LES PROBLÈMES D'ORDONNANCEMENT

META-HEURISTIQUE, 2014

REALISE PAR :

AYOUB ELHATARTI - YAHYA EL FAKIR - OLAYINKA SUNDAY FOLORUNSO - SARA WISSAD - OUSSAMA ZAKI - TAOUFIQ ELHATTAMI

ENSA Marrakech

Introduction générale

La situation actuelle des entreprises manufacturières a connu de grands changements. S'intéresser uniquement à *produire,* comme c'était le cas autrefois, n'est plus un atout de réussite. L'objectif majeur de toute entreprise actuelle n'est pas seulement la *productivité* mais aussi et surtout la *compétitivité* .Cette compétitivité passe forcément par la diversification du travail, un outil de production de plus en plus flexible et une gestion de production fiable qui devient de plus en plus complexe. L'amélioration de la gestion de production est alors, à la base de toute tentative de changement. Cette fonction vise en effet à organiser le fonctionnement du système de production, et à mieux gérer ses différentes opérations.

Les problèmes d'ordonnancement d'ateliers constituent sûrement pour les entreprises une des difficultés importantes de leurs systèmes de gestion et de pilotage de la production. En effet, c'est à ce niveau que doivent être prises en compte les caractéristiques réelles multiples et complexes des ateliers. Dans ce type d'ordonnancement, les ressources sont généralement des machines, et chaque travail à ordonnancer concerne un produit ou un lot de produits à fabriquer en respectant les gammes de fabrication.

Vu la diversité des systèmes de production au niveau opérationnel, les problèmes étudiés en ordonnancement sont extrêmement divers. Théoriquement, la littérature a l'habitude de distinguer : le Job Shop, le Flow Shop, l'Open Shop, le Mix Shop, les machines parallèles etc.

Parmi les problèmes d'ordonnancement les plus difficiles et les plus étudiés, nous avons le problème d'ordonnancement d'ateliers de type *Job Shop*. Considéré comme un modèle d'ordonnancement intéressant, le problème de Job Shop est très représentatif du domaine général de production. Ce problème correspond, réellement, à la modélisation d'une unité de production disposant de moyens polyvalents utilisés suivant des séquences différentes en fonction des produits. L'objectif consiste à programmer la réalisation des produits de manière à optimiser la production, en respectant un certain nombre de contraintes sur les machines utilisées pour effectuer chaque opération élémentaire entrant dans la fabrication de chaque produit.

La résolution de ce problème de manière optimale s'avère dans la plupart des cas impossible à cause de son caractère fortement combinatoire. Les méthodes exactes requièrent un effort

calculatoire qui croît exponentiellement avec la taille du problème. Alors, des méthodes approchées ont été proposées pour résoudre ce problème en temps raisonnable. Parmi ces méthodes, apparaissent celles dites "Métaheuristiques" dont trois méthodes ont démontré leur efficience dans nombreuses applications : les Algorithmes Génétiques appartiennent aux méthodes évolutives ; la Recherche Tabou et le Recuit Simulé qui sont basés sur le principe de la recherche locale.

L'avantage de ces Métaheuristiques se résume dans leur adaptabilité à tous les problèmes, et le bon compromis qu'elles présentent entre le temps de recherche, et la qualité de résolution. Grâce à ces Métaheuristiques, on peut proposer des solutions approchées pour des problèmes difficiles et de plus grande taille qu'il était impossible de traiter auparavant.

Les Problèmes d'Ordonnancement

1. *Définition du problème d'ordonnancement*

Nombreuses sont les définitions proposées au problème d'ordonnancement. Dans ces définitions, on retrouve l'aspect commun de l'affectation de ressources aux tâches, en recherchant une certaine optimisation.

On peut évoquer parmi ces définitions ce qui suit :

« Etant donné un ensemble de tâches à accomplir, le problème d'ordonnancement consiste à déterminer quelles opérations doivent être exécutées, et à assigner des dates et des ressources à ces opérations de façon à ce que les tâches soient, dans la mesure du possible, accomplies en temps utile, au moindre coût et dans les meilleures conditions »

« Résoudre un problème d'ordonnancement consiste à ordonnancer *i.e.* programmer ou planifier dans le temps, l'exécution des tâches en leur attribuant les ressources nécessaires, matérielles ou humaines de manière à satisfaire un ou plusieurs critères préalablement définis, tout en respectant les contraintes de réalisation »

D'une façon générale, il y a problème d'ordonnancement quand il existe

- Un ensemble de travaux (tâches ou encore jobs ou lots) à réaliser,
- Un ensemble de ressources à utiliser par ces travaux,
- Un programme (calendrier) à déterminer, pour allouer convenablement les ressources aux tâches.

Le problème de l'ordonnancement consiste donc, à programmer dans le temps l'exécution de certaines tâches, en leur allouant les ressources requises, et en respectant les contraintes imposées afin d'atteindre certains objectifs.

Pour la notion d'ordonnancement, on peut envisager sa définition de deux points de vue :

- Comme une fonction de la gestion de production :

Se définit alors, comme étant « l'ensemble des actes de gestion visant à l'établissement d'un

ordre de déroulement des opérations de production qui puisse permettre d'atteindre un certain optimum économique préalablement défini » .

Ou encore « c'est une fonction permettant de déterminer les priorités de passage des produits et leur affectation sur les différentes machines »

- Comme une solution réalisable au problème d'ordonnancement défini ci-avant. C'est le sens qui correspond à l'emploi du terme dans la suite de ce mémoire.

2. *Eléments du problème d'ordonnancement*

Dans la définition du problème d'ordonnancement, quatre éléments fondamentaux interviennent : les tâches, les ressources, les contraintes et les objectifs. Alors, la formulation et la description de ce problème se fait par la détermination de ces quatre éléments dits "de base"

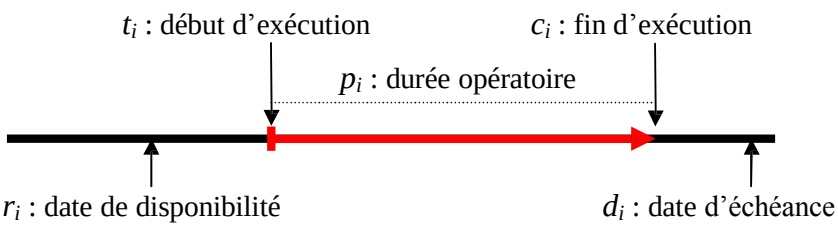
a. Les tâches

Une tâche est un travail mobilisant des ressources et réalisant un progrès significatif dans l'état d'avancement du projet compte tenu du niveau de détail retenu dans l'analyse du problème [Gia, 88].

D'une façon plus schématique : une tâche qu'on note i est une entité élémentaire de travail localisée dans le temps par une date de début t_i ou de fin c_i, dont la réalisation nécessite une durée p_i telle que $p_i = c_i - t_i$, et qui utilise des ressources k avec une intensité a_i^k

Généralement trois paramètres caractérisent une tâche:

- La durée opératoire p_i (*processing time*) : c'est la durée d'exécution de la tâche.
- La date de disponibilité r_i (*release time*) : c'est la date de début au plus tôt.
- La date d'échéance d_i (*due date*) : c'est la date de fin au plus tard.



Caractéristiques d'une tâche i.

Selon les problèmes, une tâche peut être exécutée par morceau ou sans interruption. Notre cas (Job Shop) appartient à la première catégorie, chaque tâche est constituée d'un ensemble d'opérations liées entre elles par des contraintes technologiques. Effectivement, en production manufacturière, on distingue souvent plusieurs phases dans l'exécution d'une tâche : la préparation, la phase principale, la finition, le transport etc.

Nous remarquons ici que certains auteurs utilisent le terme de "*tâche*" pour désigner une opération, et le terme de "*travail*" ou "*job*" pour l'ensemble des opérations constituant le même travail. Dans ce mémoire, nous adoptons la terminologie courante, par l'utilisation des termes tâche et opération, où une tâche est constituée d'un ensemble d'opérations.

b. Les ressources

Une ressource est un moyen destiné à être utilisé pour la réalisation d'une tâche, et disponible en quantités limitées

Dans le contexte industriel, les ressources peuvent être des machines, des ouvriers, des équipements, des locaux ou encore de l'énergie, des budgets, etc.

La disponibilité d'une ressource peut varier dans le temps suivant une fonction $a_k(t)$. Cette disponibilité qui s'appelle la capacité a_k de la ressource k, est une caractéristique qui détermine la quantité de la ressource.

Plusieurs classifications des ressources peuvent être distinguées :

1. Ressources renouvelables *vs*. ressources consommables

Une ressource est dite renouvelable, si après avoir été utilisée par une ou plusieurs tâches, elle est à nouveau disponible en même quantité comme : les hommes, les machines, l'espace, etc.

Par contre, une ressource est consommable, lorsque elle devient non disponible après avoir été utilisée ; comme la matière première, l'énergie, etc.

2. Ressources disjonctives *vs*. ressources cumulatives

Une ressource est dite disjonctive (ou non partageable), si elle ne peut être affecter qu'à une seule tâche à la fois. Ce type concerne surtout les ressources renouvelables.

Par contre, une ressource cumulative (partageable) peut être utilisée par plusieurs tâches simultanément (équipe d'ouvriers, poste de travail, …)

Dans notre problème, les ressources qui sont des machines, sont renouvelables et disjonctives.

c. Les contraintes

Les contraintes expriment des restrictions sur les valeurs que peuvent prendre conjointement une ou plusieurs variables de décision. Donc les contraintes représentent les limites imposées par l'environnement, tandis que l'objectif est le critère d'optimisation.

Plusieurs types de ces contraintes doivent être respectés. On distingue notamment :

1. Les contraintes de localisation temporelle :

Sont issues des impératifs de gestion, et relatives aux dates limites des tâches ou du projet entier. On a surtout :

- La date de disponibilité r_i (avant laquelle la tâche ne peut pas commencer) ;
- La date d'échéance d_i (avant laquelle la tâche doit être achevée).

2. Les contraintes d'enchaînement :

Nous qualifions de contrainte d'enchaînement ou de succession, une contrainte qui lie le début ou la fin de deux activités par une relation linéaire. Ce sont des contraintes imposées généralement par la cohérence technologique (les gammes opératoires dans le cas d'ateliers) qui décrivent des positionnements relatifs devant être respectés entre les tâches.

D'autres contraintes plus spécifiques entre deux tâches ou plus, telles que : la synchronisation, la simultanéité, le recouvrement,… sont également imposées dans certains systèmes.

3. Les contraintes de ressources :

Une contrainte de ressources représente le fait que les activités utilisent une certaine quantité de ressources, tout au long de leur exécution .Ces contraintes sont essentiellement soit :

- Des contraintes d'utilisation des ressources qui expriment la nature, la quantité et les caractéristiques d'utilisation de ces ressources.
- Des contraintes de disponibilité des ressources qui déterminent les quantités des ressources disponibles au cours du temps.

On peut distinguer les contraintes suivant qu'elles sont strictes ou non. Les contraintes strictes sont des obligations à respecter. Alors que, les contraintes dites "relâchables" (appelées aussi préférences) peuvent éventuellement n'être pas satisfaites .

d. Les objectifs

Tout ordonnancement est guidé par un ou plusieurs objectifs qu'il doit chercher leur optimisation. Les objectifs que doit satisfaire un ordonnancement sont variés. D'une manière générale, on distingue plusieurs classes d'objectifs concernant un ordonnancement donnés:

- Les objectifs liés au temps : on trouve par exemple, la minimisation du temps total d'exécution, du temps moyen d'achèvement, des durées totales de réglage ou des retards par rapport aux dates de livraison,
- Les objectifs liés aux ressources : par exemple, maximiser la charge d'une ressource ou minimiser le nombre de ressources nécessaires pour réaliser un ensemble de tâches.
- Les objectifs liés au coût : ces objectifs sont généralement de minimiser les coûts, de lancement, de production, de stockage, de transport, etc.

3. Classification des problèmes d'ordonnancement

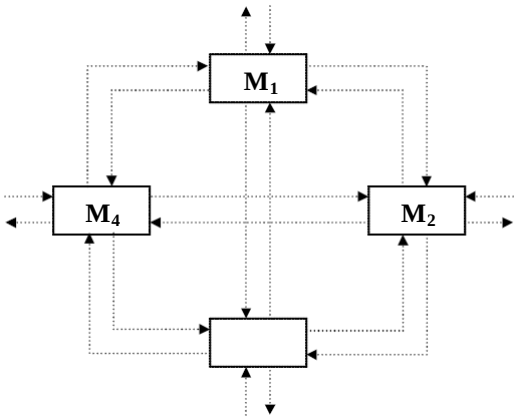
a. Atelier à cheminement multiple : Job Shop

Dans un atelier de type Job Shop les tâches ne s'exécutent pas sur toutes les machines dans le même ordre. En effet, chaque tâche emprunte un "chemin" qui lui est propre.

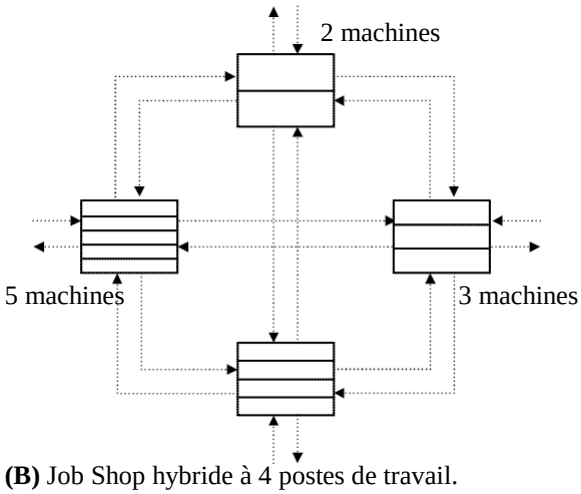
Ce type d'organisation correspond généralement à une production par lot, notamment dans une unité de production disposant de moyens polyvalents utilisés suivant des séquences différentes afin de réaliser des produits divers .

Si, pour chaque machine, on dispose d'un et un seul exemplaire, on dit que l'organisation est un Job Shop simple.

Si, au contraire, pour un au moins des machines (postes de travail), on dispose de plus d'un exemplaire, on l'appelle Job Shop Hybride ;



(A) Job Shop simple à 4 machines.



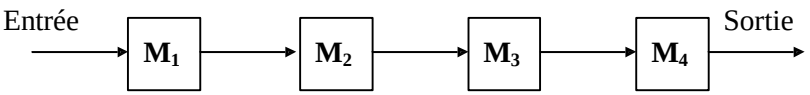
(B) Job Shop hybride à 4 postes de travail.

b. Atelier à cheminement unique : Flow Shop

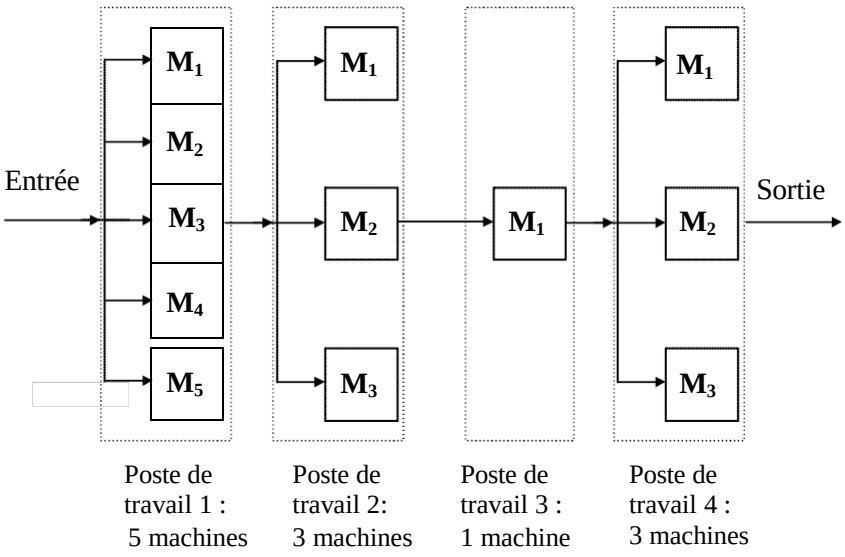
C'est une organisation de production linéaire caractérisée par une séquence d'opérations identiques pour tous les produits. Une tâche est ainsi constituée d'opérations passant par différentes machines de manière linéaire, l'ordre de passage par les machines est le même pour toutes les tâches. Autrement dit, on dispose de *m* machines en série, tous les produits (tâches) doivent passer par chacune des machines : *i.e* toutes les tâches passent par *la machine* 1, puis *la machine* 2, etc.

Ce type se rencontre fréquemment en pratique : montage, chaîne de traitement etc.

Le terme Flow Shop hybride est utilisé pour désigner des variantes de ce problème. C'est le cas rencontré dans plusieurs situations : Pour au moins un des postes de travail on dispose de plus d'un exemplaire de machine pour effectuer le même travail ou lorsque certaines machines sont capables de réaliser simultanément plusieurs travaux ou encore lorsque les temps de rejet sont pris en compte .



(A) Flow Shop simple à 4 machines.



(B) Flow Shop hybride à 4 postes de travail.

Exemples de Flow Shop simple et hybride.

c. Atelier à cheminement libre : Open Shop

Dans les organisations de type Open Shop, les contraintes de précédence sont relâchées [Duv, 00]. Autrement dit, les opérations nécessaires à la réalisation de chaque tâche peuvent être effectuées dans n'importe quel ordre (les gammes sont libres). Ce cas se présente lorsque chaque produit à fabriquer doit subir une séquence d'opérations, mais dans un ordre totalement libre.

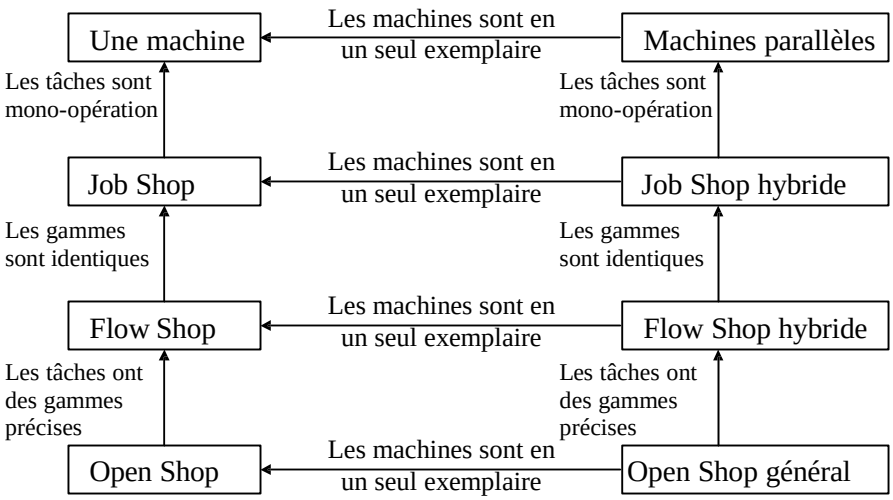
Relations entre organisations

Les organisations présentées précédemment ne sont pas indépendantes. En effet, des relations de ressemblance, de réduction et de différentiation existent entre elles.

L'étude des relations qui existent entre les différents problèmes d'ordonnancement revêt un grand intérêt, dans la mesure où cela permet d'appliquer des algorithmes de résolution connus pour certaines classes de problèmes à d'autres classes qui leurs sont réductibles .

La figure ci-dessous illustre ces différentes relations : chaque arc de *A* à *B* s'interprète que *B* est un cas particulier de *A* en considérant la condition associée. Par exemple, le Flow Shop simple est un cas particulier du Flow Shop hybride, où toutes les machines existent en un seul exemplaire. Il est également, un cas particulier du Job Shop simple avec gammes identiques pour toutes les tâches.

Dans certains cas, ce schéma peut être lu en terme de domaine de solutions réalisables. Par exemple, l'ensemble des solutions réalisables du problème de "machine unique" appartient également à l'ensemble des solutions réalisables du problème de Flow Shop simple .



Relations entre les différentes organisations .

4. Modélisation graphique du problème de Job Shop

Traiter un problème d'une telle complexité que celle du Job Shop nécessite une modélisation claire et précise. Il existe diverses manières de modéliser ce problème, et qui conduisent à choisir un mode de description adéquat.

Si la modélisation par graphe disjonctif est la plus classique et la plus répandue, les modélisations en programmation mathématique sont les plus anciennes .Il existe également, des modélisations algébriques, polyédrales, par les graphes potentiels-tâches, par les réseaux de Pétri, des modélisations basées sur UML etc.

a. Modélisation par graphe disjonctif

La modélisation sous forme d'un graphe disjonctif est très utilisée pour représenter les problèmes d'ordonnancement et en particulier les problèmes de Job Shop. Présentée pour la première fois par B. Roy et B. Sussmann en 1964, cette formulation a été à l'origine des premières méthodes de résolution du problème .

Un problème de Job Shop peut alors se modéliser par un graphe disjonctif G (X , C D) où :

- X : est l'ensemble des *sommets* : chaque opération correspond à un sommet, avec deux opérations fictives S (source) et P (puits) désignant le début et la fin de l'ordonnancement.
- C : est l'ensemble des arcs *conjonctifs* représentant les contraintes d'enchaînement des opérations d'une même tâche (gammes opératoires). Pour un arc (ij,ik) de la partie conjonctive, on a :

$$t_{ij} - t_{ik} \quad p_{i,k} \quad (ij,ik) \quad C$$

On aura donc un arc entre tous les sommets (i,j) , (i,j+1) pour $i = 1...n$ et $j = 1...n_i-1$; (n : nombre de tâches, n_i : nombre d'opérations de la tâche i). De plus, il existe des arcs entre S et tous les sommets (i,1), et d'autres arcs entre les sommets (i, n_i) et le puits P.

- *D* : l'ensemble des arcs *disjonctifs* associés aux conflits d'utilisation d'une machine. Pour un arc (*ij*,*ik*) de la partie disjonctive, on a:

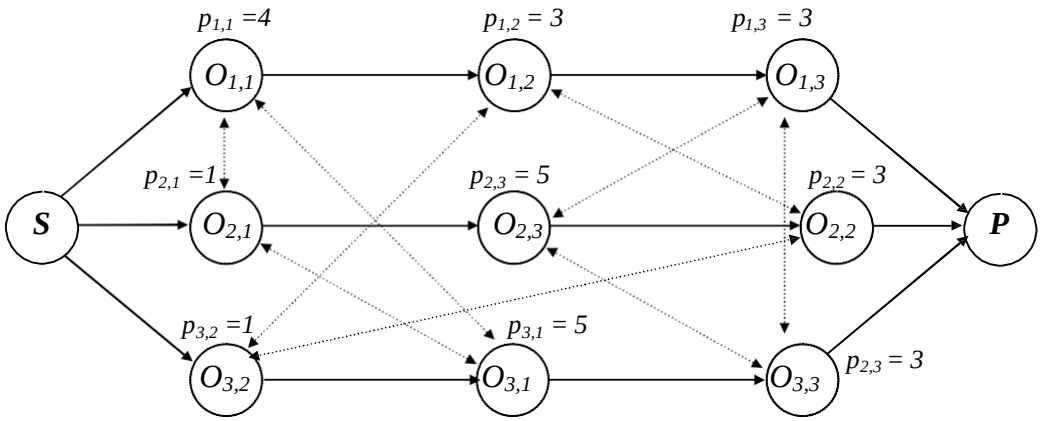
$$t_{jk} - t_{ik} \quad p_{ik} \text{ ou } t_{ik} - t_{jk} \quad p_{jk} \qquad (ik,jk) \quad D$$

Que l'on modélise par un arc et un arc retour entre les sommets (t_{jk}, t_{ik}) représentant deux opérations utilisant la même machine.

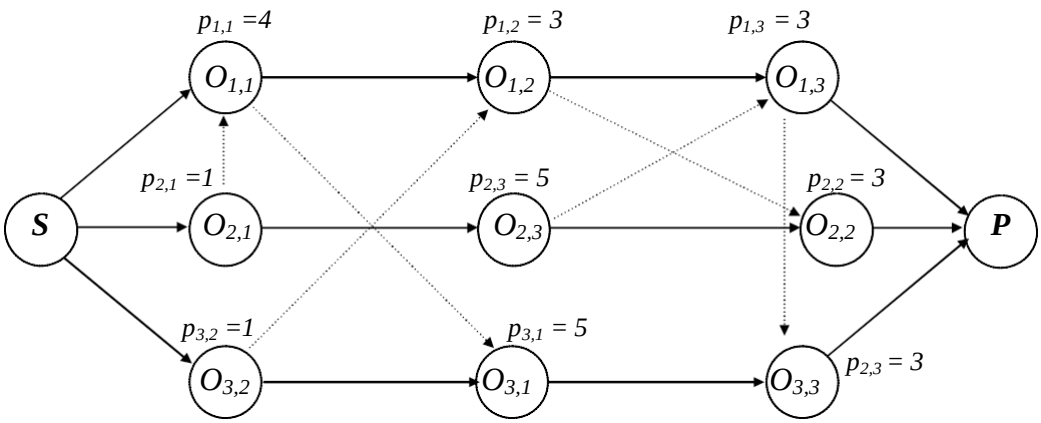
L'ensemble des paires de disjonction associées à une même machine forment une *clique de disjonction*.

Le graphe disjonctif sous cette forme, modélise un problème d'ordonnancement. Pour construire un ordonnancement admissible sur ce graphe, il suffit de choisir pour chaque paire d'arcs, l'arc qui déterminera l'ordre de passage des deux opérations sur la machine ; c'est à dire *arbitrer* chacune des disjonctions. L'arbitrage doit être complet (toutes les disjonctions sont arbitrées), et compatible (le graphe est sans circuits) .

La figure correspond au graphe disjonctif non arbitré du problème précédent .Le graphe arbitré présentant l'ordonnancement de la figure est illustré sur la figure



Le graphe disjonctif du problème de Job Shop



Le graphe disjonctif arbitré simplifié de l'ordonnancement.

Pour un ordonnancement donné, certaines notions bien évidentes sur le graphe disjonctif sont intéressantes à évoquer :

- **L'opération critique :**

Soit un ordonnancement actif, une opération est dite *critique*, si elle provoque nécessairement l'augmentation du makespan de l'ordonnancement, lorsqu'elle est retardée (alors que l'ordre d'exécution des opérations défini par l'ordonnancement ne change pas) [Duv, 00].

- **Le chemin critique :**

Le chemin critique est une suite d'opérations critiques liées par des relations de précédence. La longueur d'un chemin critique est égale à la somme des durées des opérations qui le composent.

Pour un ordonnancement donné, il peut exister plus d'un chemin critique.

- **Le bloc critique :**

Le bloc critique est une succession d'opérations critiques s'exécutant sur la même machine [Hur & Knu, 05]. Tout chemin critique peut se décomposer en *b* blocs critiques. Pour un problème *n*-tâches *m*-machines, *b* est compris entre 1 et *n* : 1 b n.

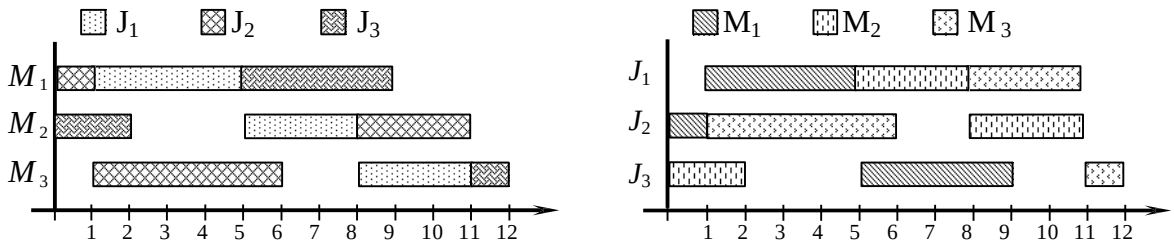
b. Représentation de l'ordonnancement par le diagramme de Gantt

La représentation la plus courante pour l'ordonnancement est le *diagramme de Gantt*. Celui-ci représente une opération par un segment ou une barre horizontale, dont la longueur est proportionnelle à sa durée opératoire.

Donc, sur ce diagramme sont indiqués, selon une échelle temporelle : l'occupation des machines par les différentes tâches, les temps morts et les éventuelles indisponibilités des machines dues aux changements entre produits

Deux types de diagramme de Gantt sont utilisés : Gantt ressources et Gantt tâches :

- Le diagramme de Gantt ressources, est composé d'une ligne horizontale pour chaque ressource (machine). Sur cette ligne, sont visualisées les périodes d'exécution des différentes opérations en séquence et les périodes de l'oisiveté de la ressource.
- Le diagramme de Gantt tâches, permet de visualiser les séquences des opérations des tâches, en représentant chaque tâche par une ligne sur laquelle sont visibles, les périodes d'exécution des opération et les périodes où la tâche est en attente des ressources.



(A) Gantt ressources. **(B)** Gantt tâches.

Diagrammes de Gantt

- ## Complexité du problème de Job Shop

La théorie de complexité permet de classer les problèmes en deux classes P et NP. La classe P regroupe les problèmes résolubles par des algorithmes polynomiaux. Un algorithme est dit polynomial, lorsque son temps d'exécution est borné par *O(p(x))* où *p* est un polynôme et *x* est la longueur d'entrée d'une instance du problème . Les algorithmes dont la complexité ne peut pas être bornée polynomialement sont qualifiés d'exponentiels et correspondent à la classe NP . Les problèmes NP-complets ou NP-difficiles, représentent une large classe parmi la classe NP.

- ## L'importance de l'espace de solutions

Le problème de Job Shop est classé par la théorie de complexité parmi les problèmes NP-difficiles au sens fort .Afin de donner une idée sur l'importance de l'espace de solutions pour une instance du problème, rappelons-nous qu'il existe $(n!)^m$ différentes solutions pour un problème de *n* tâches à réaliser sur *m* machines. Ainsi, pour un problème de taille de
10 10, il existe 39594 10^{65} solutions différentes (par comparaison, l'âge de la Terre ne dépasse pas 10^{18} secondes). Sachant que ce nombre ne comptabilise pas les ordonnancements semi-actifs. Enumérer toutes ces possibilités pour arriver à la solution optimale n'est pas une chose réaliste.

Il est à noter que ce nombre évolue plus vite en fonction de *J* qu'en fonction de *M*. Autrement dit, l'ajout d'une tâche a beaucoup plus d'impact que l'ajout d'une machine. Par exemple :

- Pour un problème de 4 tâches et 5 machines (4 5) : $(n!)^m = 7962624$;
- Alors que, pour un problème de 5 tâches et 4 machines (5 4) : $(n!)^m = 207360000$.

La difficulté du Job Shop est fonction du nombre de tâches, du nombre de machines, du nombre d'opérations par tâche, et de la durée des opérations.

Le tableau, suivant .résume les cas où le problème peut être résolu en temps polynomial et les cas où il devient NP.

P	***NP***	
$m = 2$, j n_j 2	$m = 2$, j n_j 3 [$k, n_k = 3$]	$m = 2$ autre cas
$m = 2$, o d(o) = 1	$m = 2$, o $d(o)$ 2 [$k, d(k) = 3$]	
	$m = 3$, j n_j 2	m 2
	$m = 3$, o $d(o) = 1$	
$n = 2$	n 3	
C_{max} 3	C_{max} 3	

Tableau: Complexité du Job Shop.

n : nombre de tâches, m: nombre de machine, n_j : nombre d'opération par tâche, $d(o)$: durée de l'opération.

Application des Métaheuristiques au problème d'ordonnancement de Job Shop

1. Introduction

Dans le but de résoudre le problème complexe d'ordonnancement des ateliers de type Job Shop par les trois Métaheuristiques retenues : les Algorithmes Génétiques, la Recherche Tabou et le Recuit Simulé, nous présentons dans ce chapitre, de manière synthétique l'implantation et la justification des différents choix adoptés pour notre application qui concernent les techniques, ingrédients et paramètres de ces trois méthodes adaptées au problème de Job Shop.

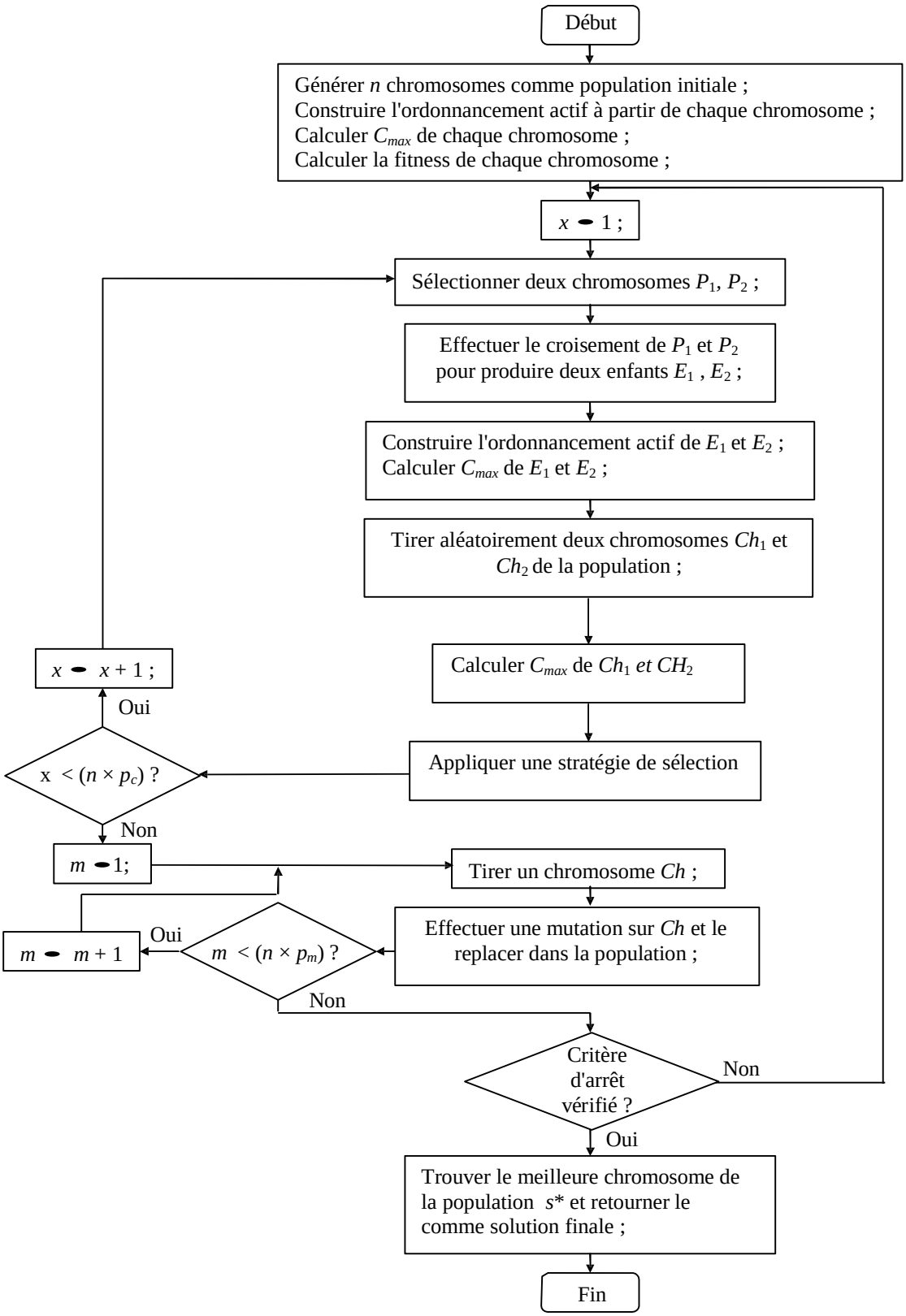
Ce chapitre est subdivisé en quatre sections. Nous consacrons les trois premières à l'implémentation des trois Métaheuristiques retenues. Nous présentons respectivement la résolution du problème par les Algorithmes Génétiques, par la Recherche Tabou et par le Recuit Simulé. Pour chaque méthode, nous discutons les différents choix implémentés, et surtout les algorithmes d'implémentation. Nous terminons le chapitre par une quatrième section qui décrit succinctement le logiciel développé "Meta-JSS" comme résultat de notre implémentation, et qui sert à l'expérimentation sur des problèmes de test.

1. Application des Algorithmes Génétiques

Les Algorithmes Génétiques sont considérés par plusieurs chercheurs comme une méthode bien adaptée au problème de Job Shop, même si elle ne peut pas arriver à l'optimum dans certains cas difficiles .

Notre application porte uniquement sur l'Algorithme Génétique standard, c-à-d, la version basale de la méthode. Des versions plus évoluées et plus sophistiquées ne sont pas appliquées. Cependant, à la place des opérateurs binaires classiques, nous avons utilisé des codages et des

opérateurs améliorés, basés sur des connaissances spécifiques du problème. L'algorithme implémenté est détaillé par l'organigramme de la figure.



Organigramme général de l'Algorithme Génétique implémenté.

La mise en œuvre de notre Algorithme Génétique passe par :

- L'implémentation de la représentation (codage) convenable au problème ;
- L'implémentation des opérateurs génétiques : croisement, mutation, sélection ;
- La détermination des différents paramètres de la méthode : taille de la population, nombre de générations, taux de croisement, taux de mutation,…

a. Le codage des solutions

Le codage est le déterminant important de l'efficacité de la méthode .
Il signifie la transcription d'un ordonnancement réel en représentation adéquate permettant la réalisation des différents opérateurs génétiques. Par conséquent, les deux mots "codage" et "représentation" seront employés dans cette section dans le même sens.

Dans notre approche, un mécanisme de réparation de l'ordonnancement est inclus "implicitement" dans tous les algorithmes de codage. Ceci évite l'utilisation d'un module séparé de réparation. En conséquence, chaque chromosome généré correspond impérativement à un ordonnancement réel actif, donné par l'algorithme de construction de l'ordonnancement correspondant. L'avantage de cette approche est d'alléger les algorithmes d'implémentation ; et d'éviter le remaniement d'ordonnancements non admissibles. Pour cette implantation, comme le montre l'organigramme de la figure 4.1, le passage de l'ordonnancement réel à sa représentation n'existe pas ; tous les algorithmes de codage n'intéressent que le passage d'un chromosome à l'ordonnancement actif.

Dans ce travail, nous adoptons quatre stratégies de codage qui sont : le codage basé sur les tâches (Job Based), le codage basé sur les opérations (Operation Based), le codage basé sur les règles de priorité (Priority Rules Based) et le codage basé sur les listes de préférences (Preference List Based). Notons que les deux premiers codages sont directs, alors que ces deux derniers sont indirects.

- Le codage basé sur les tâches

Un ordonnancement avec cette représentation utilisée par Holsapple & al. consiste en une liste ordonnée de tâches. La construction de l'ordonnancement se fait en fonction de la séquence des tâches sur le chromosome : toutes les opérations de la première tâche s'ordonnancent d'abord, puis celles de la deuxième tâche, et ainsi de suite jusqu'à le dernier gène (dernière tâche). Chaque fois que le rôle d'une tâche arrive, toutes ses opérations seront ordonnancées suivant leur ordre dans la gamme opératoire. Donc, la longueur du chromosome est *n* gènes, où *n* est le nombre de tâches.

Ce principe de construction de l'ordonnancement est implanté suivant l'algorithme 4.1.

Algorithme codage basé sur les tâches ;
Entrée : 1. $Ch[1,...,n]$: un chromosome de n gènes constitué de permutation de $\{J_1,...,J_n\}$;
2. $J_1:[O_{1,1}: p_{1,1},...,O_{1,m} : p_{1,m}],..., J_n:[O_{n,1} : p_{n,1},...,O_{n,m} : p_{n,m}]$: les gammes opératoires des tâches ;
Sortie : • : un ordonnancement actif ;
Début
De j 1 **jusqu'à** n
Faire
Début
De k 1 **jusqu'à** m
Faire Insertion($O_{Ch[j],k}$) ;
Fin ;
Fin.

Algorithme 4.1 : Construction d'ordonnancement avec le codage basé sur les tâches.

La procédure "insertion" que nous avons utilisée à l'intérieur de cet algorithme, est détaillée par l'algorithme 4.2. Cette procédure est également utilisée par d'autres algorithmes de codage.

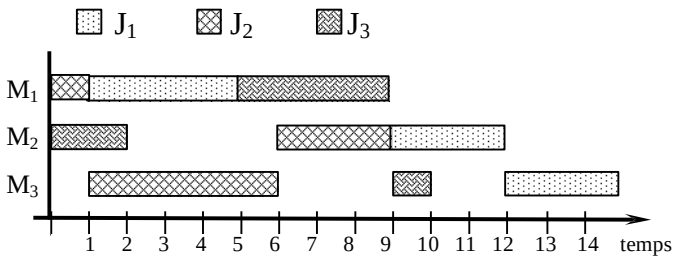
Algorithme Insertion($O_{i,j}$) ;
Entrée : 1. $O_{i,j}$: l'opération à insérer ;
2. M_k / $R(O_{i,j}) = M_k$: La machine demandée par $O_{i,j}$;
3. $J_1:[O_{1,1}: p_{1,1},...,O_{1,m}: p_{1,m}],..., J_n:[O_{n,1}: p_{n,1},...,O_{n,m}: p_{n,m}]$: la gamme opératoire.
Sortie : M_k $O_{i,j}$; où désigne l'opérateur d'ordonnancement de l'opération $O_{i,j}$ sur M_k ;
Début
1. Déterminer $c_{i,j-1}$: fin d'exécution de $O_{i,j-1}$;
2. Trouver sur M_k une période vide p de début t / p $p_{i,j}$, et t $c_{i,j-1}$;
3. $t_{i,j} \prec \max(t, c_{i,j-1})$; $c_{i,j} \prec t_{i,j} + p_{i,j}$;
4. Mise à jour des intervalles vides et occupées de M_k ;
Fin ;

Algorithme 4.2 : Procédure d'insertion d'une opération dans un ordonnancement partiel.

En guise d'exemple, pour le problème défini par les gammes opératoires suivantes :

$J_1[M_1:4 ; M_2:3 ; M_3:3]$; $J_2[M_1:1 ; M_3:5 ; M_2:3]$; $J_3[M_2:2 ; M_1:4 ; M_3:1]$;

La construction de l'ordonnancement pour le chromosome [2,1,3], se fait en ordonnançant d'abord les trois opérations de J_2, puis celles de J_1, ensuite celles de J_3



Ordonnancement construit à partir du chromosome [2 , 1 , 3] codé à base de tâches.

- 2.1.2. Le codage basé sur les opérations

Le codage basé sur les opérations représente un ordonnancement comme une suite d'opérations. Chaque opération est codée par un gène. La désignation de chaque opération peut se faire de deux façons : soit l'opération est désignée par un symbole qui la distingue de toutes les autres, comme le codage utilisé dans le problème du voyageur de commerce (TSP), mais qui est moins fiable pour le Job Shop .Soit, toutes les opérations d'une tâche donnée portent le même symbole (le numéro de la tâche). L'interprétation de ce symbole se fait suivant l'ordre de son occurrence dans la liste. C'est cette dernière approche utilisée par Gen & al. et Bierwirth , que nous avons implémentée pour notre application.

Avec ce codage, pour un problème $n \times m$, le chromosome comprend $n \ \ m$ gènes. Chaque tâche apparaît dans le chromosome exactement m fois (nombre d'opérations). L'occurrence k^{ieme} d'une tâche désigne sa k^{ieme} opération.

L'avantage de cette représentation est qu'elle n'exige qu'un simple constructeur d'ordonnancement , comme celui de notre application, qui est montré par l'algorithme 4.3. Son autre avantage, est qu'elle couvre l'espace de tous les ordonnancements actifs possibles, même si elle génère une certaine redondance.

Algorithme codage basé sur les opérations ;

Entrée : 1. $Ch[1,... , n \ \ m]$: un chromosome constitué de permutation de $\{J_1,...,J_n\}^m$;
2. J_1:$[O_{1,1}$: $p_{1,1},...,O_{1,m}$: $p_{1,m}],..., J_n$:$[O_{n,1}$: $p_{n,1},...,O_{n,m}$: $p_{n,m}]$: la gamme opératoire.

Sortie : • : un ordonnancement actif ;

Début

De j 1 **jusqu'à** $n \ \ m$

Faire

1. Déterminer la dernière opération ordonnancée de $Ch[j] = J_i$, soit $O_{i,k-1}$;
2. Insertion($O_{i,k}$) ;

Fin ;

Fin ;

Algorithme 4.3 : Construction d'ordonnancement par codage basé sur les opérations.

A titre d'exemple, considérons le problème précédent (§ 2.1.1), la figure 4.3 présente le principe de construction de l'ordonnancement à partir du chromosome : CH = [2,3,1,2,1,3,1,2,3], codé à base d'opérations.

Pour construire l'ordonnancement représenté par ce chromosome, on ordonnance les neuf opérations dans l'ordre déterminé par le chromosome qui correspond à : $O_{2,1}$ sur M_1, $O_{3,1}$ sur M_2, $O_{1,1}$ sur M_1, $O_{2,1}$ sur M_3, $O_{1,2}$ sur M_2, $O_{3,2}$ sur M_1, $O_{1,3}$ sur M_3, $O_{2,3}$ sur M_2, $O_{3,3}$ sur M_3.

Où, ($O_{i,k}$ sur M_r : signifie que la $k^{ème}$ opération de la tâche i s'ordonnance sur la machine M_r)

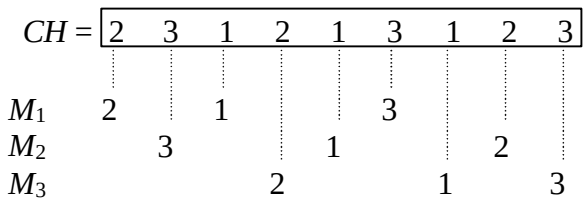


Figure 4.3 : Construction d'ordonnancement à partir d'un chromosome codé à base d'opérations.

- Le codage basé sur les règles de priorité

Les règles de priorité sont une sorte d'heuristiques qui tentent à guider le processus de recherche de solutions. Dans ce codage utilisé d'abord par Dorndorf & al. , le chromosome est une suite de gènes qui représentent des règles de priorité. La construction de l'ordonnancement dans notre approche à partir d'un chromosome ainsi codé, se fait selon l'algorithme Giffler & Thompson (GT) dont la décision sur la prochaine opération à chaque étape est guidée par les gènes du chromosome. Ce principe est détaillé par l'algorithme 4.4.

Algorithme codage basé sur les règles de priorité ;

Entrée : 1. Un chromosome $Ch[1,... , n \quad m \quad 1]$ / $Ch[j] \quad \{R_1,...,R_r\}$: R_i règle de priorité ;
2. J_1:[$O_{1,1}$: $p_{1,1}$,...,$O_{1,m}$: $p_{1,m}$],..., J_n:[$O_{n,1}$: $p_{n,1}$,...,$O_{n,m}$: $p_{n,m}$] : la gamme opératoire.

Sortie : • : un ordonnancement actif ;

Début

Initialiser C comme l'ensemble des premières opérations de toutes les tâches : C={$O_{i,1}$,...$O_{n,1}$} ; et poser pour chaque opération de C, $ES(O_{ij})$ =0, et $EC(O_{ij}) = p_{ij}$;

De $i \quad 1$ **jusqu'à** $n \quad m \quad 1$ **Faire**

Début

1. Déterminer l'opération ordonnançable au plus tôt : $O_{ij} \quad C$, et la machine M_r correspondante ;
2. Déterminer l'ensemble de conflit $C[M_r,i] \quad C$, où i-1 est le nombre d'opérations déjà ordonnancées sur M_r ;
3. Choisir selon $Ch[i]$ une opération de $C[M_r,i]$, et ordonnancer la sur la machine M_r ;
4. Mettre à jour C : enlever la dernière opération ordonnancée de C, ajouter l'opération suivante dans C et calculer ES et EC pour les opérations de C ;

Fin ;

Ordonnancer la dernière opération ;

Fin.

Algorithme 4.4 : Principe d'ordonnancement par codage basé sur les règles de priorité.

C (appelé coupe) est l'ensemble de toutes les opérations ordonnançables pour chaque itération. ES : le temps de début au plus tôt. EC : le temps de fin au plus tôt. $C[M_r,i]$: l'ensemble de conflits qui comprend toutes les opérations susceptibles d'être ordonnancées sur M_r à l'étape i.

Les règles de priorité sont nombreuses. Nous avons utilisé dans notre application dix qui sont déjà employées par plusieurs auteurs comme

EST	*Earliest Starting Time* : La prochaine opération ordonnancée est celle dont la date de début est antérieure à celles des autres opérations ;
LST	*Latest Starting Time* : La prochaine opération ordonnancée est celle dont la date de début est postérieure à celles des autres opérations ;
EFT	*Earliest Finish Time* : La prochaine opération ordonnancée est celle dont la date de fin est antérieure à celles des autres opérations ;
LFT	*Latest Finish Time* : La prochaine opération ordonnancée est celle dont la date de fin est postérieure à celles des autres opérations ;
SPT	*Shortest Processing Time* : La prochaine opération ordonnancée est celle dont la durée est inférieure à celles des autres opérations non encore ordonnancées ;
LPT	*Longest Processing Time* : La prochaine opération ordonnancée est celle dont la durée est supérieure à celles des autres opérations non encore ordonnancées ;
MOPNR	*Most OperatioNs Remaining ou MTR Most Task Remaining* : La prochaine opération ordonnancée est la première opération non encore ordonnancée de la tâche contenant le plus grand nombre d'opérations non encore ordonnancées.
MWKR	*Most WorK Remaining ou LRT Longest remaining processing Time* : La prochaine opération ordonnancée est la première opération non encore ordonnancée de la tâche dont la somme des durées des opérations non encore ordonnancées est la plus grande.
LWKR	*Least WorK Remaining ou SRT Shortest Remaining processing Time* : La prochaine opération ordonnancée est la première opération non encore ordonnancée de la tâche dont la somme des durées des opérations non encore ordonnancées est la plus courte.
RANDOM	La prochaine opération ordonnancée est choisie aléatoirement parmi les opérations non encore ordonnancées.

Les dix règles de priorité utilisées pour le codage à base de règles de priorité.

Dans ce codage, la longueur de chromosome est égale au nombre d'opérations moins un, puisque la dernière opération est prise automatiquement, soit $n \ m$-1 pour un problème $n \times m$.

Par exemple, le chromosome suivant : *CH* = [LPT, LFT, EFT, SPT, LST, EST, SPT] représente un ordonnancement pour un problème de huit opérations.

- Le codage basé sur les listes de préférence

Ce codage proposé d'abord par Davis est utilisé par Falkenauer & al et Croce & al pour la résolution des problèmes de Job Shop simples.

Ce type, appelé aussi "codage avec permutations réparties" (Partitionned Permutation Encoding), utilise pour un problème de Job Shop $n \times m$, des chromosomes constitués de m sous-chromosomes. Chaque sous-chromosome est constitué de n gènes qui sont des listes de préférences de chaque machine. Ces listes, ne décrivent pas la séquence des opérations sur les machines mais l'ordre dans lequel les machines "préfèrent" exécuter les tâches. Ainsi, à chaque étape, devant chaque machine, il existe une file d'opérations qui attendent l'exécution, la machine doit prendre une d'elles suivant sa liste de préférence (algorithme 4.5).

Algorithme codage basé sur les listes de préférences ;
Entrée : 1. $Ch[SCh_1,... , SCh_m]$ où $SCh_i = [1,... , n]$ est une permutation de $\{J_1,...,J_n\}$;
2. J_1:[$O_{1,1}$: $p_{1,1}$,...,$O_{1,m}$: $p_{1,m}$],..., J_n:[$O_{n,1}$: $p_{n,1}$,...,$O_{n,m}$: $p_{n,m}$] : la gamme opératoire.
Sortie : • : un ordonnancement actif ;
Début
Tant que il y a des opérations non ordonnancées **Faire**
Début
Déterminer les opérations actuelles préférées pour chaque machine *Pref*[1,…,*m*] ;
De *j* 1 **jusqu'à** *m*
Faire
Si *Pref*[*j*] = $O_{j,k}$ est ordonnançable i.e $O_{j,k-1}$ est déjà ordonnancée
Alors insertion($O_{j,k}$) ;
Sinon Passer à *Pref*[*j*] +1 sur la liste des préférences;
Fin ;
Fin ;

Algorithme 4.5 : Principe du codage basé sur les listes de préférences.

Ce type de codage est indirect et nécessite par conséquent un constructeur robuste d'ordonnancement afin de décoder les chromosomes en ordonnancement réels.

Pour illustrer le principe de ce codage, considérons le problème de 3-tâches 3-machines précédent (§ 2.1.1) et la représentation suivante : *CH* = [(2,3,1) (1,3,2) (2,1,3)]. (2,3,1) est la liste de préférence de la machine M_1, (1,3,2) de M_2, (2,1,3) de M_3.

Le tableau 4.2 détaille les étapes de construction de l'ordonnancement pour ce problème et la figure 4.4 donne le diagramme de Gantt de son ordonnancement final.

Etape	Opérations préférées	Opérations disponibles	Opérations effectuées
1	$J_2 \rightarrow M_1$, $J_1 \rightarrow M_2$, $J_2 \rightarrow M_3$	$J_1 \rightarrow M_1$, $J_2 \rightarrow M_1$, $J_3 \rightarrow M_2$	$J_2 \rightarrow M_1$
2	$J_3 \rightarrow M_1$, $J_1 \rightarrow M_2$, $J_2 \rightarrow M_3$	$J_1 \rightarrow M_1$, $J_2 \rightarrow M_3$, $J_3 \rightarrow M_2$	$J_2 \rightarrow M_3$
3	$J_3 \rightarrow M_1$, $J_1 \rightarrow M_2$, $J_1 \rightarrow M_3$	$J_1 \rightarrow M_1$, $J_3 \rightarrow M_2$, $J_2 \rightarrow M_2$	-
4	$J_1 \rightarrow M_1$, $J_1 \rightarrow M_2$, $J_1 \rightarrow M_3$	$J_1 \rightarrow M_1$, $J_3 \rightarrow M_2$, $J_2 \rightarrow M_2$	$J_1 \rightarrow M_1$
5	$J_3 \rightarrow M_1$, $J_1 \rightarrow M_2$, $J_1 \rightarrow M_3$	$J_1 \rightarrow M_2$, $J_3 \rightarrow M_2$, $J_2 \rightarrow M_2$	$J_1 \rightarrow M_2$
6	$J_3 \rightarrow M_1$, $J_3 \rightarrow M_2$, $J_1 \rightarrow M_3$	$J_1 \rightarrow M_3$, $J_3 \rightarrow M_2$, $J_2 \rightarrow M_2$	$J_3 \rightarrow M_2$
7	$J_3 \rightarrow M_1$, $J_2 \rightarrow M_2$, $J_1 \rightarrow M_3$	$J_1 \rightarrow M_3$, $J_2 \rightarrow M_2$, $J_3 \rightarrow M_1$	$J_3 \rightarrow M_1$, $J_2 \rightarrow M_2$, $J_1 \rightarrow M_3$
8	$J_3 \rightarrow M_3$	$J_3 \rightarrow M_3$	$J_3 \rightarrow M_3$

Tableau 4.2 : Exemple de construction d'ordonnancement avec un codage basé sur les listes de préférences.

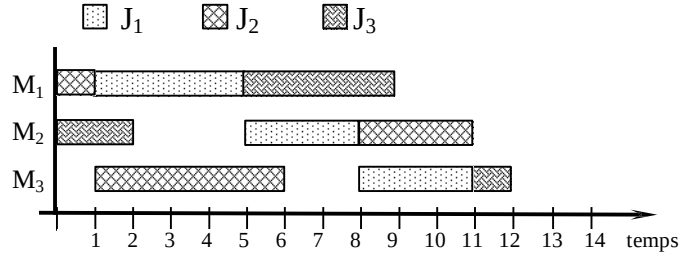


Figure 4.4 : Ordonnancement final donné par les étapes du tableau 4.2.

b. Le croisement

Le croisement est un opérateur très important des Algorithmes Génétiques. L'emploi de croisements simples (uni-point, multi-points) ne semble pas fiable pour le problème de Job Shop. Ainsi, plusieurs types de croisement spécifiques du domaine sont proposés. Dans notre application, nous avons implémenté cinq opérateurs de croisement convenables aux codages utilisés :

- Croisement par échange de sous-séquences, (Subsequence Exchange Crossover SXX).
- Croisement de l'ordre des tâches, (Job Order Crossover JOX).
- Croisement de l'ordre linéaire, (Linear Order Crossover LOX).
- Croisement de l'ordre généralisé (Generalized Order Crossover GOX).
- Croisement par préservation de la précédence, (Precedence Preservation Crossover PPX).

c. La mutation

Le rôle de la mutation est d'apporter une certaine diversité à la population et d'empêcher que celle-ci converge trop vite vers un seul type d'individu. Généralement, la mutation suit deux grands principes : mutation de valeur et mutation de position.

Pour notre problème, plusieurs stratégies de mutation sont proposées dans la littérature, comme : *Multi-step mutation* (MSM), *Multi-Step Mutation Fusion* (MSMF), *Time horizon exchange mutation* (THX-M), *Neighbor Search Mutation* (NSM) Puisque l'effet de la mutation est limité sur l'évolution, nous avons utilisé deux stratégies simples et représentatives des grand principes de mutation, il s'agit de : mutation de position (position-based mutation) et mutation à base de tâche (Job-based Mutation).

d. La sélection

La sélection est appliquée afin de favoriser au cours du temps les individus les mieux adaptés, à les faire se produire (duplication). Dans notre application, trois stratégies de sélection sont utilisées :

e. La population initiale

Avant de lancer l'Algorithme Génétique, une population initiale doit être générée. Cet ensemble d'individus qui servira de base pour les générations ultérieures doit être non

homogène. Afin de satisfaire cette non homogénéité, la génération de la population initiale dans notre application se fait de deux façons :

- Soit, en générant des chromosomes aléatoires au lieu d'ordonnancements réels. Cette approche est exploitable pour tous les codages implantés.
- Soit, en utilisant l'algorithme de GT : cette approche n'est implantée qu'avec le codage basé sur les opérations et celui basé sur les règles de priorité.

f. L'évaluation des individus

Le seul critère que nous avons utilisé pour l'évaluation des individus est le makespan de l'ordonnancement. Sur ce critère, s'appuie l'estimation de *fitness* des individus. L'évaluation de *fitness* des chromosomes se fait en trois étapes :

- La construction de l'ordonnancement correspondant au chromosome,
- Le calcul du makespan de cet ordonnancement,
- L'attribution au chromosome de son makespan comme estimation de son *fitness*.

Ensuite, le tri des chromosomes et leur sélection se fait selon cette valeur de fitness.

2. Application de la Recherche Tabou

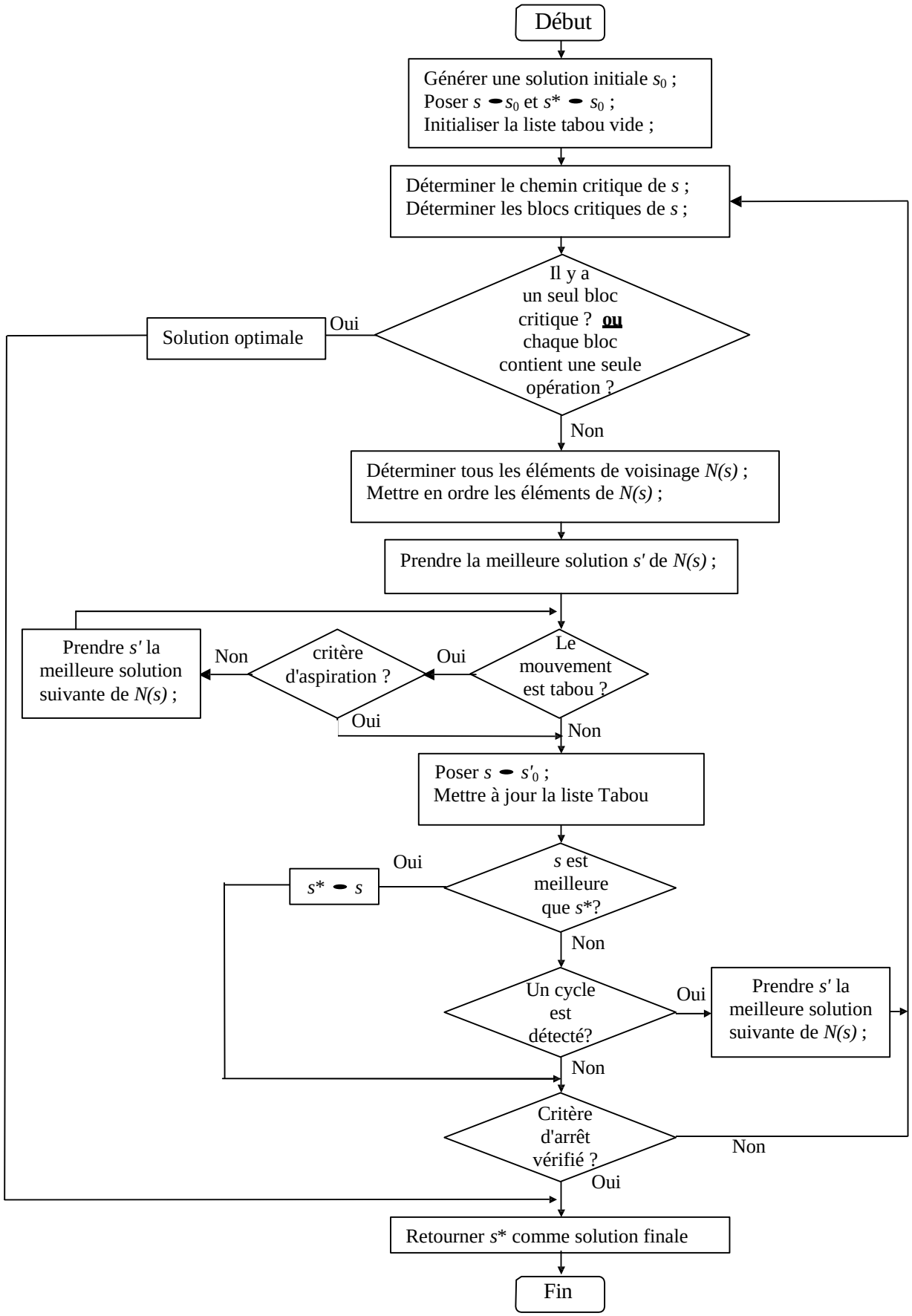
Cette méthode a montré son efficacité pour la résolution de plusieurs problèmes difficiles, parmi lesquels figure le problème de Job Shop .

Notre approche appliquée ici et détaillée par l'organigramme de la figure 4.10, suit l'algorithme standard de la Recherche Tabou. Commençant par une solution (ordonnancement) initiale, le processus de la recherche consiste, à chaque itération, à choisir la meilleure solution qui n'est pas taboue dans le voisinage de la solution courante, même si cette solution n'entraîne pas une amélioration.

Dans le même souci de vouloir exploiter les différents ingrédients de la méthode, nous avons procédé à son implémentation suivant plusieurs stratégies.

Nous présentons dans cette section la mise en œuvre des éléments suivants nécessaires à l'implémentation de la méthode Tabou :

- La génération de la solution initiale,
- La fonction de génération de voisinage,
- L'évaluation de voisinage,
- L'implantation de la liste des tabous,
- L'intégration de quelques mécanismes "standards" renforçant la méthode.



Organigramme général de la méthode de Recherche Tabou implémentée.

a. La génération de la solution initiale

Il a été démontré que l'efficacité des méthodes basées sur le principe de la recherche locale dépend étroitement de la qualité de la solution initiale choisie Dans notre application, trois possibilités de génération de la solution initiale sont exploitées selon la qualité voulue :

- Initialisation par Algorithme Génétique avec un nombre de générations de l'ordre d'une centaine afin d'obtenir une solution initiale de bonne qualité ;
- Initialisation par Algorithme Génétique avec un nombre de générations plus réduit, la solution initiale est alors de qualité moyenne ;
- Initialisation par l'algorithme de Giffler & Thompson aléatoire, la solution initiale générée est de qualité inférieure.

Dans les deux premières possibilités, L'Algorithme Génétique utilisé est codé à base d'opérations.

Le recours à l'Algorithme Génétique pour initialiser la Recherche Tabou (et le Recuit Simulé) est considéré comme une phase d'initialisation de la méthode de recherche locale et non plus une hybridation proprement dite de Métaheuristiques. Effectivement, cette initialisation ne fait pas partie du processus de recherche et peut se faire éventuellement par d'autres méthodes.

b. Les fonctions de voisinage

Le voisinage d'une solution courante est constitué de toutes les solutions obtenues en effectuant un *mouvement élémentaire* sur cette solution. Ce mouvement est défini par la fonction de voisinage qui génère pour une solution donnée, un ou plusieurs voisins.

Toute fonction de génération de voisinage doit, dans certaines limites, respecter deux conditions qui sont : la faisabilité et la connectivité :

- La faisabilité signifie que la fonction de voisinage doit générer des solutions admissibles. Toutes les approches que nous avons utilisées garantissent cette condition, puisque toutes les fonctions ne génèrent que des voisins actifs. Si un voisin n'est pas admissible, il est automatiquement omis, et aucun mécanisme de réparation ne sera appliqué.
- La connectivité ou l'accessibilité d'un optimum global est garantie si l'on peut atteindre une solution optimale depuis n'importe quelle état initial admissible. La majorité des voisinages utilisés satisfont également cette condition. De plus, ne satisfaire pas à cette condition dans

notre application ne pose pas de problèmes, puisque le but n'est pas d'arriver à la solution optimale à tout prix, mais plutôt à une solution proche de l'optimale.

Dans notre application, six fonctions de voisinage sont implémentées. Tous ces voisinages sont basés sur la notion de *bloc critique*. Rappelons-nous, que le bloc critique est l'ensemble d'opérations critiques s'exécutant consécutivement sur une même machine.

Ces voisinages, issus des travaux de plusieurs auteurs sont : *N*1, *N*2, *NA*, *RNA*, *NB* et *NS*.

En s'aidant de quelques exemples, nous présentons dans ce qui suit ces différentes fonctions de voisinage.

c. L'évaluation de voisinage

La complexité d'une approche de résolution basée sur la recherche locale dépend étroitement de la méthode d'évaluation de voisinage afin de déterminer le meilleur voisin selon le critère retenu. Pour cela, il faudrait pouvoir évaluer tous les voisins. Cependant, l'évaluation complète, *i.e.* calcul des dates de début de toutes les opérations, de chaque voisin prend un temps considérable. Il a été démontré que près de 90% du temps de résolution est pris par l'évaluation des voisinages .

Par conséquent, il est intéressant d'utiliser des voisinages aussi restreints que possible afin de diminuer au maximum la complexité de résolution.

Mais, de l'autre côté, la taille de voisinage influe sur le nombre de choix qui s'offrent à la méthode de recherche pour passer de la solution courante à la voisine, plus ce nombre est élevé, plus elle a de chances pour s'échapper de l'optimum local.

Dans notre travail, le compromis entre ces deux situations est obtenu par les considérations suivantes :

- Les voisinages choisis pour notre application sont de taille raisonnable, permettant ainsi de réduire l'effort de leur évaluation, et aussi d'exploiter suffisamment le voisinage.
- Pour évaluer le voisinage nous avons implémenté l'algorithme 4.13, qui vise à calculer seulement les dates de début d'un sous-ensemble d'opérations qui est effectivement concerné par le mouvement et non pas toutes les opérations.

Algorithme Mettre à jour O_j ;
Initialisation : ▪ = [O_j]: ▪ : Liste d'opérations à mettre à jour leurs dates de début ;
Début
Tant que ▪ ▪ ▪ **Faire**
Début
O_i ▪ ▪[1] ; ▪ ▪ ▪ – ▪[1] ;

Mettre à jour la date de début de O_i dans l'ordonnancement ;
Si la date de début de O_i est changée **Alors**
Début
Si $SJ(O_i)$ existe **Alors** : ▬ ▬ ▬ + $SJ(O_i)$;
Si $SM(O_i)$ existe **Alors** : ▬ ▬ ▬ + $SM(O_i)$;
Fin ;
Fin ;
Fin ;

Algorithme 4.13 : Mise à jour de l'ordonnancement après mouvement sur O_j.

$SJ(O_i)$ *:* l'opération suivante de O_i dans la gamme opératoire ;
$SM(O_i)$ *:* l'opération suivante de O_i sur la même machine

Cet algorithme réduit considérablement le temps alloué à l'évaluation de voisinage, puisque seulement un sous-ensemble d'opérations est concerné par la mise à jour des dates de début.

d. La mémoire Tabou

Afin d'éviter le piège des optima locaux dans lequel le processus de recherche peut être facilement attrapé, la Recherche Tabou utilise la mémoire des tabous.

- Gestion de la mémoire des tabous

La structure de la mémoire implantée est dans sa version élémentaire. Elle se réalise à l'aide d'une liste circulaire de taille *k* qui peut être constante ou variable. La gestion de la liste suit la stratégie FIFO, l'ordre de dégagement des éléments de la liste est celui de leur insertion. La mise à jour de la liste des tabous se fait à chaque itération de la recherche. A chaque itération un élément nouveau est introduit et un autre le plus ancien est dégagé de la liste.

Les éléments de la liste doivent comporter suffisamment d'informations pour mémoriser fidèlement la solution visitée. La mémorisation de configurations entières serait trop coûteuse en temps de calcul et en place mémoire et ne serait sans doute pas la plus efficace. Par conséquent, elle n'est pas applicable pour la majorité des problèmes. Couramment, ce sont des caractéristiques des solutions, ou des mouvements, qui se gardent dans la liste, au lieu de solutions complètes.

Pour notre application, les éléments de la liste Tabou sont des mouvements : un mouvement marque la transition faite sur une solution pour passer à son voisin.

- Dans le cas de : *N*1, *N*2 et *NS* : le mouvement est constitué des deux opérations échangées ;
- Dans le cas de : *NA*, *RNA* et *NB* : le mouvement est constitué de l'opération déplacée et sa nouvelle position.

Un mouvement tabou est le mouvement inverse d'un mouvement déjà mémorisé dans la liste des tabous.

- Redimensionnement de la mémoire des tabous

Dans notre implémentation, la taille de la liste des tabous peut avoir deux états : soit fixe, soit dynamique. Dans le cas de la taille dynamique, le redimensionnement de la liste se fait suivant la qualité des solutions découvertes. Ainsi, la liste Tabou raccourcit quand l'itération est amélioratrice, et s'allonge si le contraire. La justification de ce choix est que, lorsque une solution meilleure est découverte, alors il est possible de découvrir des meilleures davantage. Il

faut donner donc, plus de chances au voisinage, en réduisant les mouvements tabous. Mais lorsque la solution découverte est pire, la suivante a de grande chance d'être de même. Il faut augmenter le nombre de mouvements tabous afin d'obliger la recherche de sortir de la zone actuelle.

Le raccourcissement et l'allongement se fait d'un élément à chaque fois en ne dépassant pas les limites supérieure et inférieure de la liste. Ces limites sont fixées préalablement à l'instar de comme suit :

- La limite inférieure *min* est choisie dans l'intervalle [2, 2 + (*n*+*m*) / 3] ;
- La limite supérieure *max* est choisie dans l'intervalle [*min*+6, *min*+6 + (*n*+*m*) / 3] ;

Où *n* et *m* désignent respectivement le nombre de tâches et le nombre de machines.

- Mécanismes de renforcement

Bien que l'objet de notre travail ne concerne que les approches standards de chaque Métaheuristique, nous avons employé deux mécanismes couramment utilisés, pour renforcer la recherche par la méthode Tabou. Ces deux mécanismes sont le critère d'aspiration et la détection de cycle.

- . Le critère d'aspiration

Le critère d'aspiration est « une condition nécessaire et satisfaisante pour qu'un mouvement tabou soit accepté malgré son statut tabou » .

Le critère d'aspiration appliqué ici est celui adopté dans plusieurs applications de la Recherche Tabou au problème de Job Shop, et qui semble plus performant. Ce critère consiste à révoquer le statut tabou d'un mouvement si ce dernier conduira à une meilleure solution obtenue jusqu'à lors.

- Détection de cycle

Dans certaines circonstances, en revanche de l'existence de la liste des tabous, la recherche peut tomber dans un cycle de longueur supérieure à la taille de la liste Tabou qui ne permet pas de l'éviter. La détection de ce cycle est importante, afin de permettre à la recherche de sortir de ce bouclage. Dans notre application cela se fait par le biais de deux mécanismes :

- La technique de la mémoire Tabou dynamique, qui est efficace pour l'évasion de ce cyclage, même si la longueur de cycle est plus grande que la limite maximale de la liste des tabous.

- L’intégration d’un mécanisme de mémorisation à long terme sous forme d’une liste de petite taille (ne dépassant pas une vingtaine d’éléments). Dans cette liste, sont mémorisé par intervalle fixe une petite suite de solutions (en effet les makespans de ces solutions). Lorsque le makespan de la solution actuelle fait partie de cette liste mémorisée, on vérifie le deuxième, si celui-ci existe également, on vérifie le troisième et ainsi de suite. Si enfin, la liste mémorisée est identique à la suite des solutions visitées, alors un cycle est détecté. Son évitement se fait en déposant avec les tabous, le mouvement faisant passer du premier élément de la liste au deuxième. Ce mécanisme peut détecter et éliminer efficacement les cyclages francs où la séquence qui se répète est constante.

3. Application du Recuit Simulé

Le Recuit Simulé est une variante de la recherche locale utilisant la température *T* pour guider la recherche en passant d’une solution à sa voisine.

L’algorithme de Recuit Simulé que nous avons appliqué ici pour la résolution du problème de Job Shop est également standard, et suit le schéma général montré par l’organigramme de la figure 4.15.

Tout comme pour la Recherche Tabou, l’application du Recuit Simulé au problème de Job Shop, nous a amené à déterminer plusieurs choix pour implanter les stratégies de la méthode. Ces choix concernent :

- La génération de la solution initiale,
- La fonction de génération de voisinage,
- L’évaluation de voisinage,
- La fonction de température,
- La fonction "Energie",
- L’acceptation de transition.

Les choix adoptés pour les trois premiers éléments sont exactement ceux présentés pour le cas de la Recherche Tabou. Les mêmes voisinages utilisés auparavant sont réutilisés ici. La génération de la solution initiale et l’évaluation de voisinage se font également de la même manière. Nous allons présenter dans ce qui suit le reste des mécanismes qui sont spécifiques de la méthode : la température, l’énergie et l’acceptation de transition.

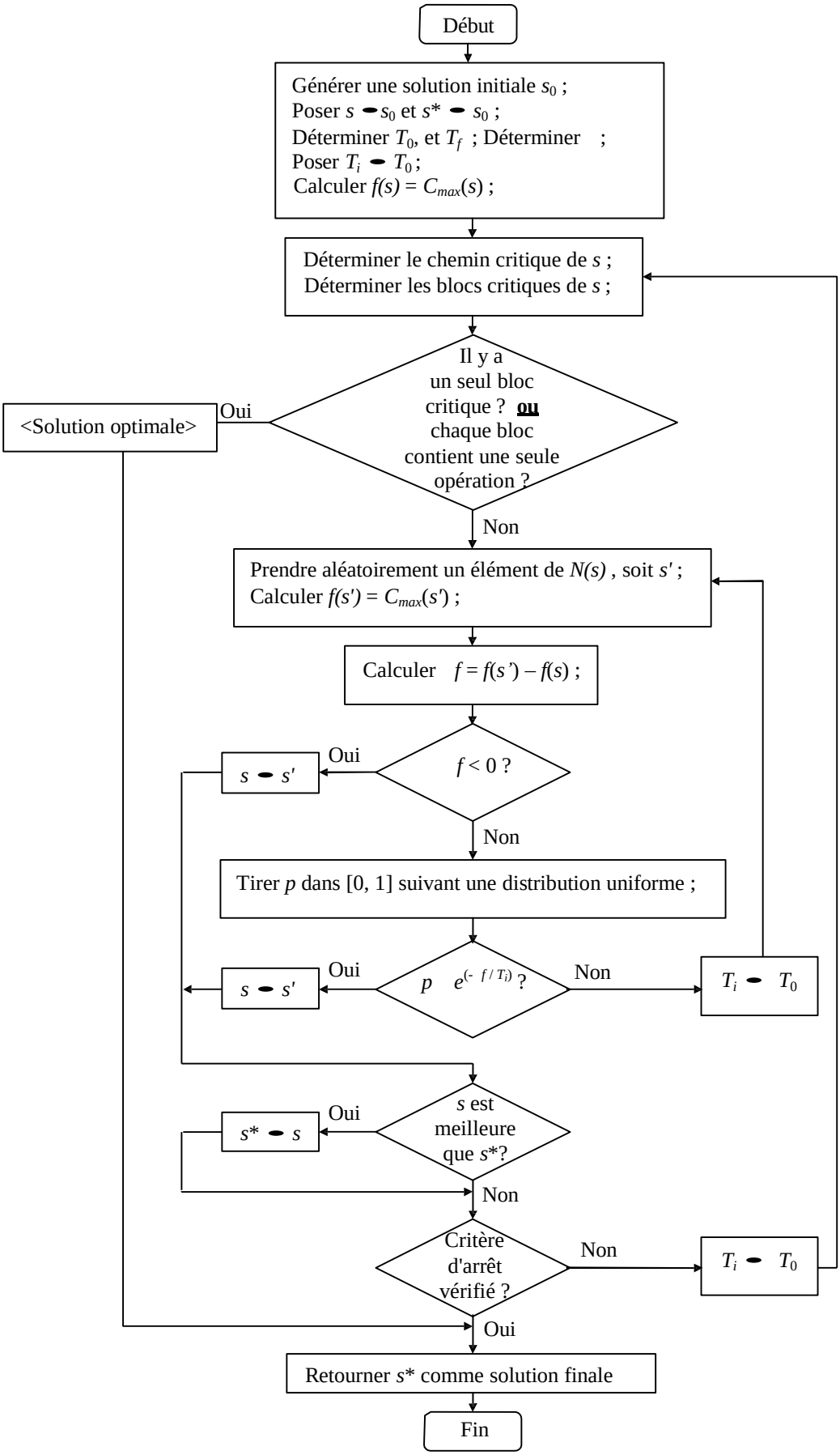


Figure 4.15 : Organigramme général du Recuit Simulé implémenté.

a. La fonction de température

La température est le plus important ingrédient de Recuit Simulé. C'est le mécanisme contrôleur, au cours du temps, de transition d'une solution courante à sa voisine. Dans l'algorithme de Recuit Simulé, la fonction de température s'implémente au travers la détermination de : La Température Initiale, la Température Finale et le schéma de Refroidissement.

- La Température Initiale

Certains auteurs font intervenir des procédures complexes pour déterminer la Température Initiale Dans notre application, l'initialisation de la Température peut se faire, soit par des valeurs arbitraires, soit empiriquement, après une série de tests.

- La Température Finale

Quant à la Température Finale, Nous avons utilisé deux possibilités de détermination. Soit, elle est fixée, comme la valeur initiale (c'est la procédure couramment employée). Soit, la méthode se déroule avec un facteur de décroissance de température arbitraire, la Température Finale dépendra alors du nombre d'itérations.

- Le schéma de Refroidissement

La décroissance de la température se fait graduellement suivant un schéma de Refroidissement. La fonction régissant la température est décroissante. Dans notre application, elle est déterminée par l'une des formules suivantes :

- $T_k = T_0 e^{-ck}$ où :
  - T_k : est la température calculée à l'itération k ;
  - T_0 : est la Températures Initiale ;
  - c : est une constante définie au début par la relation : c = $\log(T_f / T_0)/k$; T_f : est la Température Finale.
- $T_k =$ T_{k-1} avec : [0.85, 0.99]. Dans le cas où la Température Finale est laissée indéterminée [Yam & Nak, 96].

Le Refroidissement avec ces deux fonctions, peut se faire de deux manières :

- Continue : la nouvelle valeur de la température est recalculée à chaque itération k.
- Par paliers : la température est recalculée au début de chaque palier (qui correspond à un certain nombre d'itérations), et demeure inchangée jusqu'à un nouveau palier.

b. La fonction Energie

A chaque itération, une solution voisine *s*' de la solution courante *s* est générée suivant une distribution uniforme aléatoire, la probabilité de choisir la solution *s*', *g*(*s*') est :

$g(s') = 1/n$ où s' $N(s)$, n : nombre de voisinage de s.

Une valeur appelée "énergie" est associée à chaque état (solution). Cette énergie correspond à l'évaluation de la solution. Dans notre application, c'est le makespan qui est retenu comme énergie de l'état. La différence d'énergie occasionnée par une transition est donnée par :

$$E = E_{s'} \quad E_s = C_{max}(s') \quad C_{max}(s).$$

c. L'acceptation de voisinage

Pour passer d'une solution courante *s* à une solution voisine *s*', deux situations sont rencontrées :

- Soit le mouvement améliore la qualité de la solution courante, *i.e.*, $C_{max}(s')$ $C_{max}(s)$. L'acceptation de passage est alors triviale.
- Soit le mouvement détériore la qualité de la solution courante, la probabilité *p* d'accepter un tel mouvement dépend :
  - D'une part, de l'importance de la dégradation : $C_{max}(s')$ $C_{max}(s)$, les dégradations les plus faibles sont plus facilement acceptées,
  - D'autre part, de la température courante T_k : une température élevée correspond à une probabilité plus grande d'accepter les dégradations.

  Cette probabilité d'acceptation est définie par : $p = e^{E / kT}$

Implémentation :

Voire le code source.