REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MOHAMED BOUDIAF - M'SILA

FACULTE MATHEMATIQUES ET DE L'INFORMATIQUE

DEPARTEMENT D'INFORMATIQUE

N° :.....

DOMAINE : Mathématiques et Informatique

FILIERE : Informatique

OPTION : Réseaux

Mémoire présenté pour l'obtention
Du diplôme de Master Académique

Par: BELLACHE Nour El Islam

Intitulé

Développement et implémentation d'un solveur bio inspiré pour la résolution d'un problème d'ordonnancement d'atelier

Soutenu devant le jury composé de :

MOUHOUB Nacer Eddine	Université de M'sila	Président
Dr.LAMICHE Chaabane	Université de M'sila	Rapporteur
HEMMAK Allaoua	Université de M'sila	Examinateur

Année universitaire : 2016 /2017

Remerciement

Je tien à saisir cette occasion et adresser mes profonds remerciements et mes profondes reconnaissances à :

- Mon encadrant de mémoire de fin d'étude Dr **LAMICHE Chaabane**, pour ses précieux conseils et son orientation ficelée tout au long de ma recherche.

- A ma famille et mes amis en particulier : **Boudour Ouissem Eddine Amir**, **Laredj Lazher ,Ouenoughi Nedjm Eddine**,**Oussama Mousaoui** et **Benghedfa zoubir**, qui par leurs prières et leurs encouragements, on a pu surmonter tous les obstacles.

- Je voudrais remercier aussi toutes les personnes qui ont participé de près ou de loin à mes recherches et à l'élaboration de ce mémoire.

À tous ces intervenants, je présente mes remerciements, mon respect et ma gratitude.

TABLE DES MATIERES

CHAPITRE 3

APPLICATION DE LA METHODE PSO AU PROBLEME SMTWTP

CHAPITRE 4

REALISATION ET EXPERIMENTATIONS

TABLE DES FIGURES

INTRODUCTION GENERALE

Dans le contexte actuel d'évolution des entreprises, il faut disposer des méthodes et d'outils de plus en plus performants pour l'organisation et la conduite de la production. L'organisation repose en général sur la mise en œuvre d'un certain nombre de fonctions, Parmi ces fonctions, l'ordonnancement qui joue un rôle essentiel.

L'objectif de cette fonction d'ordonnancement est d'attendre un ordre des taches pour satisfaire certains objectifs économiques, parmi lesquels on peut citer:

- La réduction des retards.
- La réduction des avances.
- La réduction des délais de fabrication.
- La réduction des coûts de fabrication, etc.

Et pour cela il y a beaucoup de méthodes lesquelles on peut les utiliser pour répondre à ces objectifs économiques.

Dans ce travail on va utiliser une méthode qui est la méthode PSO (Particul Swarm Optimisation) et l'appliquer sur le problème SMWTP (Single Machine Total Weighted Tardeness Problem).L'objectif de cette étude est de proposer une méthode approchée efficace capable a résoudre les problèmes d'ordonnancements a une seule machine avec l'optimisation de critére.

Chapitre 1 **:** est consacré a la définition du problème d'ordonnancement et ses différents éléments(les taches, les opérations, les gammes et les ressources), ensuit on parle de la représentation des résultats(le diagramme de GANT), et la classification des problèmes d'ordonnancement (champ α, champ β et champ γ), enfin on cite les classes des problèmes P et NP difficile.

Chapitre 2 **:** Dans ce chapitre j'ai présenté les différents méthodes de résolution, Les méthodes exactes (programmation dynamique, méthode de séparation et élévation), Les méthodes heuristiques constructives, et les méthodes amélioratrices (méthode de descente, recuit simulé, la recherche Tabou avec leurs algorithme) et enfin notre intension c'est centrés sur l'optimisation par particule d'essaim.

Chapitre 3 : Est réservé a la description du problème considéré et l'adaptation de l'algorithme d'optimisation par particule d'essaim avec le problème considéré.

Chapitre 4 **:** dans ce chapitre je parle d'environnement matériel et logiciel, je parle aussi des données utilisée pour réaliser notre simulation, le critère d'évaluation, aussi présenter les résultats obtenus après l'expérimentation et enfin le fonctionnement du logiciel développé.

CHAPITRE I

CHAPITRE 1

LES PROBLEMES D'ORDONNANCEMENT D'ATELIERS

1. Introduction

Les problèmes d'ordonnancement font partie de notre vie quotidienne. Actuellement, l'ordonnancement apparait dans tous les domaines de l'économie, l'informatique, la construction, l'industrie et l'administration.

Dans ce chapitre on va présenter les notions de base relatives aux problèmes d'ordonnancement. Il rappelle aussi quelques concepts sur la théorie de la complexité, et donne un aperçu des approches classiques pour la résolution des problèmes d'ordonnancement, leurs composants et leurs classifications.

2. Définition d'un problème d'ordonnancement

Tout problème d'ordonnancement se définit donc par un ensemble de tâches, encore appelées activités, à exécuter sur un ensemble de ressources. Il s'agit d'affecter, à chaque activité, une date d'exécution et un sous-ensemble de ressources de façon à satisfaire à la fois des contraintes temporelles et des contraintes d'accès aux ressources. Enfin, la programmation des activités doit généralement répondre à un critère d'optimisation [1].

3. Les éléments d'un problème d'ordonnancement

Un problème d'ordonnancement est composé d'un ensemble défini d'éléments, on trouve ces composants dans tous les problèmes d'ordonnancement :

3.1 Les Tâches

Une tâche est un ensemble d'opérations mobilisant des ressources et réalisant un progrès significatif de l'état d'avancement du projet compte tenu du niveau de détail retenu dans l'analyse du problème [20]. Une tâche est une entité élémentaire de travail localisée dans le temps par une date de début ti ou de fin ci, dont la réalisation est caractérisée par une durée de traitement $p_i = c_i - t_i$, et par l'intensité a_{ik} avec laquelle elle consomme certains moyens k, ou ressources [12].

Généralement, les notations utilisées pour caractériser une tâche, qu'on note 'i', sont les suivantes :

• Une date de disponibilité ri : l'exécution de la tâche i ne peut pas débuter avant cette date.

• Une date échue notée di : la tâche i doit être achevée avant cette date.

• La durée opératoire –de traitement- notée pi.

On note [ri, di], l'intervalle temporel dans lequel la tâche i devrait s'exécuter [9].

3.2 Les opérations

En général, la plupart des tâches donnent lieu à l'utilisation de différentes ressources et se décomposent donc en plusieurs opérations. Dans le cas d'un problème d'ordonnancement sur une machine unique ou atelier unique, on parle de tâches mono-opération. L'opération est caractérisée par son temps d'exécution, appelé aussi durée opératoire. Parfois, une opération peut également être caractérisée par son temps de préparation sur la machine, ou encore les temps de montage et de démontage de la pièce (matérialisant l'opération ou l'outillage), les temps de transport et d'attente. La séquence des opérations est décrite dans ce qu'on appelle la gamme d'une tâche [7].

3.3 Les ressources

Une ressource *k* est un moyen technique et/ou humain, requis pour une réalisation d'une tâche et disponible en quantité limitée, sa capacité a_{ik} (*supposée constante*). Plusieurs types de ressources sont à distinguer :

- *Ressources renouvelables* qui après avoir être utilisées par une ou plusieurs tâches, sont à nouveau disponibles en même quantités (*les hommes, les machines, l'espace, l'équipement en général, …etc*), les quantités des ressources utilisables à chaque instant sont limitées.
- Ressource consommable qui devient non disponible après avoir été utilisée par une ou plusieurs tâches, (*matière première, budget,…etc*) ; la consommation globale (*ou cumul*) au cours du temps est limitée.
- Ressource doublement contraintes lorsque son utilisation instantanée et sa consommation globale sont toutes les deux limitées (source d'énergie, financement,… etc.).
- *Ressources disjonctives (ou non partageables)* principalement dans le cas de ressources renouvelables, sont des ressources qui ne peuvent exécuter qu'une tâche à la fois (machine-outil, robot manipulateur, … etc).
- Ressources cumulatives (ou partageables) qui peuvent être utilisées par plusieurs tâches simultanément (équipe d'ouvriers, poste de travail).

La nature de ressources prises en considération permet de dresser une typologie des problèmes d'ordonnancement(Figure.1.1).

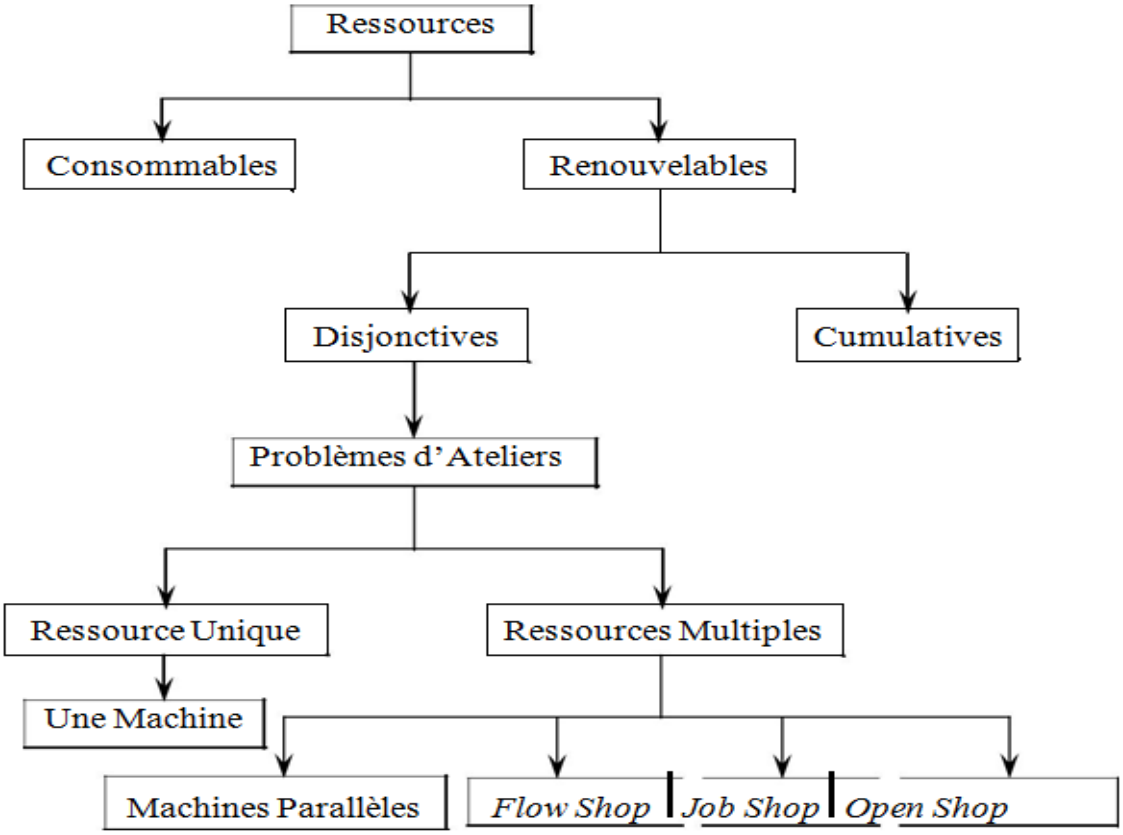


Figure 1-1 Typologie par les ressources des problèmes d'Ordonnancement

3.4 Les gammes

La gamme est un document décrivant en détail la séquence d'opérations de fabrication, d'assemblage, d'inspection ou de transport d'un composant ou d'un produit fini [20].

Les gammes de fabrication décrivent la succession des opérations à réaliser pour passer d'une matière au composant ou du produit semi-fini au produit fini ; c'est le processus d'élaboration. On trouve dans la littérature non spécialisée, l'appellation de routage de la tâche et dans la littérature anglophone le terme de *routing*, il existe plusieurs sortes de gammes parmi les quelles on distingue :

- Les gammes techniques dues essentiellement au choix prédéterminé de certaines tâches à passer sur des machines spécifiques.
- Les gammes libres qui sont l'une des principales caractéristiques des problèmes d'ordonnancement de type Open Shop. En effet, quand l'exécution des opérations est indépendante de l'ordre, il n'existe aucune contrainte de succession.

- Les gammes linéaires où l'ordre d'exécution des opérations entièrement imposé et prédéterminé. Ce genre de gammes est présent dans le cas du Job Shop (chaque tâche a sa propre route à suivre autrement dit toutes les tâches ont des gammes prédéterminées mais non identiques) et celui de Flow Shop (toutes les tâches ont le même chemin (route ou gamme) à suivre).
- Les gammes mixtes ou semi-linéaires où l'ordre d'exécution des opérations est partiellement déterminé, on trouve des tâches qui ont des gammes prédéterminées (on sait à priori la route à suivre par ces tâches), et d'autres non (on ne sait pas, à priori, la route à suivre par ces tâches). [7]

4. Représentation des résultats

Le graphique de Gantt, encore appelé diagramme de Gantt, est une technique de visualisation de l'utilisation de moyens productifs et/ou de l'avancement de l'exécution de tâche popularisée par Gantt en 1917, (mais dont retrouve des utilisations chez les prêtes égyptiens de l'antiquité) et est classiquement utilisé en ordonnancement en atelier.

Une tâche y est représentée sur un axe, habituellement horizontal, par un segment dont la longueur est, en principe, proportionnelle au temps d'exécution. Lorsque l'on étudie l'évolution de L'utilisation de plusieurs facteurs productifs (par exemple des machines), l'utilisation de chaque facteur productif est portée sur un axe différent , on a alors un faisceau de droites parallèles sur un même document (tableau mural par exemple).La même échelle temporelle est utilisée pour tous les axes, ce qui fait que les intersections de ces droites avec une même perpendiculaire repèrent le même instant. Pour faciliter ce repérage, un papier quadrillé est utilisé et l'échelle des temps est explicitée en haut du document. Un curseur vertical permet de visualiser un instant précis du temps, ce qui facilite l'utilisation de ce document pour le suivi d'un atelier.

Au dessus de chaque segment on porte le code d'identification de la tâche (ou ordre de fabrication), ou des quantités s'il s'agit de la production d'un article connu sans ambiguïté.

Ce document est utilisé pour le lancement des travaux en atelier. Conventionnellement une tâche est programmée comme le montre la Figure.1.2 où la longueur du segment est proportionnelle à la durée d'exécution et *x* est le numéro de la tâche, ou la quantité à produire. Notons que le diagramme de Gantt est le meilleur moyen pour visualiser la solution d'ordonnancement et calculer le temps de chaque tâche ainsi que les différents temps d'achèvement de ces tâches 'Makespa*Cmax*'.[2]

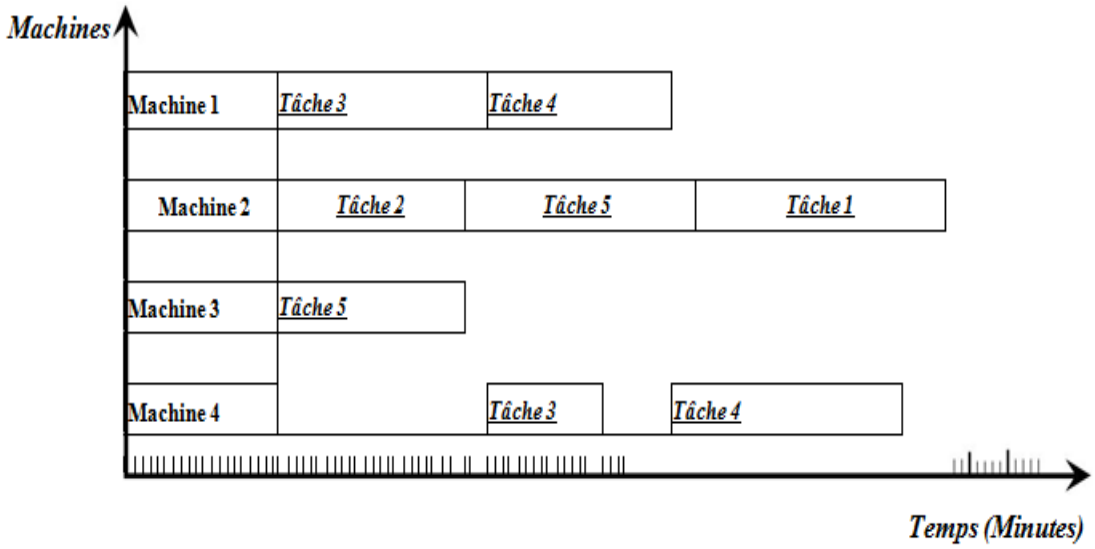


Figure 1-2 Exemple de visualisation d'un ordonnancement sur un Diagramme de Gantt

5. Classification des problèmes d'ordonnancement

Etant donné la diversité des problèmes d'ordonnancement, nous utilisons couramment un formalisme de classification, permettant de distinguer les problèmes d'ordonnancement entre eux et de les classer. Ce formalisme, comporte trois champs : α ,β ,γ **,** permettant de décrire les différentes entités d'un problème d'ordonnancement.

5.1 Champ α : Organisation des ressources

Le champ α = α1 α2 décrit l'environnement de la machine.

Le paramètre α1 ∈ { 1 ou ∅, P, Q, R, O, F, J, FH } caractérise le type de machines utilisées :

- **α1** = 1, auquel cas nous avons un problème à une seule machine. α1 = P : machines identiques parallèles.
- **α1** = Q : machines parallèles uniformes. α1 = R : machines parallèles quelconques (unrelated).
- **α1** = O : il s'agit d'un open shop.
- **α1** = F : machines dédiées : système Flow shop.
- **α1** = J : machines dédiées : système Job shop.
- **α1** = FH : flow shop hybride α2 ∈ {∅, k } est un entier qui représente le nombre de machines dans le problème.
- **α2** = ∅ : le nombre de machines est supposé être variable.
- **α2** = k : le nombre de machines est égal à k (k entier positif). [2]

5.2 Champ β : contraintes et caractéristiques du système

Le second champ β = β1 β2 β3 β4 β5 β6 β7 β8 décrit la tâche et les caractéristiques de la ressource.

- Le paramètre β1 ∈ {∅, pmtn } indique la possibilité de la préemption de la tâche.
  - β1 = ∅ : la préemption n'est pas autorisée.
  - β1 = pmtn : la préemption est autorisée.
- Le paramètre β2 ∈ {∅, res } caractérise les ressources additionnelles.
  - β2 = ∅ : aucune ressource additionnelle n'existe.
  - β2 = res : il y a des contraintes de ressources spécifiées.

- Le paramètre β3 ∈ {∅, prec, uan, tree, chains } reflète les contraintes de précédence.
  - β3 = ∅, prec, uan, tree, chains : respectivement tâches indépendantes, contraintes de précédence générales, réseau d'activité uni connexe, contraintes de précédence formant un arbre ou une chaîne.
- β4 ∈ {∅, rj } : les temps au plus tôt sont zéro, soit les temps au plus tôt différent par tâche.
- β5 ∈ {∅, pj = p, p ≤ pi ≤ p--} : les tâches ont des temps d'exécution arbitraires, le temps d'exécution dans l'intervalle défini.
- β6 ∈ {∅, ~ d } décrit les deadlines: aucune deadline n'est supposée dans le système (date au plus tard peuvent être définie si le critère utilisé pour évaluer l'ordonnancement est celui de date au plus tard), deadlines sont imposées sur la performance de l'ensemble des tâches.
- β7 ∈ {∅, nj ≤ k} : décrit le nombre maximum de tâches constituant un job dans le cas du système job shop: le nombre est arbitraire ou le problème d'ordonnancement n'est pas du job shop; le nombre de tâches pour chaque job n'est pas plus grand que k.
- β8 ∈ {∅, no-wait } décrit une propriété sans attente de l'ordonnancement sur des machines dédiées : la région tampon est de capacité illimitée; sans attente les capacités d'attente sont nulles. Une tâche qui termine son exécution sur une machine passe instantanément sur une autre machine. [2]

5.3 Champ γ : critères d'optimisation

Le troisième champ, le champ γ, concerne les critères d'évaluation d'un ordonnancement. Les objectifs visés sont liés à une bonne utilisation des ressources, une minimisation du délai

global ou encore le respect d'un maximum de contraintes. Nous donnons ici les critères les plus couramment utilisés :

- $\gamma = C_{Max}$: *Makespan*. Ce critère vise à minimiser la date de sortie du système de la dernière tâche ($\gamma = C_{Max} = Max_j\, C_j$,$C_j$ représente la date fin d'exécution de la tâche *j*)
- $\gamma = w_j\, C_j$: Somme pondérée des dates de fin d'exécution. Ce critère représente la somme des encours des tâches dans le système.
- $\gamma = L_{Max}$: Décalage temporel maximal. Ce critère mesure la plus grande violation des dates d'échéances souhaitées ($L_{Max} = Max_j\, L_j = Max_j\, (C_j - d_j)$).
- $\gamma = \sum w_j T_j$: Somme pondérée des retards (($T_j = Max_j\, (C_j - d_j, 0)$), qui permet d'évaluer les pénalités dues aux retards (w_j : poids de la tache *j*).
- $\gamma = \sum w_j U_j$: Somme pondérée du nombre de tâches en retard (U_j =1 si la tâche *j* est en retard, 0 sinon). [2]

6. Complexité

L'expérience montre que certains problèmes sont plus faciles que d'autres à résoudre. Une théorie de la complexité a été développée et permet mathématiquement de classer les problèmes faciles et difficiles en deux classes : les classes P et NP [12]. Nous exposerons dans la partie qui suit, les grands principes de la théorie de la complexité des problèmes d'ordonnancement.

6.1 Les classes P et NP

Pour pouvoir exposer la notion de classe de problèmes, il est tout d'abord nécessaire de distinguer les problèmes de décision des problèmes d'optimisation. Un problème de décision est un problème pour lequel la réponse est "oui" ou "non". Il est possible d'associer à chaque problème d'optimisation, un problème de décision en introduisant un seuil *k* correspondant à la fonction objectif *f*. Le problème de décision devient : "existe-t-il un ordonnancement réalisable *(S)* tel que $f(S) \leq k$?".

Il est alors possible de définir la classe *P* qui regroupe les problèmes de décision résolubles par des algorithmes polynomiaux [19]. Un algorithme polynomial est défini comme un algorithme dont le temps d'exécution est borné par *O(p(x))* ou *p* est un polynôme et *x* la longueur d'entrée d'une instance du problème. Les algorithmes dont la complexité ne peut pas être bornée polynomialement sont qualifiés d'exponentiels.

La classe *NP* regroupe les problèmes qui peuvent être résolus en temps polynomial par un algorithme non déterministe (Algorithme qui comporte des instructions de choix) [19]. Pour

ces algorithmes, si à chaque instruction le bon choix est effectué, le temps de calcul est polynomial. Si au contraire tous les choix sont énumérés, l'algorithme devient déterministe et son temps de calcul devient exponentiel.

Les algorithmes "ordinaires" sont évidemment des cas particuliers des algorithmes non déterministes [4]. Aussi tout problème de décision qui peut être résolu par un algorithme polynomial, et qui donc appartient à la classe *P*, appartient également à la classe *NP*. D'ou $P \subseteq NP$.

Parmi les problèmes de la classe *NP*, une large classe de problèmes, les problèmes *NP*-complets, sont équivalents entre eux quant à l'existence d'un algorithme polynomial pour les résoudre. C'est à dire, s'il existe un algorithme polynomial pour résoudre un seul de ces problèmes, alors il en existe un pour chaque problème de la classe *NP*. Afin de décrire cette classe d'équivalence, définissons tout d'abord la réduction polynomiale entre deux problèmes. Soient *P1* et *P2*, deux problèmes de décision. On dit que *P1* se réduit polynomialement à *P2* (noté $P1 \propto P2$) s'il existe un algorithme de résolution de *P1*, qui fait appel à un algorithme de résolution de *P2*, et qui est polynomial lorsque la résolution de *P2* est comptabilisée comme une opération élémentaire. On dit alors qu'un problème de décision est *NP*-complet si tout problème de la classe *NP* se réduit polynomialement à lui [19]. Les problèmes d'optimisation combinatoire dont le problème de décision associé est *NP*-complet sont qualifiés de *NP*-difficiles.

7. Conclusion

Les problèmes d'ordonnancements se trouvent dans tous les systèmes de production qui nécessite une organisation de leurs taches pour trouver un ordonnancement optimale. On a vu dans ce chapitre les éléments fondamentaux des formes d'organisation des productions et les méthodes de présentation des résultats avec le diagramme de Gantt . La suite de notre travail sera consacrée à présenter les différentes approches existantes pour la résolution de ces problèmes d'ordonnancement.

CHAPITRE II

CHAPITRE 2
METHODES DE RESOLUTION D'UN PROBLEME D'ORDONNANCEMENT

1 Introduction

Résoudre un problème d'ordonnancement consiste a trouver une planification des taches sur les ressources en optimisant les objectifs et en respectons les contraints. Pour les problèmes NP-difficiles, on ne connaît pas d'algorithmes polynomiaux pour les résoudre, et la conjecture P ≠ NP fait qu'il est peu probable qu'il en existe. En pratique, on résout un tel problème en tirant parti de :

- La taille des données. L'énumération complète peut être valable sur des instances de petite taille.
- La complexité moyenne : un algorithme exponentiel sur son pire cas, peut être assez rapide en moyenne, comme l'algorithme du simplexe.
- La nature des données : le problème peut devenir facile sur des cas particuliers.

Méthodes diminuant la combinatoire : méthodes par séparation et évaluation, Programmation dynamique, etc.

Méthodes approchées ou heuristiques. On accepte des solutions de bonne qualité sans garantie d'optimalité.

Dans le reste de ce chapitre, on présente les méthodes de résolution exacte et approchée.

2 Les méthodes exactes

Les trois familles de méthodes exactes sont : la programmation dynamique, la méthode de séparation et évaluation (Branch and Bound) et la résolution à partir d'une modélisation analytique. Ces méthodes se caractérisent par un temps de calcul exponentiel, ce qui explique qu'elles ne sont utilisables que sur des problèmes de petite taille.

Nous présentons d'abord quelques méthodes de la classe des algorithmes complets ou exacts, ces méthodes donnent une garantie de trouver la solution optimale pour une instance de taille finie dans un temps limité et de prouver son optimalité, Les techniques de résolution exacte sont définies pour les problèmes combinatoires en général. L'énumération exhaustive est la méthode la plus simple. Les méthodes par séparation et évaluation et la programmation dynamique font une énumération intelligente des solutions possibles.

Pour des problèmes difficiles de grande taille, la durée d'exécution est démesurée et il faut envisager une résolution approchée. Ils sont exposés dans la suite.

2.1 La programmation dynamique

Introduite par Bellman dans les années 50, la programmation dynamique décompose un problème de dimension n en n problèmes de dimension.

Le système est alors constitué de n étapes que l'on résout séquentiellement, le passage d'une étape à une autre se faisant à partir des lois d'évolution du système et d'une décision.

Le principe d'optimalité de Bellman est basé sur l'existence d'une équation récursive permettant de décrire la valeur optimale du critère à une étape en fonction de sa valeur à l'étape précédente. Ainsi, pour appliquer la programmation dynamique à un problème combinatoire, le calcul du critère pour un sous-ensemble de taille k nécessite la connaissance de ce critère pour chaque sous-ensemble de taille k -1 et porte le nombre de sous-ensembles considérés à 2_n (où n est le nombre d'éléments considérés dans le problème). [4]

La programmation dynamique est donc de complexité exponentielle. Pourtant, pour les problèmes *NP*-difficiles au sens faible, c'est-à-dire pour lesquels il existe des algorithmes de résolution pseudo-polynomiaux, il est souvent possible de construire un algorithme de programmation dynamique pseudopolynomial, pouvant alors être utilisé pour des problèmes de tailles raisonnables.

2.1.1 Méthode de séparation et évaluation (Branch & Bound)

L'algorithme de séparation et évaluation, plus connu sous son appellation anglaise Branch and Bound (B&B) (Land et Doig 1960), repose sur une méthode arborescente de recherche d'une solution optimale par séparations et évaluations, en représentant les états solutions par un arbre d'états, avec des noeuds, et des feuilles.

Le branch-and-bound est basé sur trois axes principaux :

- L'évaluation.
- La séparation.
- La stratégie de parcours.

L'évaluation :

L'évaluation permet de réduire l'espace de recherche en éliminant quelques sous ensembles qui ne contiennent pas la solution optimale.

L'objectif est d'essayer d'évaluer l'intérêt de l'exploration d'un sous-ensemble de l'arborescence. Le branch-and-bound utilise une élimination de branches dans l'arborescence de recherche de la manière suivante : la recherche d'une solution de coût minimal, consiste à mémoriser la solution de plus bas coût rencontré pendant l'exploration, et à comparer le coût

de chaque nœud parcouru à celui de la meilleure solution. Si le coût du nœud considéré est supérieur au meilleur coût, on arrête l'exploration de la branche et toutes les solutions de cette branche seront nécessairement de coût plus élevé que la meilleure solution déjà trouvée.

2.1.2 La séparation

La séparation consiste à diviser le problème en sous-problèmes. Ainsi, en résolvant tous les sous-problèmes et en gardant la meilleure solution trouvée, on est assuré d'avoir résolu le problème initial. Cela revient à construire un arbre permettant d'énumérer toutes les solutions. L'ensemble des neouds de l'arbre qu'il reste encore à parcourir comme étant susceptibles de contenir une solution optimale, c'est-à-dire encore à diviser, est appelé ensemble des nœuds actifs.

2.1.3 La stratégie de parcours

- La largeur d'abord :

Cette stratégie favorise les sommets les plus proches de la racine en faisant moins de séparations du problème initial. Elle est moins efficace que les deux autres stratégies présentées.

- La profondeur d'abord :

Cette stratégie avantage les sommets les plus éloignés de la racine (de profondeur la plus élevée) en appliquant plus de séparations au problème initial. Cette voie mène rapidement à une solution optimale en économisant la mémoire.

- Le meilleur d'abord

Cette stratégie consiste à explorer des sous problèmes possédant la meilleure borne. Elle permet aussi d'éviter l'exploration de tous les sous-problèmes qui possèdent une mauvaise évaluation par rapport à la valeur optimale. [6]

3 Les méthodes approchées

Le but d'une méthode approchée est de trouver une solution réalisable, tenant compte de la fonction objectif mais sans garantie d'optimalité. Son utilisation offre de multiples avantages par rapport à une méthode exacte, citons :

- Elles sont plus simples et plus rapides à mettre en œuvre quand la qualité de la solution n'est pas trop importante.
- Elles sont plus souples dans la résolution des problèmes réels. En pratique, il arrive souvent que l'on doive prendre en considération de nouvelles contraintes qu'on ne pouvait pas formuler dès le départ. Ceci peut être fatal pour une méthode exacte si

les nouvelles contraintes changent les propriétés sur lesquelles s'appuyait la méthode.

- Elles fournissent des solutions et des bornes qui peuvent être utiles dans la conception de méthodes exactes.

Pour les problèmes d'ordonnancement, si on cherche une permutation optimale des tâches, l'espace des solutions admissibles comporte n! Permutations. Un exemple de voisinage d'une solution est défini par toutes les permutations obtenues en échangeant deux tâches consécutives de la permutation.

Les méthodes approchées diffèrent aussi, les unes des autres, par le type de compromis qualité / complexité qu'elles offrent. Nous en distinguons deux classes : les heuristiques et les métaheuristiques.

3.1 Les méthodes Heuristiques

En optimisation combinatoire, une heuristique est un algorithme approché qui permet d'identifier en temps polynomial au moins une solution réalisable rapide, pas obligatoirement optimale. L'usage d'une heuristique est efficace pour calculer une solution approchée d'un problème et ainsi accélérer le processus de résolution exacte. Généralement une heuristique est conçue pour un problème particulier, en s'appuyant sur sa structure propre sans offrir aucune garantit quant à la qualité de la solution calculée.

Les heuristiques peuvent être classées en deux catégories :

– Méthodes constructives qui génèrent des solutions à partir d'une solution initiale en essayant d'en ajouter petit à petit des éléments jusqu'à ce qu'une solution complète soit obtenue.

– Méthodes de fouilles locales qui démarrent avec une solution initialement complète (probablement moins intéressante), et de manière répétitive essaie d'améliorer cette solution en explorant son voisinage.

3.2 Les méthodes Métaheuristiques

3.2.1 La méthode de descente

C'est l'une des heuristiques de recherche locale les plus simples. Elle consiste à rechercher dans le voisinage de la solution courante, une solution de coût plus faible.

Elle procède ainsi jusqu'à arriver à un optimum local. Cette méthode a l'avantage d'être rapide, mais s'arrête dès qu'un optimum local est atteint, même si celui-ci n'est pas de bonne qualité.

Parmi ces méthodes, nous décrivons ici les méthodes d'échanges de type *r-opt*. Ces méthodes d'optimisation ont été initialement proposées pour résoudre le problème du voyageur de commerce (PVC), mais elles s'appliquent également à tout problème combinatoire dont la solution consiste en une permutation de *r* composantes parmi *n*.

Le terme *r-opt* indique qu'une solution ne peut plus être améliorée en échangeant *r* éléments. La méthode consiste donc à sélectionner *r* composantes et à regarder si en interchangeant ces composantes, on obtient une meilleure solution.

Nous remarquons donc qu'une solution *n-optimale* est une solution optimale (dans le cas d'un problème de taille n). Ainsi, plus *r* augmente, plus on se rapproche de la solution optimale, mais plus les calculs sont difficiles. En effet, il y a (r_n) façons de choisir *r* composantes et r! façons de les échanger. En pratique, on se limite à *r = 2 ou 3.* [8]

Pour un problème de minimisation d'une fonction f, la méthode descente peut être décrire comme suit :

Algorithme : Méthode de descente

1 : Solution initial S ;

2 : **Répéter :**

3 : Choisir S() dans un voisinage N(S) de S ;

4: Si $f(S') < f(S)$ alors $S := S'$;

5: jusqu'à ce que $f(S') \geq f(S)$, $\forall S' \in N(S)$.

3.2.2 Recherche de tabou

La recherche tabou (TS) est une méthode de recherche locale combinée avec un ensemble de techniques permettant d'éviter d'être piégé dans un minimum local ou la répétition d'un cycle. La recherche tabou est introduite principalement par Glover (Glover 1986), Hansen (Hansen 1986), Glover et Laguna dans (Glover et Laguna 1997).

Cette méthode a montré une grande efficacité pour la résolution des problèmes d'optimisation difficiles. En effet, à partir d'une solution initiale s dans un ensemble de solutions local S, des sous-ensembles de solution N(s) appartenant au voisinage S sont générés. Par l'intermédiaire de la fonction d'évaluation nous retenons la solution qui améliore la valeur de f, choisie parmi l'ensemble de solutions voisines N(s).

L'algorithme accepte parfois des solutions qui n'améliorent pas toujours la solution courante. Nous mettons en œuvre une liste tabou (tabu list) T de longueur k contenant les k

dernières solutions visitées, ce qui ne donne pas la possibilité a une solution déjà trouvée d'être acceptée et stockée dans la liste tabou. Alors le choix de la prochaine solution est effectué sur un ensemble des solutions voisines en dehors des éléments de cette liste tabou.

Quand le nombre k est atteint, chaque nouvelle solution sélectionnée remplace la plus ancienne dans la liste. La construction de la liste tabou est basée sur le principe FIFO, c'est-à-dire le premier entré est le premier sortie. Comme critère d'arrêt on peut par exemple fixer un nombre maximum d'itérations sans amélioration de s*, ou bien fixer un temps limite après lequel la recherche doit s'arrêter. [6]

Algorithme : La recherche de Tabou

---

1 : Initialisation :

S_0 Une solution initiale

$S \leftarrow S_0 , S^* \leftarrow S_0 , C^* \leftarrow f(S_0)$

$T = \emptyset$

2 Générer un sous-ensemble de solution au voisinage de S

$S' \in N(S) Tel\ que \forall\ x\ \in N(S), f(x) \geq f(S') et\ S' \notin T$

$Si\ f(S') < C * alors\ S * \leftarrow S' etc * \leftarrow f(S')$

Mise-a-jour de T

3 : Si la condition d'arrêt n'est pas satisfaite retour a l'étape 2

- Avantage :

L'efficacité de la méthode tabou offre son utilisation dans plusieurs problémes d'optimisation combinatoire classique tels que le problème de voyageur de commerce, le problème d'ordonnancement, le problème tournées de véhicule, etc.

- Inconvénient :

L'inconvénient de cette dernière démarche est que parfois, une itération ne suffit pas pour s'en sortir d'un minimum local et un déplacement peut provoquer un déplacement inverse à une étape ultérieure, ce qui provoquera un cycle autour du minimum local. C'est pour cette raison que RT garde en mémoire les dernières solutions visitées pour éviter d'y revenir, c'est ce qu'on appelle *liste tabou* [3].

3.2.3 Le recuit simulé

Le recuit simulé est une métaheuristique s'inspirant de la métallurgie créée au début des années 1980. C'est une méthode bien éprouvée. Elle a été présentée pour la première fois par S. Kirkpatrick,C. Gelatt et M. Vecchi dans.

Cette section est divisée en deux sous-sections. La première présente la méthode du recuit simulé. La seconde décrit nos deux adaptations de cette méthode au problème du partitionnement de graphe non contraint. [18]

Description de l'algorithme du recuit simulé :

La métaheuristique du recuit simulé s'inspire de la technique expérimentale du recuit utilisée en métallurgie. Celle-ci permet d'obtenir un état stable, ou d'énergie minimale, du métal.

Cet état est obtenu quand le matériau a trouvé une structure cristalline. Alors que la technique bien connue de la trempe fige le matériau dans un état méta stable, la technique du recuit permet d'éviter au matériau d'être dans un état méta stable caractérisant un minimum local d'énergie.

Pour éviter au matériau de rester bloqué dans un état méta stable, la technique du recuit consiste à porter le matériau à haute température, puis à abaisser lentement celle-ci. En informatique, la méthode du recuit simulé transpose cette technique à la résolution d'un problème d'optimisation. Ainsi, la température, paramètre essentiel du recuit physique, est directement utilisée dans la méthode du recuit simulé comme paramètre de contrôle. De même, par analogie avec le processus physique, l'énergie du système devient la fonction de coût à minimiser. Pour simuler l'évolution thermodynamique du système, le recuit simulé utilise l'algorithme de Metropolis [16].

Ce dernier s'appuie sur la distribution de Boltzmann utilisée en physique pour déterminer la distribution des particules en fonction de leur niveau d'énergie. L'algorithme du recuit simulé est décrit dans de nombreux ouvrages [17],[3].

Son principe est simple : partant d'un état initial quelconque, aussi appelé configuration initiale, un nouvel état est créé à partir de l'état précédent par une modification élémentaire de cet état. Il est accepté si son énergie est plus faible, sinon il est accepté avec une certaine probabilité. Ce schéma est réitéré en remplaçant l'état initial par l'état accepté tant qu'un »équilibre thermodynamique« n'est pas atteint.

Lorsque cet équilibre est atteint, la température est diminuée. Puis les équilibres thermodynamiques sont renouvelés tant que le système n'est pas figé. Il existe différentes variantes de cette méthode [14].

Cette méthode permet donc d'accepter un état dégradé par rapport à l'état précédent. Cette particularité, commune à toutes les métaheuristiques, permet d'explorer une plus grande partie de l'espace des solutions et d'éviter les minima locaux.

Algorithme : Recuit simulé

1: Engendrer une configuration initiale S_0 de S : S → S_0

2: Initialiser la temperature T en fonction du schéma de refroidissement

3:Répéter

4 : Engendrer un voisin aléatoire S′ de S

5 :Calculer $\Delta E = f(S') - f(S)$

6 : Si $\Delta E \leq 0$ alors S ← S′

7: Sinon accepter S_0 comme la nouvelle solution avec la probabilité

$$P(E, T) = e^{-\frac{\Delta E}{T}}$$

8 : Fin si

9 : Mettre T a jour en fonction du schéma de refroidissement (réduire la température)

10 : Jusqu'à la condition d'arrêt

11 : Retourner la meilleure configuration trouver

Le choix de la température est primordial pour garantir l'équilibre entre l'intensification et la diversification des solutions dans l'espace de recherche.

Premièrement, le choix de la température initiale dépend de la qualité de la solution de départ. Si cette solution est choisit aléatoirement, il faut prendre une température relativement élevé.

On utilise souvent la règle suivante :

$$T_{k+1} \leftarrow T_k \cdot \alpha$$

Ou $\alpha \in [0,1]$, un paramètre qui exprime la dimension de la température de l'itération K à K+1 .

La décroissance de la température peut également étre réalisée par paliers(en d'autres termes, elle ne change qu'après un certain nombre d'itérations).

Certains préconisent l'utilisation de stratégies non monotones. On peut ainsi rehausser la température lorsque la recherche semble bloquée dans une région de l'éspace de recherche.

On peut alors considérer une grande augmentation de la température comme un processus de diversification alors que la décroissance de la température correspond a un processus d'intensification.

- Avantage :

- La méthode du recuit simulé a 'avantage d'être souple vis-à-vis des évolutions du problème et facile a implémenter.
- Contrairement aux méthodes de descente, SA évite le piége des optima locaux.
- Excellent résultats pour un nombre de problèmes complexes.

- Inconvénient :

- Nombreux tests nécessaires pour trouver les bons paramètres.
- Définir les voisinages permettant un calcul efficace de DE.

3.2.4 La méthode de PSO

L'optimisation par essaim de particules est une technique évolutionnaire qui utilise "une population" de solutions candidates pour développer une solution optimale au problème. Le degré d'optimalité est mesuré par une fonction fitness (aptitude) définie par l''utilisateur (Clerc et al. 2001, Dutot et al. 2002, Ji et al. 2007, Kennedy et al. 1995 et 2001). L'OEP diffère des autres méthodes de calcule évolutionnaire de façon que les membres de la population appelés ´´ particules´´, sont dispersées dans l'espace du problème (Kennedy et al. 1995 et 2001). Le comportement de l'essaim doit être décrit en se plaçant du point de vue d'une Particule (Kennedy et al. 2001, Omran 2004, Van den Bergh 2002, Venter et al. 2002). Au départ de l'algorithme, un essaim est réparti au hasard dans l'espace de recherche, chaque particule ayant également une vitesse aléatoire.

Principe du PSO :

Un essaime est disposé de façon aléatoire et homogène dans l'espace de recherche et chaque particule possède la capacité de se déplacer avec une vitesse aléatoire. Ainsi, à chaque pas de temps, chaque particule :

- Evalue la qualité de sa position et garde en mémoire sa meilleure performance, c'est-à-dire la meilleure position atteinte jusqu'ici (elle peut être la position courante) et sa qualité (la valeur de la fonction à optimiser en cette position).
- Interroge un certain nombre de particules pour obtenir de chacune d'entre elles sa propre meilleure performance.

- Choisit la meilleure des meilleures performances dont elle a connaissance, puis adapte sa vitesse en fonction de cette information et de ses propres données et se déplace en conséquence.

Le principe de la méthode d'essaime de particule est résumé par la figure (2.1). Pour réaliser son prochain mouvement, chaque particule combine trois tendances : suivre sa vitesse propre, revenir vers sa meilleure performance, aller vers la meilleure performance de ses informatrices.

Le principe de PSO développé par Kennedy et Eberhart se base sur le comportement des nuées d'oiseaux. Ainsi, le PSO a été fondamentalement développé à travers la simulation du comportement des nuées d'oiseaux dans l'espace bidimensionnel. La position de chaque agent (individu ou particule) est représentée par ses coordonnées suivant les deux axes XY auxquels on associe les vitesses exprimée par V_x (vitesse suivant l'axe X) et V_y (vitesse suivant l'axe Y). La modification du comportement de chaque agent se base sur les informations de position et de vitesse.

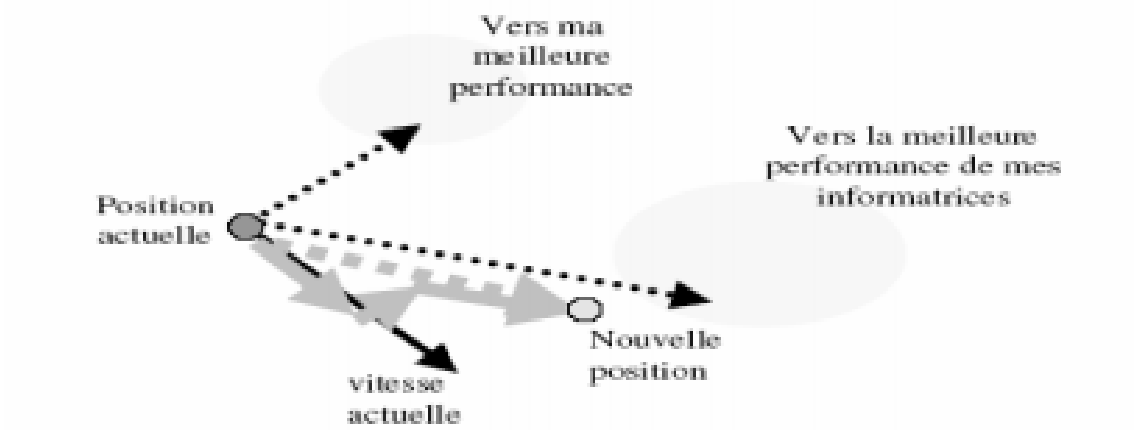


Figure 0-1 : Déplacement d'une particule

A chaque itération l'agent procède via une fonction objective à l'évaluation de sa meilleure valeur jusque là (P_{best}) et sa position suivant les deux axes XY. Cette information est obtenue à partir de l'analyse des expériences personnelles de chaque agent. En plus, chaque agent connaît la meilleure valeur globale du groupe (G_{best}) parmi les P_{bests}. Cette information représente la valeur autour de laquelle d'autres agents sont performants. Ainsi, chaque agent essaye de modifier sa position en se basant sur les informations suivantes :

- ✓ Position courante (x, y).
- ✓ Vitesse courante (V_x, V_y).
- ✓ Distance entre la position courante et P_{best}.
- ✓ Distance entre la position courante et G_{best}.

Cette modification peut être représentée par le concept de vitesse. La vitesse modifiée de chaque agent s'écrira de la manière suivante :

$$V_i^{K+1} = \omega V_i^K + C_1.rand1.\left(X_{Pbesst} - X_i^k\right) + C_2.rand2.\left(X_{Gbesst} - X_i^k\right) \qquad \textbf{(2.1)}$$

Ou :

- V^k: Vitesse de l'agent i à l'itération k.
- C_1 et C_2 : sont deux constantes d'accélérations régulant les vitesses relatives par rapport aux meilleurs postions locales et globales. Ces paramètres sont considérés comme des facteurs d'échelle utilisés pour déterminer les mouvements relatifs de la meilleure position de la particule ainsi que de la meilleure position globale. Ce sont des facteurs qui déterminent le degré d'influence des positions passées de la particule elle mêmes et celles des autres particules dans l'essaim.
- rand1 et rand2 : sont des variables aléatoires générés d'une distribution uniforme dans l'intervalle [0,1] afin de fournir un poids stochastique aux différentes composantes participant dans la définition de la vitesse de la particule.[14]
- X_i^k : Position courante de l'agent i à l'itération k qui réfère à une solution candidate pour le problème d'optimisation considéré à l'itération k.
- X_{pbesti} : La meilleure position locale de l'agent i.
- X_{gbesti} : la meilleure position globale identifiée dans le processus de recherche pour toutes les particules dans léessaim. La position optimal est mesurée avec une fonction dit fitness définit suivant le problème d'optimisation. Durant l'optimisation, les particules se déplacent suivant les équations (2.1) et (2.3).

- ω : Fonction de pondération, elle est utilisée comme un compromis entre l'exploration locale et globale du swarm. Des valeurs élevées de ce paramètre permettent une bonne exploration globale, tandis que pour des valeurs minimales une recherche fine est réalisée. La fonction de pondération habituellement utilisée dans l'équation (2.1) est la suivante :

$$\omega = \omega_{max} - \frac{\omega_{max} - \omega_{min}}{iter_{max}} \times iter \qquad \textbf{(2.2)}$$

Où:

- ω_{max}: Poids initial.
- ω_{min} : Poids final.
- $iter_{max}$: Nombre maximum d'itération.
- $iter$: Nombre courant d'itération

La partie de droite de l'équation (2.1) comprend trois termes (vecteurs). Le premier terme représente la vitesse précédente de l'agent. Le second terme ainsi que le troisième sont utilisés pour modifier l vitesse de l'agent.

Le modèle utilisé dans l'équation (2.2) est appelé approche poids inertie « Inertia Weights Approch (IWA) ». La position courante est modifiée suivant l'équation :

$$X_i^{k+1} = X_i^k + V_i^{K+1} \qquad \textbf{(2.3)}$$

La figure (2.2) montre le concept de modification d'un point de recherche par le PSO alors que la figure (2.3) illustre un concept de recherche avec des agents dans un espace de solution. Chaque agent change sa position courante en moyennant l'intégration des vecteurs comme présentés dans la figure (2.3).

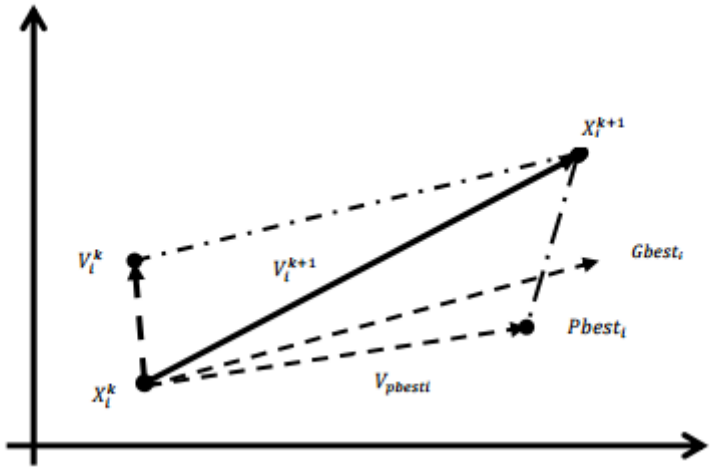


Figure 2-0-2 Concept de modification d'un point de recherche par PSO.

Où :

X_i^k : Position courante de l'agent.

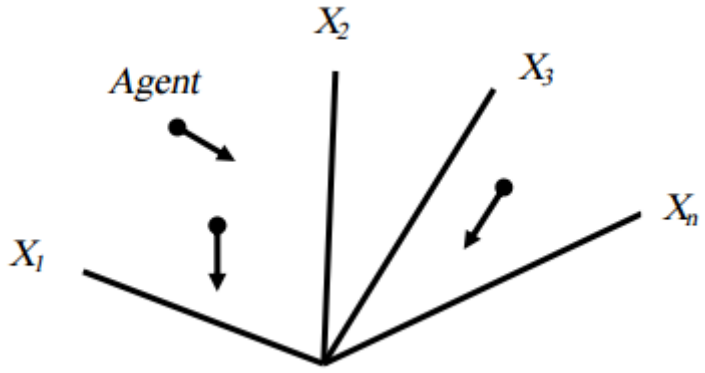
X_i^{k+1}: Position modifiée de l'agent.

V_i^K: Vitesse courante de l'agent.

V_i^{K+1}: Vitesse modifiée de l'agent.

V_{pbest} : Vitesse de l'agent basée sur la position P_{best}.

V_{gbest} : Vitesse de l'agent basée sur la position G_{best} choisi parmi toutes les meilleures positions des agents de la population. Figure 2.



Figue 2-3 Concept de recherche avec des agents dans l'espace de solution par PSO.

3.2.5 Algorithme PSO

Chaque particule représente une solution potentielle dans l'espace de recherche. La nouvelle position d'une particule est déterminée en fonction de sa propre valeur et celle de ses voisines. Soit $X_i(T)$ la position de la particule P_i au temps t, sa position est modifie en ajoutant une vitesse $V_i(T)$ a sa position courante [16].

- Avantage

L'algorithme du PSO présente un avantage certain sur les méthodes classiques dans le sens où il permet une exploration aléatoire de l'espace de solution. De plus de sa simplicité à mettre en œuvre et son exécution conduit à l'obtention de très bon résultat, rapidement et facilement avec peu de paramètre à ajuster. Bien que le PSO trouve de bonnes solutions dans un temps beaucoup plus court que d'autres algorithmes évolutionnaires, l'amélioration de la qualité des solutions ne peut être garantie en augmentant le nombre d'itération.

- Inconvénient :

L'algorithme de PSO a deux inconvénients principaux :

- Le premier inconvénient est que l'essaim peut prématurément converger.

- Le deuxième inconvénient est que les approches stochastiques ont un problème de dépendance tout changement d'un de leurs paramètres peut avoir un effet sur le fonctionnement de l'algorithme tout comme sur la solution obtenue [6].

Algorithme : Optimisation par essaim de particule

[Les variables et paramètres de l'algorithme]

N : nombre de particules

X_i : Position de la particule P_i

V_i : vitesse de la particule P_i

P_{besti}: Meilleur fitness obtenue pour la particule P_i

X_{pbesti}: Position de la particule P_i pour la meilleur fitness

P1 , P2 : valeur aléatoire positive.

1 : Initialiser aléatoirement la population

2 : Répéter

3 : Pour i de 1 à N faire

Si $f(X_i) > P_{besti}$ Alors

| $P_{besti} = f(X_i)$

| $X_{pbesti} = X_i$

Fin Si

4 : Si $f(X_i) > P_{besti}$ Alors

$G_{besti} = f(X_i)$

$X_{gbesti} = X_i$

Fin Si

Fin Pour

5 : Pour i de 1 à N faire

$$V_{i+1} = \omega V_{i+1} + P1(X_{pbest} - X_i) + P2(X_{gbest} - X_i)$$

$$X_{i+1} = X_i + V_{i+1}$$

Fin Pour

6 : Jusqu'à ce que le processus converge

---

4 Conclusion

Nous avons vu, dans ce chapitre, Les différentes méthodes de résolution d'un problème d'ordonnancement. Nous avons exposé plusieurs méthodes de résolution exactes et approchés.

Dans le type de résolution heuristique, l'inconvénient c'est qu'on ne peut pas garantir théoriquement la bonne qualité des résultats (minimisation des coûts, de temps) pour cas d'étude bien défini. Nous sommes donc invités à tester une méthode de résolution et dans le cas échéant faire une étude comparative des différentes méthodes, cela s'avère lent et pénible numériquement.

La méthode d'optimisation par essaim a prouvé sa robustesse et sa grande efficacité pour résoudre des problèmes compliqués de grande taille tels que les problèmes d'ordonnancement, plusieurs études ont sélectionné la méthode PSO comme l'outil le plus performant pour résoudre des problèmes d'ordonnancement de type SMTWTP. Dans le chapitre suivant nous allons présenter le problème SMTWTP et exposer l'algorithme proposé.

CHAPITRE III

CHAPITRE 3
APPLICATION DE LA METHODE PSO AU PROBLEME SMTWTP

1 Introduction

Pour de nombreux problèmes, il n'existe pas de solution déterministe qui donne le résultat en un temps raisonnable, et ceci malgré la création d'ordinateurs de plus en plus performants. Pour pallier à ce problème, on a recours à des méthodes dites heuristiques, c'est-à-dire des méthodes qui fournissent une solution approchée. Toutefois, il faut reproduire le processus sur plusieurs itérations pour tendre vers une solution acceptable.

On retrouve parmi ces heuristiques, certains algorithmes qui possèdent un principe générique adaptable et qui s'applique donc à plusieurs problèmes d'optimisation, On les appelle des métaheuristiques.

La plus courante est la descente stochastique : on part d'une solution initiale, on la compare à tous ses voisins en conservant à chaque fois le *meilleur* résultat.

Dans se chapitre ont va parler brièvement sur l'optimisation par particule essaim, présenter le problème d'ordonnancement SWTMTP, et en fin parler sur l'algorithme proposer pour résoudre se problème.

2 Description du problème SMTWT

Le problème de la somme pondérée des retards des taches sur une seule machine est défini comme suit. Considérer n taches à traiter sans interruption sur une seule machine qui ne peut gérer qu'un seule tache à un moment. Chaque tache j disponible pour son exécution à l'instant zéro, a un temps de traitement positif pj, un poids positif W_j , et une Date d'échéance dj. Pour une séquence de taches donnée, le retard de la tache j est défini comme suit : Tj = max {0, Cj - dj}, où Cj est le temps d'achèvement de tache j.

L'objectif du problème SMTWTP, est de trouver une séquence de traitement de toutes les taches, Cette séquence est un ordre qui minimise la somme pondérée des retards donnée par : $\sum_{j=1}^{n} wi * Ti$

3 Les méthodes pour résoudre le problème

3.1 Pourquoi métaheuristique

Comme le temps de calcul nécessaire pour trouver une solution risque d'augmenter exceptionnellement avec la taille du problème, et la représentation de l'arbre de recherche est

impossible pour les problèmes de taille importante. Lés méthodes exactes ne satisfont pas les besoins, malgré les progrès réalisés sur ces méthodes, ils restent toujours impraticables dans les domaines pratiques, pour cette raison la, et pour résoudre ces problèmes, en a besoins des méthodes efficaces pour résoudre les problèmes de grande taille et dans un temps de calcul raisonnables, Ces méthodes sont les méthodes métaheuristiques.

3.2 Heuristique utilisée

3.2.1 Règle EDD

C'est une règle simple, elle est très util dans certain cas d'ordonnancement, cette règle est efficace elle donne des résultats approchés pour le problème SMTWT, le principe de fonctionnement de la règle EDD est de planifier la séquence des taches par rapport a leurs date d'échues , la séquence est ordonnancés suivant un ordre croissant , de la tache qui a la plus petite date d'échus vers la plus grande .

Exemple :

TACHES	1	2	3	4
PROCESSING TIME	8	6	11	4
WEIGHT	2	1	5	3
DUE DATE di	9	4	7	1

Figure 3-1 Un exemple d'une matrice de données traité par la régle EDD

La séquence des taches obtenue avec la règle EDD est :

Tache4 - Tache2 – Tache3 – Tache 1.

La valeur de la fonction objective est :

$\sum Wi\ Ti = (W_4 * T_4)+(W_2 * T_2)+(W_3 * T_3)+(W_1 * T_1)$

- $T_i = \max (0 , C_i – d_i)$ ou C_i est la date de fin d'exécution de la tache i

$\sum Wi\ Ti$ = 3*3 + 1*6+ 5*16 + 2*20

$\sum Wi\ Ti$ = 9 + 6 + 80 + 40 =135

La somme pondéré des retards de la solution obtenue par la règle EDD est : 135

4 Algorithme proposé

Dans cette section on va parler sur le travail réalisé, on va détailler les différents composants de l'algorithme PSO choisit pour résoudre le problème est sur comment faire

pour adapter la méthode choisit avec le problème SMTWT , en basant sur les élément de PSO déjà vu dans le chapitre 2.

4.1 Description du travail

Dans cette section on va parler sur le travail réaliser, un solveur bio inspirer pour le problème d'ordonnancement SMTWT qui a était d'écrit d'une manière détaillé dans la rubrique précédente. L'algorithme et basé sur la méthode PSO (PARTICLE SWARM OPTIMISATON) pour planifier un nombre de taches réaliser par une seul machine afin de trouver une séquence optimale qui répond aux critères de temps de réalisation du produit finis. Les données sont générer d'une manière aléatoire, les nombre de taches et données par l'utilisateur du solveur. Le solveur va utiliser la méthode de PSO pour trouver la séquence la plus mieux dans un temps réduit par rapport au nombre des tache données.

4.2 Les composant de l'algorithme

4.2.1 Génération de population

La méthode PSO se base sur un essaim de particule qui fait la recherche, pour trouvé la solution optimale dans l'espace de recherche. La population de PSO est l'ensemble des particules. Alors la première étape de notre algorithme est la génération de population .La population de particules est construite au hasard pour l'algorithme PSO du problème SMTWT. Voici l'algorithme de la fonction de génération de la population.

Les valeurs continues des positions sont établies au hasard. La formule suivante est utilisée pour construire les valeurs initiales de la position du particule :

- X_{ij}^{k} : désigne la ième particule dans l'essaim à l'itération k et il est représentée par n nombre de dimensions $X_{ij}^{k} = [X_{il}^{k}$,.., X_{in}^{k}] où X_{ij}^{k} est la valeur de position de la ième particule qui correspond à la jème dimension (j = 1, .., n).

  $$X_{ij}^{0} = X_{min} + (X_{max} - X_{min}) \times U(0,1)$$

  X_{min} = 0.0, X_{max}= 4.0

  U(0,1) Est un nombre uniforme aléatoire compris entre 0 et 1.

- V_{ij}^{k} : est la vitesse de la particule i à l'itération k. Elle peut être défini comme $V_{ij}^{k} = \left[V_{il}^{k} \ldots, V_{in}^{k}\right]$, où V_{ij}^{k} est la vitesse de la particule i à l'itération k par rapport à la jéme dimension. Les vitesses Initial continu sont générées par une formule similaire à celle Suit:

  $$V_{ij}^{0} = V_{min} + (V_{max} - V_{min}) * U(0,1)$$

  $V_{min} = -4.0$, $V_{max} = 4.0$

U(0,1) Est un nombre uniforme aléatoire compris entre 0 et 1.

Algorithme : Génération de population (PSO)

Particle[6][nbrTache] : un tableau de deux dimention des doubles

1: Pour i de 1 a 3 faire

2 : Pour j de 1 à nbrTache faire

Switch(j) :

Cas :1

Particule[i][j] = j ;

Cas :2

Particule[i][j] = $X_{min} + (X_{max} - X_{min}) \times U(0,1)$

Cas : 3

Particule[i][j] = $V_{min} + (V_{max} - V_{min}) * U(0,1)$

Fin Pour

Fin Pour

Le tableau suivant décrit une particule générer par l'algorithme proposé :

N	1	2	3	4
Xi	1,32	0,48	2	3,76
Vi	3,24	3,4	2,48	1,6

Figure 3-2 Un exemple d'une matrice qui représente une particule

4.2.2 Fonction SPV (Smalest Position Value)

L'évaluation de chaque particule dans l'essaim nécessite la détermination de la permutation des taches pour la PFSP puisque la valeur pondérée totale des retards est un résultat de la séquence. Nous utilisons une règle heuristique appelée plus petite valeur de position (SPV) pour permettre à l'algorithme PSO d'être appliqué à toutes les classes des problèmes d'ordonnancement, qui sont NP-hard dans la littérature. Le principe de cette règle est déterminé dans l'algorithme suivant :

Algorithme : Fonction SPV

```
Temp[nbrTache] : un tableau d'une dimention des doubles
1: Pour i de 1 à  nbrTache faire
  Temp[i] = Particule[2][j]                    //copie la ligne Xi dans un tempon
Fin Pour
Seq = 1 ;
2 : Tantque (Seq≤ nbrTache) faire
  Pour i de 1 à  nbrTache faire
3 :                Si Temp[i]< Min alors
                   Min = Temp[i] ;             // ordoner la table par la positionXi
                   Indice = i ;
            Fin Si
4 :                Effacer  la tache traiter de la table tempon.
5 :                Particule[i][j] = Seq ;

            Fin Pour
    Fin Tantque
```

La permutation est déterminée à travers les valeurs de position de la particule de sorte que la valeur Fitness ($\Sigma W_i T_i$) de la particule peut être calculée avec cette permutation.

Le principe de la règle SPV est simple, il consiste a ordonnancés les tache dans un ordre croissant par leurs valeur de position X_{ij}^k , a chaque itération on utilise la règle SPV pour déduire la nouvelle séquence de chaque particule. Le tableau suivant est l'application de la fonction SPV sur le tableau précède :

N	1	2	3	4
Xi	1,32	0,48	2	3,76
Vi	3,24	3,4	2,48	1,6
Si	2	1	3	4

Figure 3-3 L'application de SPV sur l'exemple précèdant

On a appliqué la règle SPV sur l'exemple précédent, la séquence obtenu est la suivante : Tache2 – Tache1 – Tache3- Tache4.

4.2.3 Fonction objective (Fitness)

La fonction fitness mesure la performance de chaque particule de l'essaim, dans notre travail la fonction fitness est la somme des Ti les retards pondérés, le but est de minimiser cette valeur pour avoir une séquence optimale.

La fonction fitness est :

$$\text{Fitness} = f(x) = \sum Wi\ Ti$$

- Wi : est le poids correspond a la tache.
- $Ti = \max(0, C_i - d_i)$ Ou C_i est la date de fin d'exécution de la tache i.
- D_i : date d'échus date a partir duquel le on exécution de la tache i.

Les particules qui ont la plus petit fitness ce sont les solutions les plus optimale, la fonction fitness est appliquer après le calcul des valeurs de C_i et T_i de chaque tache de la particule.

Algorithme : Fonction Fitness

Data[4][nbrTache] : un tableau de deux dimention des données

1: Tantque (i≤ nbrTache) faire

3 : S = Particule[4][j] ;

4 : fitness = Particule[6][j] * Data[3][S] ;

Fin Tantque

5: Retourner la valeur fitness

4.2.4 Sélection de Gbest

La phase sélection de G_{best} se fait a chaque itération, elle se fait après le calcul de fitness de chaque particule, P_{best} est le résultat de l'application de la fonction fitness sur une particule, après l'application de la fonction objective sur l'ensemble de la population on obtient les P_{best} des particules, G_{best} est la valeur minimum entre les p_{best}.

4.2.5 Déplacement de la particule

A chaque itération de l'algorithme on calcule la nouvelle vitesse en fonction de plusieurs éléments, pour calculer cette vitesse il nous faut la vitesse précédente de la particule, le P_{best} de la particule et le G_{best}.la fonction pour la mis-a-jour de la vitesse de la particule est donnée come suit :

$$V_i^k = W^{k-1} * V_i^{k-1} + C_1\, r_1 \left(Pb_{ij}^{k-1} - X_{ij}^{k-1}\right) + C_2\, r_2 (Gb_{ij}^{k-1} - X_{ij}^{k-1})$$

La mise-a-jour de la vitesse est appliqué sur toutes les taches de la particule et sur toute la population. Après l'étape de calcule de la nouvelle vitesse, on calcule la nouvelle position.

$$X_{ij}^{K} = X_{ij}^{K+1} + V_{ij}^{K+1}$$

Algorithme : Déplacement de la particule

1: Pour i allant de 1 à nbrTache faire
2 : Val1 = ω * Particule[3][i] ;
3 : Val2 = C_1 r_1 (P_{best} − Particule [2][j] ;)
4 : Val3 = C_2 r_2(G_{best} − Particule [2][j] ;)
5 : Vélocité = Val1+Val2+Val3 ; //nouvelle vitesse
6 : Particule [2][i] = Particule [2][i] + Vélociti //déplacement
FinPour

5 Critère d'arrêt

Le critère d'arrêt utilisé est le nombre d'itération égal a $iter_{max}$. Après $iter_{max}$ itération, l'algorithme PSO s'arrête et donne la meilleure séquence de particule entre touts les particules de l'essaime. Cette séquence qui possède un $\sum Wi\,Ti$ minimale. Généralement le nombre d'itération dans les problèmes SMTWT est grand , dépond de la taille du problème a traiter, ce nombre peut arriver jusqu'à des somme multiple de 10000.Dans notre travaille nous avons essayé de prendre un nombre d'itération qui respecte le nombre de tache du problème.

Algorithme : PSO

1: Data[4][nbrTache] : matrice des donnés

2 : Population={Particule1 ;Particule2,……..,ParticuleN}

3 : Pour touts les Particule fair

4 : Génération population(Particule)

FinPoure

7 : Pour i allant de 1 à $iter_{max}$ *faire*

6 : Pour touts les Particule fair

8 : SPV(Particule) ;

9 : Fitnesse(Particule) ;

10 : Choisir G_{best}

FinPour

FinPour

11 : Retourner la meilleure particule

6 Conclusion

Dans ce chapitre 3 on a parlé brièvement sur la méthode choisit qui est la méthode PSO, et ses fonction. On a aussi présenté le problème d'ordonnancement SMTWT d'une seule machine. L'optimisation par PSO est utilisée dans plusieurs domaines de recherche parmi ses domaines on à l'ordonnancement. Dans le chapitre suivant nous parlons sur le travail réaliser, présenter les résultats obtenus et faire une comparaison avec l'ordonnancement par EDD, par des simulations sur plusieurs nombre de taches afin d'évaluer notre travail et le taux d'amélioration de la méthode.

CHAPITRE IX

CHAPITRE 4

REALISATION ET EXPERIMENTATIONS

1 Introduction

Dans le chapitre 4 on a introduire la partie de réalisation du solveur bio-inspirer, on parler sur l'environnement matériel et logiciel. Aussi nous avons présenté les donnés utilisés dans l'étude. Pour l'étude expérimentale. Nous avons procédé à une série de tests, ces tests sont réaliser sur des données généré aléatoirement, Les données sont changer a chaque fois (nombre de tache, population, nombre et nombre itération).Les tests sont réaliser avec les deux méthodes déjà vu dans le chapitre 3, Avec la méthode PSO et la règle EDD, Enfin on a parlé sur les résultats obtenu avec une petit comparaison entre les deux méthodes.

2 Environnement matériel

Pour développer l'application, nous avons utilisé comme environnement matériel un ordinateurs Dell qui possède comme caractéristiques :

- Un processeur Intel Pentium® Core (TM) i3, 1.80 GHz.
- Une mémoire vive de 4Go.
- Un disque dur 450 Go.
- Un écran 17 pouces.•

3 Environnement logiciel

Plateforme utilisé est Windows 7 professionnel pack1 .le langage de devloppement choisit est java.On a utilisé l'environnement NetBeans IDE 8.2 pour simplifiait le travail a fin de développer notre application . L'environnement NetBeans adopte l'IDE (Integrated Développent Environment) java gratuitement. L'IDE est un logiciel open source développé par SUN Microsystems utilisé pour exécuter, compilé. Un IDE est dédié à un seul langage de programmation. Le choix se fait pour les raison suivante :

- Les logiciels écrits dans ce langage sont très facilement portables sur plusieurs systèmes d'exploitation.
- NetBeans IDE est un outil gratuit et flexible.
- Nous souhaitions, à travers ce projet, améliorer notre connaissance du langage.

4 Données utilisés

Les données utilise dans notre travail sont des données génère d'une façon aléatoire le tableau suivant donne les détails sur la génération des données.

Job Parameters	Value
Processing Time	Uniform between [1,100]
Weight	Uniform between [1,10]
Due Date	Uniform between [P*(1-TFRDD/ 2), P*(1-TF+RDD/2)] If P*(1-TF-RDD/2)<=0 then [1, P*(1-TF+RDD/2)]

Figure 4-1 : Données utilisées.

P est le temps de traitement total de tous les taches, alors que TF et RDD est le facteur d'étanchéité et la gamme de dates d'échéance respectivement. Une grande valeur de RDD montre que les dates des délais sont très larges. Une faible valeur de RDD montre que dates d'échéance des dates d'échéance est très étroite. Sur d'autre part, une valeur élevée de TF montre que les dates d'échéance sont étroites et une faible valeur de TF montre que les dates d'échéance sont lâches. Dans ce papier des problèmes de taille, 25, 50, 100,200 et 250 sont effectué.

- Processing time P_i : est le temps d'exécution de la tache i, un nombre aléatoire entre 1 et 100.
- Weight w_i :est le poids de la tache, chaque tache a un poids d'importance sur l'ensemble de projet, ce poids est entre 1 et 10.
- Due date d_i: la date d'échéance est généré aléatoirement dans l'intervalle : [P(1-TF-RDD/2) , P(1-TF+RDD/2)] .

TACHES	1	2	3	4
PROCESSING TIME	97,61	38,35	17,47	22,97
WEIGHT	9,97	9,86	5,22	6,81
DUE DATE di	361,8	141,1	368,2	623,8

Figure 4-2 une matrice de donnée

5 Critères d'évaluation

Le critère d'évaluation utilisées dans notre travail est le taux d'amélioration, letaux d'amélioration est calculer avec la somme des retards pondérés $\sum Wi\ Ti$ initiale moins la solution finale sur la solution initiale.

$$Taux\ d'amelioration = \frac{\text{Solution initial} - \text{Solution final}}{Solutionfinal}$$

6 Résultats obtenus

Population : 5			Nombre d'itération : 50		
N	Solution Initiale aléatoire	Solution finale par PSO	Solution finale par EDD	Taux d'amélioration PSO	Taux d'amélioration EDD
25	3836	3362	3753	**12,38%**	2,20%
50	16787,49	15272,13	16006	**9,03%**	4,70%
100	76932,14	39131	71535	**49,14%**	7,02%
200	221412,92	212630	249442	**3,97%**	0,00%
250	347996,14	268381	378154	**22,88%**	0,00%

Figure 4-3 : Résultats obtenue pour P_{size}= 5 ,$Iter_{max}$=50

Population : 5			Nombre d'itération : 100		
N	Solution Initiale aléatoire	Solution finale par PSO	Solution finale par EDD	Taux d'amélioration PSO	Taux d'amélioration EDD
25	3345,33	2357	3594	**29,55%**	0,00%
50	19334,91	16835	23640	**12,93%**	0,00%
100	55791,54	45076	64901,35	**19,20%**	0,00%
200	210948,78	179746	232202	**14,79%**	0,00%
250	364000,55	294657	349272	**19,05%**	4,05%

Figure 0-4 : Résultats obtenue pour P_{size}= 5 , $Iter_{max}$=100

Population : 5			Nombre d'itération : 150		
N	Solution Initiale aléatoire	Solution finale par PSO	Solution finale par EDD	Taux d'amélioration PSO	Taux d'amélioration EDD
25	4868,71	3334	4668	**31,50%**	4,01%
50	15417,75	13413	16779	**13,02%**	0,00%
100	54927,58	38026,94	61747	**30,80%**	0,00%
200	253657,43	211457	240440	**16,60%**	5,20%
250	377261,44	332800	407868	**11,80%**	0,00%

Figure 4-5 : Résultats obtenue pour P_{size}= 5 , $Iter_{max}$=150

Population : 10			Nombre d'itération : 50		
N	Solution Initiale aléatoire	Solution finale par PSO	Solution finale par EDD	Taux d'amélioration PSO	Taux d'amélioration EDD
25	2511	2275	3345	**9,40%**	0,00%
50	16695	12174	16683	**27,08%**	0,06%
100	56317	53145	57834	**5,63%**	0,00%
200	235606	169418	235606	**28,09%**	0,00%
250	432636	313150	389339	**27,61%**	10 ,0%

Figure 0-6 : Résultats obtenue pour P_{size}= 10 ,$Iter_{max}$=50

Population : 10			Nombre d'itération : 100		
N	Solution Initial	Solution Final PSO	Solution Final EDD	Taux d'amélioration PSO	Taux d'amélioration EDD
25	5472	1680	2568	**69,29%**	53,07%
50	24472	17056	21364	**30,30%**	12,70%
100	88727	51194	62394	**42,30%**	29,67%
200	327477	229464	323608	**29,92%**	1,18%
250	368860	324981	383480	**11,89%**	0,00%

Figure 0-7 : Résultats obtenue pour P_{size}= 10 , $Iter_{max}$=100

Population : 10			Nombre d'itération : 150		
N	Solution Initial	Solution Final PSO	Solution Final EDD	Taux d'amélioration PSO	Taux d'amélioration EDD
25	3128	1728	2149	**44,75%**	31,29%
50	13337	6337	11650	**52,48%**	12,64%
100	60430	54757	58323	**9,38%**	3,48%
200	235959	199365	262109	**15,50%**	0,00%
250	402039	379463	533060	**5,61%**	0,00%

Figure 0-8 : Résultats obtenue pour P_{size}= 10 , $Iter_{max}$=150

7 Analyse des résultats

Les résultats présents dans les tableaux précédents soulignent les remarques suivantes :

- En remarque une amélioration importante entre la solution initiale et la solution finale, pour PSO et la règle EDD. Ce qui confirme l'importance des méthodes approchés pour résoudre le problème SMTWT.
- L'algorithme PSO donne des résultats élève a cause de l'ensemble des technique utilisés au niveau des éléments qui composent cet algorithme. Donc PSO a réussit de parcourir l'espace des solutions afin d'attendre la meilleure solution.
- Pour la comparaison entre l'algorithme PSO et la régle EDD ont peut remarquer clairement que l'algorithme PSO est plus efficace que EDD, il donne un taux d'amélioration supérieurs que EDD.

8 Interface du logiciel développé

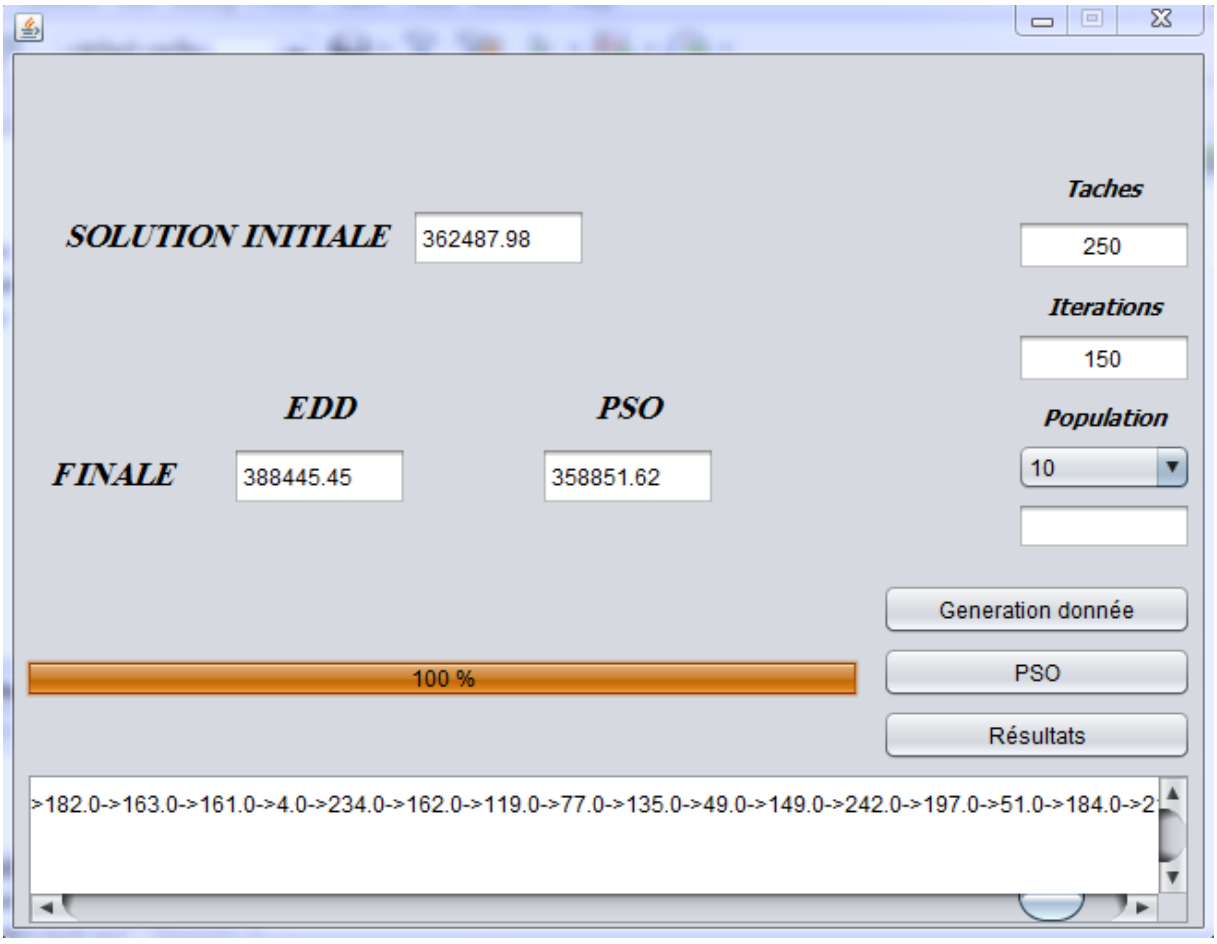


Figure 0-9 : Interface du logiciel développé.

Cette figure définit l'interface du logiciel, le logicielle contient une seul fenêtre ,cette fenetre permet a l'utilisateur de saisir les paramètre souhaiter ,comme le nombre de tache ou

le nombre d'itération , et permet aussi de visualiser les résultats obtenu . le logiciel donne la solution initial pour la séquence initial générer aléatoirement, après la simulation il donne les résultats final obtenus avec PSO et avec la règle EDD, et donne aussi la séquence associer a la meilleur solution.

9 Conclusion

Dans le chapitre 4 nous avons présenté le logiciel développé, On a commencer par présenter l'environnement matériel et logiciel d travail réaliser, ensuit en a parler sur les critères d'évaluations de travail, nous avons représenté et analyser les résultats obtenus avec plusieurs combinaison de nombre tache et nombre d'itération sur l'algorithme PSO et la règle EDD.

CONCLUSION GENERALE

Le travail exposé dans ce mémoire s'intéresse à l'étude d'une problématique industrielle concernant l'ordonnancement dans un atelier sur une seule machine. Dans cette problématique, chaque tache est caractérisée par une durée d'exécution ; une date d'échue et un poids lié a son importance, l'objectif est de déterminer une séquence de taches sur la machine afin de minimiser la somme pondérée des retards des taches. Nous avons justifié le choix qui nous a amené à étudier ce type de système de production.

Après avoir présenté les différents problèmes d'ordonnancement d'ateliers et un certain nombre de méthodes de résolution, nous avons choisi d'appliquer l'algorithme d'optimisation par essaim de particules (*PSO : Particle Swarm Optimization*) pour trouver une solution approchée au problème posé dans notre travail.

Basés sur un processus d'évolution naturelle, l'algorithme PSO est parmi les techniques d'optimisation qui sont basées sur l'exploration aléatoire de l'espace des solutions dites particules, ils cherchent l'optimum à partir d'une population de solutions et non à partir d'une seule solution. Dans le cas des problèmes d'ordonnancement, chaque particule correspond à une solution d'ordonnancement possible, la meilleure particule obtenue après plusieurs générations correspond à la meilleure solution d'ordonnancement.

A travers une série d'expérimentations numériques sur des problèmes de différentes tailles, nous avons montré que l'algorithme PSO appliqué au problème SMTWTP est efficace et donne des résultats encourageantes dans un temps raisonnables ce qui souligne la performance de notre méthode de résolution d'un point de vue qualité de la solution trouvée.

Comme perspective de ce travail, il est possible d'utiliser d'autres heuristiques pour générer la solution initiale afin d'assurer la rapidité de la méthode. De plus, on peut faire des comparaisons avec d'autres métaheuristiques à populations telle que les algorithmes génétiques sur les mêmes données pour évaluer la performance de la méthode PSO utilisée.

REFERENCES

[1] **Annexe 6**. Notions d'ordonnancement. APP3 Optimisation Combinatoire: Problèmes sur-contraints et ordonnancement. Mines-Nantes, option GIPAD, 2011-2012.

[2] B.Taha, **Optimisation des Problèmes d'Ordonnancement Multi Objectifs dans les Systèmes de Production**, Mémoire magister, Soutenu le 18- 12-2005.

[3] B. Meroine Hocine ,**Les problèmes d'ordonnancements a mchines parallèle, De Taches Dépendantes**, Mémoire de magister, Université Saad Dahleb de Blida, juillet 2006.

[4] C. Flipo-Dhaenens, **Optimisation d'un réseau de production et de distribution**, Thèse de doctorat, INPG de Grenoble. 1998, p 197.

[5] **Cours des Méthodes de Résolution Exactes Heuristiques et Métaheuristiques Université Mohammed V**, Faculté des Sciences de Rabat Laboratoire de Recherche Mathématiques, Informatique et Applications,Chapitre1,P14.

[6] D. Fadila , **Méthodologie d'optimisation par les techniques intelligentes d'un contrôleur PID pour un système CSTR** , Mémoire de Master , Université Ferhat Abess Setif , Soutenu le /06/2014.

[7] H. Hentous**, Flow Shop hybride sous contraintes**, Rapport de DUA de recherche opérationnelle, Université de Joseph Fourier, 1995.

[8] H. Hentous, **Contribution au pilotage des systèmes de production de type Job-Shop**. Thèse de doctorat, INSALYON, 1999, p 149.

[9] J. Carlier, P. Chrétienne, **Problèmes d'ordonnancement : modélisation, complexité, algorithmes**, Paris : Masson, 1988.

[10] J. DRÉO, Alain PÉTROWSKI, Patrick SIARRY et Eric TAILLARD : **Métaheuristiques pour l'optimisation difficile**. Eyrolles, 2003. Cité p 59, 60, 67, 71.

[11] J.Shwimer, "On the n-job, One-machine, **Sequence-independent Scheduling Problem with Tardiness Penalties**: A Branch-Bound Solution", Management Science, 18(6), B-301-B-313, 1972.

[12] Lopez, P. Roubellat, **Ordonnancement de la production**. Paris Hermès, 2001, 432p.

[13] L. Schrage, and Baker, K.R., "**Dynamic Programming Solution of Sequencing Problem with Precedence Constraints**", Operations Research, 26, pp. 444-449, 1978.

[14] L. INGBER : **Very fast simulated re-annealing. Mathematical Computer Modelling**, 12(8):967–973, 1989. Cité p 61.

[15] M. Fatih Tasgetiren ,Yun-Chia Liang ,Mehmet Sevkli ,Gunes Gencyilmaz, Particle **Swarm Optimization Algorithm for Single Machine Total Weighted Tardiness Problem** , Page 14.

[16] N. METROPOLIS, Arianna W. ROSENBLUTH, Marshall N. ROSENBLUTH, Augusta H. TELLER et Edward TELLER **: Equations of State Calculations by Fast Computing Machines**. Journal of Chemical Physics, 21(6):1087–1092, 1953. Cité p 60.

[17] P. SIARRY et G. DREYFUS **: La méthode du recuit simulé : théorie et applications. ESPCI - IDSET**, 10 rue Vauquelin Paris, 1989. Cité p 60.

[18] S. KIRKPATRICK, C. D. GELATT et M. P. VECCHI **: Optimization by Simulated Annealing. Science**, 220(4598):671–680, 1983. Cité p 60.

[19] V. Giard, **Gestion de la production et des Flux**, Paris Economica, 2003.

[20] V. Giard, **Gestion de production**, Paris Economica, 1988.

ملخص :

الهدف من هذه المذكرة هو دراسة مشكلة الترتيبات لمجموعة من الأشغال على آلة واحدة حيث يرتبط كل عمل بزمن تنفيذ مدة التأخر و وزن يحدد أهميته بحيث نحصل على الترتيبية المناسبة لتقليص دالة المجموع الترجيحي لمدة تأخر الإشغال عن الوقت المحدد لهم و للعلم انه لا يمكن تحديد خوارزمي محدد يضبط حل نهائي للمشكل المطروح. لذلك لجانا الى استعمال طريقة التحسين بالسرب مع تحديد وسائط هذه الطريقة التقريبية من اجل إيجاد حل تقريبي للمشكل مع مراعاة تقليل زمن التنفيذ و ذلك عن طريق تغيير بعض الإعدادات.

Abstract :

The work of this thesis concerns the study of the problem of scheduling tasks on a single machine, where each task has duration of execution, a due date and a weight related to its importance. The objective is to determine a sequence of tasks on the machine to minimize the weighted sum of tardiness. The problem is known to be NP-hard. To solve this problem a there is a method called Particul Swarm Optimization algorithm(PSO) our best knowledge here we proposed the application of this metaheuristic to determine an approximate solution and less costly in terms of time execution by changing some settings.

Résumé :

Le travail de ce mémoire concerne l'étude du problème d'ordonnancement des taches sur une seule machine ou chaque tache à une durée d'exécution, une date d'échus et un poids lié à son importance et l'objectif est de déterminer une séquence de taches sur la machine afin de minimiser la somme pondéré des retards. Le problème est connue d'être NP-difficile. Pour résoudre ce problème une méthode dite algorithme d'optimisation par essaim particulaire a notre meilleur connaissance ici ou a proposé l'application de cette méta heuristique pour déterminer une solution approchée et moins couteuse de point de vue temps d'exécution par la modification de certains paramètres.