REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOHAMED BOUDIAF de M'SILA

FACULTE DE MATHEMATIQUES ETD'INFORMATIQUE

DEPARTEMENT D'INFORMATIQUE

MEMOIRE de fin d'études

Présenté pour l'obtention du diplôme de MASTER
Domaine : Mathématiques et Informatique
Filière : Informatique
Spécialité : Informatique Décisionnelle et Optimisation

Par: BENHADDAD Hind et BELABBAS Manar

THEME

ALGORITHME GENETIQUE PARALLELE POUR L'ORDONNANCEMENT FLOW SHOP

Soutenu publiquement le : / /2022 devant le jury composé de :

Dr. MOUHOUB Nassereddine	**UMB M'sila**	**Président**
Dr. BOUNIF Mohamed	**UMB M'sila**	**Examinateur**
Dr. HEMMAK Allaoua	**UMB M'sila**	**Rapporteur**

Année Universitaire : 2021 - 2022

Dédicace

A Dieu Tout-Puissant, qui m'a permis d'apprécier cette étape de ma carrière.

A mon cher père…que Dieu le guérisse de sa maladie,Merci beaucoup pour les ailes que vous m'avez données, pour m'avoir appris à me lever et à élargir mes horizons vers les cieux…Quelques personnes ne croient pas aux héros, mais c'est parce qu'ils ne vous connaissent pas. Merci papa…

A mon amour et généreuse mère, Merci pour tout ce que tu m'as donné, pour ce que vous m'avez appris et pour l'amour que vous m'avez inculqué. pour, ta patience, ta compréhension et pour supporter mon butin tout le temps. Merci maman…

A mes frères, mes sœurs, ma famille, mes proches.

A mes amis surtout.. Nour Elyakin, Mbarka , Asma et Mouna .

A mes chers professeurs, à tous ceux qui m'ont enseigné tout au long de ma vie scolaire, et tout ceux qui m'ont soutenu même avec un simple mot.

finalement…

Je consacre tous mes efforts et mes efforts à moi-même et à ma confiance en elle, car mes objectifs qui accompagnaient ma respiration à chaque inspiration et expiration, pour mes larmes et mon sourire, pour ma croyance en mes capacités et mon courage, sont devenus une réalité qui était juste un rêve hier, et cette étoile pour laquelle je me bats est très proche.

Hind

Dédicace

قال تعالى{ يَرْفَعِ اللَّهُ الَّذِينَ آمَنُوا مِنكُمْ وَالَّذِينَ أُوتُوا الْعِلْمَ دَرَجَاتٍ}

Loué soit Dieu, qui nous a éclairés sur le chemin de la connaissance et de la connaissance, nous a aidés à accomplir ce devoir et nous a accordé le succès dans l'achèvement de ce travail

La locomotive de recherche a franchi de nombreux obstacles, mais j'ai essayé de la contourner régulièrement, grâce à Dieu et de sa part…

***A** qui je le préfère à moi-même et pourquoi pas, elle s'est sacrifiée pour moi, et n'a ménagé aucun effort pour me rendre toujours heureux (ma mère adorée)…*

*Nous marchons sur les chemins de la vie, et celui qui contrôle nos esprits reste dans chaque chemin que nous prenons Le propriétaire d'un bon visage et de bonnes actions, **il** ne m'a pas épargné toute sa vie (cher père)…*

***A** mes frères, je suis mon soutien dans ce monde et je ne les compte pas autant*

***A** tous mes proches et à tous les amis et proches sans exception, à mes chers professeurs et à tous les compagnons d'études*

Enfin, je demande à Dieu Tout-Puissant de faire de ce travail qui est le nôtre un bienfait dont tout le monde peut bénéficier.

Manar

Remerciements

Nous remercions Dieu qui nous a accordé le don de la connaissance et nous a permis d'accomplir ce travail…

Nous tenons à exprimer nos sincères remerciements et notre gratitude à tous ceux qui ont contribué à la réalisation de ce travail…

De la pose de sa première brique à son achèvement, cependant, la reconnaissance de la beauté dicte que nous avançons…

Merci au docteur ***HammakAllaoua*** *- qui a gentiment supervisé cette note*

Nous remercions également les membres du jury pour leur accord et Ils nous ont accordé leur attention et ont évalué notre travail…

TABLE DES MATIERES:

Liste des Figures

Liste des Tables

Liste des Algorithmes

Introduction générale

La recherche opérationnelle (aussi appelée aide à la décision) peut être définie comme l'ensemble des méthodes et techniques rationnelles d'analyse et de synthèse des phénomènes de management du système d'information utilisables pour élaborer de meilleures décisions, avec des objectifs limités. Elle fournit des outils (algorithmes et structures des données, optimisation combinatoire, graphes, complexité, programmations linéaire et mathématique, probabilités et statistiques…) pour rationaliser, simuler et optimiser l'architecture et le fonctionnement des systèmes industriels et économiques. Elle propose des modèles pour analyser des situations complexes et permet aux décideurs de faire des choix efficaces et robustes. Donc La recherche opérationnelle est une discipline exploitant ce qu'il y a de plus opérationnel dans les mathématiques, l'économie et l'informatique. Elle est en prise directe avec l'industrie et joue un rôle-clé dans le maintien de la compétitivité.

Les problèmes d'ordonnancement consistent généralement en un ensemble de tâches contraintes dont l'exécution nécessite des ressources. Résoudre des problèmes d'ordonnancement consiste à organiser ces tâches, c'est-à-dire à déterminer leurs dates de début et de fin, et à leur affecter des ressources dans le respect des contraintes.Il existe plusieurs modes de planification où : un modèle de flow shop tel qu'il contient n jobs qui seront traités sur m machines.chaque la machine traite une tâche pendant un moment. le temps qu'il nous faut pour finir le traitement de tous les jobes dans toutes les machines est Makespane Cmax.

En ce moment, nous examinons comment optimiser Makespan; il existe plusieurs façons Pour l'optimisation, nous choisissons des algorithmes génétiques issus de la nature, qui Le plus utilisé dans de nombreux domaines. En effet, les algorithmes génétiques sont a pu trouver une solution qui n'est peut-être pas exacte, mais Rapide et efficace Nous essaierons également d'utiliser un algorithme génétique parallèle Étudier la différence et l'ampleur de l'effet du parallélisme sur l'accélération Trouver une solution.

Ce mémoire est composé de cinq chapitres dont on présente une brève description dans les paragraphes suivants :

Dans Le premier chapitre nous allons étudier l'ordonnancement en général, On présente le problème d'ordonnancement en général, puis expliquons Les éléments d'un problème dans la troisième section, Comme pour les quatrième et cinquième sections, nous présenterons respectivement une classification et une représentation des problèmes d'ordonnancement, Enfin, dans la dernière section de ce chapitre en expliquant les différentes méthodes de résolution du problème d'ordonnancement.

Dans le deuxième chapitre, On explique c'est quoi les algorithmes génétiques, et leur principes, en plus comment ils fonctionnent outre les opérateurs des algorithmes génétiques, et enfin introduisons leur parallélisme et leurs modèles.

Dans le troisième chapitre de cette thèse est l'état de l'art,on représente des articles des chercheures sur notre sujet de thèse, et résumons donc chaque article.

Ensuite en le quatrième chapitre, on se propose de conception un algorithme génétique parallèle pour la résolution de problème d'ordonnancement de flow shop.

Dans le dernier chapitre, va vous présenter la réalisation de cette étude, nous avons conçu commençons par la présentation de martiale de développement après ca nous expliquant les logiciels qui nous utilisons que nous avons explique dans le précédent chapitre, à savoir les algorithme génétiques, adapté au problème d'ordonnancement flow shop, enfin nous testons notre programme avec des paramètres déférent.

Nous terminerons notre travail par une conclusion générale qui expose un résumé de ce qu'a été étudié dans ce mémoire et les résultats obtenus de cette recherche.

CHAPITRE I

ELEMENTS D'ORDONNANCEMENT

1. Introduction

Dans ce premier chapitre, nous présentons le problème d'ordonnancement en général, puis expliquons Les éléments d'un problème dans la troisième section, Comme pour les quatrième et cinquième sections, nous présenterons respectivement une classification et une représentation des problèmes d'ordonnancement.

Enfin, dans la dernière section de ce chapitre en expliquant les différentes méthodes de résolution du problème d'ordonnancement.

2. Définitions

« Ordonnancer, c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution»[25].

L'ordonnancement est une branche de la recherche opérationnelle et de la gestion de la production qui vise à améliorer l'efficacité d'une entreprise en termes de coûts de production et de délais de livraison. Les problèmes d'ordonnancement sont présents dans tous les secteurs d'activités de l'économie, depuis l'industrie manufacturière jusqu'à l'informatique[21].

Ordonnancer le fonctionnement d'un système industriel de production consiste à gérer l'allocation des ressources au cours du temps, tout en optimisant au mieux un ensemble de critères .C'est aussi programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution[8].

Un problème d'ordonnancement consiste à affecter des tâches à des ressources à des instants donnés pour répondre au mieux aux besoins exprimés par un client, au meilleur coût et dans les meilleurs délais, tout en tenant compte des contraintes[5].

3. Les éléments d'un problème d'ordonnancement

Le problème d'ordonnancement est constitué principalement de quatre éléments suivants :

3.1 Les tâches

Une tâche est une entité élémentaire localisée dans le temps, par une date de début et/ou de fin, et dont la réalisation nécessite une durée préalablement définie.

Elle est constituée d'un ensemble d'opérations qui requiert, pour son exécution, certaines ressources et qu'il est nécessaire de programmer de façon à optimiser un certain objectif.
On distingue deux types de tâches[7] :

- **les tâches morcelables (préemptibles) :**qui peuvent être exécutées en plusieurs fois, facilitant ainsi la résolution de certains problèmes.
- **les tâches non morcelables (indivisibles) :**qui doivent être exécutées en une seule fois et ne sont interrompues qu'une fois terminées.

3.2 Les ressources

Une ressource est un moyen technique ou humain utilisé pour réaliser une tâche. On trouve plusieurs types de ressources:

- **Les ressources renouvelables :** qui après avoir été allouées à une tâche, redeviennent disponibles (machines, personnel, etc.).
- **Les ressources consommables :** qui après avoir été allouées à une tâche, ne sont plus disponibles (argent, matières premières, etc.).

Qu'elle soit renouvelable ou consommable, la disponibilité d'une ressource peut varier au cours du temps. Par ailleurs, dans le cas des ressources renouvelables, on distingue principalement, les ressources disjonctives qui ne peuvent exécuter qu'une tâche à la fois et les ressources cumulatives qui peuvent être utilisées par plusieurs tâches simultanément mais en nombre limité[7].

3.3 Les contraintes

Les contraintes expriment les restrictions que peuvent prendre conjointement les variables de décision. Donc les contraintes nous renseignent sur les limites imposées par l'environnement. On distingue plusieurs types de contraintes, qui sont[5][16]:

- **Les contraintes temporelles :** Pour la réussite d'un projet ou d'un plan de production, on doit se soumettre aux impératifs de production (réalisation) traduits par le respect d'un planning fixé avant. Au niveau global on parlera d'une date de lancement d'une tâche et d'une date de livraison. A un autre niveau de détail plus fin, pour une tâche ou une opération. Ce dernier niveau se traduit par la définition des dates suivante : (i) date de disponibilité : ri ; (ii) date d'échéance : di.
- **Les contraintes de précédence :** une contrainte qui lie le début d'une activité à la fin d'un autre est appelée contrainte de succession ou de précédence. Les gammes opératoires sont un exemple de ces contraintes, d'autres contraintes sont imposées dans certain cas telles que les contraintes de synchronisation, de simultanéité, ou de recouvrements etc[16].
- **Les contraintes de ressources :** Les contraintes de ressources représentent le fait que les activités requièrent tout au long de leur exécution une certaine quantité de ressources[2]**.**Les contraintes de ressources concernent les deux points suivants: (i) l'utilisation de ces ressources, (leur nature, la quantité nécessaire, et les caractéristiques d'utilisation) ; (ii) la disponibilité et la quantité de ces ressources[16].

3.4 Les critères

Un critère correspond à des exigences qualitatives et quantitatives à satisfaire permettant d'évaluer la qualité de l'ordonnancement établi.

Les critères dépendant d'une application donnée sont très nombreux; plusieurs critères peuvent être retenus pour une même application. Le choix de la solution la plus satisfaisante

dépend du ou des critères préalablement définis, pouvant être classés suivant deux types, réguliers et irréguliers.

Les différents critères ne sont pas indépendants; certains même sont équivalents. Deux critères sont équivalents si une solution optimale pour l'un est aussi optimale pour l'autre et inversement[10]:

- **Les critères réguliers :** constituent des fonctions décroissantes des dates d'achèvement des opérations. Quelquesexemplessontcités ci-dessous:

  -la minimisation des dates d'achèvement des actions.
  -la minimisation du maximum des dates d'achèvement des actions.
  -la minimisation de la moyenne des dates d'achèvement des actions.
  -la minimisation des retards sur les dates d'achèvement des actions.
  -la minimisation du maximum des retards sur les dates d'achèvement des actions.

- **Les critères irréguliers :** sont des critères non réguliers, c'est-à-dire qui ne sont pas des fonctions monotones des dates de fin d'exécution des opérations, tels que:
  -la minimisation des encours.
  -la minimisation du coût de stockage des matières premières.
  -l'équilibrage des charges des machines.
  -l'optimisation des changements d'outils.

La satisfaction de tous les critères à la fois est souvent délicate, car elle conduit souvent à des situations contradictoires[1]et à la recherche de solutions à des problèmes complexes d'optimisation.

Les critères les plus utilisés font intervenir la durée totale, les retards et le coût des stocks des encours.

Minimiser la durée totale : La durée totale de l'ordonnancement notée C_{max} est égale à la date d'achèvement de la tâche la plus tardive : $C_{max} = max\ (C_i)$. Minimiser la durée totale, revient à minimiser C_{max} c'est-à dire$min\ (max\ (C_i))$.

Minimiser les retards : On peut rencontrer plusieurs problèmes dans lesquels il faut respecter les délais di. Les critères retenus sont les retards vrais $T_{max} = \max(T_i)$ et les retards algébriques $L_{max} = \max(L_i)$. Parfois on retient le nombre de tâches en retard $\sum U_i$.

Minimiser les encours : Les encours sont déterminés par le temps de présence des tâches dans le système : $F_i = C_i - r_i$. Le critère essentiel est de minimiser la somme de ces temps $\sum_{i=1}^{n} F_i$. Lorsqu'on tient compte des coûts de présence des tâches dans le système ou de leur importance, on attache un poids w_i au temps de présence, le critère devient $\sum_{i=1}^{n} w_i F_i$. Puisque les dates de disponibilité ri sont fixées (constantes), on retient seulement le critère $\sum_{i=1}^{n} C_i$ (somme de la date de fin d'exécution) et le critère $\sum_{i=1}^{n} w_i C_i$ (somme pondérée des dates de fin d'exécution).

4.Classification des problèmes d'ordonnancement

4.1 Schémas de classification

Nous suivons le schéma de classification proposé par Graham 1979, la classification à trois champs **α, β, γ** [19]:

- **α : environnement machine.**
  - α**1 :** Prend des différents caractères selon le type de problème**.**
  - α**2 :** Désigne le nombre de machine .Si α2 est entier positive, le nombre de machine supposé constant .si α2 est absent alors ce nombre est supposé arbitraire.

- **β : les caractéristiques des tâches.**
  - β**1 =** pmtn si la préemption des tâches est autorisée, sinon β1 est absent. S'il y a des contraintes de précédence entre les tâches β**2** {prec, chain, tree }, sinon β**2** est vide.
  - S'il y a des contraintes de précédence entre les tâches β2 {prec, chain, tree }, sinon $\beta 2$ est vide
  - $\beta 3 = ri$ si les dates de début au plus tôt ri (ou dates de disponibilité) des tâches ne sont pas forcément identiques, sinon (j, $ri = 0$) β3 est absent.3
  - $\beta 4$ = j si la tâche ou le job possède une date de fin obligatoire.

- **γ : le (ou les) critère(s) à optimiser.**

4.2 Les modèles des problèmes d'ordonnancement

Les problèmes d'ordonnancement sont généralement divisés en deux modèles principaux basés sur le nombre d'opérations requises pour un travail : les modèles à opération unique (machines simples et parallèles) et les modèles à plusieurs opérations (flow-shop, open shop et job shop).

4.2.1 Modèles à une opération

Il est représenté dans un Modèle à machine unique et un Modèle à machines parallèle.

4.2.1.1 Modèle à machine unique (α = 1) :

Il y a une seule machine dans le système. Chaque tâche est constituée d'une seule opération. Dans les problèmes à une machine, il faut déterminer une séquence des tâches optimisant le critère de performance. Les résultats obtenus pour les problèmes à une machine sont souvent utilisés pour résoudre des problèmes plus généraux[12].

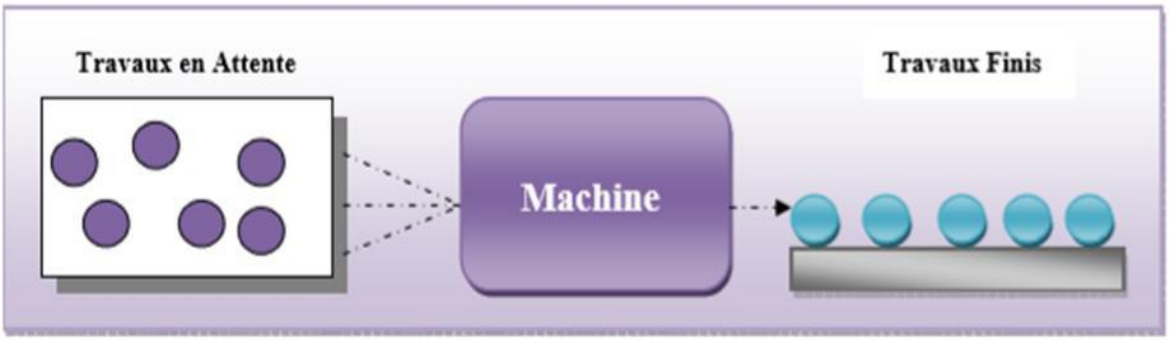


Figure 1.1 Modèle à machine unique[22].

4.2.1.2 Modèle à machines parallèle

Le système est composé de m machines parallèles. Chaque tâche est constituée d'une seule opération et elle peut être traitée sur n'importe laquelle des m machines. Dans les problèmes à machines parallèles, il y a deux types de décisions à prendre: affecter les tâches aux machines et décider de l'ordre des tâches sur une même machine [12] .

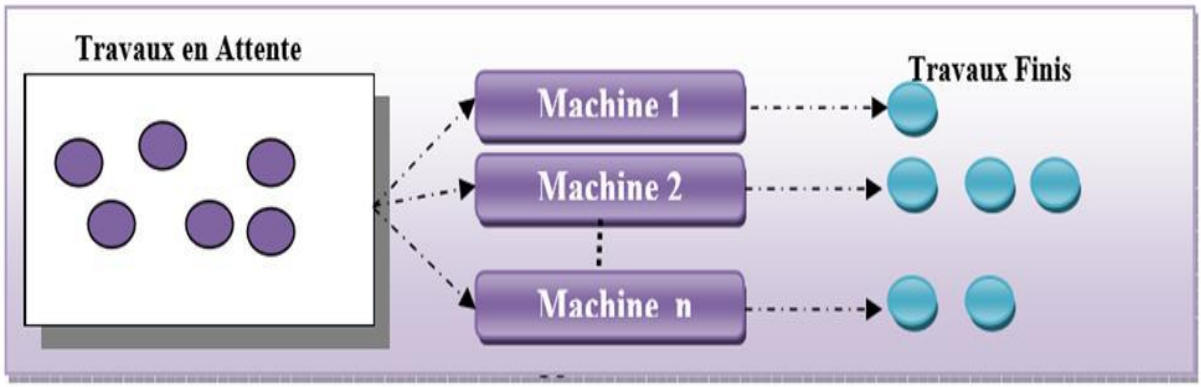


Figure 1.2 Modèle à machine parallèle[22].

4.2.2 Modèles à plusieurs opérations

Le modèle à plusieurs opérations est constitué Les situations dans lesquelles job, pour se réaliser, doit passer par plusieurs machines, chacune de ces machines ayant ses spécificités. On distingue trois modèles selon l'ordre de passage des jobs sur les machines, à savoir les modèles de flow-shop, job-shop et open-shop.

4.2.2.1 Flow-shop (α = Fm) :

Un atelier à cheminement unique est un atelier où le processus d'élaboration de produits est dit « linéaire », c'est-à-dire lorsque les étapes de transformation sont identiques pour tous les produits fabriqués. Si on trouve plusieurs exemplaires identiques et parallèles de la même machine, l'atelier devient Flow-Shop Hybride [24].

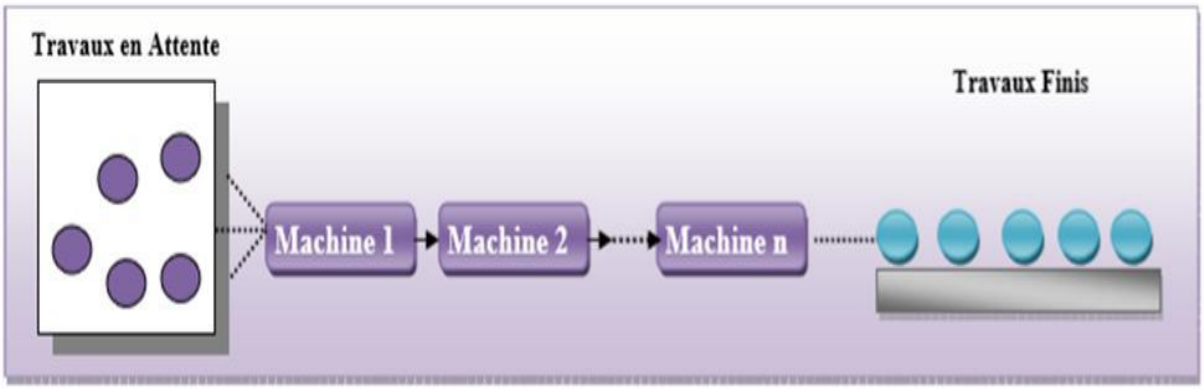


Figure 1.3 Modèle flow-shop[22].

4.2.2.2 Job-shop (α = Jm) :

Dans un Job shop à m machines, chaque tâche a un cheminement prédéterminé à suivre. Elle peut visiter certaines machines plus d'une fois et elle peut ne pas visiter certaines machines du tout. L'ordre de visite des machines pour une tâche dépend de sa gamme (l'ordre des opérations qui la composent). Les durées des opérations sur les différentes machines font parties des données du problème [12].

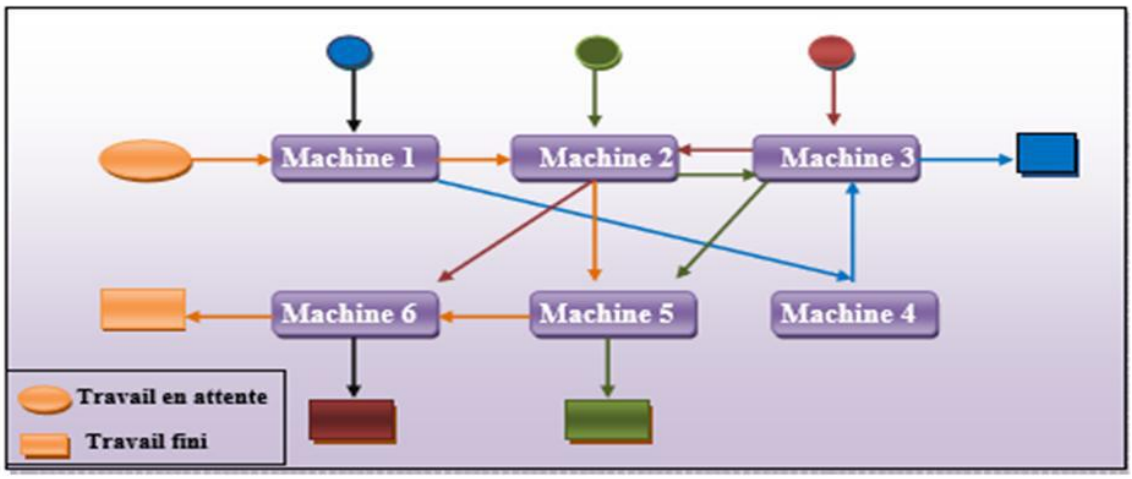


Figure 1.4 Modèle job-shop[22].

4.2.2.2.1 Formalisation linéaire

Plusieurs formulations linéaires existent pour le job-shop et certaines d'entre elle se basent sur la formulation de Manne. Dans Pham (2008), une évaluation des formulations linéaires du job-shop est proposée[4].

Pour un problème de Job-Shop de n jobs et m machines, on considère les notations suivantes:

Notations	Description
$\tilde{J}$	l'ensemble de tous les jobs;$\tilde{J}$ = {1;2;…; n} ;
I	l'ensemble de toutes les opérations
M	l'ensemble de machines M= {1;2;…; m} ;
Aj	l'ensemble de tous les couples d'opérations consécutives pour le job j $\in\tilde{J}$;
B	l'ensemble de tous les couples d'opérations (i, j) $\in I$, $i\neq j$ exécutées sur la même machine ;
Ik	l'ensemble des opérations exécutées sur la machine $k \in M$;
Pi	Le temps opératoire de l'opération $i \in I$;
H	un nombre entier positif suffisamment grand
xi	La date de début de l'opération $i \in I$
$x\tau$	La date de début de l'opération *

Table 1.1 La table de les notations

Pour chaque couple d'opérations ∀ (i, j) ∈I s'exécutant sur la même machine k ∈M
$Y_{\chi\gamma}$ {si l′opération i exécutée avant l′opération j. 0 sinon.

4.2.2.2.2 Formalisationmathématique

Nembers	les contraintes
(1)	$xj - xi \geq Pi \forall\ (i,j) \in Ak, \forall k \in \tilde{J}$
(2)	$xj + H\ (1 - yij) - xi \geq Pi \forall\ (i, j) \in B$
(3)	$xi + Hyij - xj \geq Pi \forall\ (ij) \in B$
(4)	$x\tau - xi - Pi \geq 0 \qquad \forall i \in I$
(5)	$yij \in \{i, j\}\ \forall\ (i,j) \in B$
(6)	$x_i \geq 0\ \forall i \in I$
(7)	$x\tau \geq 0$

Table 1.2 La table des contraintes

- **les contraintes (1) assurent :** aucune opération ne peut commencer avant la fin d'exécution de l'opération qui la précède.
- **les contraintes (2) et (3) assurent:** des contraintes de disjonction machine et assurent que deux opérations s'exécutant sur la même machine doivent être ordonnées grâce à l'utilisation de la variable binaire $yij \in \{0,1\}$ de **la contrainte (5).**
- **Les contraintes (4) :** fixent la date de début de l'opération *, ceci est assuré par le fait que $x\tau$est supérieur à toutes les dates de début plus la procession time $xi+Pi$ de chaque opération par $i \in I$.
- **Les contraintes (6) et (7) :** imposent que toutes les dates de débuts des opérations sont positives ou nulles[4].

4.2.2.3 Open-shop (α = Om) :

Dans un open shop avec m machines, chaque tâche a besoin d'être traitée exactement une fois sur chacune des machines (comme dans un flow shop). Sauf que l'ordre de passage des opérations constituant la tâche n'est pas fixé a priori (cheminement libre). Contrairement aux autres modèles, l'open shop n'est pas courant dans les ateliers et il n'est pas beaucoup étudié dans la littérature [15].

5. Représentation des problèmes d'ordonnancement

Il ya trois représentations possibles des problèmes d'ordonnancement : Le diagramme de Gantt, la méthode PERT et graphe Potentiel-Tâches.

5.1 Le diagramme de Gantt

Ce diagramme a été créé par Henry Gantt en 1910, est le plus courant pour l'ordonnancement, on peut avoir deux représentations de ce diagramme :

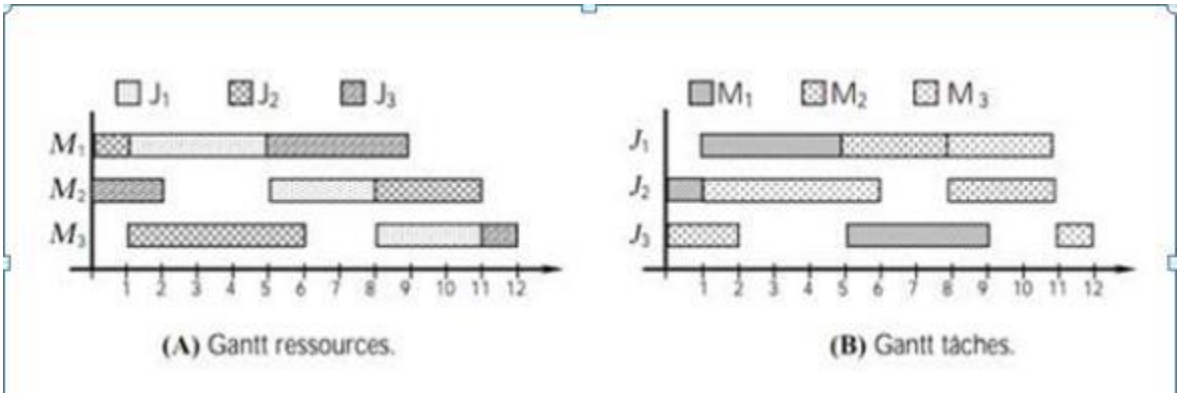


Figure 1.5Le diagramme de Gantt ressource et Le Gantt tâche

Pour les problèmes d'ordonnancement avec contraintes cumulatives, la représentation par diagramme de Gantt peut encore être élargie en considérant une représentation hybride entre le diagramme à barres et la courbe de charge. Ce mode de représentation, qui visualise l'exécution temporelle des tâches sur une ressource ainsi que leur consommation de cette ressource, sert de support pour l'illustration de caractéristiques générales des ordonnancements[9].

5.2 Méthode PERT (Program Evaluation and Research Task)

PERT est un outil de planification de projet. Il est utilisé pour calculer, de façon réaliste, le temps nécessaire pour terminer un projet. PERT signifie «Program Evaluation Review Technique» (technique d'évaluation et d'examen de programmes). Les diagrammes de PERT permettent de planifier les tâches au sein d'un projet en vue de faciliter la planification et la coordination des membres de l'équipe qui accomplissent le travail. Les diagrammes de PERT ont été créés dans les années 1950 afin de gérer la fabrication d'armes et les projets de défense pour la Marine américaine. Tandis que l'outil PERT était introduit dans la Marine, une méthode similaire appelée Critical Path (chemin critique) voyait le jour dans le secteur privé. PERT est similaire au chemin critique dans la mesure où ils sont tous deux utilisés pour visualiser le calendrier et le travail qui doit être effectué pour un projet. Cependant, avec PERT, vous créez trois estimations de temps différentes pour le projet : vous estimez le temps le plus court possible que prendra chaque tâche, le temps le plus probable et le temps le plus long si les choses ne se passent pas comme prévu. La valeur PERT est calculée à rebours à partir d'une date de fin fixe, les dates limites des sous-traitants ne pouvant généralement pas être déplacées[28].

5.3 Graphe Potentiel-Tâches(MPM)

C’est la représentation originale d’un graphe, aujourd’hui la méthode MPM (Méthode des Potentiels et antécédents Métra) lui est préférée.

Dans un graphe potentiel-tâches (Activity-on-Arrow, AoA), les tâches sont représentées par un arc reliant deux étapes symbolisées par deux cercles. Le premier cercle « étape de départ » symbolise le début de la tâche, le second cercle« étape de fin » marque la fin de la tâche.

Le nom de la tâche est inscrit au dessus de l'arc et sa durée en dessous.

Chaque cercle indique le numéro de l'étape, la date au plus tôt et la date au plus tard qui résultent du calcul du chemin critique.

Sur l'étape de départ, la date au plus tôt représente la date au plus tôt du début de la tâche et la date au plus tard représente la date de début au plus tard qui ne retardera pas l'ensemble du projet.

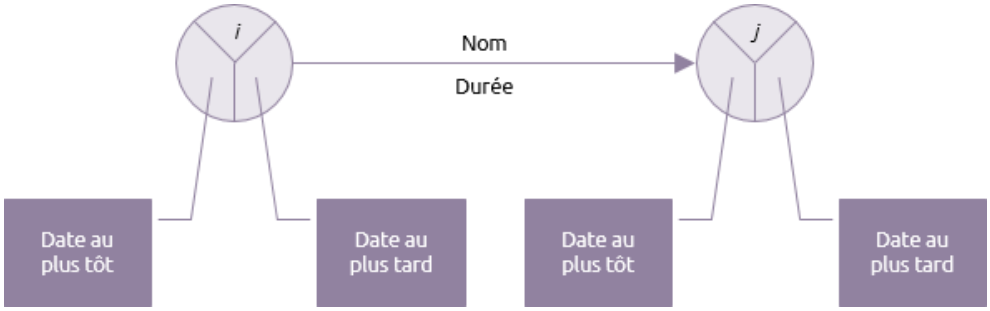


Figure 1.6 : Graphe Potentiel-Tâches

.

Sur l'événement de fin, la date au plus tôt représente la date au plus tôt de la fin de la tâche et la date au plus tard représente la date de fin au plus tard qui ne retardera pas l'ensemble du projet.

Un graphe est crée en connectant les tâches dépendantes. Par exemple, dans le graphe ci-dessous, les tâches 10-15 et 10-25 ne peuvent commencer avant que la tâche 5-10 ne soit complétée. La tâche 30-35 ne peut commencer avant que les tâches 20-30 et 25-30 ne finissent.

Une ligne pointillée est une tâche fictive, elle représente une dépendance entre deux événements.

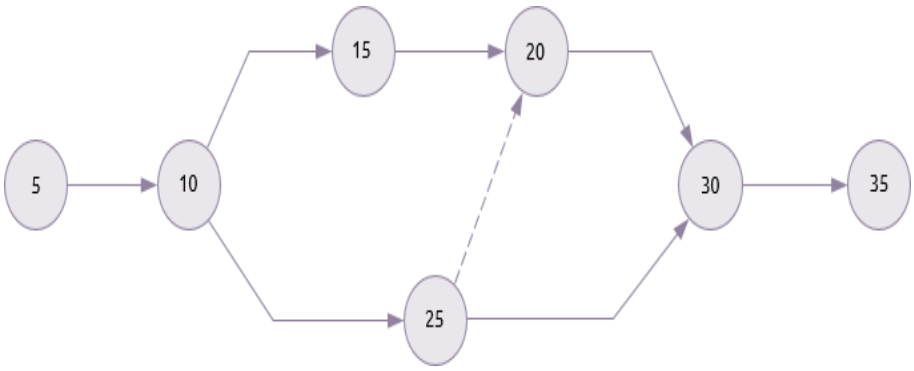


Figure 1.7 Exemple de Graphe Potentiel-Tâches.

Cette méthode était répandue avant l'arrivée des logiciels d'analyse de chemin critique et donc les calculs et graphes étaient établis manuellement. Par convention les étapes

étaient numérotées avec un créneau de 5 afin de permettre l'ajout d'étapes sans perturber la séquence numérique[29].

6. Les méthodes de résolution du problème d'ordonnancement

6.1 Les méthodes exactes

Les méthodes exactes utilisent surtout deux approches de résolution très connues: la programmation dynamique et les procédures par séparation et évaluation. Ces méthodes sont souvent utilisées pour résoudre les problèmes combinatoires de manière exacte, en ordonnancement tout particulièrement. Ce sont des méthodes d'énumération implicite:

L'énumération explicite construit toutes les solutions réalisables et retient une parmi les meilleures. L'énumération implicite consiste à explorer l'ensemble de toutes les solutions réalisables en éliminant des sous-ensembles de solutions moins intéressants sans avoir à les construire. Nous pouvons citer trois approches particulièrement célèbres : La programmation dynamique introduite par Bellman dans les années 50[10], la méthode par séparation et évaluation (Branch and Bound en anglais : notée B&B) et l'algorithme de retour arrière (Backtracking).

6.2 Les méthodes approchées

Une méthode approchée est une méthode d'optimisation qui a pour but de trouver une solution réalisable de la fonction objectif en un temps raisonnable, mais sans garantie d'optimalité. L'avantage principal de ces méthodes est qu'elles peuvent s'appliquer à n'importe quelle classe de problèmes, faciles ou très difficiles, D'un autre côté les algorithmes d'optimisation tels que les algorithmes de recuit simulé, les algorithmes tabous et les algorithmes génétiques ont démontré leurs robustesses et efficacités face à plusieurs problèmes d'optimisation combinatoires.

Les méthodes approchées englobent deux classes : les heuristiques et les métaeuristiques. La particularité qui différencie les méthodes métaheuristiques des méthodes heuristiques c'est que les métaheuristiques sont applicables sur de nombreux problèmes. Tandis que, les heuristiques sont spécifiques à un problème donné[10].

6.2.1 Les heuristiques

Une heuristique est une règle empirique simple basée sur l'expérience. C'est une technique de calcul approchée qui exploite d'une façon satisfaisant le problème à optimiser et qui fournit une solution admissible, non nécessairement exacte, dans un temps polynomial pour un problème d'optimisation difficile. En d'autres termes, une heuristique est un algorithme qui sacrifie en partie, la qualité de la solution au sens purement mathématique pour accélérer le processus de résolution. Elle est également une stratégie de bon sens pour se déplacerintelligemment dans l'espace des solutions, afin d'obtenir une solution approchée, la meilleure possible, dans un délai de temps raisonnable[5].

Elle est là, à la place d'une méthode exacte qui donne la solution dans un temps de résolution exponentiel. Elle est dédiée, généralement, à un problème bien particulier.

Les heuristiques sont dépendantes du problème à résoudre, principalement dans le choix du voisinage (donc dans le déplacement dans l'espace des solutions).

De nombreuses méthodes heuristiques ont été proposées dans la littérature pour résoudre les problèmes d'ordonnancement d'atelier. On distingue:

- **FIFO (First In First Out) :** la première tâche qui vient est la première tâche ordonnancée,
- **SPT (ShortestProcessing Time) :** la tâche ayant le temps opératoire le plus court est traité en premier lieu,
- **LPT (LongestProcessing Time) :** la tâche ayant le temps opératoire le plus important est ordonnancé en premier lieu,
- **EDD (Earliest Due Date) :** cet algorithme choisit parmi les tâches exécutables celle dont le délai est échu le plus tôt. Si aucune tâche n'est disponible, alors un temps libre est généré,
- **SRPT (ShortestRemainingProcessing Time) :** cette règle, servant à lancer la tâche ayant la plus courte durée de travail restant à exécuter, est très utilisée pour minimiser les encours et dans le cas des problèmes d'ordonnancement préemptifs,
- **ST (Slack Time) :** à chaque point de décision, l'opération ayant la plus petite marge temporelle est prioritaire. Faute de disponibilité des ressources de production, cette marge peut devenir négative.

6.2.2 Métaheuristiques

Les métaheuristiques d'optimisation sont des algorithmes généraux d'optimisation applicables à une grande variété de problèmes. Elles sont apparues à partir des années 80, dans le but de résoudre au mieux des problèmes d'optimisation. Les métaheuristiques s'efforcent de résoudre tout type de problème d'optimisation.

Elles sont caractérisées par leur caractère stochastique, ainsi que par leur origine discrète. Elles sont inspirées par des analogies avec la physique (recuit simulé, recuit micro canonique), avec la biologie (algorithmes évolutionnaires) ou encore l'éthologie (colonies de fourmis, essaims particulaires). Cependant, elles ont l'inconvénient d'avoir plusieurs paramètres à régler. Il est à souligner que les métaheuristiques se prêtent à toutes sortes d'extensions, notamment en optimisation mono-objectif et multi-objectif[3].

6.2.2.1 Le Recuit Simulé

Le Recuit Simulé (SimulatedAnnealing) est l'une des méthodes de voisinage les plus anciennes. Il a acquis son succès essentiellement grâce à des résultats pratiques obtenus sur de nombreux problèmes NP-difficiles[13].

Algorithme 1 Pseudo code de la méthode du Recuit Simulé.

```
Début
1 :  s ← s0                                          ▷ s0 est la solution initiale
2 :  T ← T0                                          ▷ T0 est la température initiale du système
3 :  Tant que T > Tf
4 :      Générer une solution s′ ∈ N(s) aléatoirement
5 :      Calculer Δf = f(s′) − f(s)
6 :      if Δf ≤ 0 then
7 :          s ← s′
8 :      else
9 :          Calculer prob(Δf, T) ← exp(−Δf/T)
10 :         Générer q uniformément dans l'intervalle [0, 1]
11 :         if q < prob(Δf, T) then
12 :             s ← s′
13 :         else
14 :             s′ est rejetée
15 :         end if
16 :     end if
17 :     T ← T × θ                                   ▷ 0 < θ < 1 coefficient de refroidissement
18 :  Fin Tant que
Fin
```

Algorithme 2.1 L'algorithme de RS.

6.2.2.2GRASP

Le GRASP (GreeclyRandomized Adaptive SearchProcedure) est une méta-heuristique qui consiste en une suite de solutions construites par une approche Vorace et à leur optimisation par l'exploration de leurs voisinages respectifs [26]. Chaque itération de cette méta-heuristique consiste en une phase de construction de solution et en une exploration de voisinage. La construction se fait élément par élément, le choix du prochain élément se faisant par une approche Vorace.essaie de tirer à la fois avantage de l'approche Vorace et de l'approche aléatoire. GRASP garde une trace de la meilleure solution et la retourne à la fin de l'algorithme [20].

```
s* = ∞
Tant que (Condition d'arrêt non satisfaite)
        Construire_Solution_Vorace(s)
        Trouver le minimum local s' dans N(s)
        si f(s')<f(s*)
                s*=s'
Retourner meilleure solution de s*
```

Algorithme 2.2 L'algorithme de GRASP.

6.2.2.3 Recherche avec Tabous

La recherche tabou (TS : TabuSearch) est une métaheuristique d'optimisation proposé par Glover en 1990 [6]. Son idée de base consiste à introduire la notion de mémoire dans la politique d'exploration de voisinage. Cette idée est inspirée de la mémoire humaine.
La recherche tabou est une procédure itérative qui, partant d'une solution initiale (cette solution peut être construite par une heuristique ou générée aléatoirement) tente de converger

vers la meilleur solution en exécutant à chaque itération un mouvement dans l'espace de recherche. Chaque itération consiste d'abord à engendrer un ensemble de solutions voisines de la solution courante pour ensuite en choisir la meilleure.

Algorithme 2 Pseudo code de la recherche tabou.

```
Début
1 :  s ← s0                                   ▷ s0 est la solution initiale
2 :  L = {}                                   ▷ la liste des tabous est vide initialement
3 :  Tant que la condition d'arrêt n'est pas vérifiée
4 :      Générer un ensemble N' ⊆ N(s) de solution voisines de s
5 :      Choisir la meilleur solution s' ∈ N' telle que s' ∉ L
6 :      Mettre à jour la liste L des solutions taboues
7 :      s ← s'
8 :  Fin Tant que
Fin
```

Algorithme 2.3 L'algorithme de RT.

7. Conclusion

L'ordonnancement est souvent décrite comme une fonction décisionnelle spécifique dans un système de gestion du travail concernant la production de biens, de travaux ou de services.

La majorité des problèmes d'ordonnancement sont NP-difficile, ça veut dire que, dans la pratique, la complexité croît exponentiellement avec le nombre de tâches et de ressources. Il n'est, donc, pas envisageable de résoudre de tels problèmes avec les méthodes exactes. C'est pour cela qu'il faut développer des nouveaux méthodes qui donnent des solutions certes sous-optimales, mais obtenues rapidement. dont l'objectif est de fournir des solutions aussi proches que possible de la solution exacte en un temps raisonnable.

CHAPITRE 2
LES ALGORITHMES GENETIQUES

1. Introduction

Dans ce chapitre on explique c'est quoi les algorithmes génétiques, et leur principes, en plus comment ils fonctionnent outre les opérateurs des algorithmes génétiques, et enfin introduisons leur parallélisme et leurs modèles.

2. Définition

Les algorithmes génétiques (**AG**s) sont des algorithmes d'optimisation stochastique fondés sur les mécanismes de la sélection naturelle et de la génétique. Leur fonctionnement est extrêmement simple. On part avec une population de solutions potentielles (*chromosomes*) initiales arbitrairement choisies. On évalue leur performance (*fitness*) relative. Sur la base de ces performance on crée une nouvelle population de solutions potentielles en utilisant des opérateurs évolutionnaires simples : la sélection, le croisement et la mutation. On recommence ce cycle jusqu'à ce que l'on trouve une solution satisfaisante.

Les AGs ont été initialement développés par John Holland (1975).[21]

3. Principes des algorithmes génétiques

Les algorithmes génétiques utilisent la théorie de Darwin sur l'évolution des espèces. Elle repose sur trois principes : le principe de variation, le principe d'adaptation et le principe d'hérédité. Les algorithmes génétiques utilisent la théorie de Darwin sur l'évolution des espèces.

Elle repose sur trois principes : le principe de variation, le principe d'adaptation et le principe d'hérédité[27].

3.1 Principe de variation

Chaque individu au sein d'une population est unique. Ces différences, plus ou moins importantes, vont être décisives dans le processus de sélection.

Figure 2.1 représentation du principe de variation.

3.2 Principe d'adaptation

Les individus les plus adaptés à leur environnement atteignent plus facilement l'âge adulte. Ceux ayant une meilleure capacité de survie pourront donc se reproduire davantage.

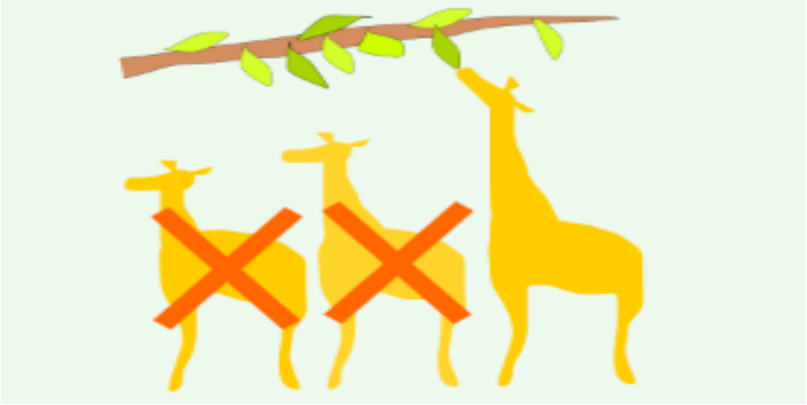


Figure 2.2représentation du principe d'adaptation.

3.3 Principe d'hérédité

Le principe d'hérédité : Les caractéristiques des individus doivent être héréditaires pour pouvoir être transmises à leur descendance. Ce mécanisme permettra de faire évoluer l'espèce pour partager les caractéristiques avantageuse à sa survie[27].

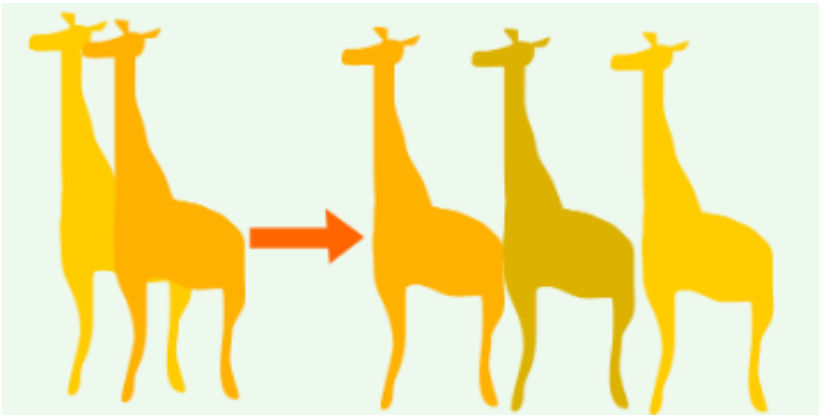


Figure 2.3représentation Principe d'hérédité.

4. Les opérateurs d'algorithmes génétiques

Il y a trois opérateurs d'évolution dans les algorithmes génétiques :

4.1 L'opérateur de sélection

La sélection consiste à choisir les individus les mieux adaptés afin d'avoir une population de solution la plus proche de converger vers l'optimum global. Cet opérateur est l'application du principe d'adaptation de la théorie de Darwin[27].

Il existe plusieurs techniques de sélection. Voici les principales utilisées :

- **Sélection par rang :** Cette technique de sélection choisit toujours les individus possédant les meilleurs scores d'adaptation.
- **Probabilité de sélection proportionnelle à l'adaptation :** Technique de la roulette ou roue de la fortune, pour chaque individu, la probabilité d'être sélectionné est proportionnelle à son adaptation au problème.
- **Sélection par tournoi :** Cette technique utilise la sélection proportionnelle sur des paires d'individus, puis choisit parmi ces paires l'individu qui a le meilleur score d'adaptation.
- **Sélection uniforme :** La sélection se fait aléatoirement, uniformément et sans intervention de la valeur d'adaptation.

Voici un exemple avec des individus en représentation binaire une fois la sélection effectuée :

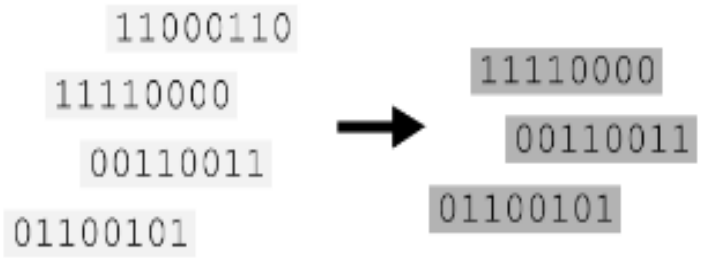


Figure 2.4 Individus en représentation binaire une fois la sélection effectuée.

4.2 L'opérateur de croisement

Le croisement, ou enjambement, crossing-over, est le résultat obtenue lorsque deux chromosomes partage leurs particularités.

Celui-ci permet le brassage génétique de la population et l'application du ***principe d'hérédité*** *de* la théorie de Darwin[27].

Il existe deux méthodes de croissement : simple ou double enjambement.

- **Le simple enjambement** consiste à fusionner les particularités de deux individus à partir d'un pivot, afin d'obtenir un ou deux enfants :

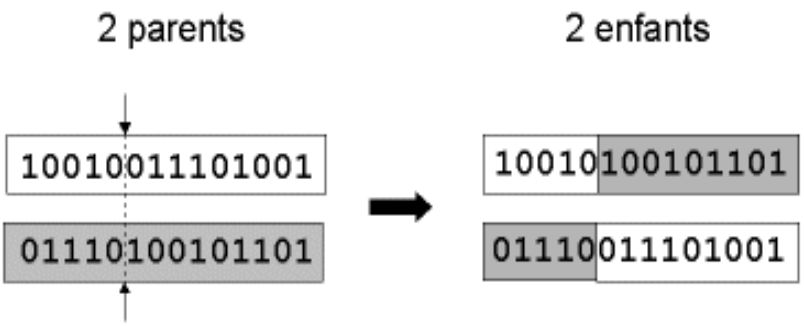


Figure 2.5 croisement sur un seul point.

- **Le double enjambement** repose sur le même principe, sauf qu'il y a deux pivots :

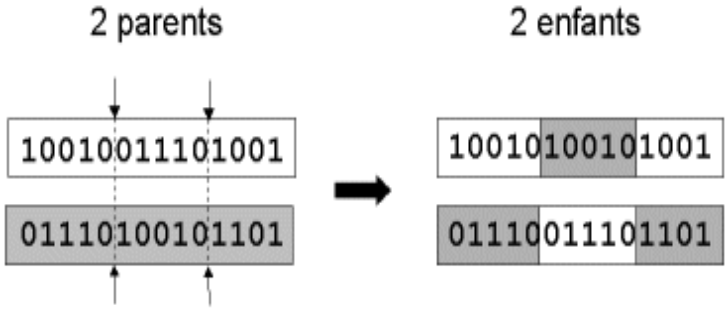


Figure 2.6 croisement sur deux points.

4.3 L'opérateur de mutation

La mutation consiste à altérer un gène dans un chromosome selon un facteur de mutation. Ce facteur est la probabilité qu'une mutation soit effectuée sur un individu.
Cet opérateur est l'application du principe de variation de la théorie de Darwin et permet, par la même occasion, d'éviter une convergence prématurée de l'algorithme vers un extremum local[27].

Voici un exemple de mutation sur un individu ayant un seul chromosome :

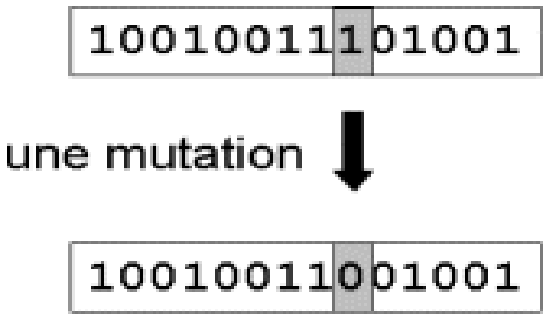


Figure 2.7 représentation d'opérateur de la mutation

Les principes de bases étant expliqués, voici le fonctionnement des algorithmes génétiques :

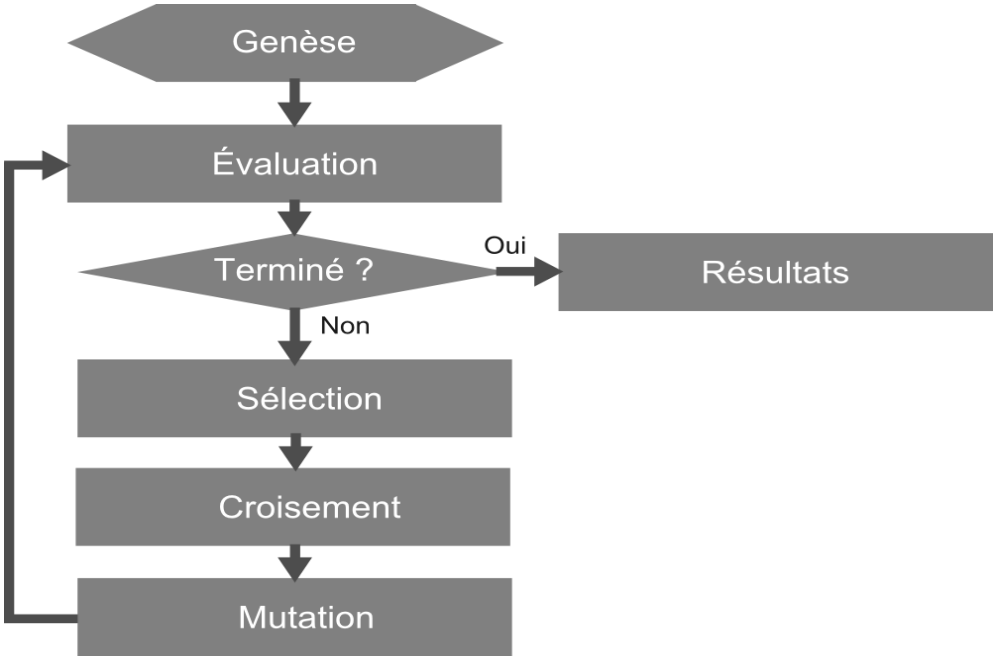


Figure 2 .8 le fonctionnement général des algorithmes génétiques.

Quelques explications :

- La genèse est l'étape de la création d'une population aléatoire. C'est le point de départ de notre algorithme
- L'évaluation est l'analyse des individus pour analyser si une solution est disponible. Pour ceci, nous utilisons une fonction de coût, ou d'erreur, afin de définir le score d'adaptation des individus lors du processus de sélection.
- Nous effectuons une boucle tant que l'évaluation estime que la solution n'est pas optimale[27].

5. Le parallélisme

Le parallélisme est une architecture pour traiter l'information de manière simultanée, c'est-à-dire on peut faire plusieurs operateurs au même temps, pour gagner le temps.

5.1 Les modèles des algorithmes génétiques parallèles(AGPs)

5.1.1 AGP maître-esclave

On suppose qu'il n'y a qu'une seule population panmictique, c'est-à-dire une AG canonique. Cependant, contrairement à l'AG canonique, l'évaluation des individus sont distribuées en ordonnant des fractions de la population parmi les nœuds esclaves de traitement. Un tel modèle présente l'avantage d'être facile à mettre en œuvre et ne modifie pas le comportement de recherche d'un AG canonique[4].

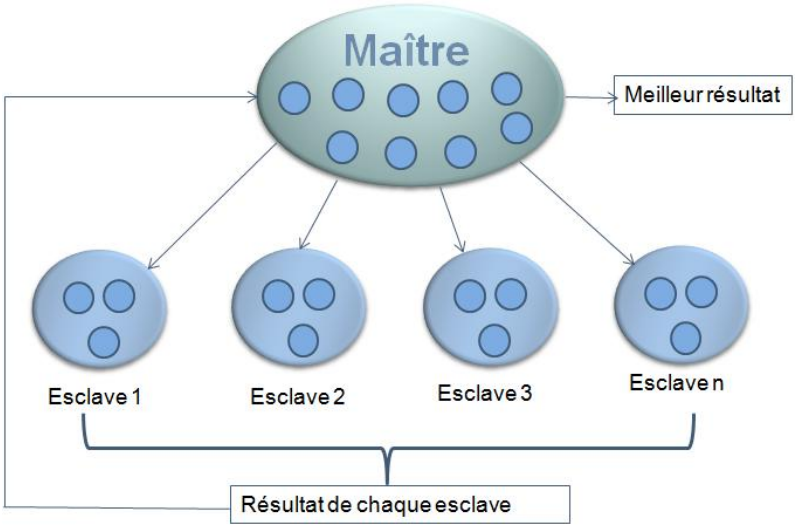
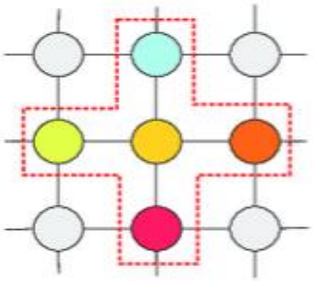


Figure 2 .9 Le modèle AGP maître-esclave

5.1.2 AGP à grains fins

L'AGP à grains fins ne se compose d'une seule population, qui est structurée spatialement. Il est conçu pour fonctionner sur un système de traitement massivement parallèle étroitement lié, c'est-à-dire un système informatique composé d'un grand nombre d'éléments de traitement et connecté dans une topologie à grande vitesse spécifique. Par exemple, la population d'individus dans un AGP grain fin peut être organisée comme une grille bidimensionnelle. Par conséquent, la sélection et l'accouplement dans un AG parallèle à grains fins sont limités à de petits groupes. Néanmoins, les groupes se chevauchent pour permettre certaines interactions entre tous les individus afin que les bonnes solutions puissent se diffuser à travers les populations entières. Parfois, AG parallèle à grain fin est également appelée modèle AGP cellulaire[4].



(c)

Figure 2 .10 Le modèle AGP à grains fins

5.1.3 La AGP à populations multiples

La AGP à populations multiples (ou dèmes) peut être plus sophistiquée, car elle se compose de plusieurs sous-populations qui échangent des individus à l'occasion. Cet échange d'individus est appelé migration et il est contrôlé par plusieurs paramètres. Les AGP à population multiple sont également connus sous divers noms. Puisqu'ils ressemblent au

«modèle insulaire» en génétique des populations qui considère les dèmes relativement isolés, il est aussi souvent connu sous le nom de modèle insulaire AGP[4].

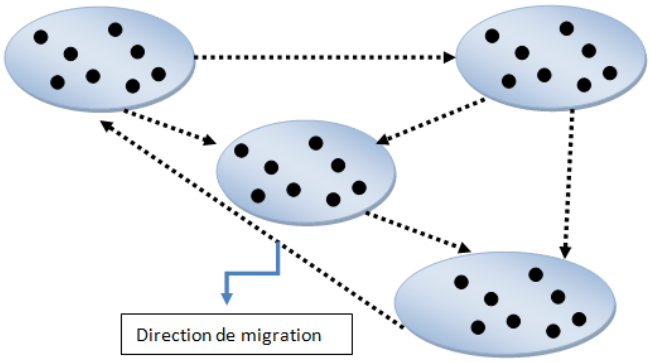


Figure 2 .11 Le modèle AGP à populations.

6. Conclusion

Nous avons présente dans ce chapitre les concepts des algorithmes génétiques et ses principes (principe de mutation, principe d'adaptation, principe génétique) Après ;avoir compris les opérateurs de l'algorithme génétique (sélection, croisement , mutation), Ensuite, nous étudions l'algorithme génétique parallèle et leurs modèles.

CHAPITRE 3
L’ETAT DE L’ART

1. Introduction

Dans ce chapitre, on représente des articles des chercheures sur notre sujet de thèse, et résumons donc chaque article comme suit :

- **(1993) Hsiao-Lan Fang, Peter Ross et Dave Corne** décrivent une approche AG qui produit des résultats raisonnablement bons très rapidement sur les problèmes job-shop Scheduling, mieux que les efforts précédents utilisant des algorithmes génétiques pour cette tâche, et comparable aux méthodes de recherche conventionnelles existantes. La représentation utilisée est une variante de celle connue pour fonctionner assez bien pour le problème du voyageur de commerce. Il a le mérite considérable que le croisement produira toujours des horaires légaux. Une nouvelle méthode d'amélioration des performances est examinée sur la base d'un échantillonnage dynamique des taux de convergence dans différentes parties du génome. Leur approche promet également de résoudre efficacement le problème d'Open Shop Scheduling et le problème dejob-shop Scheduling.

- **(1997) Shyh-Chang Lin, Erik D. Goodman et William F. Punch** décriassent une AG pour les problèmes de Job Shop Scheduling. En utilisant l'algorithme de Giffler et Thompson, ils avent créés deux nouveaux opérateurs, le croisement THX et la mutation, qui transmettent mieux les relations temporelles dans le planning. L'approche a produit d'excellents résultats sur les problèmes de job shop Scheduling. Ils avent testé de nombreux modèles et échelles d'AG parallèles dans le contexte de problèmes de job shop scheduling. Le modèle hybride composé d'Algorithme génétique Coarse-grain à connectés dans une topologie de style AG fine-grain a donné les meilleurs résultats, semblant intégrer avec succès les avantages des AG coarse-grain et fine-grain.

- **(2003) Michelle Moore** présente algorithmes génétiques parallèles qui sont appliquées au problème NP-complet de planification de plusieurs tâches sur un cluster d'ordinateurs connectés par un bus partagé. Les expériences révèlent que l'algorithme de programmation parallèle développe des programmes très précis lorsque les directives de paramètres sont utilisées.

- **(2004) Murat Yildizoglu et Thomas Vallée** présentent les mécanismes de base de ces algorithmes et un panorama de leurs applications en économie, accompagnés d'une bibliographie représentative.

- **(2004)Souquet Amédée et RadetFrancois-Gérard** vous travaux consisterez à définir ces algorithmes évolutionnaires, et à les comparer aux méthodes déterministes afin de pouvoir cerner leur utilité en fonction du problème posé. Le devoir s'articulera sur deux axes principaux qui serviront de fils conducteurs au passage de la théorie à la pratique.
  Dans un premier temps, attachèrent à l'origine des algorithmes génétiques, leur appartenance dans le monde de la vie artificielle, et à la théorie de l'évolution des espèces formulée par le naturaliste Charles Darwin, sur laquelle ces premiers sont basés.
  Montrerent quels en sont les principes, de telle manière à ce que puissent poursuivre vers notre premier axe, qui se penchera sur les différentes méthodes de sélection et introduira les différents operateurs, qui soulignent la nécessité de diversité.
  Sur ces bases, proposèrent, dans dernière partie, de souligner l'aspect pragmatique de ces algorithmes en illustrant leur utilisation et leur domaine d'application dans les sciences contemporaines.

- **(2004)Nourah Al-Angari, AbdullatifALAbdullatif**Application réussie de l'algorithme génétique parallèle pour résoudre le problème de planification des tâches. L'évaluation de « fitness » est l'opération la plus gourmande en CPU et affecte les performances de l'AG. Dans le cas de problèmes complexes et de haute génération, l'algorithme maître-esclave synchrone proposé surpasse l'algorithme séquentiel.

- **(2007) Kheireddine MERHOUM et Messaoud DJEGHABA** développent une application qui permet de minimiser le makespan pour un problème d'ordonnancement job-shop flexible ils utilisèrent les Algorithme génétique. Cependant le problème d'ordonnancement job-shop flexible dans la littérature est considéré comme une problématique difficile à résoudre dans le domaine de l'optimisation combinatoire, sa complexité est de type NP-complet au sens fort. L'objectif est montré les performances des algorithmes génétiques (métaheuristique) dans la résolution de ce genre de problème.

- **(2006) R.Neduncheliane, K.Koushik, N.Meiyappan, V.Raghu**développent Un algorithme génétique pour planifier dynamiquement des tâches hétérogènes à des processeurs hétérogènes dans un environnement distribué. Le problème de planification est connu pour être NP-complet. Les algorithmes génétiques, une technique de recherche méta-heuristique, ont été utilisés avec succès dans ce domaine. L'algorithme proposé utilise plusieurs processeurs avec un contrôle centralisé pour la planification. Les tâches sont prises en lots et sont planifiées pour minimiser le temps d'exécution et équilibrer les charges des processeurs. Selon leurs résultats expérimentaux, l'algorithme génétique parallèle proposé (PPGA) diminue considérablement le temps de programmation sans affecter négativement Makespan des programmes résultants.

- **(2013) FrankWerner**donne un aperçu de certains algorithmes génétiques pour les problèmes de Shop Scheduling. Dans un problème de shop Scheduling, un ensemble de Jobs doit être traité sur un ensemble de machines de telle sorte qu'un critère d'optimisation spécifique soit satisfait. En fonction des restrictions sur les itinéraires technologiques des emplois, on distingue Flow Shop (chaque Job est caractérisé par le même itinéraire technologique), Job Shop (chaque Job a un itinéraire spécifique) et Open Shop (aucun itinéraire technologique n'est imposé sur les Jobs). Il considère également certaines extensions des problèmes de Shop Scheduling tels que Shop hybrides ou flexibles (à chaque étape de traitement, nous pouvons avoir un ensemble de machines parallèles) ou l'inclusion de contraintes de traitement supplémentaires telles que les temps de traitement contrôlables, les temps de publication, les temps de configuration ou le condition sans attente. Après avoir donne une introduction aux algorithmes génétiques de base discutant des représentations de solutions brèves, de la génération de la population initiale, des principes de sélection, de l'application d'opérateurs génétiques tels que le croisement et la mutation, et des critères de terminaison, il discute de plusieurs algorithmes génétiques pour les types de problèmes particuliers en mettant l'accent sur leur caractéristiques et différences communes. Ici, il concentre principalement sur les problèmes à critère unique (minimisation de la durée de validité ou d'un critère de somme particulier tel que le temps total d'achèvement ou le retard total), mais mentionne brièvement quelques travaux sur les problèmes multicritères. Il discute de certains résultats de calcul et les comparons avec ceux obtenue par d'autres heuristiques. En outre, il résume également la génération d'instances de référence pour les problèmes de durée de fabrication et donne une brève introduction à l'utilisation du package de programmes « LiSA-ALibraryof SchedulingAlgorithms » développe à « Otto-von-Guericke-

University Magdeburg » pour résoudre les problèmes de Shop Scheduling ,qui comprend également un algorithmegénétique.

- **(2016)Rakesh Kumar PHANDEN** représente Job Shop Scheduling qui est un problème important et complexe pour un système de fabrication. C'est un problème bien connu et populaire ayant une caractéristique NP-hard (non polynomiale) de trouver rapidement la solution optimale ou quasi optimale (horaires). Dans Job Shop Scheduling, un ensemble de nombres "N" de Jobs est traité par le biais d'un nombre "M" d'un ensemble donné de machines. Il doit être traité dans l'ordre prescrit en utilisant la séquence d'opérations réalisable pour un travail. Par conséquent, en raison de sa nature complexe, la recherche de solutions approximatives est choisie plutôt que la recherche de la solution exacte qui implique un coût plus élevé. Diverses techniques métaheuristiques sont utilisées afin de trouver la solution sous-optimale pour le problème de planification de l'atelier de travail. L'algorithme génétique et la méthode de recherche de voisinage variable (VNS) sont les techniques préférées qui sont mieux connues pour la recherche globale et locale de solutions, respectivement. VNS fonctionne comme pour augmenter l'approche GA. Dans le présent travail, les multi-agents sont proposés pour trouver la solution presque optimale pour le problème de planification de l'atelier en utilisant l'approche GA et VNS en parallèle. Le système multi-agents est préféré en raison de sa capacité à fonctionner en parallèle et de sa robustesse ainsi que de l'élucidation de l'intelligence. Dans le système proposé, de nombreux hôtes du réseau sont hébergés par des agents. JADE est utilisée pour configurer les communications. Chaque agent est conçu pour effectuer la tâche spécifique à savoir l'agent d'initialisation (IA), l'agent de traitement (PA) et l'agent de coordination (CA) pour la génération initiale de la population, pour planifier les opérations sur les machines, pour trouver l'hôte distinctif et pour effectuer les migrations entre différentes populations respectivement. L'objectif est de trouver une valeur optimale de makespan pour le problème de planification de l'atelier de travail. La performance du système est évaluée par une étude de cas et elle révèle que l'approche proposée est suffisamment efficace pour trouver la solution optimale. Les travaux futurs consistent à introduire des agents de perturbation (DA) pour les perturbations internes et externes.

- **(2017)CHERGUIAbderrahman DAHMANI Abd Naceur et ADDA ABBOU Abdellah** intéressent à l'adaptation et l'hybridation de l'algorithme de colonies d'abeilles (BCO) pour la résolution de problème d'ordonnancement Flow Shop. L'idée consiste à insérer une méthode de recherche locale au niveau de l'algorithme de base pour améliorer l'intensification et la diversification de la technique de recherche de l'algorithme.
  Les résultats de simulation montrent que l'algorithme BCO hybride est plus performant que le BCO de base pour différentes classes de systèmes proposées.

- **(2018) J.Adan,A.Akcay, J.Stokkermansb et R.VandenDobbelsteen**traitent Un algorithme génétique hybride pour améliorer le processus de planification, dont les principales caractéristiques sont un mécanisme de croisement amélioré pour la recherche locale, deux procédures de recherche locale rapide supplémentaires et une fonction d'ajustement multi-objectif contrôlée par l'utilisateur. Les tests avec des données de production réelles montrent que cette approche multi-objectifs peut atteindre l'équilibre souhaité entre le temps de production, le temps de configuration et les retards, produisant des calendriers de production de haute qualité pratiquement réalisables

- **(2019)Yuri N. Sotskov, Natalja M. Matsveichuk et Vadzim D. Hatsura**étudient les problèmes de Shop Scheduling à deux machines à condition que les limites inférieures et supérieures des durées de ‘ n’ Jobs soient données avant la planification. Une valeur exacte de la durée du travail reste inconnue jusqu'à la fin du Job. L'objectif est de minimiser le makespan (durée du planning). Ils abordent la question de la meilleure façon d'exécuter un calendrier si la durée du travail peut prendre une valeur réelle du segment donné. Les décisions de programmation peuvent comprendre deux phases: une phase hors ligne et une phase en ligne. En utilisant des informations sur les limites inférieures et supérieures pour chaque durée de Job disponible lors de la phase hors ligne, un planificateur peut déterminer un ensemble dominant minimal de plannings (DS) sur la base de conditions suffisantes pour la domination du programme. Le DS couvre de manière optimale toutes les réalisations (scénarios) possibles des durées de Job incertaines dans le sens où, pour chaque scénario possible, il existe au moins un calendrier dans le DS qui est optimal. Le DS permet à un planificateur de prendre rapidement une décision de planification en ligne chaque fois que des informations supplémentaires sur l'achèvement des travaux sont disponibles. Un planificateur peut choisir un calendrier optimal pour les scénarios les plus possibles. Nous avons développé des algorithmes pour tester un ensemble de conditions pour une domination d'horaire. Ces algorithmes sont polynomiaux dans le nombre de travaux. Leur complexité temporelle ne dépasse pas. Des expériences informatiques ont montré l'efficacité des algorithmes développés. S'il n'y avait pas plus de 600 tâches, les 1000 instances de chaque série testée ont été résolues en une seconde au plus. Un cas avec 10 000 tâches a été résolu en 0,4 s en moyenne. La plupart des instances de neuf classes testées ont été résolues de manière optimale. Si l'erreur relative maximale de la durée du travail n'était pas supérieure à, alors plus que les instances testées ont été résolues de manière optimale. Si l'erreur relative maximale était égale à, alors les instances testées des neuf classes ont été résolues de manière optimale.

- **(2019) KRAM Zakaria**Le problème traité dans ce mémoire concerne l’insertion d’un novelle opération dans l’ordonnancement à machines parallèles en vue de minimiser le makespan. Les opérations peuvent être exécutées en parallèle sur plusieurs machines. Une méthode a été suggérée basée sur les algorithmes génétiques, et l’algorithme glouton ont été, ensuite, proposée pour la résolution de problèmes.

- **(2020) DAHMANE Lamia et HADROUG Selma** consistaient à appliquer les algorithmes génétiques pour résoudre le problème d’ordonnancement de job shop qui est un problème extrêmement complexe. Il est classé parmi les problèmes combinatoires.

- **(2020) YAHIAOUI Khadidja et BOUSBA Achwak**Le travail exposé dans ce mémoire s'intéresse au problème d'ordonnancement d'un Flow Shop à deux machines avec des temps de latence. L’objectif est de trouver une séquence appropriée de tâches en fonction des temps de latence, de manière à minimiser le Makespan (temps d’exécution maximal). Plusieurs méthodes peuvent être utilisées pour résoudre ce problème. En effet, pouvent trouver des méthodes exactes et des méthodes approchées. Et c'est dans cette optique que

CHAPITRE 4
ALGORITHME GENETIQUE PARALLELE D'ORDONNANCEMENT FLOW SHOP

1. Introduction

Dans ce chapitre, on se propose de conception un algorithme génétique parallèle pour la résolution de problème d'ordonnancement de flow shop.

2. Problématique

Etant n tâche et m jobs. Chaque job contient exactement n opérations. La ième opération du job doit être exécuté sur la i-ème machine. Aucune machine ne peut effectuer plus d'une opération simultanément. Pour chaque opération de chaque job, le temps d'exécution est spécifié. Les opérations d'un job doivent être effectuées dans le ordre. La première opération est exécutée sur la première machine, puis (lorsque la première opération est terminée) la seconde opération sur la deuxième machine, et ainsi de suite jusqu'à la n-ième opération. Cependant, les tâches peuvent être exécutées dans n'importe quel ordre. La définition du problème implique que cet ordre de job est exactement le même pour chaque machine. Le problème est de déterminer l'arrangement optimal, c'est-à-dire celui dont l'exécution totale des tâches est la plus courte possible.

leursfonctions objectifs sont généralement :

- Le Makespan Cmax : date de fin de la dernière tâche sur la machine M, Tell que $Cmax=max\{C_{ij}\}1\leq n,1\leq j\leq m.$

Exemple:

On considérons un exemple d'un Flow-Shop à 3 jobs et 5 machines.

Machines \ Jobs	M1	M2	M3
J1	**3**	**2**	**1**
J5	**1**	**1**	**2**
J4	**4**	**3**	**3**
J2	**2**	**1**	**2**
J3	**4**	**2**	**2**

Tableau4.1.exemple d'un flow shop à j=3 et m=5.

On représente le diagramme de Gantt pour la permutation de jobs $J1 \longrightarrow J5 \longrightarrow J4 \longrightarrow J2 \longrightarrow J$ obtenue avec le Flow-Shop de permutation comme suivant:

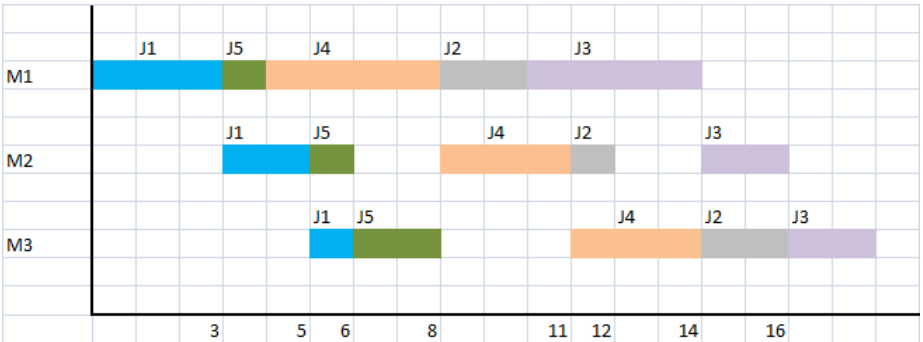


Figure4.1 Diagramme de Gantt de flow shop.

3. Formulation mathématique

Les paramètres ainsi que les indices utilisés pour la formulation mathématique du problème d'ordonnancement de type Flow-Shop sans contraintes de ressources non-renouvelables sont les suivants:

- *n*: nombre de jobs.
- *m*: nombre de machines.
- *i*: indice de job, où *i 2 f*1, ...,*ng*.
- *j*: indice de machine, où *j 2 f*1, ...,*mg*.
- *k*: indice de position, où *k 2 f*1, ...,*ng*.
- *t*: Indice d'arrivée de ressource, où *t ϵ { u*1*j* , ...,*uq j}*.
- *u*1*j* : date de la première livraison arrivée à la machine *Mj*.
- *uq j* : date de la dernière livraison arrivée à la machine *Mj*.
- *pi j* : temps du traitement du job *Ji* sur la machine *Mj*.
- *ai j* : quantité de ressource que le job *Ji* consomme sur la machine *Mj*.
- *bt j* : nombre total de la ressource dédiée arrivée à la machine *Mj*à l'instant *t* .
- *q*: nombre de dates de livraison.
- *M* : Constante suffisamment grande.

4. Complexité

Nous étudions dans cette section la complexité du problème considéré:

Fm/ /Cmax est NP-difficile.

- **Preuve 1** Soit p le problème d'ordonnancement Flow-Shop de permutation sous contraintes de ressources non-renouvelables avec minimisation de makespan. Construisant le problème p′ comme étant un problème d'ordonnancement Flow-Shop classique (sans contraintes). Clairement, le problème p′ est un cas particulier de problème p (puisque le besoin en ressources peut être limité à 0 dans le problème p). En effet, le problème p′ est connu pour être NP-difficile au sens fort pour m > 2[18]. Il en résulte que le problème p est NP-difficile au sens fort.

5. Approche utilisée

5.1 Les Algorithmes Génétique

5.1.1 Générer la solution initiale

Plusieurs méthodes existent pour définir le mécanisme de la génération de la population initiale qui représente le point de départ pour la constitution des générations futures : le tirage aléatoire, les heuristiques ou une combinaisons de solution heuristiques et aléatoire.

Machine	Opération(job)		
M1	O_j	…	…

M2	$O_{j'}$	…	…
…	…	…	…

Tableau4.2 Représentation du codage d'un P_i.

Une solution ou Pi qui compose cette population est codée par une matrice de séquence d'opérations qui définie l'ordre des opérations que doit effectuer chaque machine comme le montre la table4. Les solutions constituants la population initiale sont choisies aléatoirement dans cette étude[14].

- **Exemple:**

  On a un exemple de Flow Shop aves 3 jobs et 3 machines comme suite:

Job	Machine(le temps de traitement)		
J1	1(19)	2(38)	3(30)
J2	1(19)	3(10)	2(19)
J3	2(13)	1(25)	3(9)

Tableau4.3exemple de flow shop.

On représente le problème aves la matrice de séquence des Jobs $\{T_{ik}\}$ et la matrice de temps de traitement $\{P_{ik}\}$ comme suite :

$$Tik = \begin{vmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \end{vmatrix} \qquad Pik = \begin{vmatrix} 19 & 38 & 30 \\ 19 & 10 & 19 \\ 13 & 25 & 9 \end{vmatrix}$$

On code les opérations du problème précédent comme suite:

Machine	Opération(job)		
M1	10	11	12
M2	13	14	15
M3	16	17	18

Table 4.4 les opérations de l'exemple.

a partir la matrice $\{Tjk\}$ les opérations $\{10,12,17\}$ sont traiter en $M1$, alors $M2$ traite les opérations $\{11,15,16\}$ et $M3$ traite $\{12,14,18\}$.

Machine	Opération(job)		
M1	10(J1)	11(J2)	12(J3)

M2	13(J3)	14(J2)	15(J1)
M3	16(J2)	17(J1)	18(J3)

Table 4.5 La solution initiale de l'exemple.

On proposons la matrice de solution initiale S_{jk} :

$$S_{jk} = \begin{vmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 2 & 1 & 3 \end{vmatrix}$$

Nous allons créer plusieurs matrices Sjk basées sur le changement d'encodage des opérations et sélectionner aléatoirement la population initiale.

L'algorithme de génération de population initiale comme suite :

```
#region Initial Pop

1 référence
void CreateInitialPopulation()
{
    for (int i = 0; i < population.Length; i++)
    {
        Schedule ciz = createNewSchedule();
        population[i] = ciz;
        produced++;

        Progress = double.Parse((produced * 100d / 10000000).ToString("0.00"));
        if (stopped)
            break;
    }
}
#endregion
```

Algorithme 4.1 L'algorithme de génération de population initiale.

5.1.2La fonction objective « fitness »

Fitness = M - $Cmax$,tell que $Cmax$est Makespane et M est un paramètre pour change le problème minimal a maximale[4].

5.1.3 La sélection

La phase de sélection consiste à choisir parmi les N individus i de la population courante les plus forts individus à partir desquels la génération suivante sera créée. Soit pour un ordonnancement i réalisable, on calcul la valeur de la fonction objectif fi « Makespan » pour notre problème c'est évaluer le plus long chemin sur le graphe disjonctif. On associe à chaque solution une fonction d'évaluation pour calculer sa force d'adaptation Fi, dans notre cas le problème à pour but de minimiser la fonction objectif min f (i), alors la force de pi (fitness) peut être exprimer tout simplement par *Fi = 1/fi*, Les individu ainsi sélectionnés constituent une population intermédiaires[14].

L'algorithme de génération de sélection comme suite :

```
#region DoSelection

1 référence
int[] doSelection()
{
    int[] nums = null;
    switch (selectionType)
    {
        case SelectionTypes.Tournament:
            nums = tourSelection(groupSize);
            break;
        case SelectionTypes.RouletteWheel:
            nums = rouletWheelSelection(groupSize);
            break;
    }
    return nums;
}

#endregion
```

Algorithme 4.2 L'algorithme de génération de sélection.

Dans cette étude nous avons utilisé la méthode de sélection par Rang (Rankine) de classement : on affecte à chaque individu un rang *Ri* qui dépend de son fitness, les individus d'une population sont ensuite classés dans une liste selon l'ordre croissant de leur fitness, on associe pour chaque individu i une probabilité pi d'être sélectionné, la sélection sera proportionnelle à leur rang dans la liste de la population :

$$p_i = R_i / \Sigma R_i$$

Les algorithmes utilisés des ces méthodes comme suivant:

5.1.3.1 Sélection par tournoi

L'algorithme de Sélection par tournoiest:

```
int[] tourSelection(int groupSize)
{
    int[] nums = new int[groupSize];

    for (int i = 0; i < nums.Length; i++)
    {
        nums[i] = rnd.Next(population.Length - 1);
    }
    sortArray(nums);
    return nums;
}
2 références
void sortArray(int[] nums)
{
    bool sorted = true;
    while (sorted)
    {
        sorted = false;

        for (int i = 0; i < nums.Length - 1; i++)
        {
            if (population[nums[i]].FitValue > population[nums[i + 1]].FitValue)
            {
                int temp = nums[i];
                nums[i] = nums[i + 1];
                nums[i + 1] = temp;
                sorted = true;
            }
        }
    }
}
```

Algorithme 4.3 L'algorithme de Sélection par tournoi.

5.1.3.2 Sélection de Roulette

```
int[] rouletWheelSelection(int groupSize)
{
    int[] nums = new int[groupSize];
    for (int i = 0; i < nums.Length; i++)
    {
        nums[i] = rnd.Next(population.Length - 1);
    }
    sortArray(nums);
    double[] reciprocal = new double[groupSize];
    for (int i = 0; i < reciprocal.Length; i++)
    {
        reciprocal[i] = 1 / population[nums[i]].FitValue;
    }
    for (int j = 0; j < 2; j++)
    {
        double value = rnd.NextDouble();

        for (int i = 0; i < reciprocal.Length; i++)
        {
            value -= reciprocal[i];
            if (value <= 0)
            {
                nums[j] = i;
                break;
            }
        }
    }

    return nums;
}
```

Algorithme 4.4 L'algorithme de sélection de roulette.

5.1.4 Croissement

L'opérateur de croisement est le plus important dans les AGs, il permet d'explorer efficacement l'espace de recherche. L'opérateur de croisement appliquer sur deux individu parent1 et parent2 choisis parmi la population sélectionnée, permet de générer deux nouvelles solutions enfant1 et enfant2 par combinaison des propriétés des parents 1 et 2[14].
La méthode implémenté dans cette étude consiste à choisir aléatoirement une machine k et on applique un croisement entre la séquence des opérations associé à cette machine seulement, le séquencement des opérations sur les autres machines reste identique sur les fils, puis on teste la réalisabilité des solutions obtenu, ceux qui ne comporte pas de cycle seront acceptées [14].

M1	$O_{2,2}$	$O_{1,1}$	$O_{3,3}$
M2	$O_{3,1}$	$O_{2,3}$	$O_{1,2}$
M3	$O_{1,3}$	$O_{2,1}$	$O_{3,2}$

M1	$O_{1,1}$	$O_{2,2}$	$O_{3,3}$
M2	$O_{2,3}$	$O_{1,2}$	$O_{3,1}$
M3	$O_{2,1}$	$O_{1,3}$	$O_{3,2}$

Table4.6 Représente les des parents indiv1, indiv2 sélectionner pour le croisement sur la machine 2 [10].

M1	$O_{2,2}$	$O_{1,1}$	$O_{3,3}$
M2	$O_{3,1}$	$O_{2,3}$	$O_{1,2}$
M3	$O_{1,3}$	$O_{2,1}$	$O_{3,2}$

M1	$O_{1,1}$	$O_{2,2}$	$O_{3,3}$
M2	$O_{2,3}$	$O_{1,2}$	$O_{3,1}$
M3	$O_{2,1}$	$O_{1,3}$	$O_{3,2}$

Tableau 4.7 Les enfants *enf1 et enf2* obtenu après croisement[10].

- **Exemple:**

On fit le croisement avec l'utilisation Algorithme GT en trois parents $P1$,$P2$,$P3$:

Parent 1 :

Machine	Opération(job)		
M1	10(J1)	11(J2)	12(J3)
M2	13(J3)	15(J1)	14(J2)
M3	16(J2)	18(J3)	17(J1)

Parent 3:

Machine	Opération(job)		
M1	10(J1)	11(J2)	12(J3)
M2	13(J3)	15(J1)	14(J2)
M3	17(J1)	16(J2)	18(J3)

Parent 2:

Machine	Opération(job)		
M1	10(J1)	12(J3)	11(J2)
M2	13(J3)	15(J1)	14(J2)
M3	18(J3)	17(J1)	16(J2)

Table 4.8 le croisement de l'exemple précédent.

Enfant :

Machine	Opération(job)		
M1	10(J1)	11(J2)	12(J3)
M2	13(J3)	15(J1)	14(J2)
M3	16(J2)	18(J3)	17(J1)

Table 4.9 Le résultat de croisement.

L'algorithme de génération de croisement comme suite :

```
#region DoCrossover

2 références
Schedule doCrossover(Schedule mother, Schedule father)
{
    COTypes temp = crossOver;
    Schedule child = null;
    switch (temp)
    {
        case COTypes.SinglePoint:
            child = SinglePointCO(mother, father);
            break;
        case COTypes.TwoPoint:
            child = TwoPointCO(mother, father);
            break;
        case COTypes.Uniform:
            child = UniformCO(mother, father);
            child.Repair();
            break;
        case COTypes.OrderedX:
            child = OrderedCO(mother, father);
            break;
    }
    return child;
}
#endregion
```

Algorithme 4.5 L'algorithme de génération de croisement.

les méthode des croisements sont:

5.1.4.1 Croisement Uniform

```
#region Uniform CO

1 référence
Schedule UniformCO(Schedule mother, Schedule father)
{
    int N = krlength;
    int[] macs = mother.MacSchedule.Clone() as int[];
    int[] jobs = mother.JobSchedule.Clone() as int[];
    for (int i = 0; i < N; i++)
    {
        if (rnd.Next(0, 2) == 0)
        {
            macs[i] = father.MacSchedule[i];
        }
        if (rnd.Next(0, 2) == 0)
        {
            jobs[i] = father.JobSchedule[i];
        }
    }
    return new Schedule(jobs, macs, jobnum, procnum, macnum);
}
#endregion
```

Algorithme 4.6 L'algorithme de croisement Uniform

5.1.4.2 Croisement sur deux points

```
#region Two Point CO
1 référence
public Schedule TwoPointCO(Schedule mother, Schedule father)
{
    int N = krlength;
    int elNum = rnd.Next(N / 4, 3 * N / 4);
    int x = rnd.Next(0, N - 1);
    int y = x + elNum - 1;
    int tt = rnd.Next(0, N - 1);
    int[] gens = new int[N];
    for (int i = 0; i < N; i++)
        gens[i] = -1;
    int[] used = new int[jobnum];
    for (int i = x; i <= y; i++)
    {
        gens[i % N] = father.JobSchedule[tt % N];
        used[gens[i % N]]++;
        tt++;
    }
    int k = 0;
    for (int i = 0; i < N; i++)
    {
        if (gens[i] != -1) continue;

        while (used[mother.JobSchedule[k]] >= procnum)
            k++;
        gens[i] = mother.JobSchedule[k];
        used[gens[i]]++;
    }
    int[] macAS = mother.MacSchedule.Clone() as int[];
    elNum = rnd.Next(N / 4, 2 * N / 4);
    x = rnd.Next(0, N);
    tt = rnd.Next(0, N);
    int bound = x + elNum;
    while (x < bound)
    {
        macAS[x % N] = father.MacSchedule[tt % N];
        tt++;
        x++;
    }
    return new Schedule(gens, macAS, jobnum, procnum, macnum);
}
#endregion
```

Algorithme 4.7 L'algorithme de croisement sur deux points.

5.1.4.3 Croisement sur une seul point

```
#region Single Point
1 référence
public Schedule SinglePointCO(Schedule mother, Schedule father)
{
    int N = krlength;
    int x = rnd.Next(N / 4, 3 * N / 4);

    int[] gens = new int[N];
    for (int i = 0; i < N; i++)
        gens[i] = -1;

    int[] used = new int[jobnum];
    for (int i = 0; i < x; i++)
    {
        gens[i] = mother.JobSchedule[i];
        used[gens[i]]++;
    }

    int k = 0;
    for (int i = x; i < N; i++)
    {
        if (gens[i] != -1) continue;

        while (used[father.JobSchedule[k]] >= procnum)
            k++;
        gens[i] = father.JobSchedule[k];
        used[gens[i]]++;
    }

    int[] macAS = new int[N];
    x = rnd.Next(N / 4, 3 * N / 4);
    for (int i = 0; i < x; i++)
    {
        macAS[i] = mother.MacSchedule[i];
    }
    for (int i = x; i < N; i++)
    {
        macAS[i] = father.MacSchedule[i];
    }
    return new Schedule(gens, macAS, jobnum, procnum, macnum);
}
#endregion
```

Algorithme 4.8 L'algorithme de croisement sur une seul point.

5.1.4.4 Croisement surCommandé(OrderedCroissing)

```
#region Ordered CO
1 référence
Schedule OrderedCO(Schedule mother, Schedule father)
{
    int N = krlength;
    //Job sequence CO
    int[] jobOS = new int[N];
    List<int> posOS = Enumerable.Range(0, N).ToList();
    List<int> fatRem = new List<int>();
    int el = rnd.Next(N / 4, 2 * N / 4);
    int x = rnd.Next(0, N - el);
    int y = (x + el - 1);
    for (int i = y + 1; i < N; i++)
    {
        fatRem.Add(father.JobSchedule[i]);
    }
    for (int i = 0; i <= y; i++)
    {
        fatRem.Add(father.JobSchedule[i]);
    }

    for (int i = x; i <= y; i++)
    {
        fatRem.Remove(mother.JobSchedule[i]);
    }
    int a = 0;
    for (int i = 0; i < N; i++)
    {
        if (i < x || i > y)
        {
            jobOS[i] = fatRem[a];
            a++;
        }
        else
        {
            jobOS[i] = mother.JobSchedule[i];
        }
    }
    int[] macAS = new int[N];
    for (int j = 0; j < procnum; j++)
    {
        x = rnd.Next(N / 3, 2 * N / 3);
        for (int i = 0; i < x; i++)
        {
            macAS[i] = mother.MacSchedule[i];
        }
        for (int i = x; i < N; i++)
        {
            macAS[i] = father.MacSchedule[i];
        }
    }
    return new Schedule(jobOS, macAS, jobnum, procnum, macnum);
}
#endregion
```

Algorithme 4.9 L'algorithme de croisement surCommandé**.**

5.1.5 Mutation

Le deuxième opérateur génétique important est la mutation, elle vient en deuxième place sur le plan d'importance par rapport au croisement. Une mutation est une perturbation introduite sur la composante de l'individu afin de garantir la diversité et élargir le champ d'exploration.

La démarche suivie dans ce travail consiste à choisir aléatoirement un individu et la machine qui doit subir cette perturbation, ensuite on choisit aléatoirement deux opérations, l'opération consiste à permuter l'ordre entre eux [14].

L'algorithme de génération de la mutation comme suite :

```
#region DoMutation
2 références
public void doMutation(Schedule child)
{
    MutationTypes temp = mutationTypes;
    if(rnd.Next(1, 101) <= mutOdd)
    switch (temp)
    {
        case MutationTypes.ExchangeValues:
            ExchangeValuesMutation(child);
            break;
        case MutationTypes.ChangeValue:
            ChangeValueMutation(child);
            break;
        case MutationTypes.SlipBlock:
            SlipBlockMutation(child);
            break;
        case Scheduling.MutationTypes.Replacement:
            ReplacementMutation(child);
            break;
    }
}
#endregion
```

Algorithme 4.10 L'algorithme de génération de la mutation.

les méthode des mutations:

5.1.5.1 Changer la valeur Mutation

```
#region Change Value Mutation

1 référence
void ChangeValueMutation(Schedule sch)
{
    int N = krlength;
    int x1 = rnd.Next(0, N);
    int mac = rnd.Next(0, macnum);
    sch.MacSchedule[x1] = mac;
}
#endregion

#region Exchange Values Mutation
```

Algorithme 4.11 L'algorithme deChanger la valeur Mutation.

5.1.5.2 Mutation des valeurs d'échange

```
#region Exchange Values Mutation

1 référence
void ExchangeValuesMutation(Schedule sch)
{
    int N = krlength;
    int x1 = rnd.Next(0, N);
    int x2 = rnd.Next(0, N);

    int temp = sch.JobSchedule[x1];
    sch.JobSchedule[x1] = sch.JobSchedule[x2];
    sch.JobSchedule[x2] = temp;

    x1 = rnd.Next(0, krlength);
    x2 = rnd.Next(0, krlength);

    int macTemp = sch.MacSchedule[x1];
    sch.MacSchedule[x1] = sch.MacSchedule[x2];
    sch.MacSchedule[x2] = macTemp;
}
#endregion
```

Algorithme 4.12 L'algorithme deMutation des valeurs d'échange.

5.1.5.3 Mutation du bloc de glissement

```
void SlipBlockMutation(Schedule sch)
{
    //Job part
    int N = krlength;
    int el = rnd.Next(1, N);
    int x1 = rnd.Next(0, N - el);
    int x2 = rnd.Next(0, N - el);
    x1 = Math.Min(x1, x2);
    x2 = Math.Max(x1, x2);
    int[] temps = new int[el];
    int kk = x1;
    for (int i = 0; i < temps.Length; i++)
    {
        temps[i] = sch.JobSchedule[kk];
        kk++;
    }
    int step = x2 - x1;
    int[] temps2 = new int[step];
    for (int i = x1 + el; i < x1 + el + step; i++)
    {
        temps2[i - x1 - el] = sch.JobSchedule[i];
    }
    for (int i = x1; i < x1 + temps2.Length; i++)
    {
        sch.JobSchedule[i] = temps2[i - x1];
    }
    for (int i = x2; i < x2 + temps.Length; i++)
    {
        sch.JobSchedule[i] = temps[i - x2];
    }

    //Machine part
    el = rnd.Next(1, N);
    x1 = rnd.Next(0, N - el);
    x2 = rnd.Next(0, N - el);
    x1 = Math.Min(x1, x2);
    x2 = Math.Max(x1, x2);
    int[] tempMacs = new int[el];
    kk = x1;
    for (int i = 0; i < tempMacs.Length; i++)
    {
        tempMacs[i] = sch.MacSchedule[kk];
        kk++;
    }
    step = x2 - x1;
    int[] tempMacs2 = new int[step];
    for (int i = x1 + el; i < x1 + el + step; i++)
    {
        tempMacs2[i - x1 - el] = sch.MacSchedule[i];
    }
    for (int i = x1; i < x1 + tempMacs2.Length; i++)
    {
        sch.MacSchedule[i] = tempMacs2[i - x1];
    }
    for (int i = x2; i < x2 + tempMacs.Length; i++)
    {
        sch.MacSchedule[i] = tempMacs[i - x2];
    }
}
```

Algorithme 4.13 L'algorithme deMutation du bloc de glissement.

5.1.5.4 Replacement Mutation

```
void ReplacementMutation(Schedule sch)
{
    int N = krlength;
    int x1 = rnd.Next(0, N);
    int x2 = rnd.Next(0, N);
    x1 = Math.Min(x1, x2);
    x2 = Math.Max(x1, x2);
    int selected = sch.JobSchedule[x1];
    for (int i = 0; i < sch.JobSchedule.Length; i++)
    {
        if (i > x1 && i <= x2)
        {
            sch.JobSchedule[i - 1] = sch.JobSchedule[i];
        }
    }
    sch.JobSchedule[x2] = selected;
    x1 = rnd.Next(0, N);
    x2 = rnd.Next(0, N);

    int mac = sch.MacSchedule[x1];
    for (int i = 0; i < sch.JobSchedule.Length; i++)
    {
        if (i > x1 && i <= x2)
        {
            sch.MacSchedule[i - 1] = sch.MacSchedule[i];
        }
    }
    sch.MacSchedule[x2] = mac;
}
```

Algorithme 4.14 L'algorithme dereplacement mutation.

5.1.6 Evolution

L'algorithme de génération d'évolution comme suite :

```
{
    t = 0;
    Initialiser p(t);
    Evaluer p(t);
    Calcule l'évolution Eval ( p(t) );
  while (condition d'arie =1) faire
    {
        Selection();
        T← T + 1;
        Selectionner p(t) de p(t-1);
        Evaluer p(t);
        Calcule l'évolution Eval ( p(t) );
      if (Eval(p(t - 1)) ≥ Eval(p(t)))
        {
            T = T - 1; *
        }
    }
}
```

Algorithme 4.15 L'algorithme de génération d'évolution .

5.1.7 Parallélisme

On utilisant le modèle Master-Slave comme suite :

```
{
   Initialiser les threades;
   Salves: auto - intiaisation;
   t = 0;
   Initialiser p(t);
   Evaluer p(t);
   Calcule l'évolution Eval ( p(t) );
     while (condition d'arie =1)
     {
       Selection();
       T← T + 1;
       Selectionner p(t) de p(t-1);
       Evaluer p(t);
     }
   Salves: Selectionner la solution optimale et envoyer vers le Mastre
   Mastre:choisier la solution optimale parmi les solution qui recevoir
}
```

Algorithme 4.16L'algorithme Parallélisme de modèle Master-Slave

CHAPITRE 5
RESULTAT ET DICISION

1. Introduction

Dans ce chapitre on va vous présenter la réalisation de cette étude, nous avons présenté notre application après ca nous expliquant les logiciels qui nous utilisons que nous avons explique dans le précédent chapitre, à savoir les algorithme génétiques, adapté au problème d'ordonnancement flow shop, enfin nous testons notre programme avec des paramètres déférent.

2. Les éléments matériels

Nous avons utilisé comme élément matériel un ordinateurs HP qui possède comme particuliers:

- Un processeur : Intel (R) Core (TM) i5 CPU M 520 @ 2.40Hz 2.40GHz.
- Mémoireinstallée(RAM) : 6.00 Go.
- Type De System : Système d'exploitation 64 bit.

3. les élémentslogiciels

Plateforme utilisé est Windows 7 professionnel pack1 .le langage de développement choisit est java.

3.1 Le langage de programmation "C#/Sharp":

C# est un langage de programmation orientée objet, fortement typé, dérivé de C et de C++, ressemblant au langage Java. Il est utilisé pour développer des applications web, ainsi que des applications de bureau, des services web, des commandes, des widgets ou des bibliothèques de classes. En C#, une application est un lot de classes où une des classes comporte une méthode Main, comme cela se fait en Java [28].

3.2 L'environnement Microsoft Visual Studio

Visual Studio est un ensemble complet d'outils de développement permettant de générer des applications web ASP.NET, des services web XML, des applications bureautiques et des applications mobiles. Visual Basic, Visual C++, Visual C# utilisent tous le même environnement de développement intégré (IDE), qui leur permet de partager des outils et facilite la création de solutions faisant appel à plusieurs langages. Par ailleurs, ces langages permettent de mieux tirer parti des fonctionnalités du frame work .NET, qui fournit un accès à des technologies clés simplifiant le développement d'applications web ASP et de services web XML grâce à Visual Web Développer[30].

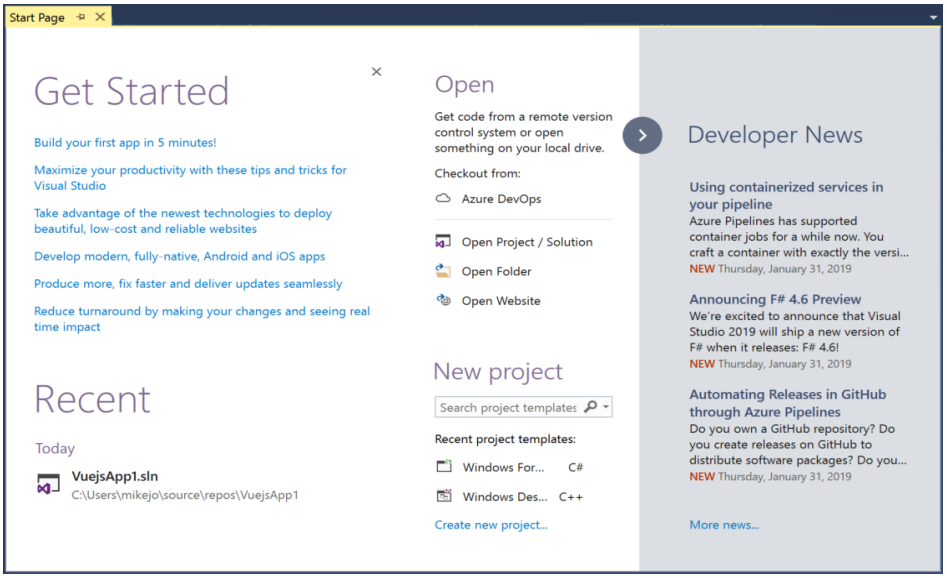


Figure 5.1 Logiciel Visual studio.

4. L'interfacegénérale du notreprogramme

Dans notre programme, nous pouvons créer le nombre de travaux, de machines et de processus, puis n'oubliez pas les paramètres AG comme la population et l'algèbre, et enfin cliquez sur exécuter pour voir le résultat. Pour exécuter en parallèle, nous cliquons sur plusieurs processus et nous pouvons voir la différence entre séquentiel et parallèle à partir du graphique de comparaison des clics.

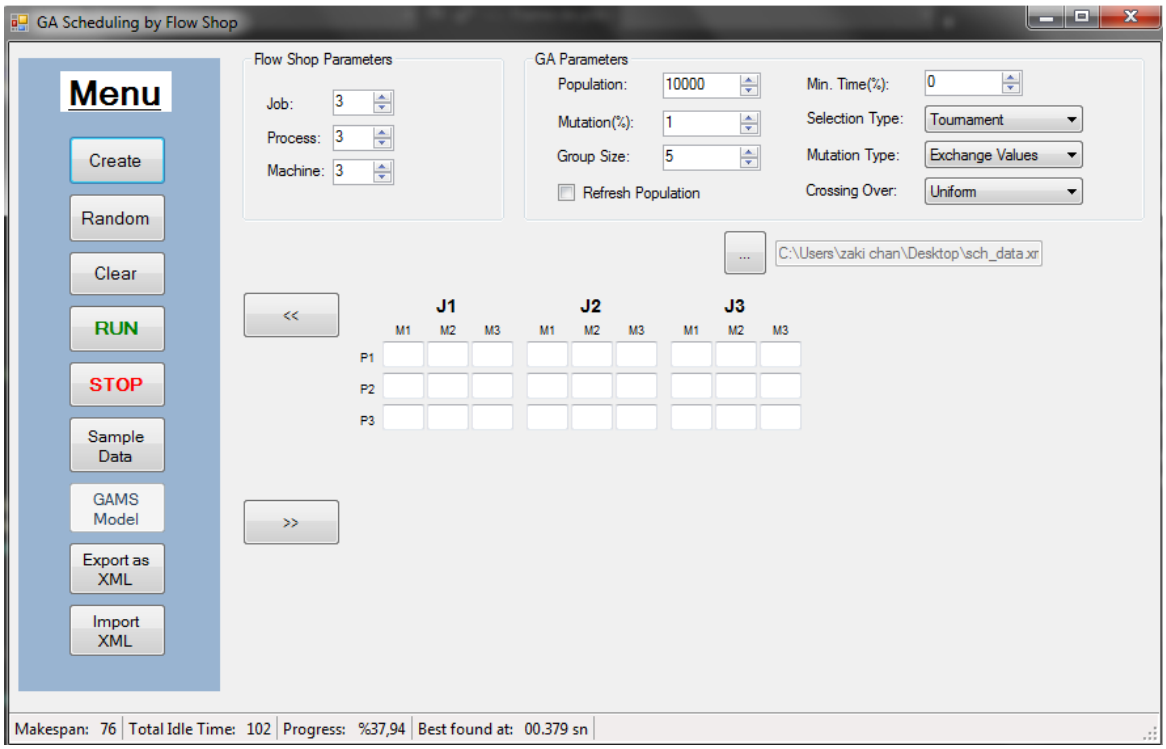


Figure 5.2 Interface principale de l'application.

5. Les résultats quelque exemples sur l'implémentation

- **Exemple 1 :**

Le tableau 5.1 présente un problème de type de flow shop composé de 3 jobs et 3 machines. on essaie d'exécuter dans deux cas (séquentiel et parallèle):

Numéro de job	Type de machine	Temps d'exécution
J1	1	19
	3	30
	2	38
J2	3	10
	1	19
	2	19
J3	1	25
	3	9
	2	13

Table 5.1 Exemple 1 en notre application.

Avec notre application on a trouvé le makespan en séquentiel :

Le Makespane= 140

Et le diagramme de Gant est qui représente la solution obtenue après application de notre algorithme en séquentiel au problème m=3/j=3 est:

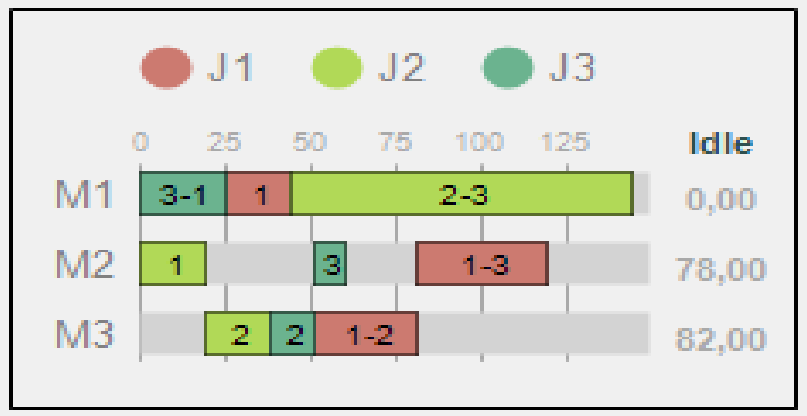


Figure 5.3 le diagramme de Gant séquentiel au problème 3x3.

On a trouvé la solution du même exemple en parallèle :

leMakespan = 108

Ainsi, le diagramme de comparaison entre le séquentiel et parallèle est le suivant :

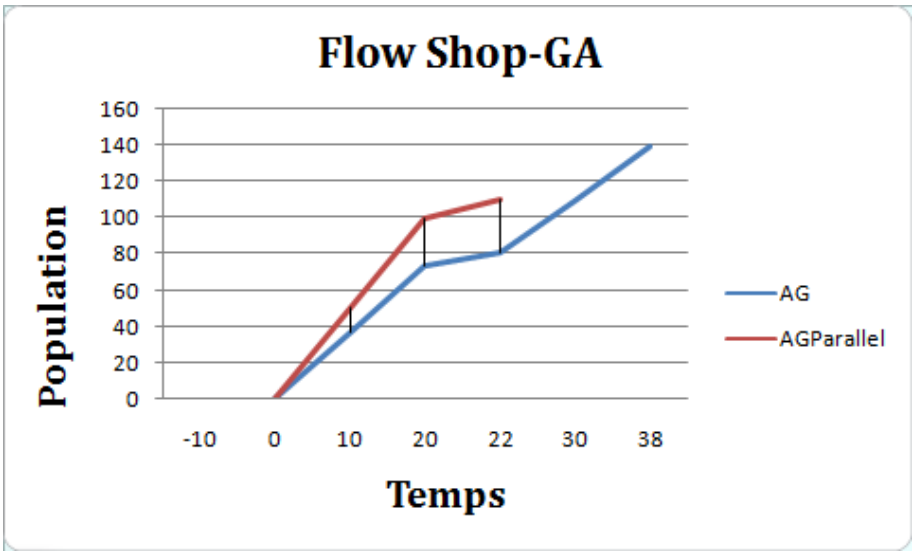


Figure 5.4 le diagramme de Gant de comparaison entre le séquentiel et parallèle.

- **Exemple 2 :**

Le tableau 4.2 présente un problème de type de flow shop composé de 3 jobs et 5 machines.

Numéro de job	Type de machine	Temps d'exécution
J1	1	63
	1	100
	2	38
	3	60
J2	2	95
	3	30
	5	52
	5	50
J3	1	24
	4	23
	4	52
	5	50
J4	2	95
	3	75
	4	80
	4	25
J5	50	3
	78	5
	25	5
	35	5

Table 5.2 Exemple 2 en notre application.

On a trouvé le makespan en séquentiel : **Makespane** =275.

Et voici le diagramme de Gantt d'une solution obtenuaprès l'application en séquentiel sur le problème 5x5 est:

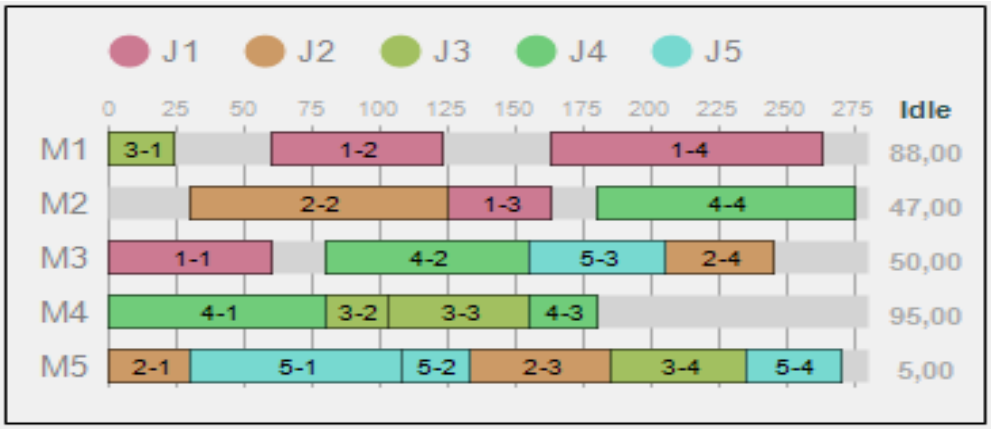


Figure 5.5 le diagramme de Gant séquentiel au problème 5x5.

Ainsi, en parallèle le makespan est :

Makespane=153

La graphe de comparaison est :

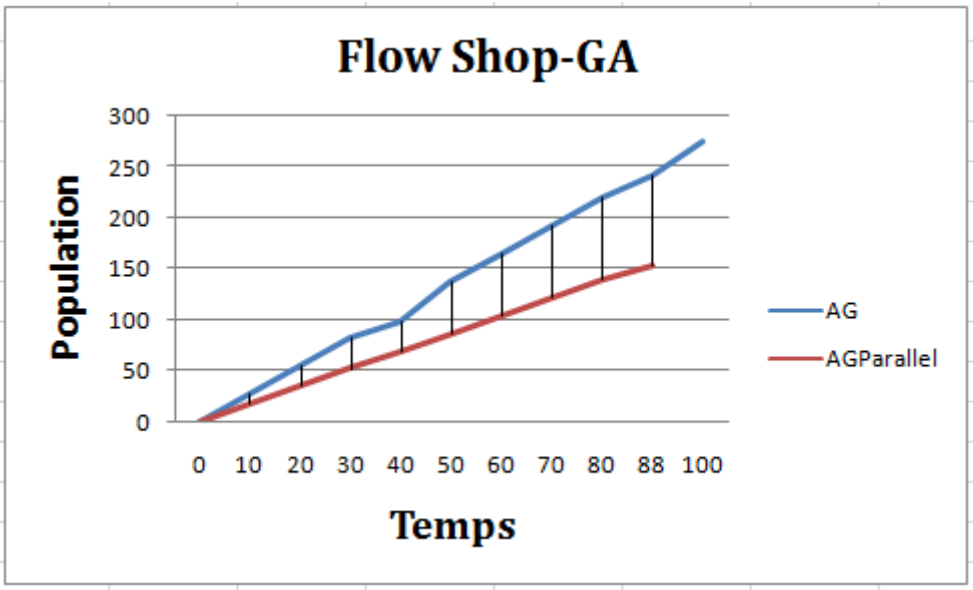


Figure 5.6 le diagramme de Gant 2 de comparaison entre le séquentiel et parallèle.

Apres la comparaison entre les exemples, nous constatons que les Algorithme génétique parallèle sont meilleurs que les Algorithme génétique séquentiel.

6. Conclusion

Dans ce chapitre nous avons vue langage et l'environnement de développement qui ont été utilisé. Ainsi, l'implémentation des algorithmes génétiques qui ont été appliquée, Nousprésentons ensuite une étude comparative des algorithmes génétiques séquentiels et des algorithmes génétiques parallèles après avoir comparé des graphes de parallélisme, et enfin nous discutons les résultats obtenus.

Conclusion générale

Le problème d'ordonnancement du flow shop est l'un des problèmes Calendrier de recherche étendu.C'est un problème très complexe, Il est un problème combinatoire difficile. Cette complexité est due à La combinaison du nombre de solutions explose, augmentant de manière exponentielle avec la taille de la solution question. Utiliser des méthodes exactes pour obtenir des solutions optimales Cela semble irréaliste. L'utilisation de méthodes d'approximation telles que l'heuristique est devenue inévitable.L'objet de ce thème est de concevoir puis implémenter un algorithme génétique parallèle pour aborder ce type de problèmes.

Nous avons conçu une application utilisant des algorithmes génétiques, puis à chaque fois que nous ajoutons du parallélisme pour comparer les algorithmes génétiques séquentiels et parallèles, nous avons ensuite essayé de déterminer lequel est le meilleur pour réduire Makespan. Ainsi, nous concluons finalement que les algorithmes génétiques parallèles sont les meilleurs pour réduire rapidement Makespan.

Dans le premier chapitre, nous avons présenté tout d'abord une présentation générale sur l'ordonnancement.Ainsi, nous avons présenté Les éléments d'un problème. Ensuite, nous avons explique classification et une représentation des problèmes d'ordonnancement. et leurs méthodes.

Dans le deuxième chapitre nous avons donné une généralité sur les algorithme génétique avec leur principes et leur mécanisme de fonctionnement.

Dans le troisième chapitre, nous résumons quelques articles liés à notre thèse.

Dans le quatrième chapitre, nous expliquons notre conception dans la résolution d'un problème d'ordonnancement de flow shop en utilisant des algorithmes génétiques parallèles.

Le dernier chapitre, nous avons vue langage et l'environnement de développement qui ont été utilisé. Ainsi, l'implémentation des algorithmes génétiques qui ont été appliquée avec succès à la résolution de problèmes d'ordonnancement de flow shop pour différents critères.

REFERENCESBIBLIOGRAPHIQUES

[1]B. Roy et D. Bouysson, Aide multi-critèresàladécision : Méthodes et cas. Collection Gestion Série : Production et technologie quantitatives appliquées à la gestion, Edition Economica, Paris, 1993.

[2] BAPTISTE P, une étude théorique et expérimentale de la propagation des contraintes de ressources, Thèse de doctorat, université Technologique de Compiègne, 1998.

[3]CHERGUIAbderrahman.DAHMANIAbd Naceur et ADDA ABBOU Abdellah: Ordonnancement d'un flow-shop par métaheuristique hybride, Mémoire de Master, Université Abou BekrBelkaid – Tlemcen,2017

[4]DAHMANE Lamia et HADROUG Selma: Algorithme génétique parallèle pour un problème d'ordonnancement d'atelier Job Shop, Mémoire de Master, Université de Msila,2020.

[5]FATHIZahira: Etude d'un problème d'ordonnancement Job-Shop avec des contraintes de transport, Thèse de MASTER, Université de Msila, 2021.

[6]Fred Glover et Tabu search : A tutorial, *Interfaces*, 20(4) :74–94, 1990.

[7]H.Boukef, Sur l'ordonnancement d'ateliers job-shop flexibles et flow-shop en industries pharma-ceutiques : optimisation par algorithmes génétiques et essaims particulaires, Thèse de Doctorat, Université de Tunis El Manar, Tunis, 2009.

[8]HANNACHEAboubakr et LEMMOUIS Abdelhamid: Résolution d'un problème d'ordonnancement de type job shop avec contrainte de transport, Thèse de doctorat, Université de Tlemcen, 2016.

[9] HEMMAK Allaoua: Support de cours d'optimisation combinatoire Focus sur les méthodes de résolution approchée,Université de M'sila, 2017.

[10]J.Carlier et P. Chrétienne: Problèmes d'ordonnancement, Edition Masson, Paris, 1988.

[11]J.Kaabi-Harrath: Contribution à l'ordonnancement des activités de maintenance dans les systèmes de production, Thèse de Doctorat, Université de Franche-Comté, França, 2004.

[12] J. Y.-T.Leung, Handbook of Scheduling : Algorithms, Models and Performance analysis. Chapman & Hall/CRC Computer and information sciences series,2004.

[13] K.Mebarek: Utilisation des stratégies Méta-heuristiques pour l'ordonnancement des ateliers de type Job Shop, Magister, BATNA, 2008.

[14]Kheireddine MERHOUM et Messaoud DJEGHABA: Algorithme génétique pour le problème d'ordonnancement de type job-shop, Article, Université Badji Mokhtar,2007.

[15]LANNAK Hanane: Adaptation de la méthode coucou pour la résolution du problème job shop. Mémoire de Master. Université de Msila,2019.

[16] LOPEZ P et ESQUIROL P: L'ordonnancement, ECONOMICA, Paris, 1999.

[17]M.larabi. Le problème de job-shop avec transport : modélisation et optimisation, Thèse de Doctorat,Université Blaise Pascal - Clermont-Ferrand II, Français, 2010.

[18]M . R. Garey and D. S. Johnson: The complexity of flowshop and jobshop scheduling. *Mathematics of operationsresearch*, 2 :117–129, 1976.

[19] MEGUIRECHE Soumia: Ordonnancement des systèmes de production flexible, Mémoire de Master, Université de Msila,2019.
[20] M.Taleb: Parallélisassion d'un algorithme génétique pour le problème d'ordonnancement sur machine unique avec temps de réglages dépendants de la séquence, Université du Québec à Chicoutimi ,2008.
[21]M.Yildizoglu et Thomas Vallée, Présentation des algorithmes génétiques et de leurs applications en économie,Article, University of Bordeaux,February 2004.

[22] N.Taghezout , Conception et Développement d'un système multi-agent d'Aide à la Décision pour la gestion de production dynamique , Doctorat, TOULOUSE ,2011.

[23] NADIR Ramla. Un problème d'ordonnancement de type Job Shop dans un environnement dynamique.Diss. Université de Msila, 2019.
[24]P.Lopez: Approche énergétique pour l'ordonnancement de tâches sous contraintes de temps et de ressources, Doctorat ,à Toulouse (France). LAAS CNRS, 1991.
[25] PINEDO M, Scheduling: Theory, Algorithms and systems, Prentice-Hall, Englewood Clis, New Jersey, 1955.

[26]T.FEO et M.RESENDE, Greedy Randomized Adaptive Search Procedures, Journal of Global Optimization 6, 2 (1995), 109-133.
[27]Wikipedia,http://igm.univmlv.fr/~dr/XPOSE2013/tleroux_genetic_algorithm/fonctionnement.html, consulté le : 25/03/2022.

[28]Wikipedia,https://www.wrike.com/fr/project-management-guide/faq/quest-ce-que-la-methode-pert-en-gestion-de-projet/,consulté le : 30/03/2022.

[29]Wikipedia,https://www.praxisframework.org/fr/library/activity-on-arrow-diagram,consulté le : 30/03/2022.

[28] Balagurusamy: *Programming In C#*, Tata McGraw-Hill Education , 7 nov. 2008.

[30]Wikipedia,https://www.techno-science.net/glossaire-definition/Visual-Studio.html ,consulté le : 30/03/2022.

Abstract

Flow shop type scheduling problems are known for their exponential complexity and their increased interest in several economic and scientific fields. For their part, genetic algorithms, despite their polynomial complexity, have always proved to be too greedy in computation time given the manipulation of populations of solutions.

The objective of this topic is to design and then implement a parallel genetic algorithm to address this type of problem. It is a matter of parallelizing gene operators with the goal of optimizing the multi-core architecture that is currently ubiquitous in the computer market.

An analytical adjustment is necessary to justify the contribution of parallelism by choosing adequate benchmarks from the literature.

Keywords: Genetic Algorithms; Parallel Algorithms; Scheduling flow Shop; Meta heuristics.

ملخص

تُعرفمشاكلجدولةنوعمتجرالتدفقبتعقيدهاالمتزايدواهتمامهاالمتزايدبالعديدمنالمجالاتالاقتصاديةوالعلمية.منجانبها،أثبتتالخوارزمياتالجينية،علىالرغممنتعقيدهامتعددالحدود،أنهاجشعةجدًافيوقتالحسابنظرًاللتلاعببمجموعاتالحلول.

الغرضمنهذاالموضوعهوتصميمثمتنفيذخوارزميةجينيةموازيةلمعالجةهذاالنوعمنالمشاكل.إنهامسألةموازاةمشغلياالجيناتبهدفا لاستغلالالأمثلللهندسةالمعماريةمتعددةالنواةالموجودةحاليًافيكلمكانفيسوقالكمبيوتر.

يعدالتعديلالتحليليضروريًّالتبريرمساهمةالتوازيمنخلالاختيارمعاييرمناسبةمنالأدبيات.

الكلماتالمفتاحية:الخوارزمياتالجينية. الخوارزمياتالمتوازيةجدولةتدفقالتسوق. الاستدلالالفوقي.

Résumé

Les problèmes d'ordonnancement de type flow shop sont connus par leur complexité exponentielle et leur intérêt accru dans plusieurs domaines économiques et scientifiques. De leur côté, les algorithmes génétiques, en dépit de leur complexité polynomiale, se sont toujours révélés être trop gourmands en temps de calcul vu la manipulation de populations de solutions.

L'objet de ce thème est de concevoir puis implémenter un algorithme génétique parallèle pour aborder ce type de problèmes. Il s'agit de paralléliser les opérateurs génétiques en vue de l'exploitation optimale de l'architecture multi-cœurs omniprésente actuellement dans le marché informatique.

Un ajustement analytique est nécessaire pour justifier l'apport du parallélisme en choisissant des benchmarks adéquats de la littérature.

Mots clés: Algorithmes Génétiques; Algorithmes Parallèles; Ordonnancement flow Shop; Metaheuristiques.