligne de production, d'une autre règle d'ordonnancement sur la performance du système étudié. Cette technique peut aider à mieux cerner les conséquences des choix potentiels d'actions, afin de mieux les maîtriser et d'améliorer ainsi le pilotage de l'atelier.

1.1. Définition de la simulation :

La simulation est un processus qui consiste à :

- Concevoir un modèle du système (réel) étudié,
- Mener des expérimentations sur ce modèle (et non pas des calculs),
- Interpréter les observations fournies par le déroulement du modèle et formuler des décisions relatives au système.

Une synthèse de différentes définitions de la simulation est proposée dans [BAK96] : « la simulation est une méthode de mesure et d'étude consistant à remplacer un phénomène ou un système à étudier par un modèle informatique plus simple mais ayant un comportement analogue ». Le but est de comprendre le comportement dynamique du système, de comparer des configurations et d'évaluer différentes stratégies de pilotage.
La simulation permet de répondre à la question « que se passe-t-il si ... ? » et non pas à la question « qu'est ce qu'il faut faire pour ? ». Simuler un système revient à imiter son comportement afin de mesurer sa réponse (output) à différents intrants (inputs) en fonction du temps. La simulation requiert donc une première étape de modélisation du système réel. Ce modèle abstrait est ensuite traduit ou interprété afin de générer un code informatique respectant la logique du modèle.

Le logiciel résultant est soumis aux différentes expériences ou scénarios préparés par le concepteur et permet, après une phase d'analyse des résultats, d'inférer des connaissances quant au comportement du système et d'évaluer les opportunités de pilotage les plus profitables.
La modélisation constitue donc le fondement même de la simulation. La réciproque n'est d'ailleurs pas forcément vraie. Effectivement, la modélisation n'a pas forcément pour finalité la simulation [BER00]. Il existe des techniques de modélisation qui ont pour objectif l'amélioration de la compréhension du système étudié, l'analyse de son organisation, sans pour autant effectuer de simulation. Mais la modélisation est la première phase essentielle dans la simulation, c'est pourquoi, nous proposons le rappel suivant sur la notion de modèle.

1.2. Définition de la modélisation :

« La modélisation est un processus de représentation qui permet d'obtenir une image approchée du système réel, suite à une phase d'abstraction, exprimée dans un langage de représentation » [BER00]. Le modèle permet une description structurelle du système réel, d'une part, et représente ainsi un substitut du système réel afin d'en analyser le comportement dans des conditions variées, d'autre part. Le modèle n'est probablement pas la solution d'un problème, mais il donne quelques idées pour l'aborder plus intelligemment. Il peut concerner un système existant ou à concevoir : on parlera alors de modélisation « a posteriori » dans le premier cas (tel que : nouvelle reconfiguration, test d'ordonnancement, ...), et de modélisation « a priori » dans le second (tel que : dimensionnement, agencement, évaluation, ...).
Voici les différents points qui doivent être abordés lors de la modélisation :

- Définir l'objectif de la modélisation (lié au cahier des charges) : Pourquoi modélise-t-on ? Qu'étudie-t-on ? Que veut-on améliorer, ou faire ?
- Définir les éléments du système (via la réalisation d'une fonction, ou d'un processus) et les limites du système (les entrées, les sorties).
- Définir les interactions entre ces éléments (hiérarchie).
- Définir la dynamique du système (entités qui circulent entre les éléments, comportement du système au cours du temps).
- Abstraction (choisir les éléments du système pertinents pour l'étude).
- Formalisation, conceptualisation : Modèle mathématique (algèbre (max, +), chaînes de Markov), modèle logiciel (Simulink, Siman-Arena), modèle graphique (réseaux de Petri).

1.3. Les étapes du processus de la simulation:

La conduite d'une étude de simulation comprend trois étapes principales :

Etape 1 : Analyse du problème

Le but est de construire un modèle valide qui soit le plus simple possible, tout en restant cohérent avec les objectifs de l'étude, il faut donc tout d'abord formuler explicitement ses objectifs et les divers scénarios à étudier.

Etape 2 : Construction du modèle

Elle comprend la modélisation logico-mathématique (choix des entités et des attributs, modélisation de l'évolution du système) et la programmation proprement dite. Il est important dès cette étape de construire un programme facilement modifiable, cette étape se termine par une validation qui consiste à comparer le comportement du modèle avec celui du système physique qu'il est censé représenter.

Etape 3 : Exploitation du modèle

Quand le modèle est validé, il peut servir à l'évaluation du comportement dynamique du système. Cette phase nécessite une définition précise des objectifs (quelles hypothèses veut-on vérifier ?, dans quel contexte ?) si les objectifs poursuivis n'ont pas été atteints, de nouveau scénario sont proposés et testés jusqu'à satisfaction. Par la suite, cette phase aboutit à l'analyse des résultats et à leur interprétation.

Du point de vue management, une étude de simulation est un projet dont le succès nécessite une gestion comme tout autre projet [AIT05]. Passer du problème à la simulation suppose de franchir un certain nombre d'étapes. Pour Pritsker, celles-ci sont au nombre de dix (figure III.7) :

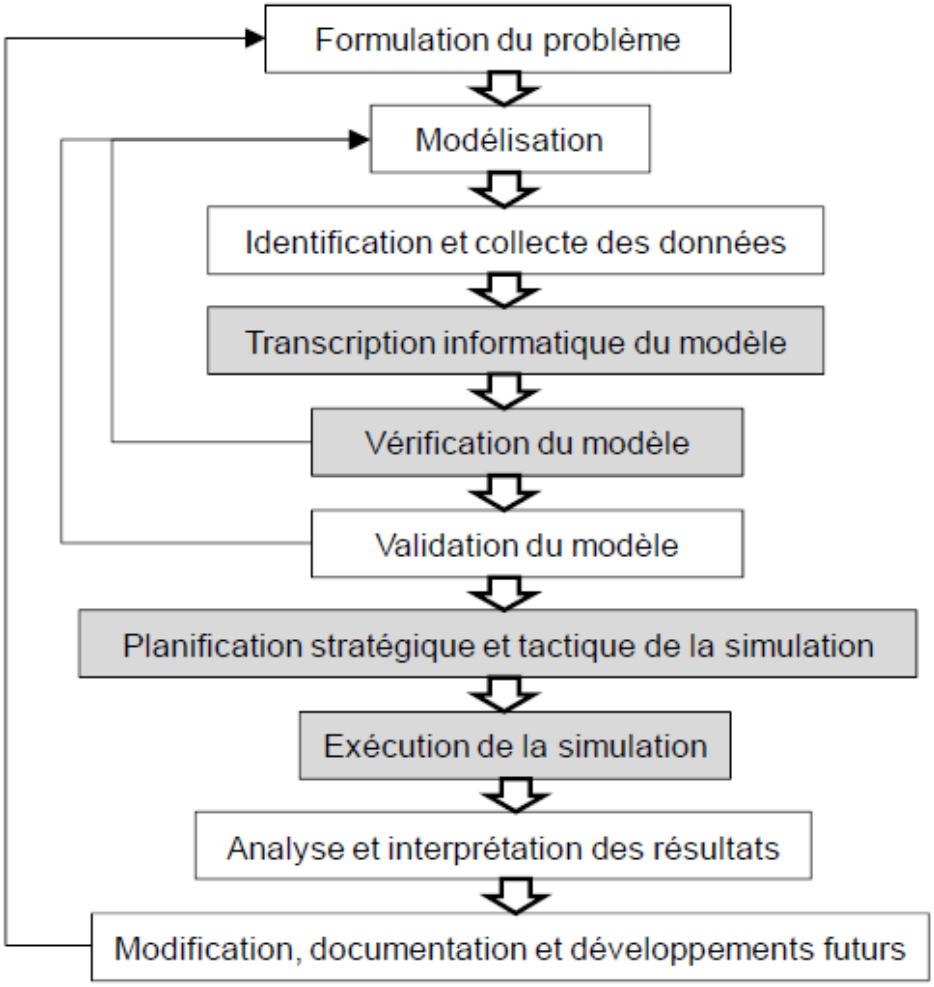


Figure III.7 : Les étapes d'un projet de simulation selon Pritsker [HAB01].

Le schéma de la figure III.7 illustre la façon dont ces différentes étapes s'articulent dans la pratique.

1.4. Les limites de la simulation :

Certaines limites sont dues à la technique elle-même, d'autres sont dues aux produits actuellement disponibles sur le marché (notons qu'une formation sur les logiciels utilisés est souvent nécessaire) [AIT05] :

- La programmation demande un certain niveau d'expertise. La qualité des résultats fournis lors de l'analyse des solutions est liée à la qualité de la modélisation et au savoir-faire du « modélisateur » (la modélisation est un métier).
- La simulation n'est pas une technique d'optimisation au sens propre. Elle ne peut qu'établir les performances d'une solution conçue et imaginée par l'utilisateur. C'est une technique entièrement itérative qui ne propose pas de solution finale mais qui permet seulement à

l'utilisateur d'envisager des choix possibles. En tout état de cause, c'est lui qui devra décider de ce qui répond le mieux aux problèmes posés.

La difficulté liée à la simulation est double :

- Les résultats de simulation sont souvent complexes à interpréter. On étudie des phénomènes aléatoires et les techniques d'analyse demandent de la rigueur ;
- Souvent pour des raisons financières, on doit aller au plus vite vers une solution finale (sans passer trop de temps à explorer d'autres solutions intermédiaires).

2. Présentation du logiciel ARENA 10 :

Pour la modélisation et la simulation nous utilisons le logiciel Siman/Arena, développé par Systems Modeling Corporation. Il est notamment dédié à la modélisation, simulation et animation des systèmes de production. Le logiciel est basé sur les concepts de programmation orientée objet et de modélisation hiérarchique, utilisant la puissance et la flexibilité de modélisation du système Siman/Cinema et le script Microsoft Visual Basic.

Les composants d'ARENA sont programmés en langage SIMAN, ce qui permet d'en créer ses propres blocs, et ce qui en rajoute à la flexibilité du simulateur. Ajoutez à cela, le simulateur comprend des animations graphiques, des courbes et de différents composants d'analyse des résultats.

A l'aide d'ARENA on peut développer un projet complet de simulation. ARENA est un support intégré pour l'analyse des données d'entrée, la construction du modèle, l'exécution interactive, l'animation, la traçabilité et l'analyse des sorties

2.1. Structure d'un programme ARENA :

Un programme (ou modèle logiciel) élaboré avec ARENA est sauvegardé dans un fichier ayant pour extension *.doe* et est constitué :

- d'une partie modèle, qui représente l'algorithme décrivant les caractéristiques statiques et dynamiques des différents blocs fonctionnels composant le modèle ;
- du cadre expérimental, qui regroupe les données précisant les paramètres spécifiques à une simulation donnée (conditions initiales, durée de la simulation, capacité des files d'attente, nombre de ressources,…).

Les deux fichiers générés par SIMAN-ARENA (au format *txt*) sont accessibles *via* le menu *Run/SIMAN/View.*

2.2. Eléments d'un modèle de simulation :

Entité : Une entité est un objet qui évolue dans les différents blocs fonctionnels constituant le modèle du système (une personne ou une pièce dans un atelier).

Attribut : Un attribut est une variable locale associée individuellement aux entités pour représenter leurs états ou des paramètres qui leur sont propres. (Type _ de _ piece afin de désigner le type d'une pièce) ;

Variable globale : Contrairement aux attributs, les variables globales n'appartiennent pas à une entité spécifique, mais à tout le système. Par exemple, la variable *TNOW* (variable prédéfinie dans SIMAN) désigne la date à laquelle se trouve la simulation, c'est le temps courant - mis à jour à chaque avancée dans l'échéancier des événements – s'écoulant durant une simulation du modèle.

Les événements : un événement est une action d'une durée nulle suite à laquelle les valeurs des attributs ou des variables changent. (L'entrée d'une nouvelle entité dans le système, une panne …etc.)

Ces événements sont stockés dans un calendrier d'événements « event calendar », ils sont triés selon leurs dates d'occurrence.

Les ressources : Une ressource est affectée à une entité, afin de la transformer, de lui rajouter de la valeur ou juste pour la faire attendre. Une ressource peut être une personne, une machine, etc.

Les files d'attente : Une entité doit attendre dans une file d'attente lorsqu'elle doit attendre la libération d'une unité appartenant à une ressource.

Le principe de fonctionnement du logiciel ARENA est de suivre chacune des entités évoluant d'un bloc fonctionnel vers un autre dans le modèle, de sa création à sa destruction.

Quand une entité est introduite dans un bloc fonctionnel, elle déclenche le « service » qui lui est associé, ce qui provoque une modification de l'état du modèle.

2.3. Utilisation du VBA dans ARENA :

Le logiciel de simulation ARENA intègre le langage de programmation Visuel Basic VBA, en utilisant les blocs VBA : à chaque fois qu'une entité passe par le bloc VBA, la procédure programmée est effectuée. En cliquant directement sur le bloc VBA, l'éditeur VBA s'ouvre automatiquement afin de permettre la génération du code.

Les primitives du VBA sont reliées aux différents évènements apparaissant lors d'une simulation ou même avant le début de celle là.

Les évènements sont classés en trois grandes familles (toutes les fonctions commencent par *ModelLogic*) :

- Les évènements d'avant exécution de la simulation : DocumentOpen, DocumentSave…

- Les évènements d'initialisation de la simulation : RunBegin, RunBeginSimulation…
- Les évènements lors de la simulation : VBA_Block_Fire, OnKeyStroke…

1. RunBegin
2. ARENA teste et initialise le modèle
3. RunBeginSimulation
4. RunBeginReplication
5. ARENA exécute la réplication
OnKeyStroke
UserFunction
6. RunEndReplication
7. RunEndSimulation
8. ARENA termine la simulation
9. RunEnd

Figure III.8 : Procédure du VBA.

ARENA fait automatiquement appel à chacune de ces procédures lors de la phase de simulation, si aucun code n'est spécifié pour un événement donné, aucune action n'est exécutée ; ARENA se comporte comme si l'événement en question n'existait pas [KEL 03].

IV. Modélisation de l'approche proposée sous ARENA

Cette section est consacrée à la présentation du modèle que nous avons développé sous ARENA:
Comme cité précédemment, l'algorithme collaboratif suppose la distribution du problème global en plusieurs sous-problèmes. Rappelons que notre méthode suppose la distribution de la fonction ordonnancement. Chaque ressource gère son propre ordonnancement local et l'ordonnancement global résulte d'une coopération entre les deux ressources.
Donc notre modèle reprend le fonctionnement du modèle collaboratif présenté par Behiri et Latrech [BL 09] avec des modifications au niveau des blocks VBA comme on le verra dans ce qui suit.
Rappelons que dans ce travail, on cherche à atteindre deux objectifs :

- Le nombre de tâches aléatoires exécutées est à maximiser
- Le C_{max} de l'ordonnancement est à minimiser.

Les différents attributs et variables définis dans le cadre de cette étude, sont :

- IndexP : cet attribut est alloué à chaque tâche programmable afin d'identifier l'ordre de son apparition.

- IndexA : cet attribut est alloué à chaque tâche aléatoire afin d'identifier l'ordre de son apparition.
- r : attribut spécifiant la date de disponibilité de la tâche.
- pr : attribut spécifiant la durée opératoire de la tâche.
- dl : attribut spécifiant la date échue de la tâche.
- tai : attribut spécifiant le début au plus tôt de la tâche programmable i.
- tbi : attribut spécifiant le début au plus tard de la tâche programmable i.
- mach : $\in \{0, 1, 2, 3\}$. Cet attribut indique sur quelle machine sera traitée la tâche (1 pour la machine 1, 2 pour la machine 2, 0 la tâche est stockée en attendant une prochaine vérification et 3 la tâche est rejetée).
- type : $\in \{1,2\}$. Cet attribut nous permet de distinguer entre les tâches programmables et aléatoires (1 pour les tâches programmables et 2 pour les tâches aléatoires).
- X6 : cette variable nous indique le nombre de tâches aléatoires traitées.
- Makespan : variable indiquant le C_{max}.

Globalement, la configuration de base de notre atelier, reste la même que celle présentée pour le modèle proposé dans [BL 09]. La seule différence est qu'on rajoute à notre modèle une subroutine programmée dans les blocs VBA qui reprend le fonctionnement du théorème des pyramides et qui est appelée à chaque fois que les conditions d'insertion des différentes tâches aléatoires ne sont pas vérifiées.

1. Le modèle ARENA :

Cette section est consacrée à la présentation du modèle que nous avons développé sous ARENA:
Le modèle contient trois parties :

- Une pour la représentation des tâches programmables,
- Une pour la représentation des tâches aléatoires et
- la dernière partie est consacrée au traitement de l'ensemble des tâches (programmables et aléatoires) sur les deux machines.

f) Modélisation des tâches programmables :

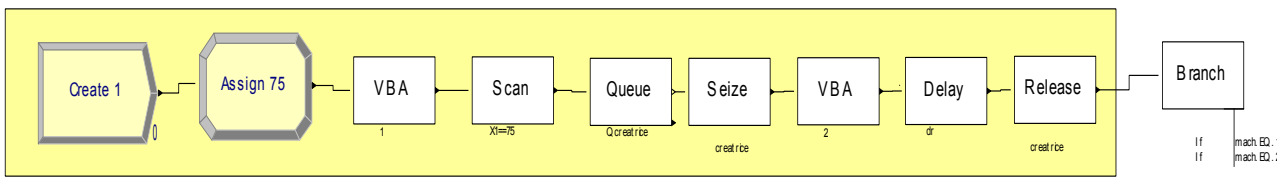


Figure III. 9: Création des tâches programmables.

Le bloc Creat permet de créer les différentes tâches programmables au même instant (t=0), ensuite, grâce au bloc Assign les différents attributs (le type de la tâche, Index P, r, pr, dl, ta, tb) sont affectés aux différentes entités.

Après, à chaque passage d'une entité par le bloc VBA, un programme correspondant au bloc s'exécute, ce programme permet dans notre cas de :

- Calculer les dates début au plus tôt de chaque tâche (tai)
- Créer deux matrices « m1 » et « m2 », contenant les caractéristiques des tâches qui s'exécutent sur la machine 1 et 2 respectivement ;
- affecter les tâches aux machines « m1 » et « m2 »
- calculer les dates début au plus tard (tbi)
- d'appliquer le théorème des pyramides sur chaque machine
- calculer l'ensemble des séquences dominantes
- calculer pour chaque séquence le C_{max} correspondant
- calculer la borne supérieure (BS)

Le bloc Scan a pour but d'attendre que toutes les tâches soient passées du bloc VBA et de les relâcher au même moment.

La séquence de blocs qui suit (Queue, Size, VBA, Delay, Release) a pour mission la simulation de la création des tâches programmables grâce au calcul des durées entre création (dr) calculées auparavant dans le premier bloc VBA et qu'on met dans le Delay.

Le second bloc VBA nous permet d'attribuer les différentes valeurs calculées aux attributs.

On a ensuite un Branch qui sépare les tâches programmables selon qu'elles doivent s'exécuter sur la machine 1, ou sur la machine 2.

Une fois les tâches créées et affectées aux deux machines, on les stocke dans une file d'attente Queue (QM).

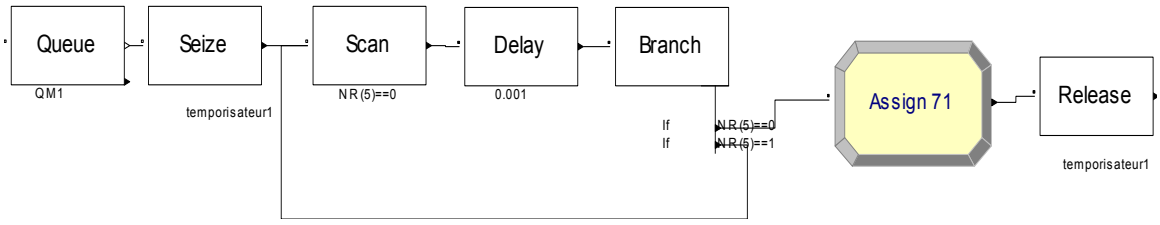


Figure III.10 : modélisation de la file d'attente machine 1 (2)

Les tâches sont en attente dans la file d'attentes QM 1 (QM2), ou elles restent dés qu'une ressource se libère.

Puis on a une combinaison de blocs qui représente un temporisateur et qui ont pour fonction de palier à un problème d'instantanéité, ce temporisateur donne le temps aux tâches aléatoires de vérifier si elles peuvent être exécutées par une machine libre.

Le Branch permet dans le cas où une tâche aléatoire a été acceptée (i.e. : la ressource est occupée et NR ( )=1) de remettre la tâche programmable en attente dans le Scan en attendant la prochaine libération de la ressource. Dans le cas contraire, si aucune tâche aléatoire n'a été acceptée, alors, la tâche programmable est libérée,

g) Modélisation des tâches aléatoires :

Cette partie se décompose en trois parties : création des tâches aléatoires, ensuite attente des tâches pour la vérification et enfin la vérification.

La création des tâches aléatoires :

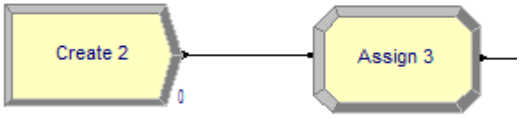


Figure III.11 : création des tâches aléatoires

Comme on n'a aucune donnée à priori sur les tâches aléatoires (à la différence des tâches programmables), les tâches aléatoires sont créées selon une loi de probabilité (EXPO) dont les paramètres sont à déterminer (le r dans ce cas constitue la date de création de la tâche). Les autres caractéristiques de chaque tâche (le type de la tâche, IndexA, pr, dl) lui sont assignées via le module Assign.

Après, un Branch nous permet de faire des tests de vérification sur une nouvelle tâche aléatoire qui vient d'apparaitre dés qu'une des ressources se libère et ou il n y a aucune tâche programmable à exécuter.

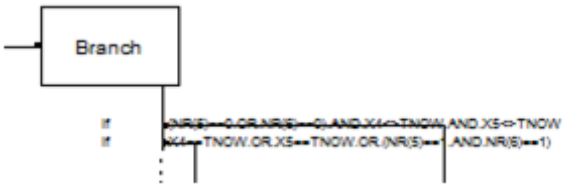


Figure III. 12: séparation des tâches aléatoires qui interviennent alors que les deux machines sont occupées ou pas.

Attente de la vérification :

Après le passage d'une tâche aléatoire dans le Branch, en fonction que les deux ressources soient libres ou pas, les tâches sont dirigées soit vers la file d'attente de vérification, soit vers
La file d'attente tâches aléatoires ou elles restent en attente jusqu'à la libération d'une ressource.

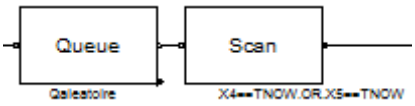


Figure III.13 : file d'attente tâches aléatoires

Vérification des tâches aléatoires :

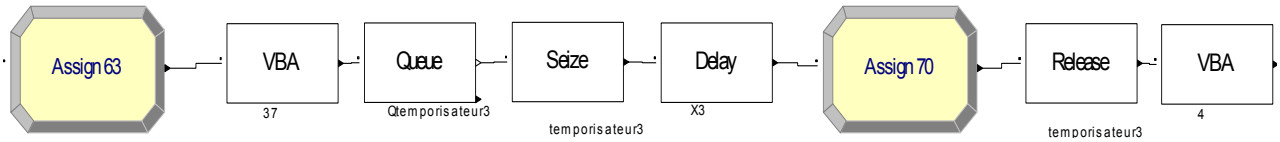


Figure III.14 : modélisation des tests d'insertion des tâches aléatoires

Cette étape nous permet dans un premier temps de déterminer les tâches aléatoires susceptibles d'être exécutées sur la (les) ressource (s) libre (s), dans un second temps, on sélectionne la tâche aléatoire à exécuter selon les critères retenus.

- Rôle du premier bloc :

Le premier bloc VBA a pour fonction le remplissage d'une matrice "M" (figure III.15) avec un certain nombre d'informations sur les tâches aléatoires, Comme informations communes à toutes les machines, la durée opératoire de la tâche aléatoire et sa date échue (ces données sont stockées dans les 2 premières lignes de la matrice "M").

Pour ce qui est de la partie relative à chaque machine (3 lignes consécutives pour chaque machine), on a, l'impact sur la borne supérieure dans le cas de l'exécution de la tâche aléatoire sur la machine concernée. Puis, une ligne est consacrée à la priorité de la tâche concernée quant à son exécution sur cette machine (car, chaque machine libre, classe les tâches aléatoires présentes dans la file d'attente selon un ordre de priorité tel que :

- Les tâches qui peuvent être exécutées sans augmenter la borne supérieure, ni décaler une tâche programmable représentent les tâches de priorité 1.
- Les tâches qui peuvent être exécutées sans augmenter la borne supérieure mais décalent les tâches programmables représentent les tâches de priorité 2.
- Les tâches qui peuvent être exécutées mais augmentent la borne supérieure, font partie des tâches de priorité 3.
- Pour les tâches qui ne peuvent être exécutées, on leur affecte dans cette ligne la valeur 0).

La troisième ligne de la partie relative à chaque machine est réservée pour le décalage induit par l'éventuelle exécution de cette tâche sur la machine concernée.

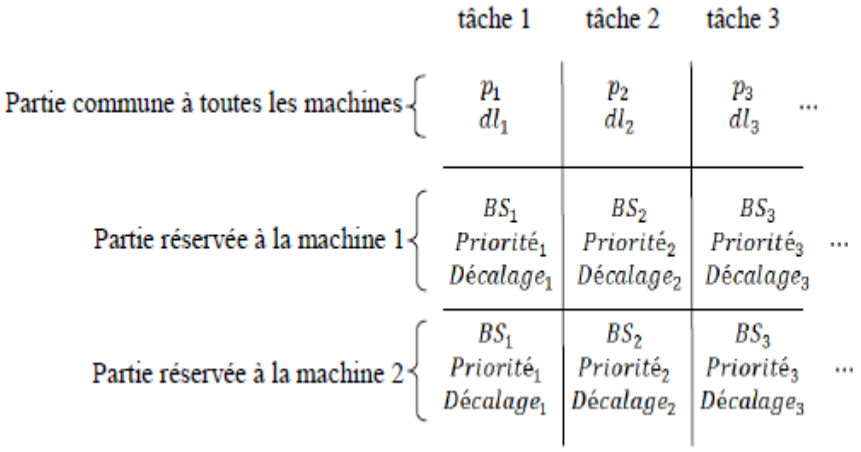


Figure III.15 : la forme de la matrice M.

Pour chaque machine une liste de tâches aléatoires candidates est dressée. Ces tâches sont triées dans la matrice "ma" (figure III.16) selon leurs priorités.
Les tâches aléatoires de la matrice "ma" sont triées selon leur ordre dans la file d'attente.

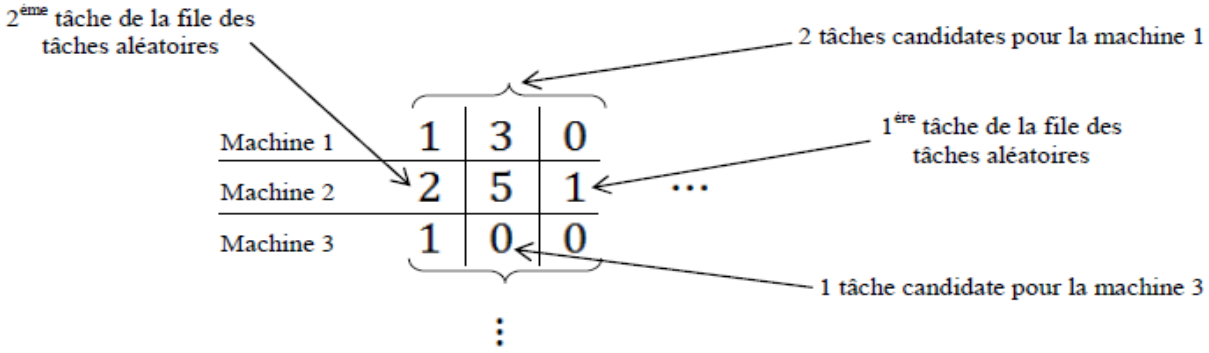


Figure III.16 : la forme de la matrice ma

Le changement important apporté se situe dans la partie tâche aléatoire, dans le deuxième bloc VBA présent au niveau de la partie vérification des tâches aléatoires (figure III.14), dont la fonction est de sélectionner les tâches aléatoires à exécuter sur les machines libres.

- Rôle du deuxième bloc VBA :

Le deuxième bloc VBA a pour fonction, la mise en œuvre de la collaboration entre les machines.

Ce bloc est structuré d'une façon telle que chaque machine libre commence par remplir sa ligne de tâches aléatoires candidates et ceci dans la matrice "ma". Le remplissage s'effectue depuis la matrice "M". Ensuite, chaque machine trie ses tâches selon leurs priorités. Au bout de cette étape, on aura pour chaque machine une tâche aléatoire optimale à exécuter sur cette dernière.

Ces informations sont, par la suite, transmises à la dernière machine qui testera, la combinaison de tâches obtenues pour l'exécution sur chaque machine libre, si cette dernière est compatible avec l'objectif global. Si c'est le cas, la machine initiatrice de la collaboration confirmera pour chaque autre machine l'exécution de la tâche qu'elle a sélectionnée. Dans le cas contraire, il sera demandé aux différentes machines, une à une, de proposer une autre tâche aléatoire pour exécution (qui est la prochaine tâche aléatoire candidate sur la liste). La machine initiatrice aura à confirmer la meilleure combinaison.

Au bout de ce processus, on obtient une combinaison, qui, si elle n'est pas optimale, tend très sensiblement vers cette dernière, grâce aux différents tests auxquels la machine initiatrice fait subir à la combinaison obtenue.

Si la combinaison de tâches obtenues après négociation contient au moins une machine oisive (c.à.d. l'une des priorités est égale à « 0 »), on fait appel au programme qui reprend le fonctionnement du théorème des pyramides sur la machine concernée, ceci nous permettra de modifier la séquence, et de choisir une meilleure séquence. Ensuite, on recalcule la nouvelle matrice M et on relance le processus de collaboration.

Les codes VBA sont présentés en annexe II.

On a après, un Branch, qui oriente les différentes tâches aléatoires selon le résultat de la vérification.

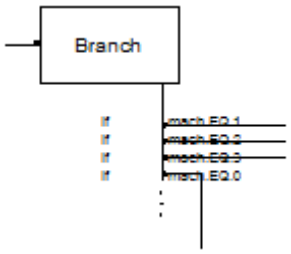


Figure III.17 : modélisation du transfert des tâches aléatoires à exécuter sur les machines

Dans le cas où l'attribut mach, de la tâche aléatoire est égal à 1, cette dernière est affectée à la machine 1.

- S'il est égal à 2, on l'affecte à la machine 2.
- S'il est égal à 3, cela signifie que cette tâche aléatoire ne peut plus être exécutée dans son intervalle d'exécution, ainsi, on l'élimine.

- S'il est égal à 0, cela signifie que la tâche aléatoire n'a pas été acceptée, mais qu'elle peut encore prétendre à être exécutée. Dans ce cas, elle est redirigée vers la file d'attente.

h) Modélisation du traitement des tâches sur les différentes ressources :

Cette partie s'occupe du traitement des tâches aléatoires et programmables sur les deux ressources.

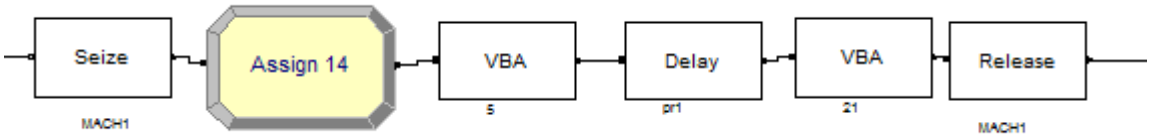


Figure III.18 : traitement des tâches sur la ressource 1 (2)

Dans cette partie, on affecte une tache à une ressource, les deux blocs VBA calculent le nombre de tâches programmables et aléatoires déjà exécutées, et nous permettent de représenter graphiquement l'exécution des tâches.

Une fois la tâche exécutée, on l'oriente vers un module Dispose.

Figure III.19 : destruction des tâches du système.

2. Vérification et validation du modèle :

Avant de tirer des inférences des résultats statistiques d'un modèle de simulation il faut s'assurer qu'il est correct au sens où il représente bien le système. C'est pourquoi, on a soumis le modèle obtenu à une suite de tests, Parmi ces tests nous citons :

- Des techniques d'animation du modèle montrant que la circulation des entités est cohérente avec les règles mises en place ;
- en appliquant le modèle obtenus à des exemples dont les sorties sont connues.
- Utilisation de fichiers de sorties pour analyser les données recueillies au cours de la simulation.
- Petits changements dans les paramètres d'entrée devraient changer les paramètres de sortie.

Par ailleurs, Visual Basic nous permet d'afficher en temps réel durant la simulation, la valeur des attributs calculés au niveau des blocks VBA, ce qui nous permet de garder une traçabilité du fonctionnement du système et de le comparer au fonctionnement souhaité.

V. Conclusion :

Dans ce chapitre, on a vu comment exploité la flexibilité séquentielle apportée par le théorème des pyramides, pour résoudre le problème d'ordonnancement temps-réel à deux machines identiques en parallèle avec minimisation du C_{max} de manière distribuée.

D'autre part, il a été démontré pour la simulation, que la modélisation constitue le fondement même de cette dernière, ce qui nous a mené à la description des différentes étapes de construction d'un modèle. Puis, nous avons présenté ARENA, le langage de simulation utilisé dans ce travail, par suite, nous avons modélisé l'approche proposée sous ARENA.

Dans le chapitre qui suit, la performance de l'approche étudiée dans ce travail sera évaluée, ceci en faisons des expérimentations sur le modèle ARENA.

Chapitre IV :

Simulation et analyse des résultats.

I. Introduction :

Dans ce chapitre, nous mettons en œuvre les étapes restantes du processus de simulation, telles que définies par Pritsker (Chapitre 3), c'est-à-dire la planification stratégique et tactique de la simulation, son exécution et enfin l'analyse et l'interprétation des résultats. Ainsi, la première section du chapitre est consacrée à l'élaboration du plan d'expérimentation pour la simulation, et à la définition des indicateurs de performance, Dans la deuxième section, des expérimentations sur le modèle ARENA seront effectuées et les résultats des différentes expérimentations seront enregistrés.
Dans la troisième section, après l'exécution de la simulation, les résultats obtenus seront examinés et seront comparés avec les résultats des approches précédentes, validées dans [BL 09], ainsi, la quatrième section sera consacrée à la validation des résultats, ceci, grâce à la variation de certains paramètres et la comparaison entre les résultats obtenus pour chaque approche.

II. La planification de la simulation (le plan d'expérimentation):

Le processus de simulation aboutit, après la phase de conception et d'implémentation informatique du modèle, à la phase d'expérimentation et d'interprétation des résultats. Selon l'objectif de l'étude, les paramètres à considérer lors des expérimentations peuvent être très nombreux (le nombre de files d'attente retenue, le nombre de machines, la fréquence d'entrée des pièces dans l'atelier), chacun pouvant avoir une influence sur la performance globale du système étudié et pouvant faire l'objet de variations diverses [GHA06].
Ceci rend nécessaire le passage par une phase de planification des expériences qui permet de rationaliser le nombre d'expériences à mener tout en assurant une couverture suffisante du problème étudié.
Pour ce faire, nous commencerons par arrêter une configuration de base de l'atelier en définissant : le nombre de machines, le nombre de files d'attente et les différents types des tâches. Nous identifierons dans un second temps les règles de pilotage à faire varier. Enfin, nous présenterons les

différents indicateurs retenus pour analyser la performance du système étudié en cohérence avec l'objectif de notre étude.

1. Le choix d'une configuration :

L'atelier qui a été choisit pour la simulation est constitué de :

- Deux types de tâches : programmables et aléatoires.
- Deux machines identiques en parallèles.
- Trois files d'attente. Une file d'attente pour les tâches aléatoires commune aux deux machines et une file d'attente des tâches programmables pour chaque machine.

Ce choix a été défini par [BL 09], ainsi, nous avons choisi la même configuration afin de comparer les résultats de l'approche proposée avec les autres approches.

2. Le plan d'expérience :

Le plan d'expérience planifie de façon rigoureuse les essais (choix des conditions initiales, des facteurs et de leurs niveaux, des variables et indicateurs de sortie), en vue d'un objectif parfaitement défini.

Les différents paramètres de ce problème (r, p, d) seront générés selon les distributions suivantes :

- Pour les tâches programmables : l'arrivée des tâches (r) est poissonnienne, leurs durées opératoires (p) suivent une loi triangulaire et leurs dates échues (d) sont données par la somme de la date de disponibilité, la durée opératoire et une variable aléatoire triangulaire ;
- Pour les tâches aléatoires : même chose que pour les tâches programmables (l'arrivée des tâches (r) est poissonnienne, leurs durées opératoires (p) suivent une loi triangulaire et leurs dates échues (d) sont données par la somme de la date de disponibilité, la durée opératoire et une variable aléatoire triangulaire) ;

Les paramètres qui ont été choisis pour la simulation des différentes expérimentations sont [4]:

Pour les tâches programmables :

- Les arrivées : $r_i = r_{i+1} + EXPO(2)$
- Les durées opératoires : $p_i = TRIA(1,2,4)$
- Les dates échues : $d_i = r_i + p_i + TRIA(1,11,37)$

Pour les tâches aléatoires :

- Les arrivées : $r_i = r_{i+1} + EXPO(2)$
- Les durées opératoires : $p_i = TRIA(1,2,5)$
- Les dates échues : $d_i = r_i + p_i + TRIA(1,3,6)$

Sur la base de ce qui précède, on doit avoir un nombre suffisant d'expériences et des valeurs aléatoires générées pour chaque expérimentation :

On effectue une centaine d'expériences [4], afin d'avoir une première tendance sur la performance de l'approche proposée (Après chaque simulation, les résultats sont enregistrés avant leur analyse).
Le nombre de tâches programmables est de 75, celui des tâches aléatoires est de 60 [4]

3. Les indicateurs de performance :

Pendant la phase de l'analyse des résultats, nous mesurerons la performance de chacune des expériences exécutées à l'aide des indicateurs de performance décrite ci-dessous :

- **Le Makespan ($C_{\max}$)** :

Nous prendrons comme référence pour le $C_{\max}$ pour chaque expérience, celui obtenu dans le cas ou l'heuristique LPT est utilisée sachant que l'application de cette heuristique implique la relaxation d'un certain nombre d'hypothèses (dates d'arrivées des tâches, le respect des dates échues n'est plus considéré), la performance de l'application de cette règle a été évaluée par Graham dans [GRA 69] comme étant égale à :

$\frac{C_{\max}(LPT)}{C^*_{\max}} \leq \frac{4}{3} - \frac{1}{3m}$ Où

($C^*_{\max}$ est la solution optimale proposée par McNaughton [MCN 59], $C_{\max}(LPT)$ est la solution obtenue avec l'heuristique *LPT* et *m* est le nombre de machines).
Cette référence a été utilisée afin d'évaluer la performance des approches de résolution citées dans [BL 09], de même, cette référence sera utilisée afin d'évaluer la performance de notre travail.

- **Le nombre de tâches aléatoires traitées (exécutées).**

Le nombre de tâches aléatoires exécutées sera enregistré pour chaque expérimentation.

III. Simulation et analyse des résultats :

Dans ce sui suit, on va présenter les résultats obtenus lors de la simulation, ensuite, on procède à l'interprétation de ces résultats.

4 : nous utilisons la même série de données proposées par [BL 09] dans le but de comparer les résultats.

Les résultats obtenus à l'issu de la simulation sont :

- **Le** C_{max}

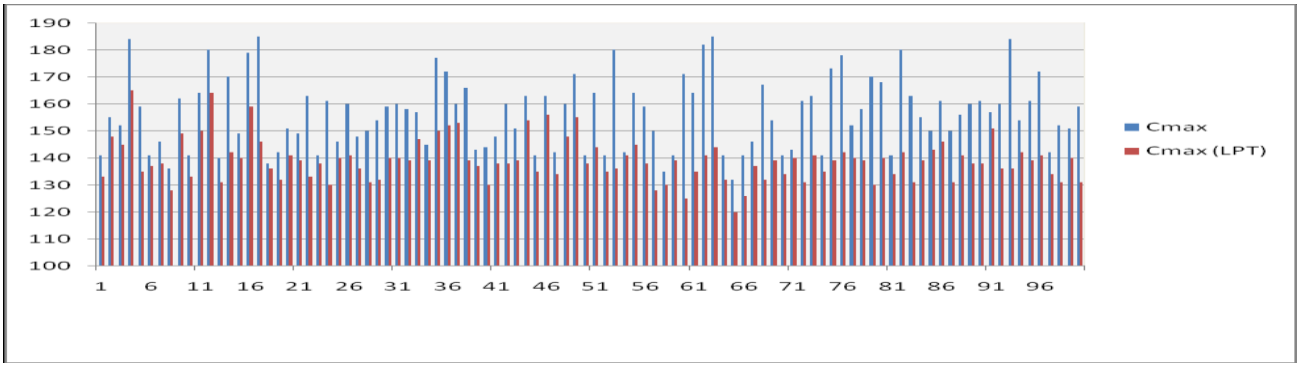


Figure IV.1 : observation du C_{max} obtenu et du C_{max} selon la règle LPT

Le C_{max} obtenu pour cette approche est, pour chaque expérimentation, supérieur à celui obtenu en ordonnançant les tâches exécutées selon la règle LPT. La plus grande valeur du C_{max} a été obtenue lors de la 48ème réplication avec un C_{max} d'une valeur de 185 U.T, quand à la plus petite valeur, elle est de 120, elle a été obtenue lors de la 65ème Réplication. La moyenne du nombre du « Makespan » est de : 156,44 U.T. Pour un écart type de : 13,11.

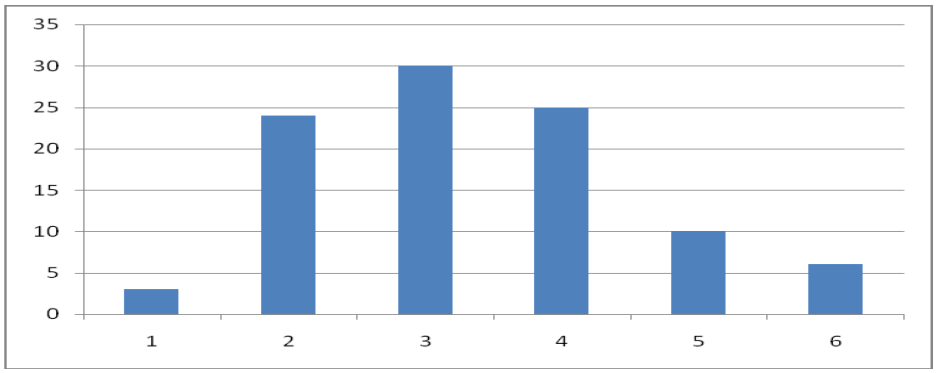


Figure IV.2 : distribution des C max.

En regroupant les C max par classe (figure IV.2) telle que les classes sont :

1 :] 130, 140] ; 2 :] 140, 150] ; 3 :] 150, 160] ; 4 :] 160, 170] ; 5 :] 170, 180] ; 6 :] 180, 190].

Nous remarquons d'après la figure IV. 2 que la distribution des C_{max} suit une Gaussienne.

Le même phénomène a été observé pour les autres approches de résolution validées dans [BL 09], ainsi, on peut comparer les résultats de notre modèle avec les autres modèles en termes de C_{max}.

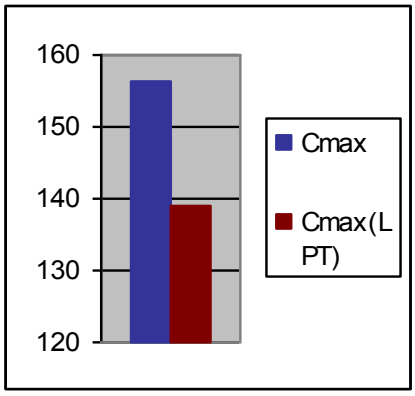


Figure IV.3 : Moyenne du C_{max} et du C_{max} selon la règle LPT

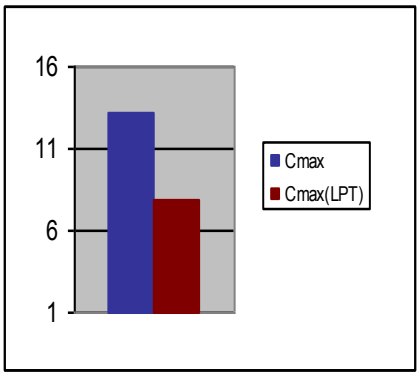


Figure IV.4 : Ecart type du C_{max} et du C_{max} selon la règle LPT.

Nous rapportons le C_{max} à un taux µ, ce taux représente le rapprochement entre le C_{max} trouvé avec l'approche étudié dans ce travail et le C_{max} obtenu avec l'heuristique LPT

$$\mu = ((C_{max} - C_{max}(LPT))/C_{max}(LPT))*100$$

La moyenne du C_{max} obtenu comparé à celle obtenu selon la règle LPT nous donne un taux de 12,5 %, le C_{max} obtenu avec la règle LPT est inférieur à celui obtenu avec l'approche étudié dans ce travail de 17,44 U.T.

L'écart type de la règle LPT est deux fois moins important que l'écart type obtenu avec l'approche étudiée dans ce travail, l'écart est de 5,28 U.T.

- **Le nombre de tâches aléatoires traitées :**

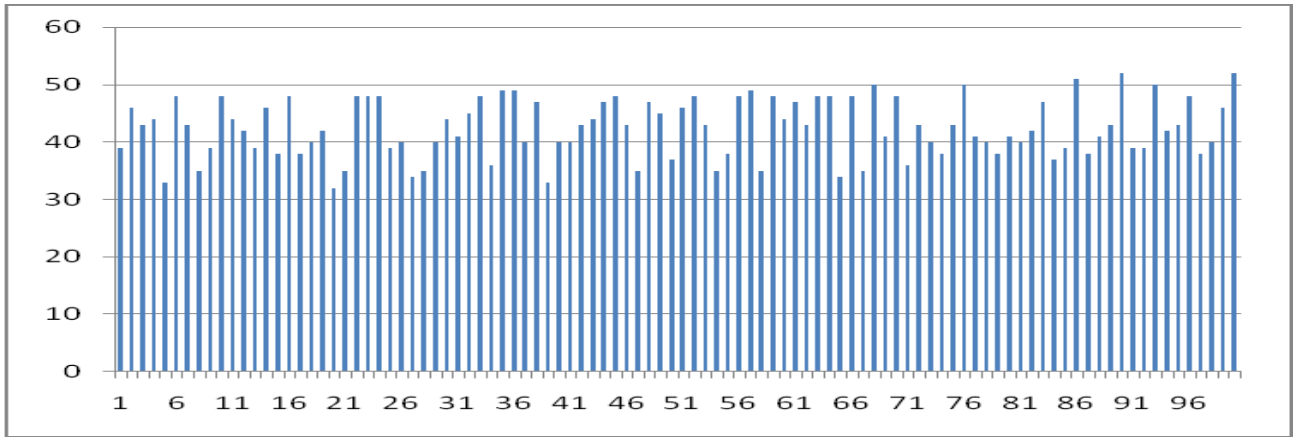


Figure IV.5 : le nombre de tâches aléatoires traitées.

La figure IV.5 représente le nombre de tâches aléatoires exécutées lors de chaque expérience.
On remarque une amélioration importante, la distribution des nombres de tâches aléatoires traitées est très dense vers le haut, le nombre maximum de tâches aléatoires exécutées a été constaté lors de la 90 et 100ème expérience avec 52 tâches exécutées, quant au minimum, il est de 32, il a été obtenu lors de la 20ème expérience. La moyenne du nombre de tâches aléatoires traitées est de : 42,48 tâches aléatoires. Pour un écart type de : 5,01 tâches aléatoires.

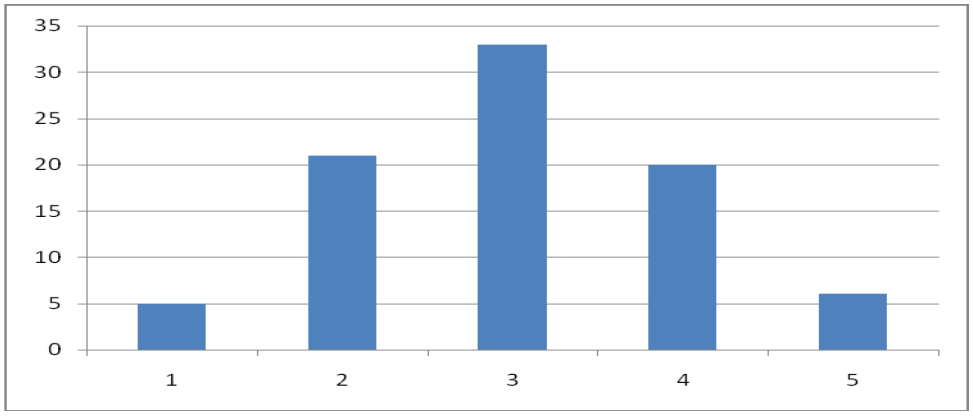


Figure IV.6 : distribution du nombre de tâches aléatoires traitées

En regroupant le nombre de tâches aléatoire exécutées par classe (figure IV.6) telle que les classes sont :
1 : [30, 35] ; 2 : [35, 40] ; 3 : [40, 45] ; 4 : [45, 50] ; 5 : [50, 55].
On obtient une distribution gaussienne. Le même phénomène constaté lors de l'analyse des résultats des autres approches validés dans [BL 09]. Ceci veut dire qu'on peut comparer les résultats de notre modèle avec les autres modèles en termes de nombres de tâches aléatoires exécutées.

IV. Interprétation des résultats et comparaison avec les autres modèles :

Dans cette partie, des expérimentations vont être réalisées sur le modèle de simulation, conformément au plan d'expérience définie plus haut. Nous commençons par rappeler les résultats qui ont été trouvés dans les autres approches de résolution et citées dans [BL 09], nous présentons les résultats trouvés dans ce travail et enfin, nous allons procéder à la comparaison des différents modèles dans le but de comparer la performance de l'approche étudiée dans ce travail.

1. Rappel des résultats trouvés dans les autres approches :

Dans ce qui suit, nous nous proposons de rappeler les résultats concernant les autres algorithmes à savoir : l'algorithme d'ordonnancement temps-réel au plus tard dans le cas d'une file d'attente,

l'algorithme d'ordonnancement temps-réel au plus tôt dans le cas d'une file d'attente, l'algorithme d'ordonnancement temps-réel au plus tard dans le cas de deux files d'attentes et enfin l'algorithme d'ordonnancement temps-réel collaboratif.

Nous avons résumé les résultats qui ont été trouvés dans un tableau récapitulatif, nous invitons le lecteur à consulter le PFE de Behiri et Laterech [BL 09] pour plus de détails.

Nous présentons le $C_{\max}$, le nombre de tâches aléatoires, et la cadence qui ont été trouvés lors des différentes expérimentations.

	La moyenne et l'écart-type du $C_{\max}$ trouvée pour les quatre modèle. (U.T).		La moyenne et l'écart-type du nombre de tâches aléatoires traitées. (U.T).		La moyenne et l'écart-type de la cadence. (***Cadence = Nombre de tâches aléatoires exécutées*** $/ C_{\max}$)	
	Moyenne	**Ecart-type**	**Moyenne**	**Ecart-type**	**Moyenne**	**Ecart-type**
Au plus tôt	155,6	14,7	32,4	5,2	0,20785	0,0252
Au plus tard à 1 FA	156,3	14,27	40	5,18	0,25573	0,0271
Au plus tard 2 FA	156,7	14,30	40,1	5,6	0,25585	0,0271
Collaboratif	156,3	14,25	41,2	5,22	0,26399	0,02658

Tableau IV.1 : Tableau récapitulatif des résultats obtenus pour les quatre modèles de résolution.

Commentaires :

Les variances sont proches les unes des autres pour le $C_{\max}$, pour le nombre de tâche aléatoires traitées et pour la cadence, ce qui rend les résultats des différents modèles comparables.

Le modèle au plus tôt présente le $C_{\max}$ le moins important (155, 6 U.T) pour un nombre de tâches aléatoires exécutées assez faible (32,4). Concernant la cadence, elle est beaucoup plus inférieure à celle trouvée dans les autres modèles (0,20785).

Le modèle au plus tard avec deux files d'attente présente le plus grand $C_{\max}$ (156,7 U.T) pour un nombre de tâches aléatoires égal à (40,1). La cadence trouvée est de (0,25573), elle est supérieure à celle trouvée pour le modèle au plus tôt à une file d'attente.

Le modèle au plus tard à une file d'attente présente un $C_{\max}$ d'une valeur de (156,3 U.T) pour un nombre de tâches aléatoires traitées égal à (40). La cadence trouvée est de (0,25585), elle est légèrement supérieure à celle trouvée pour le modèle au plus tard avec deux files d'attentes.

Le modèle collaboratif présente le nombre de tâche aléatoires le plus important pour un $C_{\max}$ de (156,3 U.T), aussi, il présente une meilleure cadence par rapport aux autres modèles (0,02658).

Concernant la cadence, elle est légèrement supérieure pour le modèle collaboratif, suivi des deux algorithmes au plus tard (à une file et à deux files d'attente). Elle est relativement médiocre pour le modèle au plus tôt.

2. Evaluation des performances et interprétation des résultats :

Dans le but de mesurer la performance de l'approche proposée, on se propose de comparer les résultats de notre approche avec les résultats des autres modèles. Pour ce faire, nous allons reprendre les résultats relatifs aux quatre modèles et intégrer les résultats de l'approche étudiée dans ce travail.

- **Le C max :**

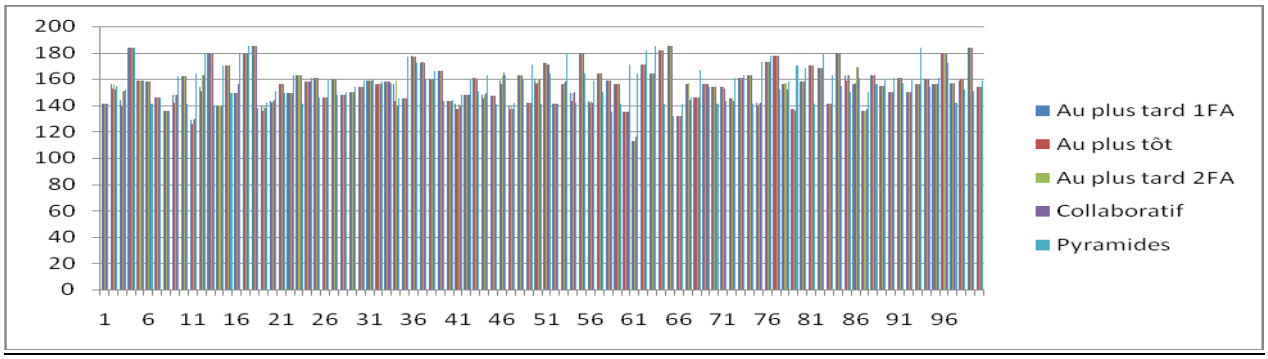


Figure IV.7 : le C_{max} des cinq modèles

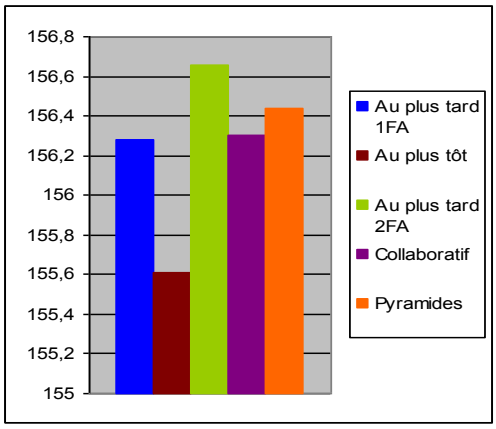


Figure IV.8 : les moyennes des C_{max} des cinq modèles.

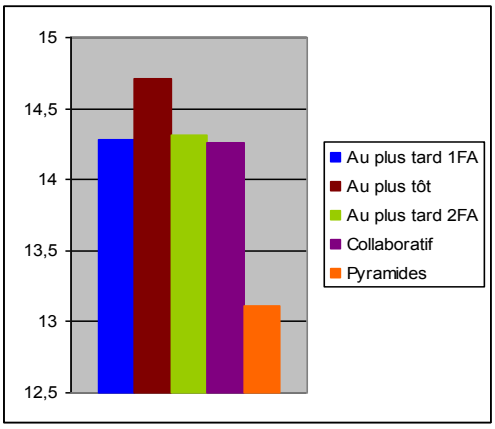


Figure IV.9 : les écarts types des C_{max} des cinq modèles.

La figure IV.7 présente la valeur des C_{max} des cinq modèles pour les 100 expérimentations.

Nous remarquons que le modèle au plus tard avec deux files d'attente, favorisant l'ordonnancement des tâches aléatoires présente le C_{max} le plus important d'une moyenne de 156,7 U.T, ce dernier donne la priorité aux tâches aléatoires. Il est suivi du modèle étudiée dans ce travail avec un écart de

0,3 U.T et un C_{max} égal à 156,4 U.T, ensuite vient le modèle collaboratif et le modèle au plus tard avec une file d'attente avec un C_{max} égal à 156,3 U.T et un écart de 0,1 U.T par rapport au modèle basé sur le théorème des pyramides .

Les variances des cinq modèles sont approximativement les mêmes, le modèle au plus tôt présente la plus grande variance (14,7), le modèle étudié dans ce travail présente la plus petite variance (13,2), l'écart entre la plus petite valeur et la plus grande valeur est de (1,5), l'écart constaté n'est pas très grand, ceci nous permet d'effectuer une bonne comparaison.

- **Le nombre de tâches aléatoires traitées :**

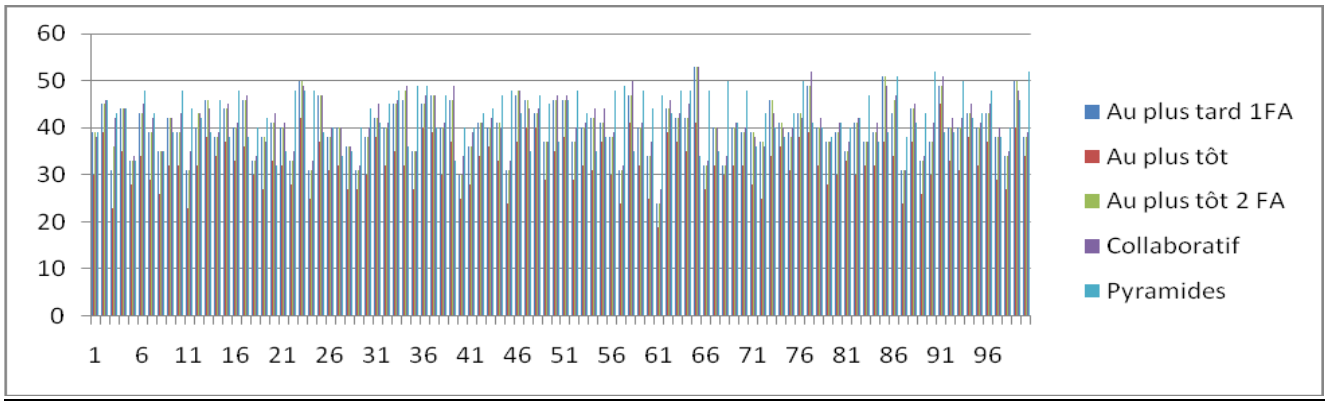


Figure IV.10 : le nombre de tâches aléatoires pour les cinq modèles.

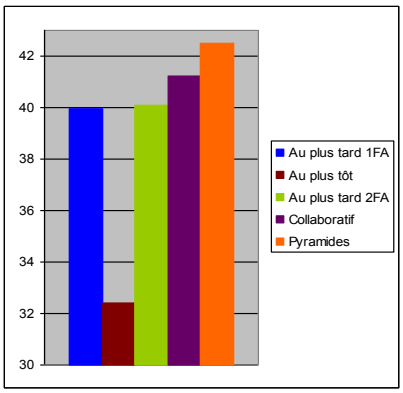


Figure IV.11 : les moyennes des nombres des tâches aléatoires traitées des cinq modèles

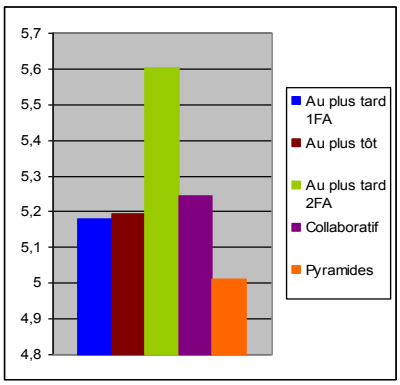


Figure IV.12 : les écarts types des nombre de tâches aléatoires traitées des cinq modèles.

L'histogramme des moyennes montre que le modèle étudié dans ce travail est en tête avec une moyenne de **42,4**, l'exécution des tâches aléatoires est avantagée, ceci s'explique par la flexibilité séquentielle engendré par le théorème des pyramides et qui cherche à traiter un nombre plus important de tâches aléatoires.

Le modèle au plus tôt traite le plus faible nombre de tâches aléatoires. Ce qui est expliqué par le fait que ce modèle priorise les tâches programmables.

Ainsi, pour le critère maximisant le nombre de tâches aléatoires exécutées, l'algorithme étudié dans ce travail est le meilleur. En présentant une moyenne de **42,4**. L'algorithme collaboratif est en seconde position avec une moyenne de **41, 2**. Il est suivi du modèle au plus tard avec deux files d'attente avec un nombre de tâches aléatoires de **40**. Le modèle qui présente le nombre de tâches aléatoires le plus faible est le modèle au plus tôt avec un nombre de tâches aléatoires traitées de **32,4**.

Les variances sont, une fois de plus, proches les unes des autres, ce qui rend les résultats des différents modèles comparables.

- **La cadence :**

La cadence d'un modèle mesure le nombre de tâches aléatoires exécutées par unité de temps. Sa formule est comme suit :

Cadence = Nombre de tâches aléatoires exécutées / C_{max}

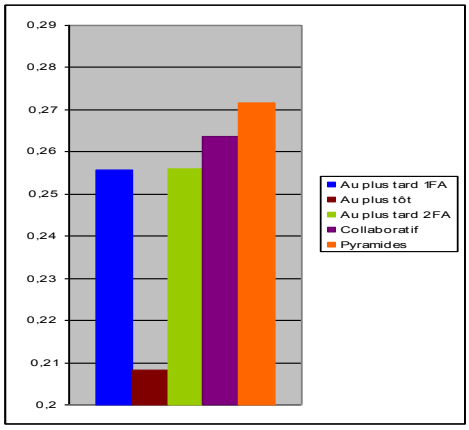


Figure IV.13 : Moyenne des cadences des cinq modèles.

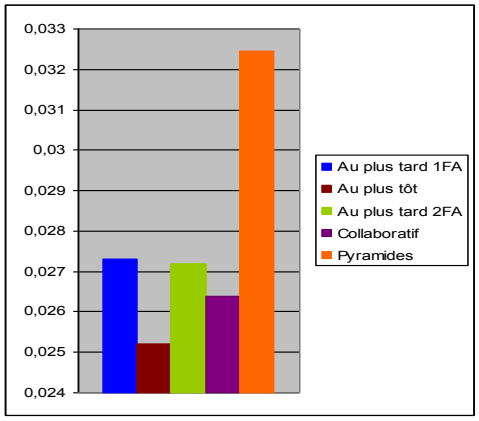


Figure IV.14 : Ecarts types des cadences des cinq modèles.

D'après la figure IV.13, le modèle étudiée dans ce travail présente la plus grande cadence, elle est de **0,27154,** elle est suivi de l'approche collaboratif avec une cadence de **0,26399**. Ensuite vient le modèle au plus tard à une file d'attente avec une cadence de **0,25573** suivi de l'algorithme au plus tard à deux files d'attente avec une cadence de **0,25585.**

Le modèle au plus tôt présente la cadence la moins importante avec **0,20785.**

Les variances des quatre modèles sont proches relativement aux moyennes, ce qui permet d'effectuer une bonne comparaison.

Cela veut dire que pour un même C_{max} = **500** unités de temps, si le modèle basé sur le théorème des pyramides exécute à peu près **136** tâches, le modèle collaboratif exécute **132** tâches, le modèle au

plus tard à une file d'attente exécute **128** tâches, celui à deux files d'attente exécute **129** tâches et le modèle au plus tôt exécute **101** tâches.

3. Etude de la performance de l'algorithme proposé :

D'après ce qui précède, le modèle basé sur le théorème des pyramides a apporté quelque résultat, pour valider ces résultats, il est nécessaire de comparer entre la différence des $C_{\max}$ obtenus et la différence entre la somme des durées opératoires (PR) du nombre de tâches aléatoires traitées, ceci nous permettra, d'une part de valider les résultats trouvés précédemment et d'autre part de s'assurer qu'on exploite au maximum les marges disponibles.

D'une part, nous allons calculer la différence entre le $C_{\max}$ obtenu avec l'approche proposée dans ce travail et le $C_{\max}$ obtenu avec l'approche collaborative (celle-ci présente de meilleurs résultats par rapport aux autres méthodes), d'autre part, nous allons calculer la différence entre la moyenne des sommes des durées opératoires des tâches aléatoires sur les deux modèles. Par la suite on va comparer ces deux résultats.

On résume les résultats trouvé dans le tableau suivant :

	Modèle basé sur le théorème des pyramides	Modèle collaboratif	La différence
$C_{\max}$	156,4 U.T	156,3 U.T	0,1 U.T.
La moyenne de la somme des durées opératoires des tâches aléatoires traitées.	53,55 U.T.	50,375 U.T.	3,18 U.T.

Tableau IV. 2 : tableau récapitulatif des résultats trouvés.

Si on compare les deux résultats des deux modèles, on trouve que la différence entre les deux $C_{\max}$ est largement inférieure à la différence entre les sommes des durées opératoires, ceci veut dire qu'on a exploité les marges disponibles, qu'on a maximisé le nombre de tâches aléatoires sans pour autant augmenter le $C_{\max}$.

4. Validation des résultats et étude de la robustesse :

Un autre critère nous semble important pour qualifier l'ordonnancement étudié dans ce travail, c'est la robustesse par rapport aux variations des durées opératoires.

Rappelons qu'on peut définir un ordonnancement robuste comme un ordonnancement peu sensible aux incertitudes [BMS 04].

Partons de là et afin d'étudier la robustesse de l'approche étudiée, nous avons procédé à tester le comportement des différents modèles, face à des paramètres différemment proportionnés. Pour ce faire, nous avons fait varier les durées opératoires des tâches programmables et celles des tâches aléatoires. Pour les mêmes dates de disponibilité et échues, la variation des durées opératoires [5] nous permet de tester la capacité des modèles à « absorber » les incertitudes.

Dans ce qui suit nous allons tester les cinq modèles avec des paramètres (durées opératoires différentes) (voir le tableau IV.3).

Numéro de l'expérience	P tâches programmables	P tâches aléatoires
1	Triangulaire (0,5 ; 1 ; 1,5)	Triangulaire (1;2;3)
2	Triangulaire (0,5 ; 1 ; 1,5)	Triangulaire (3;4;5)
3	Triangulaire (0,5 ; 1 ; 1,5)	Triangulaire (5;6;7)
4	Triangulaire (0,5 ; 1 ; 1,5)	Triangulaire (7;8;9)
5	Triangulaire (0,5 ; 1 ; 1,5)	Triangulaire (9;10;11)
6	**Triangulaire (1 ; 1,5 ; 2,5)**	**Triangulaire (1;2;3)**
7	Triangulaire (1 ; 1,5 ; 2,5)	Triangulaire (3;4;5)
8	Triangulaire (1 ; 1,5 ; 2,5)	Triangulaire (5;6;7)
9	Triangulaire (1 ; 1,5 ; 2,5)	Triangulaire (7;8;9)
10	Triangulaire (1 ; 1,5 ; 2,5)	Triangulaire (9;10;11)
11	**Triangulaire (1,5 ; 2 ; 2,5)**	**Triangulaire (1;2;3)**
12	Triangulaire (1,5 ; 2 ; 2,5)	Triangulaire (3;4;5)
13	Triangulaire (1,5 ; 2 ; 2,5)	Triangulaire (5;6;7)
14	Triangulaire (1,5 ; 2 ; 2,5)	Triangulaire (7;8;9)
15	Triangulaire (1,5 ; 2 ; 2,5)	Triangulaire (9;10;11)
16	**Triangulaire (2 ; 2,5 ; 3)**	**Triangulaire (1;2;3)**
17	Triangulaire (2 ; 2,5 ; 3)	Triangulaire (3;4;5)
18	Triangulaire (2 ; 2,5 ; 3)	Triangulaire (5;6;7)
19	Triangulaire (2 ; 2,5 ; 3)	Triangulaire (7;8;9)
20	Triangulaire (2 ; 2,5 ; 3)	Triangulaire (9;10;11)

Tableau IV.3 : paramètres des durées opératoires pour chaque expérimentation.

Les durées des tâches programmables varient dans une plage allant de 0,5 jusqu'à 3. Quant aux durées opératoires (p) des tâches aléatoires, elles varient de 2 jusqu'à 10.

5 : Les durées opératoires choisis dans ce travail sont les même qui ont été retenues par Behiri le Latrech [BL 09] afin de garantir la comparaison entre les différents modèles

On effectue une centaine d'expérimentations pour chaque expérience.

- **Le C_{max} moyen :**

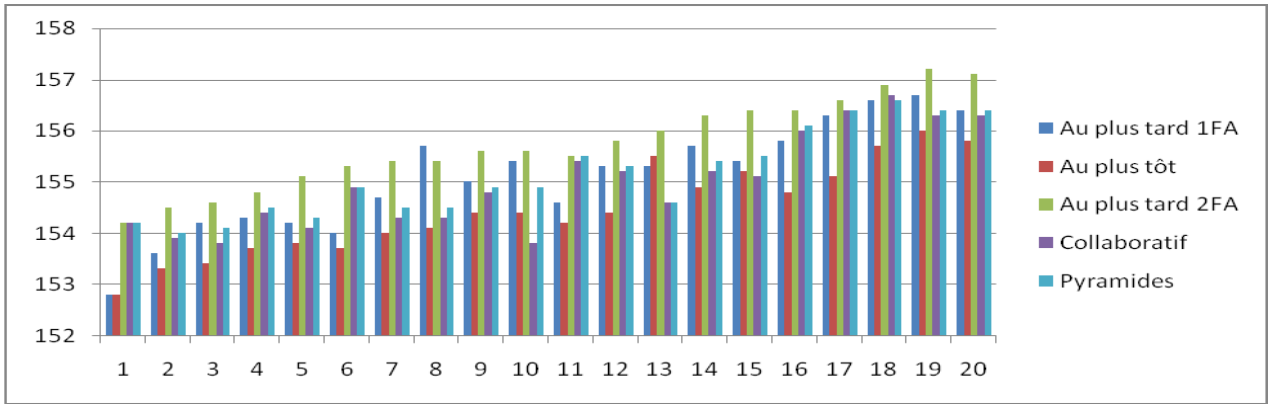


Figure IV.15 : le C_{max} moyen des cinq modèles.

D'après ces expérimentations, l'ordonnancement des cinq modèles semble capable d'absorber les incertitudes des durées opératoires vis-à-vis du C_{max}. Pour les 20 expérimentations, nous constatons que le C_{max} moyen pour les cinq méthodes varie autour de la même valeur, nous pouvons conclure que les modèles sont stables vis-à-vis du C_{max}.

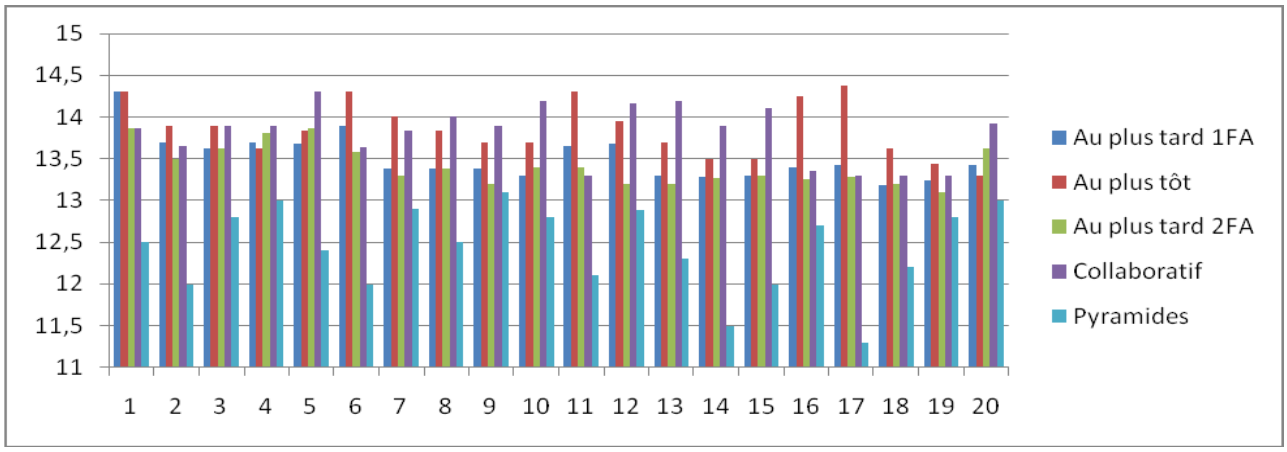


Figure IV.16 : Ecarts type du C_{max} moyen des cinq modèles

Pour chaque expérimentation, nous pouvons constater qu'il n y a pas une grande variation entre les différents écarts type des cinq modèle.

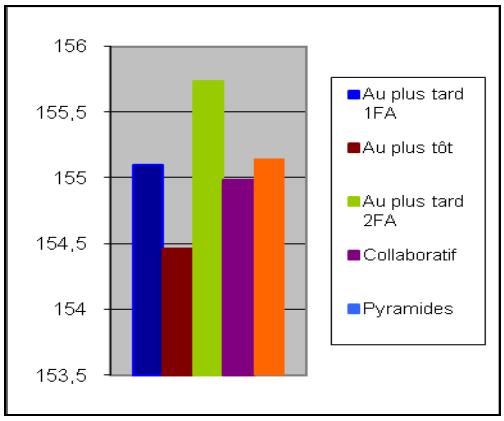


Figure IV.17 : Les moyennes des C_{max} moyens des cinq modèles.

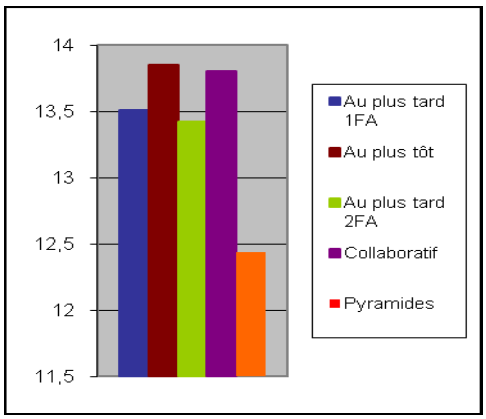


Figure IV.18 : Les écarts types des C_{max} moyens des cinq modèles.

Sur la figure IV.17, on constate que les valeurs des différents C_{max} suit le même ordre trouvé dans le cas à 100 expériences (Figure IV.7), autrement dit le modèle au plus tard avec deux files d'attente est en tête suivi du modèle étudié dans ce travail, ensuite vient le modèle au plus tard à une file d'attende et le modèle collaboratif. Le modèle au plus tôt présente le C_{max} le moins important.

- **Le nombre de tâches aléatoires exécutées :**

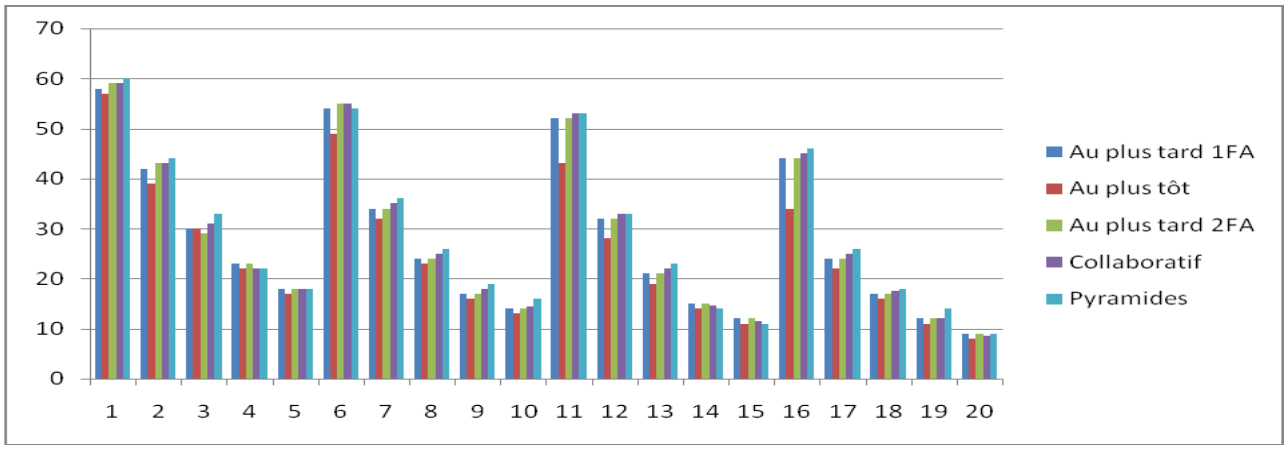


Figure IV.19 : le nombre de tâches aléatoires exécutées des cinq modèles.

Conformément à la figure IV.19, nous pouvons confirmer que le comportement des modèles est le même vis-à-vis du nombre de tâches aléatoires traitées et ceci peu importe les proportions entre les durées opératoires et leurs valeurs.

Ainsi, l'ordre des différents modèles par rapport aux nombre de tâches aléatoires traitées est conforme à celui trouvé pour le cas de 100 expériences (Figure IV.10) : Pour les 20 expérimentations, on a l'approche étudiée dans ce travail qui favorise l'exécution des tâches aléatoire suivi du modèle collaboratif, le modèle au plus tard avec 2 files d'attente, le modèle au plus tard avec 1 file d'attente et enfin le modèle au plus tôt.

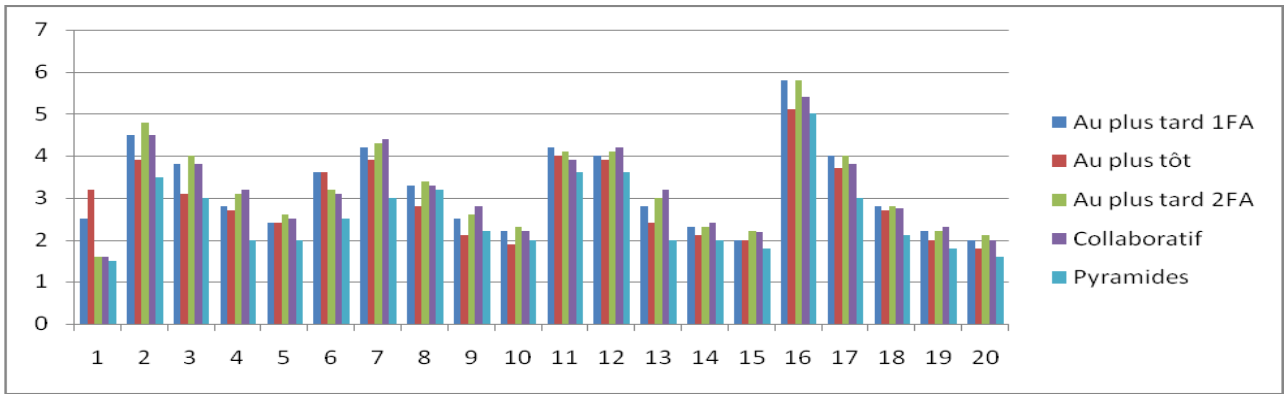


Figure IV.20 : les écarts types du nombre de tâches aléatoires exécutées des cinq modèles.

L'écart type est stable pour les 20 expériences et il varie très peu d'un modèle à un autre.

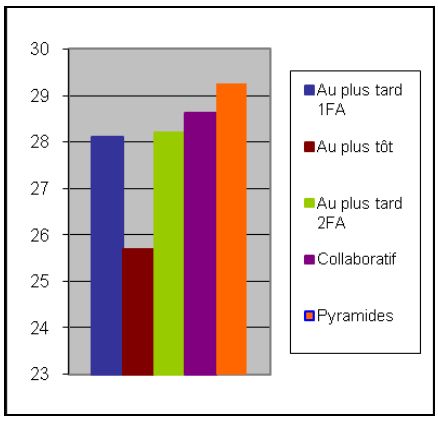


Figure IV.21 : Les moyennes des nombres de tâches aléatoires exécutées des cinq modèles

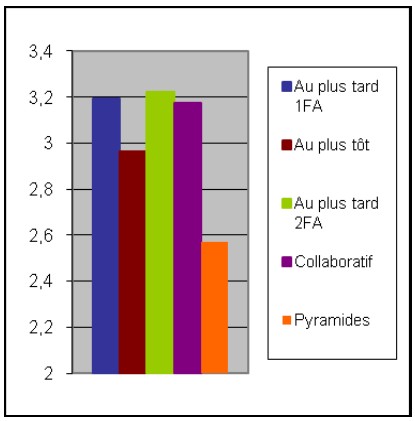


Figure IV.22 : Les écarts types des nombres de tâches aléatoires traitées des cinq modèles.

Les résultats sur le classement des modèles, selon le nombre de tâches aléatoires exécutées, présenté dans la section précédente sont confirmés avec ceux de la figure IV.21.

Les écarts type des cinq modèles présentes des écarts faible, l'écart entre la plus grande valeur (trouvée pour le modèle au plus tard à deux files d'attente) et la plus petite valeur (trouvée pour le modèle étudié dans ce travail) est égale à 0,8.

Dans le but d'étudier la performance globale du modèle Proposé par rapport aux variations qui concerne la configuration de base de notre atelier, il semble intéressant d'adapter l'approche étudiée dans ce travail au cas d'un atelier à trois machines identiques en parallèle.
Par ailleurs, l'approche coopérative permet d'alléger la complexité du problème original, ceci en distribuant l'ordonnancement sur les différentes machines constituant l'atelier de production. Il est alors possible de généraliser la solution proposée au cas de plusieurs machines en parallèle sans augmenter, pour autant, la difficulté du problème initial. Ce qui n'était pas le cas avec les autres approches déjà existantes (Algorithme d'ordonnancement temps-réel au plus tard au cas d'une file d'attente, algorithme d'ordonnancement temps-réel au plus tard au cas de deux file d'attente et l' algorithme d'ordonnancement temps-réel au plus tôt) qui voient la complexité de leur application augmenter en fonction du nombre de machines constituant l'atelier.
Nous avons entamé ce chapitre par tester la stratégie d'ordonnancement étudiée dans ce travail sur un atelier contenant deux ressources identiques en parallèle.
Nous nous proposons maintenant de tester cette même stratégie au cas de trois machines identiques en parallèle.
Les différents paramètres (r, p, d) sont générés selon les mêmes distributions choisis dans le cas de deux ressources, à savoir:
Pour les tâches programmables :

- Les arrivées : $r_i = r_{i+1} + EXPO(2)$
- Les durées opératoires : $p_i = TRIA(1,2,4)$
- Les dates échues : $d_i = r_i + p_i + TRIA(1,1\,1,3\,7)$

Pour les tâches aléatoires :

- Les arrivées : $r_i = r_{i+1} + EXPO(2)$
- Les durées opératoires : $p_i = TRIA(1,2,5)$
- Les dates échues : $d_i = r_i + p_i + TRIA(1,3,6)$

Le nombre de tâches programmables est de 75, celui des tâches aléatoires est de 60.
On effectue une centaine d'expérience.
Les mêmes indicateurs de performance seront considérés, à savoir, le makespan, et le nombre de tâches aléatoires traitées.
Après cent expérience, la moyenne du makespan trouvé est de : 152,9 U.T pour un nombre de tâches aléatoires exécutées de 57,4.
Il serait alors intéressant de généraliser l'approche au cas de quatre machines.

5. Les recommandations relatives aux choix du meilleur modèle :

En reprenant les expériences qui ont été effectuées par [BL 09] et en intégrant le modèle basé sur le théorème des pyramide, il s'avère que le $C_{\max}$ obtenu avec l'approche étudiée dans ce travail est inférieur au $C_{\max}$ obtenu avec le modèle au plus tard avec deux files d'attente, supérieur au $C_{\max}$ du modèle collaboratif, du modèle au plus tard avec une file d'attente et du modèle au plus tôt.

Et concernant le nombre de tâches aléatoires exécutées, le modèle basée sur le théorème des pyramides est le meilleur suivi du modèle collaboratif, avec un résultat loin de celui du modèle au plus tôt. Ainsi le modèle basé sur le théorème des pyramides permet de maximiser le nombre de tâches aléatoires traitées sans pour autant augmenter le $C_{\max}$.

Dans le but de mieux comprendre les résultats à l'issu de la simulation, nous pouvons comparer notre modèle avec un réel atelier de production à deux machines en parallèle ou chaque expérimentation représente une journée de production, les tâches programmables sont des commandes issues du plan de production tandis que les tâche aléatoires sont, dans la plus part des cas, des commandes urgentes.

En supposant qu'on a un nombre de tâches programmables en moyenne égale à 75 tâches /jour et un nombre de commandes urgentes en moyenne égal à 60 tâches / jour. Après un trimestre de production (100 jours, on suppose que chaque expérimentation correspond à une journée de production), nous déduisons le $C_{\max}$ moyen ainsi que le nombre de tâches aléatoires traitées en moyenne pour les cinq modèles :

- Le modèle étudié dans ce travail présente un $C_{\max}$ de 156,4 U.T et un nombre de tâches aléatoires exécutées de 42,4.
- Le modèle collaboratif donne un $C_{\max}$ = 156,3 U.T et un nombre de tâches aléatoires traitées de 41,2.
- Le modèle au plus tard avec deux files d'attente donne un $C_{\max}$ = 156,7 U.T et un nombre de tâches aléatoires traitées de 40,1.
- Le modèle au plus tard à une file d'attente présente un = 156,3 U.T et un nombre de tâches aléatoires traitées de 40.
- Le modèle au plus tôt donne un $C_{\max}$ = 155,6 U.T et un nombre de tâches aléatoires traitées de 32,4.

De ce fait, si on veut maximiser le nombre de tâches aléatoires exécutées, le modèle basé sur l'approche des pyramides est le plus adapté suivi du modèle collaboratif, ensuite, le modèle au plus

tard avec deux files d'attente. Sinon, si on ne veut optimiser que le $C_{\max}$, le modèle au plus tôt donne de meilleurs résultats.

V. Conclusion :

Dans ce chapitre et dans le but d'évaluer la performance de l'approche étudiée dans ce travail, nous avons effectué des expérimentations sur le modèle de simulation et nous avons procédé à l'analyse des résultats obtenus.

L'analyse des résultats a permis de constater les éléments suivants :

- Le $C_{\max}$ dans le cas au plus tard avec deux files d'attente est le plus important. Cependant, il présente un nombre de tâches aléatoires traitées assez satisfaisant.
- Le $C_{\max}$ obtenu pour le modèle au plus tard avec une file d'attente présente un $C_{\max}$ moins important par rapport au modèle au plus tard avec deux files d'attentes.
- L'approche collaborative est assez performante car elle présente un nombre important de tâches aléatoires pour un $C_{\max}$ satisfaisant.
- Le modèle Au plus tôt est le modèle le plus médiocre vis-à-vis du nombre de tâches aléatoires traitées, cependant il présente le plus faible « Makespane ».
- l'approche étudiée dans ce travail présente un $C_{\max}$ relativement faible si on ignore celui de l'approche au plus tôt, tandis que le nombre de tâches aléatoires traitées est le plus important, ceci veut dire qu'on a réussi à exploiter le maximum de marges libres sans pour autant augmenter le $C_{\max}$.

Les différentes expérimentations ont confirmé l'efficacité de l'approche étudiée dans ce travail, en effet, la coopération a conduit à la caractérisation d'une famille de solutions permettant de faire face aux aléas. Introduire de la flexibilité séquentielle dans une solution d'ordonnancement afin de disposer d'un ensemble de solutions s'avère très intéressant, il est alors possible de maximiser le nombre de tâches imprévues dans le temps en passant d'une solution à une autre.

Conclusion générale et perspective :

La flexibilité et la réactivité sont devenues des qualités incontournables pour les entreprises, car elles se trouvent confrontées à une demande variée et fluctuante avec des contrainte de qualité et de délais de plus en plus fortes. Pour de tels systèmes, l'ordonnancement des tâches de production devient une fonction critique de la gestion opérationnelle.

Les méthodes classiques d'ordonnancement ne laissent généralement que peu de flexibilité quant à l'ordre d'exécution des tâches ordonnancées, celui-ci étant figé dans la solution produite. Pourtant, accepter une flexibilité sur l'ordre d'exécution des tâches permet de mettre en évidence un nombre de solutions important ; les méthodes d'ordonnancement capables de déterminer, non pas une solution, mais une famille de solutions sont qualifiées de robustes dans le sens où elles permettent de réagir aux événements imprédictibles qui apparaissent durant l'exécution de l'ordonnancement sans pour cela nécessiter la remise en cause des calculs déjà effectués.

Ce travail s'inscrit dans le cadre de la contribution à la recherche d'heuristiques performantes pour la résolution du problème d'ordonnancement temps réel sur des ressources identiques en parallèle avec minimisation du $C_{\max}$, et l'approche abordée est une approche par coopération dans le cadre d'un ordonnancement robuste distribué.

Dans un premier temps, le problème global est décomposé en plusieurs sous-problèmes à une machine, chacune gérant son propre ordonnancement robuste. Cet ordonnancement robuste est calculé grâce à un théorème de dominance démontré dans les années quatre-vingt. L'avantage de ce théorème réside dans sa capacité à caractériser un ensemble dominant de séquence. Ceci a pour conséquence directe de réduire l'espace de recherche de solutions admissibles d'une part et d'apporter de la flexibilité séquentielle pour la construction d'un ordonnancement global capable de faire face à l'arrivée de nouvelles tâches, d'autre part.

Dans le but d'évaluer la performance de l'approche étudiée, nous avons simulé la stratégie d'ordonnancement proposée sous Arena 10, ce qui nous a permis de comparer la performance de notre approche avec les résultats obtenus dans [BL 09]. L'analyse des résultats obtenus à l'issue du processus de simulation a permis de mettre en évidence les éléments suivants :

- L'approche étudiée dans ce travail présente des résultats encourageants, elle a augmenté le nombre de tâches aléatoires traitées. Cette performance est réalisée grâce à la flexibilité séquentielle.
- L'approche collaborative présente un nombre de tâches exécutées moins important mais meilleur en comparant avec les autres approches de résolution, cependant, le $C_{\max}$ obtenu est inférieur à celui obtenu par l'approche étudiée dans ce travail.
- L'algorithme d'ordonnancement temps-réel au plus tard dans le cas de deux files d'attente, présente un nombre de tâches aléatoires exécutées, toujours en troisième position après l'approche collaborative. Cependant, il présente un $C_{\max}$ supérieur à ceux obtenus par les autres méthodes de résolution ;
- L'algorithme d'ordonnancement temps-réel au plus tard dans le cas d'une file d'attente, présente un nombre de tâches aléatoires exécutées, en quatrième position. Cependant, le $C_{\max}$ obtenu est, dans tous les cas, inférieur à celui obtenu par L'algorithme d'ordonnancement temps-réel au plus tard dans le cas de deux files d'attente ;
- L'algorithme d'ordonnancement temps-réel au plus tôt dans le cas d'une file d'attente, présente un nombre de tâches aléatoires exécutées, très inférieur par rapport aux méthodes de résolution précédentes. Cependant, c'est le modèle le plus médiocre vis-à-vis du nombre de tâches aléatoires traitées.

Les résultats des différentes expérimentations ont confirmé l'efficacité de l'approche étudiée. Il est alors intéressant d'introduire de la flexibilité séquentielle dans une solution d'ordonnancement afin de disposer d'un ensemble de solutions, cette flexibilité séquentielle a pour but de maximiser le nombre de tâches imprévues dans le temps en passant d'une solution à une autre.

Le travail de recherche que nous avons effectué nous a permis d'approfondir nos connaissances dans divers domaines, dans l'ordonnancement d'atelier en général, l'ordonnancement en milieu incertain, la résolution distribuée et coopérative et dans la simulation comme outil d'aide à la décision.

En définitive, ce travail a permis d'améliorer les résultats trouvés dans l'approche collaborative vis-à-vis du nombre de tâches aléatoires exécutées. Ceci permet de dégager un certain nombre de travaux en perspectives.

L'approche étudiée a été généralisée au cas de trois ressources identiques en parallèle, il serait alors intéressant de la généraliser au cas de quatre ressources voire m ressources identiques en parallèle.

Une des perspectives intéressantes de ce travail serait d'appliquer le théorème des pyramides au cas de l'algorithme d'ordonnancement temps réel au plus tard à une file d'attente, ceci nous permettra d'évaluer la contribution effective de chacune des deux méthodes, à savoir, l'approche collaborative et le théorème des pyramides.

Par ailleurs, le théorème des pyramides semble être une approche intéressante pour générer un ensemble de séquences dominante vis-à-vis d'un critère donné. Ainsi, nous proposons qu'une étude relative à son application pour d'autres problèmes avec d'autres critères (contraintes de délais, minimisation de taches en retard,.. etc.) pourrait mettre en évidence l'apport de cette approche.

Références bibliographiques :

A

[AIT 05] Ait Hssain A., 2005, « *Optimisation des flux de production* », 2ème Edition, L'usine nouvelle, DUNOD, Paris.

[ANT 65] R.N.ANTON, 1965, « planning and control systems: a framework for analysis », Harvard University Press.

[ART 02] C.ARTIGUES, C. BRIAND, J-C.PORTMANN, F.ROUBELLAT, 2002, Pilotage d'atelier basé sur un ordonnancement flexible, Hermes Sciences, 2002, p. 61-97.

[ART 97] Christian Artigues, 1997, Ordonnancement en temps réel d'ateliers avec temps de préparation des ressources. Thèse de doctorat, Université Paul Sabatier, Toulouse, 1997.

B

[BAC 95] L. Baccouche, Un Mécanisme d'Ordonnancement Distribué de Tâches Temps Réel, 1995, thèse de doctorat, l'institut national polytechnique de Grenoble, France.

[BAK 96] Bakalem M., 1996, *« modélisation et simulation orientées objet des systèmes manufacturiers »,* Thèse de Doctorat en Electronique-Electrotechnique-Automatique, Université de Savoie, France.

[BAR 03] Le Bars M, 2003, « Un Simulateur Multi-Agent pour l'Aide à la Décision d'un Collectif : Application à la Gestion d'une Ressource Limitée Agro-environnementale ». Thèse de doctorat, Université Paris IX-DAUPHINE, France.

[BER 00] Berchet C., 2000, *« Modélisation pour la simulation d'un système d'aide au pilotage industriel »*, Thèse de doctorat en génie industriel, Institut National Polytechnique de Grenoble.

[BHLL 04] Cyril Briant, Marie-José Huguet, Hoang Trung La, and Pierre Lopez, 2004, Hermes Science, pages 191–218, Paris, France.

[BIL 93] Jean-Charles Billaut, 1980, Prise en compte des ressources multiples et des temps de préparation dans les problèmes d'ordonnancement en temps réel. Thèse de doctorat, Université Paul Sabatier, Toulouse.

[BL 09] W.Behiri, R. Latrech, 2009, Contribution à la résolution d'un problème d'ordonnancement en temps réel sur deux ressources identiques en parallèle par l'approche collaborative, Mémoire du projet de fin d'étude Ecole Nationale Polytechnique d'Alger, Algérie.

[BM 92] S. Bussmann and J. Muller, 1992, A Negotiation Framework for Co-operating Agents, page 117. In M, D. S., editor, Proc. CKBS-SIG, Dake Centre, University of Keele.

[BMS 04] Jean-Charles Billaut, Aziz Moukrim, and Eric Sanlaville, 2004, Flexibilité et robustesse en ordonnancement, Hermes Science, pages 15–34. Paris, France.

[BOB 08] Cyril Briand, S. Ourari and B. Bouzouia, 2008, Une approche coopérative pour l'ordonnancement sous incertitudes, In Actes de la 7ème Conférence Francophone de Modélisation et Simulation (MOSIM'08).

[BOU 07] Bougchiche Fazia, 2007, Contribution à l'ordonnancement temps-réel sur des ressources identiques en parallèle, Mémoire de magister, Ecole Nationale Polytechnique d'Alger, Algérie.

[BRU 98] P. Brucker, Scheduling algorithms, Springer, Heidelberg, 2nd edition, 1998.

C

[CAM 00] J.P. Camalot, 2000, Aide à la décision et à la coopération en gestion du temps et des ressources. Thèse de Doctorat, Institut National des Sciences Appliquées de Toulouse, France.

[CC 88] J. Carlier and P. Chrétienne, 1988 Problèmes d'ordonnancement : Modélisation, Complexité, Algorithmes. Paris, Masson.

[CV 97] B. Chen, A. Vestjens, 1997. Scheduling on identical machines: How good is LPT in an on-line setting? , Operations Research Letters 21, p 165-169

D

[DB 00] A.J. Davenport and J.C. Beck, 2000, a survey of techniques for scheduling with uncertainty.

[DES 82]: P. Deschizeaux, 1982, "Temps et évènement dans les systèmes distribués de contrôle de procédé", R.A.I.R.O. Automatique/Systems Analysis and Control. 16(3):259-74.

[DIM 05] Christos Dimopoulos, 2005, a novel approach for the solution of the multiobjective cell-formation problem. In Proceedings of the International Conference of Production Research, 2005.

[DJE 07] S.Djellouli, 2007, Approche par coopération inter machine pour un ordonnancement distribué, magister, Ecole Nationale Polytechnique d'Alger, Algérie.

E

[EBH 03] Jacques ERSCHLER, Cyril Briand, Hoang Trung LA, du 23 au 25 avril 2003, une approche pour l'ordonnancement robuste de tâches sur une machine, 4e Conférence Francophone de Modélisation et simulation "Organisation et Conduite d'Activités dans l'Industrie et les Services" MOSIM'03, Toulouse (France)

[EFM 85] J. Erschler, G. Fontan, C. Merce, 1985, Un nouveau concept de dominance pour l'ordonnancement de travaux sur une machine. RAIRO Recherche Opérationnelle/Operations Research, Vol. 19, n°1.

[EHT 97] J.ERSCHLER, M.J. HUGUET et G.TERSSAC, 1997, Décision distribuée en gestion De production, Toulouse, p. 109-131.

[ESQ 99] P. Esquirol and P. Lopez. L'ordonnancement. Paris, Economica, 1999.

[ESQ 01] P. Esquirol, P. Lopez., and F. Roubellat, 2001, Ordonnancement de la production, Edition HERMES, ISBN, p.25-53. Paris.

[ESS 03] C. Esswein, 2003, Un apport de flexibilité séquentielle pour l'ordonnancement robuste. Thèse de Doctorat, Université François Rabelais, Tours, France.

[ETB 07] C. Briand, H. Trung and Jacques Erschler, 2007, a robust approach for the single machine scheduling problem. Journal of Scheduling, vol. 10, no. 2.

G

[GAR 79] M.R. Garey, D.S. Johnson, 1979, Computer and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco.

[GHA 06] Ghazi N., 2006, « *Problème d'ordonnancement Open Shop dans un environnement d'atelier d'assemblage cyclique »*, mémoire de magister, Spécialité : automatique, département Génie électrique, Laboratoire de Commande des Processus, ENP, Alger.

[GIA 88] V. Giard. Gestion de production, 1988, Economica.

[GKS 03] A. Girault, H. Kalla, and Y. Sorel. Une heuristique d'ordonnancement et de distribution tolérante aux pannes pour systèmes temps réel embarqués. Modélisation des Systèmes Réactifs, MSR'03, Metz, France. Hermes, pages 145–160, Octobre 2003.

[GOT 93] GOTHA, 1993, Les problèmes d'ordonnancement, R.A.I.R.O. Recherche opérationnelle/Operational Research, volume 27, n°1, chapitre 4, pages 77_150.

[GRA 79] Ronald L.Graham, Eugene L. Lawler, Jan Karel Lenstra, and A. G. H. Rinnooy Kan, 1979, Optimization and approximation in deterministic sequencing and scheduling, a survey. Annals of Discrete Mathematics, 5: 287–326, 1979.

[GRA 05]: B. GRABOT, "Ordonnancement d'ateliers manufacturiers". Techniques de l'ingénieur, AG 3 015.

H

[HAB 01] Habchi G., 2001, *« Conceptualisation et modélisation pour la simulation des systèmes de production »,* Document de Synthèse*,* Université de Savoie*,* France*.*

[HAL 2000] N.G.Hall, C. N. Potts, and C. Sriskandarajah, 2000, Parallel machine scheduling with a common server. Discrete Applied Mathematics, 102, p.223-243

[HAO 04] M. Haouari and A. Gharbi, Lower Bounds for Scheduling on Identical Parallel

[HAU 89] R. Haupt, 1989, a survey of priority rule-based scheduling. OR Spektrum, V 13, p.3–16.

[HETL 96] M.J. Huguet, J. Erschler, G. De Terssac, and N. Lompre. Negotiation based on constraints in cooperation, 1996.

[HU 61] T. Hu, 1961, parallel sequencing and assembly line problems. Operations Research 9, p. 841-848.

J

[JEL 07] Djellouli Sabrina , 2007, Approche par cooperation intermachines pour l'ordonnancement pour un ordonnancement distribute, thèse de magistère en automatique, Ecole Nationale Polytechnique.

R. Karp, 1972, Complexity of computer Computations, Plenum Press, New-York.

K

[KAR 72] R. Karp, 1972, Complexity of computer Computations, Plenum Press, New-York.

[KEL03] Kelton W. D., 2003, « *Simulation with Arena »*, 3ème Edition, International Edition, New York.

[KPB 85] G. Kallel, X. Pellet, and Z. Binder, 1985, Conduite décentralisée coordonnée d'atelier.

L

[LA 05] Hoang Trung La. Utilisation d'ordres partiels pour la caractérisation de solution robustes en ordonnancement. Thèse de doctorat, INSA, Toulouse, 2005.

[LEN 77] J.K. Lenstra, Rinnooy Kan A.H.G.et Brucker P, 1977, Complexity of machine scheduling problems. Annals of Discrete Mathematics 1, pages 343–362.

[LOP 01] P. Lopez et F. Roubellat, Ordonnancement de la production. Paris, Hermès, 2001.

M

[MC 70] Muntz and Coffman, 1970, Preemptive scheduling or real time tasks on multiprocessor systems. International Journal of Production Research

[ML 93] B.L. MacCathy and J. Liu. Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling, International Journal of Production Research, volume vol. 31, no. 1, p.59_79.

[MOK 04] E. Mokotoff, 2004, an exact algorithm for the identical parallel machine scheduling problem, European Journal of Operational Research.

[MON 05] E. Monsarrat, C.Briand & P. Esquirol, 2005, Aide à la décision pour la coopération interentreprises. Journal Européen des Systèmes Automatisés.

P

[PIN 95] M. Pinedo, 1995, Scheduling: Theory, Algorithms and Systems. Prentice Hall.

[PM 97] H. Pierreval and N. Mebarki, 1997, Dynamic selection of dispatching rules for manufacturing system scheduling. International Journal of Production Research.

S

[SB D 94] K. Swanson, J. Bresina, and M. Drummond, 1994, Just-in-case scheduling for automatic telescopes. In W. Buntine and D. H. Fisher, editors, Knowledge-Based Artificial Intelligence Systems in Aerospace and Industry, p. 10–19.

[SID 97] K. Sidhoum, 1997. Résolution de problème de partitionnement généralisé par des méthodes d'optimisation globale à base de déplacements stochastique : application à l'ordonnancement à machines parallèles, Thèse de doctorat, Ecole Centrale Paris, France.

[SPO 00] F. SPOSITO, « CSCW, Groupware and social issues - Applications, analytical perspectives and possible solutions in an ever-changing and critical study field », support du cours « Human-Computer Interaction ».

[STA 98] J.A. Stankovic, 1988, « Misconception about Real−Time Computing », IEEE Computer Magazine.

T

[THO 80] V.Thomas. Aide à la décision pour l'ordonnancement d'atelier en temps réel. Thèse de doctorat, Université Paul Sabatier, Toulouse, 1980.

[TRU 05] L. Trung. Utilisation d'ordre partiel pour la caractérisation de solutions robustes en ordonnancement. Thèse de doctorat, Laboratoire d'analyse et d'architecture des systèmes (LAAS), CNRS, Toulouse, France, 2005.

V

[VER 04] M. Verrons, 2004, un modèle général de négociation de contrats entre agents. Thèse de doctorat de l'Université des Sciences et Technologies de Lille, France, 2004.

[Yalaoui 00] F. Yalaoui, 2000. Ordonnancement de la production en présence de machines alternatives, Thèse de doctorat, Université de technologie Troyes, France.

Liste des annexes :

La notation en α/β/γ des problèmes d'ordonnancement

α	Système	α_1	Ø	une seule machine
			P	machines identiques en parallèle
			U	machines à vitesse proportionnelle en parallèle
			R	machines non reliées en parallèle
			O	open shop
			F	flow shop
			J	job shop
		α_2	Ø	le nombre de machines est fixé à une valeur donnée précise
			1,2…	le nombre de machines est fixé dans la définition de l'énoncé
			m	le nombre de machines varie avec chaque instance considérée
β	Contraintes	β_1	Ø	les opérations ne peuvent pas être interrompues
			pmtn	les opérations peuvent être interrompues
		β_2	Ø	il n'y a pas de contraintes de précédence
			tree	il existe des contraintes de précédence additionnelles sous forme d'arborescence ou d'anti-arborescence
			prec	il existe des contraintes de précédence additionnelles sous forme d'un graphe orienté sans circuit
		β_3	Ø	$\forall i\ r_i = 0$
			r_j	les dates de disponibilité sont différentes selon les travaux
		β_4	Ø	$\forall i\ p_i \in \mathbb{N}^*$
			p_j=1	chaque opération a une durée unitaire
			(p_j=p) & (β_3= r_j)	La durée des opérations est égale à p≠1 et au moins un des r_j n'est pas divisible par p
		β_5	Ø	les temps de réglage sont constants et inclus dans la durée des opérations
			Snsd[1]	des temps de réglage sont affectés à chaque opération, ils sont indépendants de la séquence mais non inclus dans la durée opératoire car ils peuvent être exécutés avant que l'opération n'arrive dans la file d'attente de la machine
			Ssd[1]	les temps de réglage ne sont pas inclus dans la durée des opérations et sont dépendants de la séquence des opérations sur la ressource
		β_6	Ø	les dates de fin souhaitées si elles existent sont des nombres entiers arbitraires non négatifs
			dj=d	toutes les dates de fin souhaitée sont identiques (« common due date »)
			Δj	les dates de fin souhaitée sont remplacées par des dates de fin impérative
γ	Critères	γ_1	C	Completion time
			L	Le retard « Lateness »
			T	Le retard vrai « Tardiness »
			E	L'avance vraie « earliness »
			G	une fonction générale non décroissante du « completion time »
			NT	le nombre de travaux en retard
		γ_2	K_{max}	Minimisation de la plus grande valeur d'une fonction donnée de la date de fin pour tous les travaux (K≡C, L, T, E, G)
			$\overline{K}$	Minimisation de la valeur moyenne d'une fonction donnée de la date de fin pour tous les travaux (K≡C, L, T, E)
			$\overline{K}_w$	Minimisation de la somme pondérée de la fonction considérée de la date de fin pour tous les travaux (K≡C, L, T, E)

Le programme VBA :

```
Dim oSIMAN As Arena.SIMAN
Dim d (7, 75), m1 (7, 75), m2 (7, 75), m3 (7, 75),
ma (10, 75), M (15, 75) As Double
Dim v(75), v1(75), v2(75), v3(75), ve(75),
VBS(75) As Double
Dim BS, SPT1, TNOW, x4, x5, x12, dn, BSP1,
BSP3, decalageglobal1As Double
Dim c, f, taille1, taille2, taille3, compteur1,
compteur2, n, e, g, p, i, j, o, ntp, nbremach, As
Double
Private Sub ModelLogic_RunBeginSimulation()
Set oSIMAN = ThisDocument.Model.SIMAN
End Sub

Sub remplirpy(mm, ee, ii)
For i1 = ee + 1 To ii
For j1 = 1 To 3

py(j1, i1 - ee) = mm(j1, i1)
Next j1
py(4, i1 - ee) = mm(6, i1)
Next i1
End Sub
Sub trii(ie)
For i1 = 1 To ie
For j1 = i1 + 1 To ie
If py(1, i1) > py(1, j1) Then
For k1 = 1 To 4
cc = py(k1, i1)
py(k1, i1) = py(k1, j1)
py(k1, j1) = cc
Next k1
End If
Next j1
'MsgBox "py( 1 , " & i1 & " ) = " & py(1, i1)
Next i1
End Sub
Sub remplirpp(ie)
colonne = 2
ns = 1
pp(1, 1) = py(4, ie)
For i1 = ie - 1 To 1 Step -1
If py(1, i1) < py(1, ie) And py(3, i1) > py(3, ie) Then
pp(1, colonne) = py(4, i1)
colonne = colonne + 1
End If
Next i1
tailles(1) = colonne - 1
'vérifier si une tâche appartirnt a deux pyramides

For i1 = ie - 1 To 1 Step -1
For k1 = 1 To ns
For k2 = 1 To tailles(k1)
If pp(k1, k2) = py(4, i1) Then
GoTo nexti1
End If
Next k2
Next k1
ns = ns + 1
pp(ns, 1) = py(4, i1)
colonne = 2
For j1 = i1 - 1 To 1 Step -1
If py(1, i1) > py(1, j1) And py(3, i1) < py(3, j1) Then
pp(ns, colonne) = py(4, j1)
colonne = colonne + 1
'MsgBox "pp( " & ns & " , " & colonne & " ) = " & pp(ns,
j1)
End If
'MsgBox "pp( " & ns & " , " & colonne & " ) = " & pp(ns,
j1)
Next j1
 tailles(ns) = colonne - 1
'MsgBox ns
'MsgBox colonne
'MsgBox "pp( " & ns & " , " & colonne & " ) = " & pp(ns,
j1)
'MsgBox "pyramide  " & ns & "  de taille =  " & tailles(ns)
nexti1: Next i1
For i1 = 1 To ns / 2
For j1 = 1 To ie
cc = pp(i1, j1)
pp(i1, j1) = pp(ns - i1 + 1, j1)
pp(ns - i1 + 1, j1) = cc
'MsgBox "pp( " & i1 & " , " & j1 & " ) = " & pp(i1, j1)
Next j1
cc = tailles(i1)
tailles(i1) = tailles(ns - i1 + 1)
tailles(ns - i1 + 1) = cc
Next i1
End Sub

Sub remplirseqq(ie)
Dim k1, ii, i1, i2, j2, jj, t, n, mm, mm2, p, l, mm11,
mm1(1000, 1000), ind(1000), fr(1000), seqq(80, 100000),
vv(100000) As Integer
Dim nb As Double
Dim seq1(1000)  As Single
k1 = 0
For i1 = 1 To ns - 1
For t = 1 To tailles(i1) - 1
seq1(k1 + t) = pp(i1, tailles(i1) - t + 1)
Next t
'seq1(k1+1:k1+tai1lles(i1)-1)=pp(i1,tailles(i1):-1:2);
k1 = k1 + tailles(i1)
seq1(k1) = pp(i1, 1)
For jj = 2 To tailles(i1)
n = 0
For mm = 2 To tailles(i1 + 1)
If (pp(i1 + 1, mm) = pp(i1, jj)) Then
n = n + 1
End If
Next mm
If (n = 0) Then
k1 = k1 + 1
seq1(k1) = pp(i1, jj)
End If
Next jj
Next i1
For t = 1 To tailles(ns) - 1
seq1(k1 + t) = pp(ns, tailles(ns) - t + 1)
Next t
k1 = k1 + tailles(ns)
```

```
seq1(k1) = pp(ns, 1)
For t = 1 To tailles(ns) - 1
seq1(k1 + t) = pp(ns, tailles(ns) - t + 1)
Next t
k1 = k1 + tailles(i1) - 1
mm = k1
For i1 = 1 To ie
For jj = 1 To mm
mm1(i1, jj) = 0
Next jj
Next i1
nb = 1
For i1 = 1 To ie
k = 0
For jj = 1 To mm
If (seq1(j) = i1) Then
k = k + 1
mm1(i1, k) = jj
End If
Next jj
nb = nb * k
fr(i1) = nb
ind(i1) = k
Next i1
k = 1
For i1 = 1 To mm
vv(i1) = 0
Next i1
For i1 = 1 To ie
vv(mm1(i1, 1)) = i1
Next i1
mm2 = 0
For i1 = 1 To mm
If (vv(i1) <> 0) Then
mm2 = mm2 + 1
seqq(mm2, 1) = vv(i1)
End If
Next i1
Do While (k < nb)
For i1 = 1 To mm
vv(i1) = 0
Next i1
k = k + 1
p = (k - 1 Mod ind(1)) + 1
vv(mm1(1, p)) = 1
For i1 = 2 To ie
p = (((k - 1) \ fr(i1 - 1)) Mod ind(i1)) + 1
vv(mm1(i1, p)) = i1
Next i1
mm2 = 0
For i1 = 1 To mm
If (vv(i1) <> 0) Then
mm2 = mm2 + 1
seqq(mm2, k) = vv(i1)
End If
Next i1
Loop
'la matrice est seqq et le nobre de lignes est k
'eliminer les séquences qui contiennent des tâches en
retard
k = mm
For i1 = 1 To k
temps = TNOW
For i2 = 1 To ie
temps = temps + d(2, seqq(i1, i2))
If d(1, seqq(i1, i2)) + d(2, seqq(i1, i2)) > temps Then
temps = d(1, seqq(i1, i2)) + d(2, seqq(i1, i2))
End If
If temps > d(3, seqq(i1, i2)) Then
seqq(i1, 1) = 0
GoTo nexti1
End If
Next i2
nexti1:  Next i1
k1 = 0
For i1 = 1 To k
If seqq(i1, 1) = 0 Then
For i2 = i1 + 1 To ie
For i3 = 1 To ie
seqq(i2 - 1, i3) = seqq(i2, i3)
Next i3
Next i2
k1 = k1 + 1
End If
Next i1
k = k - k1
For i1 = 1 To k
For i2 = 1 To ie
' MsgBox "seqq( " & i1 & " , " & i2 & ") = " & seqq(i1,
i2)
Next i2
Next i1
For col = 1 To k
temps = TNOW
For i1 = 1 To ie
If d(1, seqq(col, i1)) < temps Then
temps = temps + d(2, seqq(col, i1))
' MsgBox "oui"
 Else
 temps = d(2, seqq(col, i1)) + d(1, seqq(col, i1))
' MsgBox "non"
 End If
 Next i1
cmaxxx(col) = temps
Next col
'classer les séquence selon l'ordre croissant du C max
For i1 = 1 To k
For p1 = i1 + 1 To k
If cmaxxx(i1) > cmaxxx(p1) Then
Z = cmaxxx(i1)
cmaxxx(i1) = cmaxxx(p1)
cmaxxx(p1) = Z
For j1 = 1 To ie
zz = seqq(i1, j1)
seqq(i1, j1) = seqq(p1, j1)
seqq(p1, j1) = zz
Next j1
End If
Next p1
Next i1
End Sub

Sub validerseqq(mmm, jj, ie, ee)
For i1 = 1 To ie
For j1 = 1 To 3
mmm(j1, i1 + ee) = d(j1, seqq(jj, i1))
Next j1
mmm(6, i1 + ee) = seqq(jj, i1)
```

```
Next i1
temps = TNOW
For i1 = 1 + ee To ie + ee
'MsgBox i1
If mmm(1, i1) < temps Then
mmm(4, i1) = temps
temps = temps + mmm(2, i1)
Else
mmm(4, i1) = mmm(1, i1)
temps = mmm(1, i1) + mmm(2, i1)
End If
Next i1
temps = mmm(3, ie + ee) - mmm(2, ie + ee)
mmm(7, ie + ee) = temps
For i1 = ie - 1 To 1 Step -1
If temps < mmm(3, i1 + ee) Then
temps = temps - mmm(2, i1 + ee)
mmm(7, i1 + ee) = temps
Else
temps = mmm(3, i1 + ee) - mmm(2, i1 + ee)
mmm(7, i1 + ee) = temps
End If
Next i1
End Sub

Sub affectmachine(am, vv, ad, antp)
For R = 1 To antp
k = 1
For l = 1 To am
If vv(l) < vv(k) Then
k = l
End If
Next l
If ad(1, R) > vv(k) Then
ad(4, R) = ad(1, R)
vv(k) = ad(1, R) + ad(2, R)
ad(5, R) = k
Else
ad(4, R) = vv(k)
vv(k) = vv(k) + ad(2, R)
ad(5, R) = k
End If
Next R
End Sub
Sub matmach(ma, mi, mm, mee, md, mntp)
mi = mee
For l = 1 To mntp
If md(5, l) = ma Then
mi = mi + 1
mm(6, mi) = l
For k = 1 To 4
mm(k, mi) = md(k, l)
Next k
End If
Next l
For l = 1 To mi
For k = l To mi
If mm(1, l) = mm(1, k) And mm(3, l) > mm(3, k)
Then
For n = 1 To 6
yy = mm(n, l)
mm(n, l) = mm(n, k)
mm(n, k) = yy
Next n
End If
Next k
Next l
End Sub
Sub calcultbi(mi, mm, ce)
mm(7, mi) = mm(3, mi) - mm(2, mi)
For l = 1 To mi - ce - 1
n = mi - l
d1 = mm(3, n) - mm(2, n)
d2 = mm(7, n + 1) - mm(2, n)
If d1 <= d2 Then
mm(7, n) = d1
Else
mm(7, n) = d2
End If
Next l
End Sub
Function faisabilite(mi, mm, fe)
faisabilite = 0
For l = fe + 1 To mi
If mm(4, l) > mm(7, l) Then
faisabilite = 1
End If
Next l
End Function
Function calculBS(cm)
BS = VBS(1)
For l = 1 To cm
If VBS(l) > BS Then
BS = VBS(l)
End If
Next l
calculBS = BS
End Function
Sub affect(am)
For l = 1 To am
If v(l) = compteur2 Then
oSIMAN.EntityAttribute(oSIMAN.ActiveEntity,
oSIMAN.SymbolNumber("mach")) = l
End If
Next l
End Sub
Sub tachesuivante(mach, taille9, matrix)
If taille9 <> 0 Then
s = matrix(mach, 1)
For n = 1 To taille9
matrix(mach, n) = matrix(mach, n + 1)
Next n
matrix(mach, taille9 + 1) = s
End If
End Sub
Function decalageglobal(dnm)
dg = 0
For dgn = 1 To dnm
dg = M(dgn * 3 + 2, ma(dgn, 1)) + dg
Next dgn
decalageglobal = dg
End Function

Function taille(mac, matrix, AR)
For tl = 1 To AR + 1
If matrix(mac, tl) = 0 Then
taille9 = tl - 1
GoTo 20
```

```
End If
Next tl
20 taille = taille9
End Function
Sub tri(tnm)
BSP1 = 0
For tm = 1 To tnm
For tn = 1 To compteur1
If M(tm * 3 + 1, tn) <> 0 Then
ma(tm, tn) = tn
End If
Next tn
Next tm
For tm = 1 To tnm
For tn = 1 To compteur1
If ma(tm, tn) = 0 Then
For tk = tn To compteur1
ma(tm, tk) = ma(tm, tk + 1)
Next tk
End If
Next tn
Next tm
For n = 1 To nbremach
For l = 1 To compteur1
For k = l To compteur1
If M(1, ma(n, l)) > M(1, ma(n, k)) And M(1, ma(n,
k)) <> 0 And M(1, ma(n, l)) Then
yy = ma(n, l)
ma(n, l) = ma(n, k)
ma(n, k) = yy
Else
If M(1, ma(n, l)) = M(1, ma(n, k)) And M(2, ma(n,
l)) > M(2, ma(n, k)) And M(1, ma(n, k)) <> 0 And
M(1, ma(n, l)) Then
yy = ma(n, l)
ma(n, l) = ma(n, k)
ma(n, k) = yy
End If
End If
Next k
Next l
Next n
End Sub
Function verification(vnm, matrix)
verification = 0
For vl = 1 To vnm - 1
If matrix(vl, 1) <> 0 Then
For vk = vl + 1 To vnm
If matrix(vk, 1) = matrix(vl, 1) Then
verification = 1
End If
Next vk
End If
Next vl
End Function
Function nombreTAT(nnm)
nombreTAT = 0
For nn = 1 To nnm
If ma(nn, 1) <> 0 Then
nombreTAT = nombreTAT + 1
End If
Next nn
End Function
Function BSP(bnm)
BSP2 = 0
For bn = 1 To bnm
If ma(bn, 1) <> 0 Then
If BSP2 < M(bn * 3, ma(bn, 1)) Then
BSP2 = M(bn * 3, ma(bn, 1))
End If
End If
Next bn
BSP = BSP2
End Function
Function SPT(snm)
SPT = 0
For sn = 1 To snm
If ma(sn, 1) <> 0 Then
SPT = SPT + M(1, ma(sn, 1))
End If
Next sn
End Function
Function EDD(enm)
EDD = 0
For en = 1 To enm
If ma(en, 1) <> 0 Then
EDD = EDD + M(2, ma(en, 1))
End If
Next en
End Function
Sub affectcombine(anm, vv, maa)
For an = 1 To anm
vv(an) = maa(an, 1)
Next an
End Sub
Sub mat(mt, et, vt, xt, b, it)
pr =
oSIMAN.EntityAttribute(oSIMAN.ActiveEntity,
oSIMAN.SymbolNumber("pr"))
If xt <= TNOW Then
If et = it Then
If TNOW + pr <= BS Then
M(3 * b, compteur1) = BS
M(3 * b + 1, compteur1) = 1
M(3 * b + 2, compteur1) = 0

Else
M(3 * b, compteur1) = TNOW + pr
M(3 * b + 1, compteur1) = 2
M(3 * b + 2, compteur1) = 0
End If
Else
If TNOW + pr <= mt(4, et + 1) Then
M(3 * b, compteur1) = BS
M(3 * b + 1, compteur1) = 1
M(3 * b + 2, compteur1) = 0
Else
mt(2, et) = pr
vt(et) = TNOW
For l = et + 1 To it
vt(l) = mt(1, l)
If mt(1, l) < vt(l - 1) + mt(2, l - 1) Then
vt(l) = vt(l - 1) + mt(2, l - 1)
End If
Next l
If vt(it) + mt(2, it) <= BS And TNOW + pr <=
mt(7, et + 1) Then
M(3 * b, compteur1) = BS
```

```
M(3 * b + 1, compteur1) = 2
M(3 * b + 2, compteur1) = TNOW + pr - mt(4, et +
1)
Else
If TNOW + pr <= mt(7, et + 1) Then
M(3 * b, compteur1) = vt(it) + mt(2, it)
M(3 * b + 1, compteur1) = 3
M(3 * b + 2, compteur1) = TNOW + pr - mt(4, et +
1)
End If
End If
End If
End If
End If
End Sub
Sub test(tm)
If verification(tm, ma) = 0 Then
If BSP1 = 0 Then
NTAT = nombreTAT(tm)
BSP1 = BSP(tm)
SPT1 = SPT(tm)
EDD1 = EDD(tm)
decalageglobal1 = decalageglobal(tm)
affectcombine tm, v, ma
Else
If NTAT < nombreTAT(tm) Then
NTAT = nombreTAT(tm)
BSP1 = BSP(tm)
SPT1 = SPT(tm)
decalageglobal1 = decalageglobal(tm)
EDD1 = EDD(tm)
affectcombine tm, v, ma
Else
If NTAT = nombreTAT(tm) Then
If BSP1 > BSP(tm) Then
NTAT = nombreTAT(tm)
BSP1 = BSP(tm)
SPT1 = SPT(tm)
EDD1 = EDD(tm)
decalageglobal1 = decalageglobal(tm)
affectcombine tm, v, ma
Else
If BSP1 = BSP(tm) Then
If decalageglobal(tm) < decalageglobal1 Then
SPT1 = SPT(tm)
EDD1 = EDD(tm)
decalageglobal1 = decalageglobal(tm)
affectcombine tm, v, ma
Else
If decalageglobal(tm) = decalageglobal1 Then
If SPT(tm) < SPT1 Then
SPT1 = SPT(tm)
EDD1 = EDD(tm)
decalageglobal1 = decalageglobal(tm)
affectcombine tm, v, ma
Else
If SPT1 = SPT(tm) Then
If EDD(tm) < EDD1 Then
EDD1 = EDD(tm)
affectcombine tm, v, ma
decalageglobal1 = decalageglobal(tm)
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End Sub
‘Ordonnancement des tâches programmables
Private Sub VBA_Block_1_Fire()
ntp = 75
nbremach = 2
'Lecture des parametres
c = c + 1
d(1, c) =
oSIMAN.EntityAttribute(oSIMAN.ActiveEntity,
oSIMAN.SymbolNumber("r"))
d(2, c) =
oSIMAN.EntityAttribute(oSIMAN.ActiveEntity,
oSIMAN.SymbolNumber("pr"))
d(3, c) =
oSIMAN.EntityAttribute(oSIMAN.ActiveEntity,
oSIMAN.SymbolNumber("dl"))
'Affectation et ordonnancement des taches
programmables
If c = ntp Then

'calcul des ri et des di
d(3, 1) = d(1, 1) + d(2, 1) + d(3, 1) 'd1
For l = 2 To ntp
d(1, l) = d(1, l - 1) + d(1, l) 'ri
d(3, l) = d(1, l) + d(2, l) + d(3, l) 'di
Next l
'Affectation aux machines
affectmachine nbremach, ve, d, ntp
'Séparation de la matrice
matmach 1, i, m1, 0, d, ntp
matmach 2, j, m2, 0, d, ntp
matmach 3, o, m3, 0, d, ntp
'calcul des Tbi
calcultbi i, m1, e
calcultbi j, m2, g
calcultbi o, m3, p
'Vérification de la faisabilité de
l'ordonnancement
fsa = 0
If faisabilite(i, m1, e) = 1 Or faisabilite(j, m2, g) = 1
Or faisabilite(o, m3, p) = 1 Then
fsa = 1
End If
If fsa = 1 Then
MsgBox "L'ordonnancement des tâches
programmables ne respecte pas les délais"
End If
'Calcul de la borne supérieure de
l'ordonnancement statique
VBS(1) = m1(4, i) + m1(2, i)
VBS(2) = m2(4, j) + m2(2, j)
VBS(3) = m3(4, o) + m3(2, o)
oSIMAN.VariableArrayValue(oSIMAN.SymbolNu
mber("Makespan")) = calculBS(nbremach)
'Calcul des durées entre créations selon les Ri
d(7, 1) = d(1, 1)
```

```
For l = 2 To ntp
d(7, l) = d(1, l) - d(1, l - 1)
Next l
'Calcul du plus grand délai
dn = d(3, 1)
For l = 1 To ntp
If d(3, l) > dn Then
dn = d(3, l)
End If
Next l
End If
End Sub
‘Ordonnancement des tâches aléatoires
Private Sub VBA_Block_37_Fire()
compteur1 = compteur1 + 1
TNOW =
oSIMAN.VariableArrayValue(oSIMAN.SymbolNu
mber("X7"))
pr =
oSIMAN.EntityAttribute(oSIMAN.ActiveEntity,
oSIMAN.SymbolNumber("pr"))
dl =
oSIMAN.EntityAttribute(oSIMAN.ActiveEntity,
oSIMAN.SymbolNumber("dl"))
x4 =
oSIMAN.VariableArrayValue(oSIMAN.SymbolNu
mber("X4"))
x5 =
oSIMAN.VariableArrayValue(oSIMAN.SymbolNu
mber("X5"))
x12 =
oSIMAN.VariableArrayValue(oSIMAN.SymbolNu
mber("X12"))
'Condition d'arrêt
If TNOW >= dn Then
GoTo 1
End If
'Tâches dépassant leurs délais
If TNOW + pr > dl Then
oSIMAN.EntityAttribute(oSIMAN.ActiveEntity,
oSIMAN.SymbolNumber("mach")) = 9
GoTo 1
End If
'Remplissage de la matrice de données
M(1, compteur1) = pr
M(2, compteur1) = dl
'machine1
mat m1, e, v1, x4, 1, i
'machine2
mat m2, g, v2, x5, 2, j
'machine3
mat m3, p, v3, x12, 3, o
1 End Sub

Private Sub VBA_Block_4_Fire()
compteur2 = compteur2 + 1
If compteur2 = 1 Then
'Initialisation
tri (nbremach)
taille1 = taille(1, ma, compteur1)
taille2 = taille(2, ma, compteur1)
taille3 = taille(3, ma, compteur1)
For l = 1 To nbremach
v(l) = 0
Next l
'Négociation inter machines de la meilleure
combinaison des tâches aléatoires à traitées
2 'machine1
GoTo 8 'machine2
3 'mach1
If ma(1, 1) = 0 Then
GoTo 16 'affectation des tâches aux machines
End If
tachesuivante 1, taille1, ma
4 'machine2
GoTo 8 'TEST
5 'mach2
If jj2 <= k And jj2 > 0 Then
End If
If ma(2, 1) = 0 Then
tachesuivante 2, taille2, ma
GoTo 3 'mach1
End If
tachesuivante 2, taille2, ma
6   'machine3
GoTo 8 'TEST
7 'mach3
If jj2 <= k And jj2 > 0 Then
'MsgBox "mach3"
End If
If ma(3, 1) = 0 Then
tachesuivante 3, taille3, ma
GoTo 5 'mach2
End If
tachesuivante 3, taille3, ma
8 'TEST
ddss = 0
test nbremach
If jj2 <= k And jj2 > 0 Then
'MsgBox "test"
End If
If ddss = 1 Then
If jj1 <= k And jj1 > 0 Then
 ' MsgBox "cond vérif"
j11 = jj1
End If
If jj2 <= k And jj2 > 0 Then
j22 = jj2
End If
End If
GoTo 7 'mach3
16
 If x4 <= TNOW And v(1) = 0 And e <> i Then ' MsgBox
"machine 1 pas affectée  "
 jj1 = jj1 + 1
If jj1 = 1 Then
remplirpy m1, e, i
trii (i - e)
remplirpp (i - e)
remplirseqq (i - e)
End If
'MsgBox "kb= " & kb
If jj1 <= k Then
validerseqq m11, jj1, i - e, e
VBS(1) = m11(4, i) + m11(2, i)
VBS(2) = m2(4, j) + m2(2, j)
VBS(3) = m3(4, o) + m3(2, o)
ccc = calculBS(nbremach)
```

```
For l = 1 To 20
ma(2, l) = 0
ma(1, l) = 0
m(1 * 3, l) = 0
m(1 * 3 + 1, l) = 0
m(1 * 3 + 2, l) = 0
Next l
For i1 = 1 To compteur1
If TNOW + m(1, i1) <= m(2, i1) Then
mat m11, e, v1, x4, 1, i, i1
End If
Next i1
BSP11 = BSP1
GoTo 44
End If
End If
If j11 <> 0 Then
'MsgBox "m1 changée"
validerseqq m11, j11, i - e, e
For k1 = e To i
For k2 = 1 To 7
m1(k2, k1) = m11(k2, k1)
Next k2
Next k1
End If
If x5 <= TNOW And v(2) = 0 And j <> g Then '
 33
jj2 = jj2 + 1
If jj2 = 1 Then
remplirpy m2, g, j
trii (j - g)
remplirpp (j - g)
remplirseqq (j - g)
 End If
'MsgBox "kb= " & kb
If jj2 <= k Then
validerseqq m22, jj2, j - g, g
VBS(1) = m1(4, i) + m1(2, i)
VBS(2) = m22(4, j) + m22(2, j)
VBS(3) = m3(4, o) + m3(2, o)
ccc = calculBS(nbremach)
For l = 1 To 20
ma(2, l) = 0
ma(1, l) = 0
m(2 * 3, l) = 0
m(2 * 3 + 1, l) = 0
m(2 * 3 + 2, l) = 0
Next l
For i1 = 1 To compteur1
If TNOW + m(1, i1) <= m(2, i1) Then
mat m22, g, v2, x5, 2, j, i1
End If
Next i1
BSP11 = BSP1
GoTo 44
End If
End If
If j22 <> 0 Then
'MsgBox "m1 changée"
validerseqq m22, j22, j - g, g
For k1 = g To j
For k2 = 1 To 7
m2(k2, k1) = m22(k2, k1)
Next k2
Next k1
End If
End If
'16 'Affectation
affect (nbremach)

'Réinitialisation
If compteur1 = compteur2 Then
For k = 1 To nbremach
For l = 1 To 20
ma(k, l) = 0
M(1, l) = 0
M(2, l) = 0
M(k * 3, l) = 0
M(k * 3 + 1, l) = 0
M(k * 3 + 2, l) = 0
Next l
Next k
compteur1 = 0
compteur2 = 0
End If
End Sub
```

Glossaire :

Gestion de production : fonction générale consistant à gérer prévisionnellement et à conduire les produits (leur stockage, leur transformation) de la commande des matières premières à la livraison des produits finis.

Ordonnancement : planning des opérations complété par les affectations des ressources qui vont réaliser ces opérations. Organisation fine du travail, équilibrage de la charge.

Temps réel un système en temps réel est un système qui est capable de réagir dans un horizon temporel fixé par son environnement.

Flexibilité : c'est la liberté dont on dispose durant la phase de mise en œuvre d'un ordonnancement.

Flexibilité séquentielle : capacité à interchanger l'ordre de plusieurs opérations sur une ressource.

Incertitudes : signifie les modifications des données d'un problème d'ordonnancement qui peuvent intervenir entre le calcul d'un ordonnancement et la fin de sa mise en œuvre réelle dans son environnement.

Aléas : évènements du type discret qui entrainent des modifications dans le modèle (une panne machine, commande urgente…)

Ordonnancement robuste : est un ordonnancement qui est insensible à des perturbations imprévues de l'atelier de production.

Ensemble dominant : un ensemble de solutions est dit dominant pour l'optimisation d'un critère donné, s'il contient au moins un optimum pour ce critère.

Heuristique : une méthode de résolution qui fournit rapidement une solution réalisable, pas nécessairement optimale, pour un problème d'optimisation NP-difficile.

Problème NP-difficile : (Non Polynomial), le temps de convergence de l'algorithme est une fonction exponentielle de la taille du problème. Aucun algorithme polynomial n'est connu pour ce genre de problème.

Commande : niveau de traduction de l'ordre en une séquence d'instructions exécutables par la machine.

Évaluation des performances : mise en évidence des impacts d'une décision, passée ou future, sur le positionnement relatif de la finalité, des objectifs, des résultats et des moyens constitutifs de ces systèmes, sur la globalité du cycle de vie de ces impacts, et sous chacun des points de vue concernés.

Gestion prévisionnelle : prise en compte prévisionnelle des besoins, niveaux de stock et charge, regroupe les fonctions de planification, de programmation et d'ordonnancement.

Indicateur de performance : donnée quantifiée qui mesure l'efficacité de tout ou partie d'un processus ou d'un système par rapport à une norme, un plan ou un objectif déterminé et accepté dans le cadre d'une stratégie d'entreprise.

Modèle : vue « logique » (c'est-à-dire traitant des informations, par opposition à la vue « physique » traitant de l'énergie ou de la matière) et réductrice d'une réalité afin d'en améliorer la compréhension.

Processus : un processus est constitué d'un ensemble d'activités identifiées en nombre fini et concourant chacune à la réalisation d'un objectif commun.

Programmation : définition des besoins nets, jalonnement, création du programme prévisionnel de production.