



MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA
RECHERCHE SCIENTIFIQUE

UNIVERSITÉ ABDELHAMID IBN BADIS - MOSTAGANEM

Faculté des Sciences Exactes et d'Informatique

Département de Mathématiques et informatique

Filière : Informatique

MÉMOIRE DE FIN D'ÉTUDES

Pour l'Obtention du Diplôme de Master en Informatique

Option : **Ingénierie des Systèmes d'Information**

Présenté par :

« **RACHED RACHID** »

« **SABER AHMED AMINE** »

THEME :

**Optimisation des systèmes de production : adaptation du «
Galaxy-based Search Algorithm » pour le job shop**

Soutenu le :

Devant le jury composé de :

MECHAOUI M.D Professeur Université de Mostaganem Président

BENHAMMED S. Professeur Université de Mostaganem Examineur

HABIB ZAHMANI M. Professeur Université de Mostaganem Encadreur

Année Universitaire 2024-2025



بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

Dédicaces

Je dédie ce travail à ceux qui comptent le plus pour moi :

À ma mère, mon pilier, mon cœur, celle qui a toujours cru en moi et m'a porté même dans les moments les plus difficiles. Que Dieu te bénisse pour tout ton amour et tes sacrifices.

*À mon frère et à ma sœur, mes compagnons de vie, mes complices dans les rires comme dans les épreuves.
Merci pour votre soutien sans faille.*

*Et à mes amis, Votre présence est un cadeau précieux.
Merci à vous tous d'être ma force et ma joie*

RACHID

Dédicaces

*Je dédie ce travail à toutes ma famille, mes parents,
mon frère, mes sœurs, mes amis. Merci pour tout.*

AHMED

Remerciements

Nous remercions, en premier lieu, Allah tout puissant qui nous a accordé la force et le savoir afin de mener à bien le présent travail.

Nous exprimons nos plus profondes gratitude à nos parents pour leurs encouragements et leur soutien tout au long de notre parcours.

Nos remerciements les plus sincères vont à notre encadrant, le Professeur MOHAMED HABIB ZAHMANI pour l'opportunité précieuse qu'il nous a présenté, sa disponibilité constante, ses conseils et suggestions qui ont été d'une aide indispensable au cours du projet.

Enfin nous tenons à exprimer notre reconnaissance à tous nos amis et collègues pour leur aide et soutien dans la réalisation de ce projet.

Résumé

L'optimisation des systèmes de production représente un enjeu majeur pour améliorer la compétitivité industrielle. Dans ce travail de fin d'études, nous proposons une adaptation du *Galaxy-based Search Algorithm* (GbSA), un algorithme d'optimisation métaheuristique inspiré du mouvement des galaxies, au problème classique du *Job Shop Scheduling* (JSS). Ce dernier, connu pour sa complexité NP-difficile, consiste à organiser l'ordonnancement d'un ensemble de tâches sur plusieurs machines, tout en minimisant des critères comme le temps total de production (*makespan*). L'approche développée intègre des mécanismes d'exploration et d'exploitation améliorés pour mieux s'adapter aux contraintes du job shop. Les résultats expérimentaux montrent que l'algorithme proposé est intéressant bien que les résultats obtenus ne soient pas, à ce stade, très compétitifs.

Mots clés

Optimisation des systèmes de production, Ordonnancement (Scheduling), Métaheuristiciques, Algorithmes d'optimisation, Makespan, Tâche, Représentation des solutions.

الملخص

تمثل تحسين نظم الإنتاج تحديًا رئيسيًا لتعزيز القدرة التنافسية الصناعية. في هذا العمل نهاية الدراسة، نقترح تكييف خوارزمية البحث المستندة إلى المجرة (*Galaxy-based Search Algorithm - GbSA*) ، وهي خوارزمية تحسين متوارثة مستوحاة من حركة المجرات، لمشكلة جدولة ورشة العمل (*Job Shop Scheduling - JSS*) الكلاسيكية. تُعرف هذه المشكلة بصعوبتها من فئة NP ، وتتمثل في تنظيم جدولة مجموعة من المهام عبر عدة آلات مع تقليل معايير مثل إجمالي وقت الإنتاج (*makespan*). تعتمد المنهجية المطورة على آليات محسنة للاستكشاف والاستغلال لتتلاءم بشكل أفضل مع قيود ورشة العمل. تُظهر النتائج التجريبية أن الخوارزمية المقترحة مثيرة للاهتمام، على الرغم من أن النتائج التي تم الحصول عليها ليست، في هذه المرحلة، تنافسية للغاية.

الكلمات المفتاحية

تحسين نظم الإنتاج، الجدولة، الخوارزميات المتوارثة، خوارزميات التحسين، الإنتاجية الكلية، المهمة، تمثيل الحلول.

Abstract

Optimization of production systems is a major challenge for improving industrial competitiveness. In this final-year project, we propose an adaptation of the *Galaxy-based Search Algorithm (GbSA)*, a metaheuristic optimization algorithm inspired by galactic motion, to the classical *Job Shop Scheduling (JSS)* problem. The latter, known for its NP-hard complexity, involves scheduling a set of tasks across multiple machines while minimizing criteria such as total production time (*makespan*). The developed approach integrates enhanced exploration and exploitation mechanisms to better adapt to job shop constraints. Experimental results show that the proposed algorithm is interesting, although the results obtained are not, at this stage, very competitive.

Keywords

Production system optimization, Scheduling, Metaheuristics, Optimization algorithms, Makespan, Task, Solution representation.

Table des matières

Dédicaces	III
Remerciements.....	V
Résumé	VI
Table des matières	VIII
Table des Figures.....	XI
Introduction Générale.....	1
CHAPITRE 1 : L’Ordonnancement dans les Systèmes de Production	1
1.1 Introduction	1
1.2 Formulation d’un problème d’ordonnancement.....	1
1.2.1 Les tâches	2
1.2.2 Les ressources.....	2
1.2.3 Les contraintes.....	3
1.2.4 Les objectifs	3
1.3 Les classes d’ordonnancement	4
1.3.1 Ordonnancement semi-actif	4
1.3.2 Ordonnancement actif	5
1.3.3 Ordonnancement sans délai.....	5
1.3.4 Ordonnancement optimal	5
1.4 Les ateliers.....	5
1.4.1 Single Machine.....	6
1.4.2 Flow Shop.....	7
1.4.3 Job Shop	7
1.4.4 Open Shop	8
1.5 Modélisation et représentation des problèmes d’ordonnancement	8
1.5.1 Représentation des solutions avec Diagramme de Gant	8
1.6 Méthodes de résolution	9

1.6.1	Les méthodes approchée	10
1.6.1.1	Les méthodes recherche locale	10
1.6.2	Les Méthodes évolutives	16
1.6.2.1	Algorithme Génétique	16
1.7	Conclusion.....	17
CHAPITRE 2 : Le Galaxy-Based Search Algorithm (GbSA).....		18
2.1	Introduction	19
2.2	Galaxy-based Search Algorithm (GbSA).....	19
2.3	Procédures du Galaxy-based Search Algorithm (GbSA).....	19
2.4	Adaptation de Galaxy-Based Search Algorithm (GbSA) pour ce problème SMTWT 19	
2.4.1	SMTWT (Single Machine Total Weighted Tardiness)	19
2.4.2	Structure de L'algorithme	20
2.5	Adaptation de Galaxy-Based Search Algorithm (GbSA) sur notre problème d'atelier (JSSP) 24	
2.5.1	Problème Job Shop JSSP.....	24
2.5.2	Structure de l'algorithme.....	25
2.6	Conclusion.....	37
CHAPITRE 3 : Implémentation et Résultats		38
3.1	Introduction	38
3.2	Les outils	38
3.3	Présentation Générale d'application	38
3.3.1	Interface de Saisie des Données	39
3.3.2	Interface d'Affichage des Résultats	40
3.3.3	Interface de Visualisation Graphique	41
3.4	Résultats des problèmes d'ordonnancement d'ateliers (JSSP).....	42
3.4.1	Instances 15×15	43
3.4.2	Instances 20×15	43

3.4.3	Instances 20×20	44
3.4.4	Instances 30×15	44
3.4.5	Instances 30×20	44
3.4.6	Instances 50×15	45
3.4.7	Instances 50×20	45
3.4.8	Analyse des résultats:	45
3.5	Conclusion.....	46
Conclusion générale		46
Bibliographie.....		47

Table des Figures

Figure 1 Exemple d'une tâche	2
Figure 2 Les classes d'ordonnancement	4
Figure 3 Ordonnancement semi-actif	4
Figure 4 Types d'ateliers	6
Figure 5 Unique machine	7
Figure 6 Flow Shops.....	7
Figure 7 Job Shop.....	8
Figure 8 Open Shop.....	8
Figure 9 Exemple d'un diagramme de Gantt par Machines	9
Figure 10 Exemple d'un diagramme de Gantt par Produits.....	9
Figure 11 Principe de la recherche locale	10
Figure 12 Principe de fonctionnement de la méthode descente	11
Figure 13 Principe de fonctionnement du recuit simulé.....	13
Figure 14 Principe de fonctionnement de la recherche tabou	15
Figure 15 Fonctionnement de l'algorithme génétique	17
Figure 16 La procédure GbSA principale	20
Figure 17 La procédure SpiralChaoticMove	24
Figure 18 Procédure Random (Generation Aleatoire)	26
Figure 19 procédure SPT (shortest processing Time)	27
Figure 20 LPT (Longest Processing Time).	28
Figure 21 MOR (Most Operations Remaining).	29
Figure 22 Procédure LocalSearch	31
Figure 23 Procédure Voisinage Insertion.....	32
Figure 24 Procédure Voisinage Échange	33
Figure 25 Procédure Voisinage_Insertion_Bloc_2_Jobs	34
Figure 26 Procédure Voisinage_Insertion_Bloc_3_Jobs	35
Figure 27 Procédure Voisinage_Inversion_Bloc_Aléatoire.....	36
Figure 28 Interface de Saisie des Données.....	40
Figure 29 Interface d'Affichage des Résultats.....	41
Figure 30 Interface de Visualisation Graphique.....	42
Figure 31 Instances 15x15.....	43
Figure 32 Instances 20x15.....	43
Figure 33 Instances 20x20.....	44

Figure 34 Instances 30x15.....	44
Figure 35 Instances 30x20.....	44
Figure 36 Instances 50x15.....	45
Figure 37 Instances 50x20.....	45

Introduction Générale

Dans un contexte industriel de plus en plus concurrentiel, l'optimisation des systèmes de production représente un enjeu majeur pour améliorer la productivité, réduire les coûts et maximiser l'efficacité des ressources [1]. Parmi les problèmes d'ordonnancement les plus complexes, le *Job Shop Scheduling Problem* (JSSP) occupe une place centrale en raison de sa NP-difficulté [2] et de son applicabilité dans divers domaines manufacturiers [3, 4].

Les méthodes classiques d'optimisation, qu'elles soient exactes (comme la programmation linéaire [5]) ou heuristiques, montrent souvent leurs limites face à la combinatoire explosive du JSSP [6]. C'est pourquoi les métaheuristiques, inspirées de phénomènes naturels ou sociaux, ont émergé comme une solution prometteuse pour explorer efficacement l'espace des solutions [7]. Parmi ces approches, le *Galaxy-based Search Algorithm* (GbSA), inspiré des mouvements des corps célestes [8], se distingue par sa capacité à équilibrer exploration et exploitation dans des espaces de recherche complexes [9].

L'objectif de ce mémoire est d'adapter et d'appliquer le GbSA au problème de job shop, en proposant des améliorations pour mieux répondre aux contraintes spécifiques de ce problème [10]. Nous évaluerons la performance de cet algorithme en comparaison avec des benchmarks académiques [12].

À travers cette étude, nous souhaitons contribuer à l'avancement des méthodes d'optimisation en production [14], tout en ouvrant des perspectives pour des applications industrielles plus larges [15].

CHAPITRE 1 : L'Ordonnancement dans les Systèmes de Production

Introduction

Les problèmes d'ordonnancement constituent un domaine clé de la recherche opérationnelle avec des applications pratiques dans divers secteurs industriels. Comme le note Pinedo (2016) [1], "l'ordonnancement est au cœur de toute opération de production". Un exemple concret se trouve dans l'industrie automobile où l'assemblage d'un véhicule nécessite l'ordonnancement précis de milliers d'opérations sur des centaines de machines (Zhang et al., 2020) [4].

Formulation d'un problème d'ordonnancement

Ordonnancer, c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution. L'ordonnancement est la programmation dans le temps de l'exécution d'une suite de tâches (activités, opérations) sur un ensemble de ressources physiques (humaines et techniques), en cherchant à optimiser certains critères, financiers ou technologiques et en respectant les contraintes de fabrication et d'organisation. L'ordonnancement concerne l'affectation de ressources limitées aux tâches dans le temps. C'est un processus de prise de décision dont le but est d'optimiser un ou plusieurs objectifs.

1.1.1 Les tâches

Une tâche est une entité élémentaire localisée dans le temps par une date de début et/ou de fin, dont la réalisation nécessite une durée, et qui consomme un moyen selon une certaine intensité. Certains modèles intègrent la notion de date due, une date à laquelle la tâche doit être finie ; dans ces cas, le retard induit une pénalité.[1]

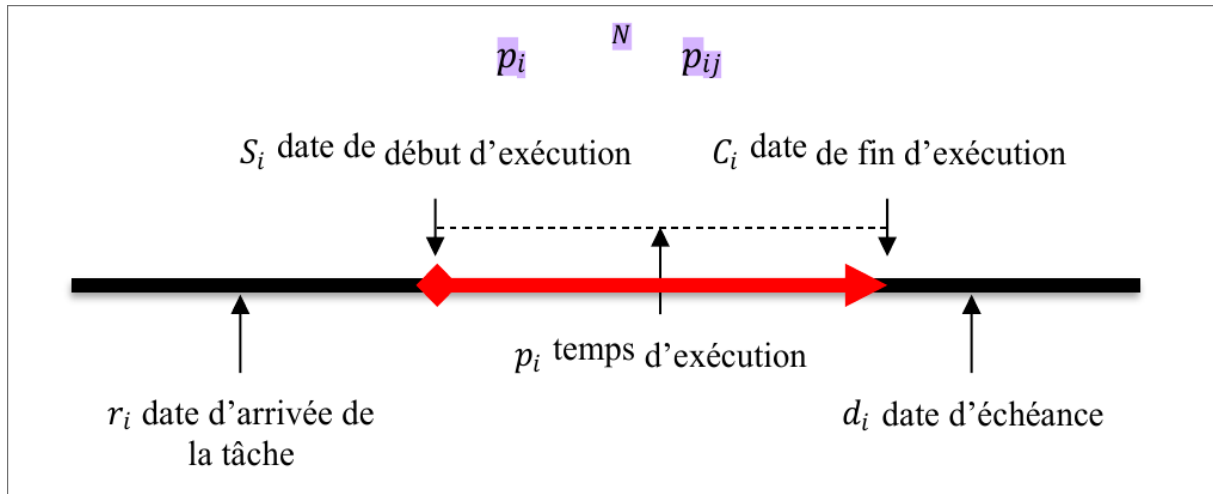


Figure 1 - Exemple d'une tâche

1.1.2 Les ressources

Une ressource est un moyen technique ou humain, destiné à être utilisé pour la réalisation d'une tâche et disponible en quantité limitée [5]. On note en général $M = \{M1, M2, \dots, Mm\}$ l'ensemble des ressources. Plusieurs types de ressources sont à distinguer. [16]

Les ressources renouvelables :

Une ressource est dite renouvelable, si après avoir été utilisée par une tâche ou allouée à une tâche, elle redevient disponible pour les autres tâches en même quantité.

Les ressources non renouvelables :

Les ressources non renouvelables sont généralement produites dans un système indépendant de la machine sur laquelle les tâches sollicitant cette ressource sont exécutées. Toutefois, certains problèmes peuvent exister où certaines tâches produisent des ressources qui peuvent être consommées plus tard par d'autres tâches.

1.1.3 Les contraintes

Les contraintes dans l'ordonnancement expriment les restrictions imposées sur les variables de décision, et se divisent en deux grandes catégories : les contraintes temporelles et les contraintes liées aux ressources.

Les contraintes temporelles peuvent être classées en deux catégories, les contraintes de temps absolu et de temps relatif. Les contraintes de temps absolu permettent de représenter la limitation des valeurs possibles pour les dates des tâches. Par exemple date de début au plus tôt, date de fin au plus tard, les contraintes de temps relatif sont les contraintes relatives.

Les contraintes de ressources concernent la capacité et la disponibilité des moyens techniques ou humains. Une ressource est dite **renouvelable** lorsqu'elle retrouve sa capacité initiale après utilisation, tandis qu'elle est **consommable** lorsqu'elle s'épuise après usage. Les ressources peuvent être **disjonctives** (une seule tâche à la fois) ou **cumulatives** (plusieurs tâches simultanées).

Enfin, Tangour Toumi [17] propose une classification supplémentaire :

Les contraintes endogènes liées au système interne (capacité et disponibilité des machines, gammes opératoires),

Les contraintes exogènes imposées par l'environnement externe (dates d'échéance client, priorités, retards permis).

1.1.4 Les objectifs

Dans la résolution d'un problème d'ordonnancement, on peut choisir entre deux grands types de stratégies, visant respectivement à l'optimalité des solutions, ou plus simplement à leur admissibilité.

L'approche par optimisation suppose que les solutions candidates à un problème puissent être ordonnées de manière rationnelle selon un ou plusieurs critères d'évaluation numériques, construits sur la base d'indicateurs de performances. On cherchera donc à minimiser ou maximiser de tels critères. On note par exemple ceux [1]

- **Liés au temps** [16] :
 - Le temps total d'exécution ou le temps moyen d'achèvement d'un ensemble de tâches.
 - Le stock d'en-cours de traitement.
 - Différents retards (maximum, moyen, somme, nombre, etc.) ou avances par rapport aux dates limites fixées.

- **Liés aux ressources** [16] :

- La quantité totale ou pondérée de ressources nécessaires pour réaliser un ensemble de tâches
- La charge de chaque ressource.
- Liés à une énergie ou un débit.
- Liés aux coûts de lancement, de production, de transport, etc., mais aussi aux revenus, aux retours d'investissements.

1.2 Les classes d'ordonnancement

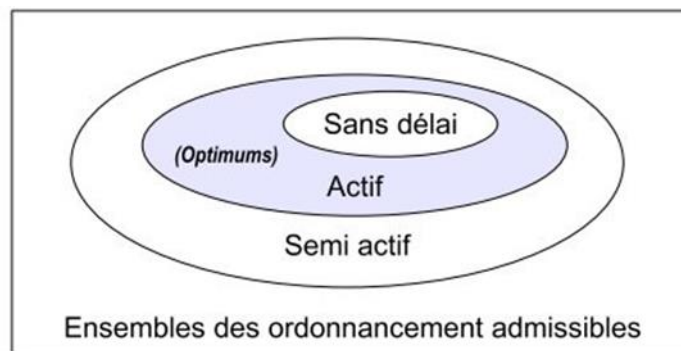


Figure 2 - Les classes d'ordonnancement

1.2.1 Ordonnancement semi-actif

Un ordonnancement est semi-actif s'il n'est pas possible de commencer une opération plus tôt sans pour autant violer une contrainte ou sans changer l'ordre des opérations sur les ressources (machines). Dans ce type d'ordonnancement la seule solution pour optimiser la fonction « objectif » est de réordonner toutes les tâches/opérations sur les ressources. Si un ordonnancement semi-actif (l'ordre de passage des tâches) est représenté avec un diagramme de Gantt, il est impossible de décaler une tâche à gauche.

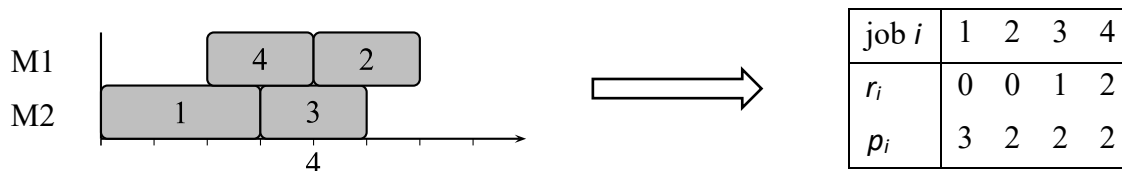


Figure 3 - Ordonnancement semi-actif

1.2.2 Ordonnancement actif

Un ordonnancement est actif si tout décalage oblige à retarder la date de début d'exécution d'une autre opération. Il est aussi impossible d'insérer une nouvelle opération entre de deux tâches sans remettre en cause l'ordre d'exécution ou violer certaines contraintes. Un ordonnancement actif est aussi en même temps semi-actif. Sur un diagramme de Gantt, un ordonnancement est considéré comme actif s'il est impossible de décaler une tâche à gauche.

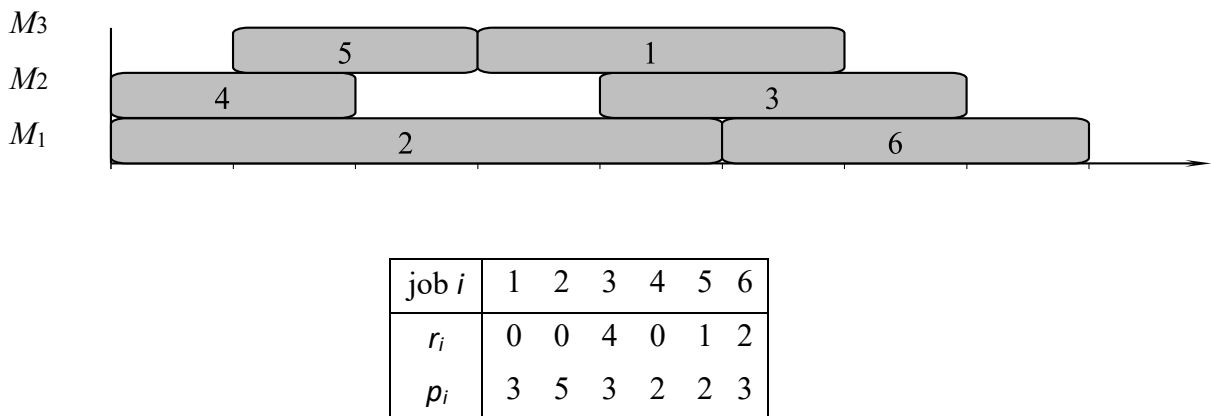


Figure 4 - Ordonnancement actif

1.2.3 Ordonnancement sans délai

Un ordonnancement est « sans délai » si à chaque instant toute opération qui dispose des ressources nécessaires est commencée. C'est-à-dire qu'une machine ne doit pas rester inactive alors qu'une tâche/opération est présente dans la file d'attente de la machine. On ne doit pas retarder l'exécution d'une tâche si celle-ci est en attente et si la ressource est disponible.

1.2.4 Ordonnancement optimal

Un ordonnancement est optimal s'il donne les meilleures valeurs possibles étant donné un ensemble d'objectifs à optimiser. Dans un problème d'ordonnancement il est possible de trouver plus d'une solution qui donnent le même résultat (même valeur de la fonction objectif).

1.3 Les ateliers

Une classification très répandue des ateliers, du point de vue ordonnancement, est basée sur les différentes configurations des machines. Les modèles les plus connus sont ceux d'une machine unique, de machines parallèles, d'un atelier à cheminement unique ou d'un atelier à cheminement multiple.

Les types: Single Machine, Flow Shop, Job Shop et Open Shop.

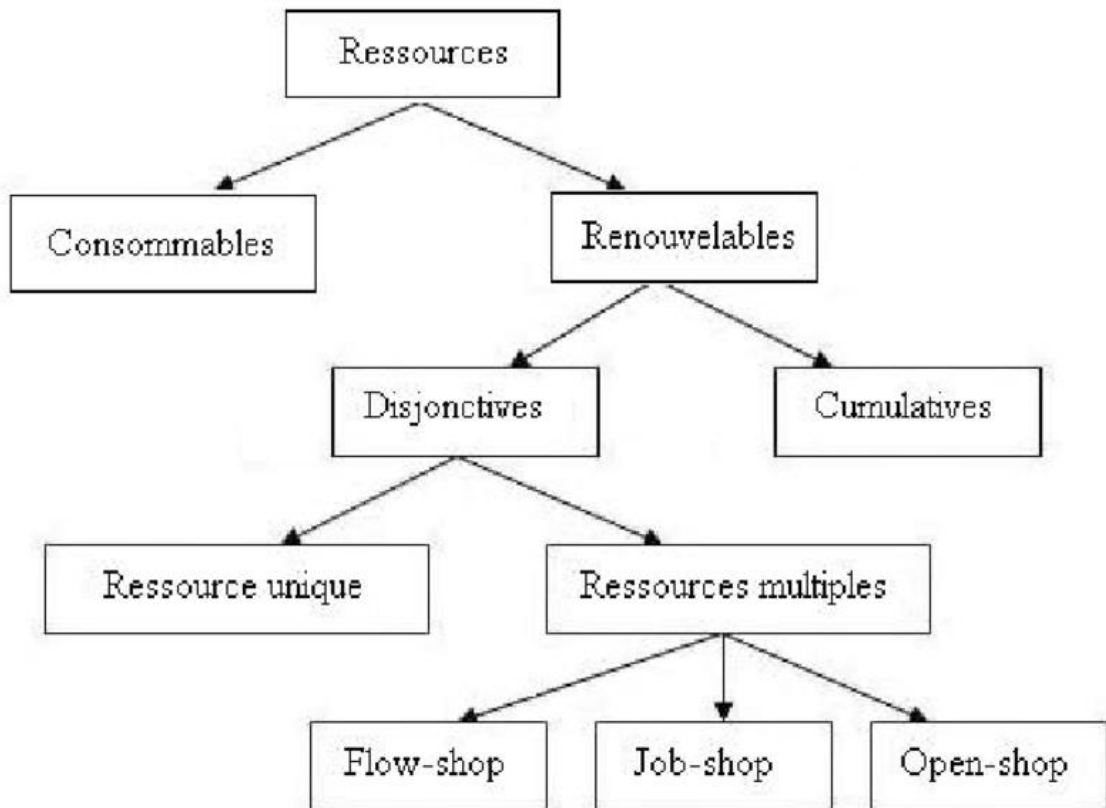


Figure 4 - Types d'ateliers

1.3.1 Single Machine

Dans L'atelier à Machine Unique, l'ensemble des tâches à réaliser est fait par une seule machine. Les tâches alors sont composées d'une seule opération qui nécessite la même machine. L'une des situations intéressantes où on peut rencontrer ce genre de configurations est le cas où on est devant un système de production comprenant une machine goulot qui influence l'ensemble du processus. L'étude peut alors être restreinte à l'étude de cette machine.

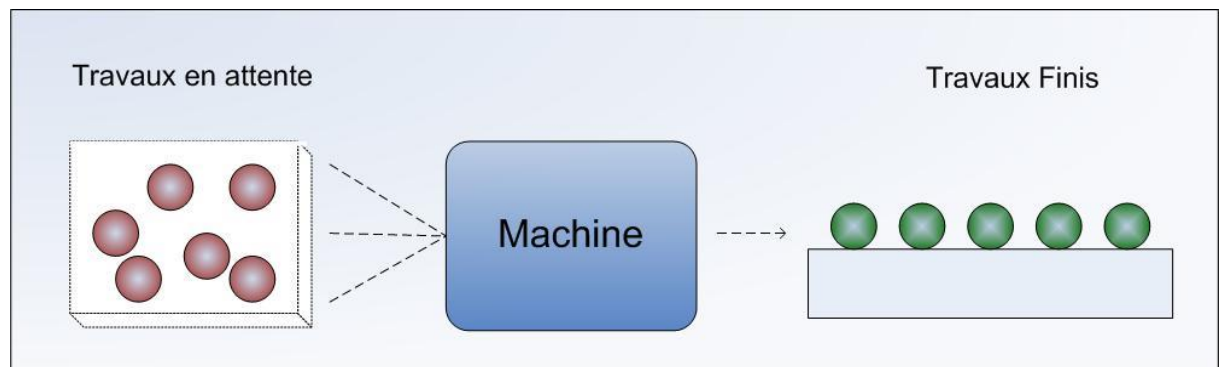


Figure 5 - Unique machine

1.3.2 Flow Shop

Appelés également ateliers à cheminement unique, ce sont des ateliers où une ligne de fabrication est constituée de plusieurs machines en série ; toutes les opérations de toutes les tâches passent par les machines dans le même ordre. Dans les ateliers de type flow-shop hybride, une machine peut exister en plusieurs exemplaires identiques fonctionnant en parallèle (Rajendran & Holthaus [18]).

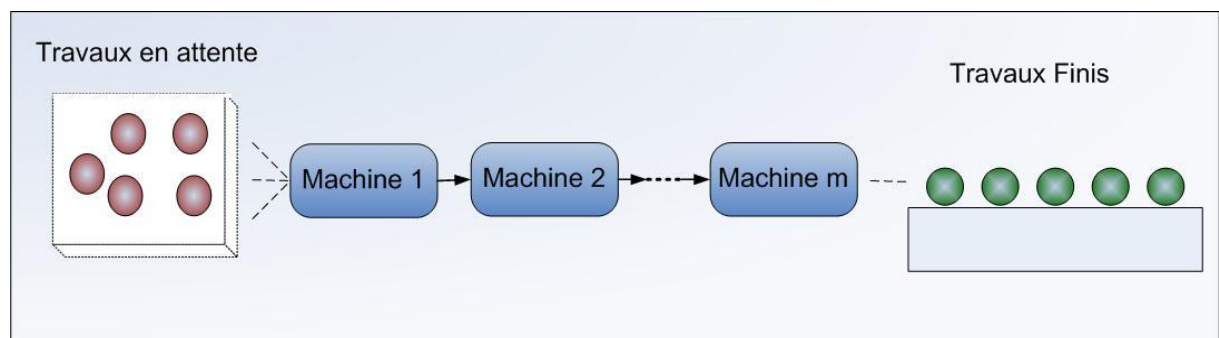


Figure 6 - Flow Shops

1.3.3 Job Shop

Dans les ateliers de type *Job Shop*, ateliers à cheminement multiple, ce sont des ateliers où les opérations sont réalisées selon un ordre bien déterminé, variant selon la tâche à exécuter ; le job-shop flexible est une extension du modèle job-shop classique, sa particularité réside dans le fait que plusieurs machines sont potentiellement capables de réaliser un sous-ensemble d'opérations. [15]

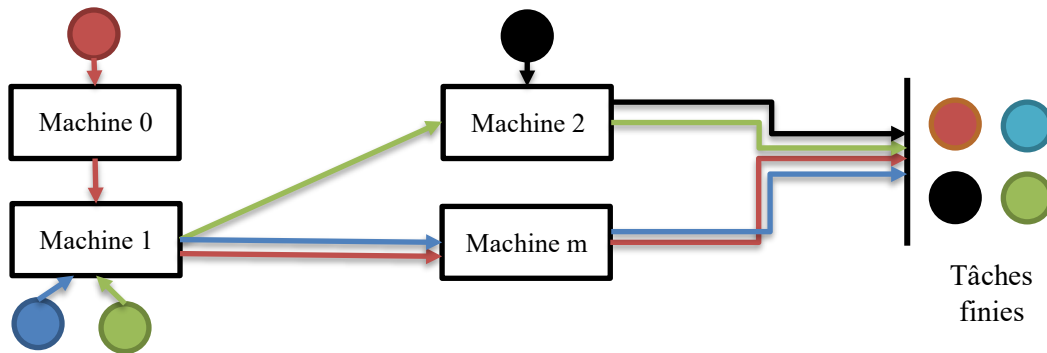


Figure 7 - Job Shop

1.3.4 Open Shop

Ce type d'atelier est moins contraint que celui de type flow-shop ou de type job-shop. Ainsi, l'ordre des opérations n'est pas fixé a priori, le problème d'ordonnancement consiste, d'une part, à déterminer le cheminement de chaque produit et, d'autre part, à ordonnancer les produits en tenant compte des gammes trouvées, ces deux problèmes pouvant être résolus simultanément.

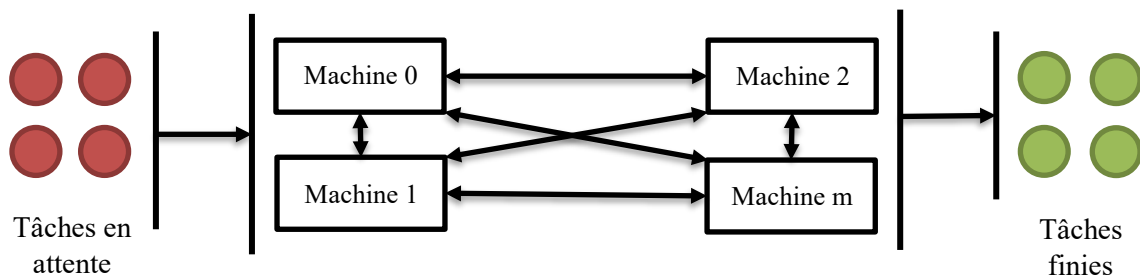


Figure 8 - Open Shop

1.4 Modélisation et représentation des problèmes d'ordonnancement

Il existe trois sortes de représentations possibles d'un problème d'ordonnancement : le diagramme de Gantt, le graphe Potentiel-Tâches et la méthode PERT

1.4.1 Représentation des solutions avec Diagramme de Gant

Afin de visualiser la solution d'un problème d'ordonnancement en utilise Ce diagramme qu'il permet de représenter l'**ordre d'exécution** des tâches ainsi que leurs **dates de début** et **dates de fin** sur les différentes machines de l'atelier tout au long du processus de production. [16]

[1]

Il existe deux types de représentations par diagramme de Gantt :

- **Diagramme par produit** : où l'on suit l'avancement des différentes opérations associées à un même produit,
- **Diagramme par machine** : où l'on visualise la séquence d'opérations exécutées par chaque machine.

Ces deux types de représentation sont illustrés dans la **Figure 10** et la **Figure 11**, à partir d'un exemple simple comportant **3 tâches** et **2 machines**.

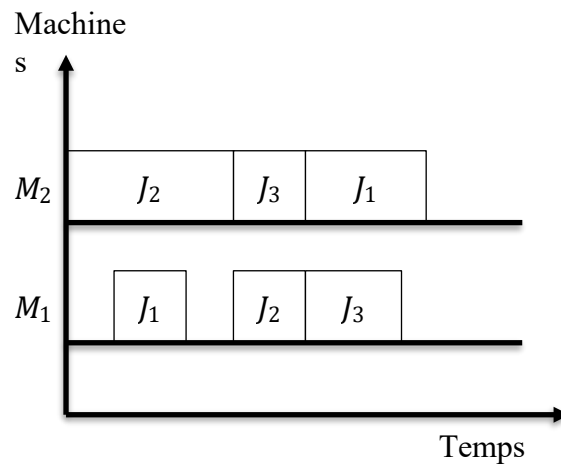


Figure 9 - Exemple d'un diagramme de Gantt par Machines

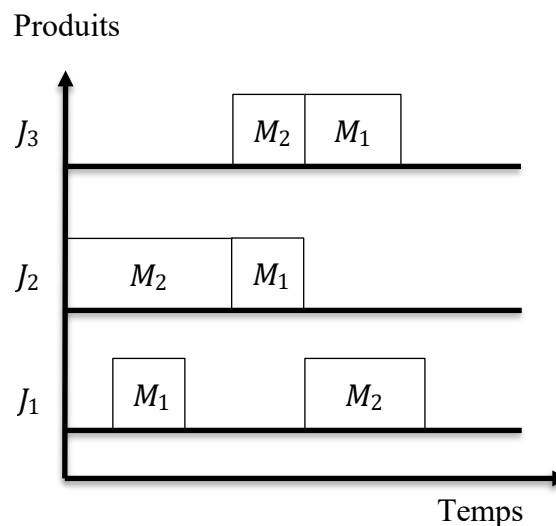


Figure 10 - Exemple d'un diagramme de Gantt par Produits

1.5 Méthodes de résolution

Méthodes de résolution développées peuvent être classées en deux catégories : les méthodes exactes qui garantissent la complétude de la résolution et les méthodes approchées qui perdent la complétude pour gagner en efficacité, les méthodes constructives construction pas à pas d'une solution, les méthodes recherche locale amélioration itérative d'une solution initiale et pour finir les méthodes évolutives. Inspirées de l'évolution biologique

1.5.1 Les méthodes approchées

Les méthodes approchées (heuristiques/méta-heuristiques) sont des techniques alternatives aux méthodes exactes pour résoudre des problèmes d'ordonnancement complexes. Elles permettent d'obtenir rapidement des solutions de bonne qualité sans garantir l'optimalité, ce qui les rend indispensables dans des cas où les méthodes exactes échouent (problèmes NP-difficiles, dynamique des environnements, multi-objectifs).

1.5.1.1 Les méthodes recherche locale

Les méthodes décrites sont des algorithmes itératifs qu'ils s'améliorent une solution initiale soit aléatoire ou bien heuristique en explorant son voisinage. A chaque itération, une solution voisine est choisie pour optimiser fonction objective.

Le processus s'arrête lorsqu'un optimum local est trouvé (Figure 12). Ces approches incluent la méthode descente, recuit simulé et la méthode tabou [7].

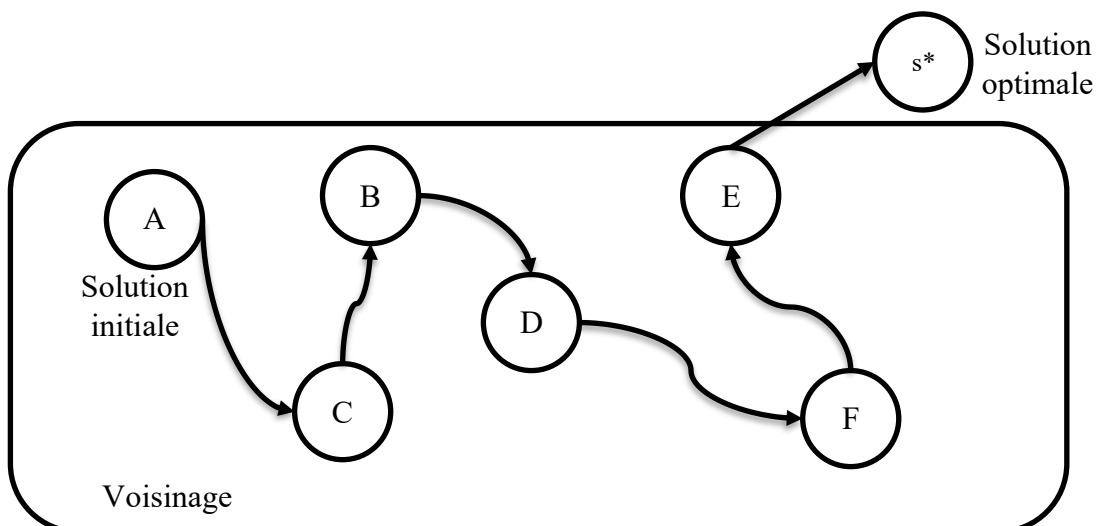


Figure 11 - Principe de la recherche locale

- **La méthode de descente**

La méthode de descente est une approche simple et intuitive pour résoudre des problèmes d'optimisation. Elle fonctionne en explorant l'espace des solutions de manière itérative : à chaque étape, elle évalue les solutions voisines de la solution courante et sélectionne celle qui offre la meilleure amélioration de la fonction objective. La **Figure-13** illustre ce principe : on observe un déplacement progressif de la solution vers un point où toute tentative de changement ne conduit plus à une amélioration.

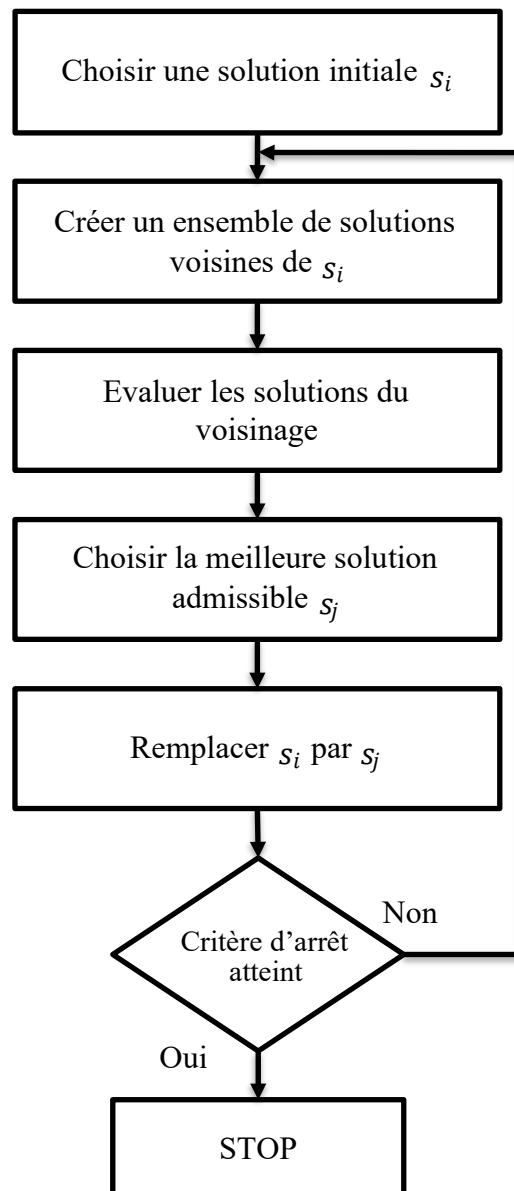


Figure 12 - Principe de fonctionnement de la méthode descente

Le principal défaut de la méthode de descente réside dans sa forte **dépendance à la solution**. En effet, le processus est **gourmand** : il poursuit uniquement des

améliorations locales sans jamais revenir en arrière. Ainsi, si la solution initiale est proche d'un **minimum local** peu satisfaisant (c'est-à-dire loin du minimum global), l'algorithme risque de s'y piéger rapidement et de s'arrêter prématurément. Ce comportement rend la méthode inefficace pour les problèmes comportant de nombreux minima locaux.

- **Recuit simulé**

Le recuit simulé est une méthode empirique (métaheuristiques) d'optimisation, inspirée d'un processus, le recuit, utilisé en métallurgie. On alterne dans cette dernière des cycles de refroidissement lent et de réchauffage (recuit) qui ont pour effet de minimiser l'énergie du matériau. Cette méthode est transposée en optimisation pour trouver les extrema d'une fonction. [20]

La méthode vient du constat que le refroidissement naturel de certains métaux ne permet pas aux atomes de se placer dans la configuration la plus solide. La configuration la plus stable est atteinte en maîtrisant le refroidissement et en le ralentissant par un apport de chaleur externe, ou bien par une isolation.

Le recuit simulé d'abord démarre avec une solution initiale aléatoire, et explore l'espace de solutions en effectuant des modifications mineures à la solution initiale. Si après modification, la nouvelle solution obtenue est meilleure que la précédente elle est retenue. Si ce n'est pas le cas, elle peut être retenue suivant la probabilité P où ΔE représente la détérioration de la nouvelle solution et T un paramètre inversement proportionnel au nombre d'itérations (voir Figure 14).

$$P = \exp(-\Delta E/T)$$

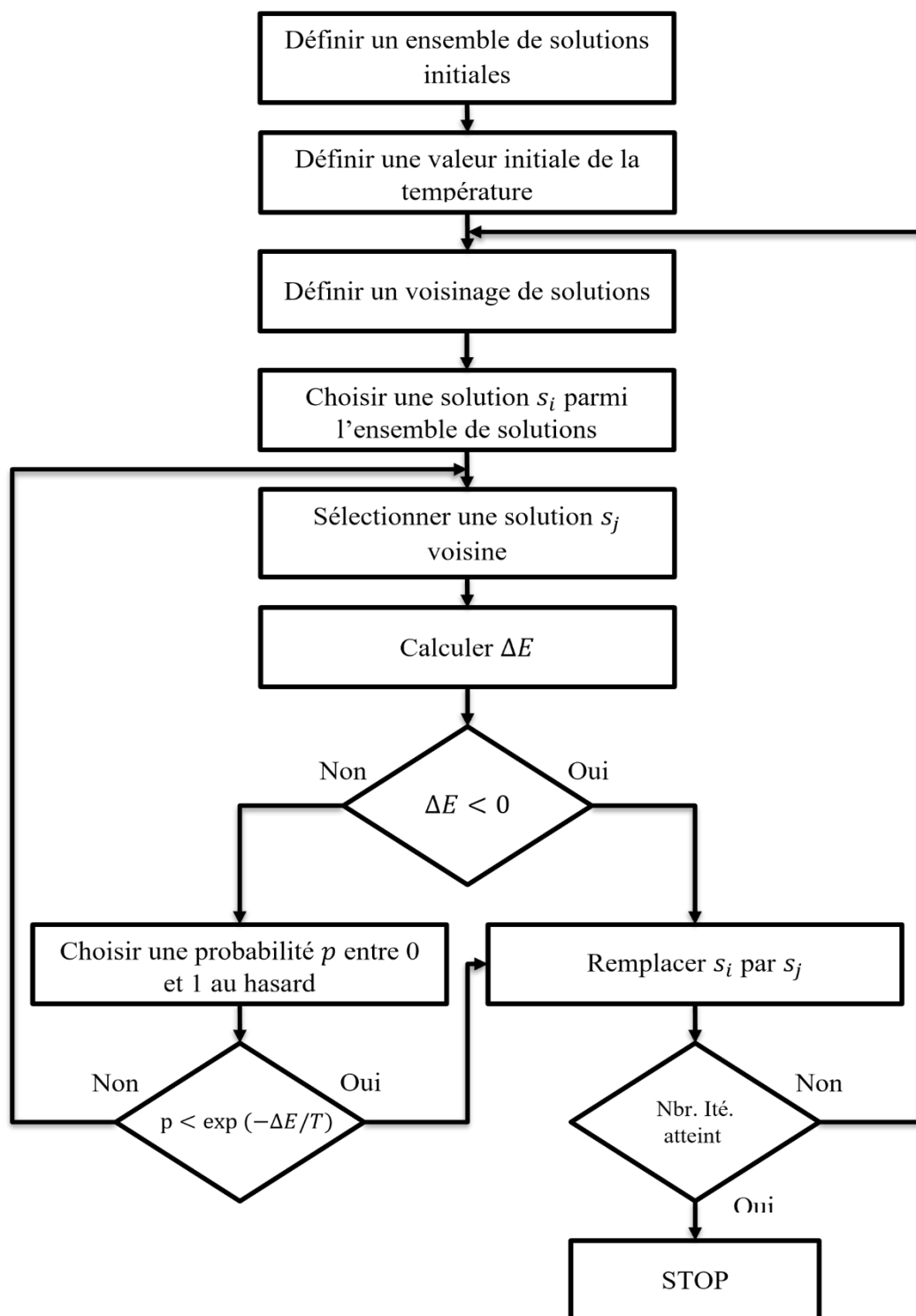


Figure 13 - Principe de fonctionnement du recuit simulé

- **Recherche taboue**

La **recherche tabou** (*Tabu Search*) est une méthode d'optimisation avancée fondée sur la recherche locale, introduite par **Fred Glover** dans les années 1980 [21]. Bien que ses concepts soient apparus dès les années 1970, ce n'est qu'en 1986 que **Glover** a formalisé et popularisé son utilisation. Comme les autres méthodes locales, elle cherche à améliorer une fonction objective en explorant l'espace des solutions de proche en proche.

Cependant, contrairement aux méthodes locales classiques, la recherche taboue introduit des mécanismes spécifiques pour éviter le piégeage dans des minima locaux et favoriser l'exploration diversifiée de l'espace de solutions.

La méthode repose sur deux principes fondamentaux :

- **Principe d'amélioration** : à chaque itération, la meilleure solution voisine est choisie. Si aucune solution voisine n'améliore la fonction objectif, l'algorithme sélectionne alors la moins mauvaise parmi les voisines disponibles.
- **Principe de mémoire taboue** : pour éviter de revenir sur des solutions récemment visitées, l'algorithme maintient une "**liste taboue**", qui mémorise un certain nombre de solutions ou de mouvements interdits pendant un temps déterminé (ou un certain nombre d'itérations) [28].

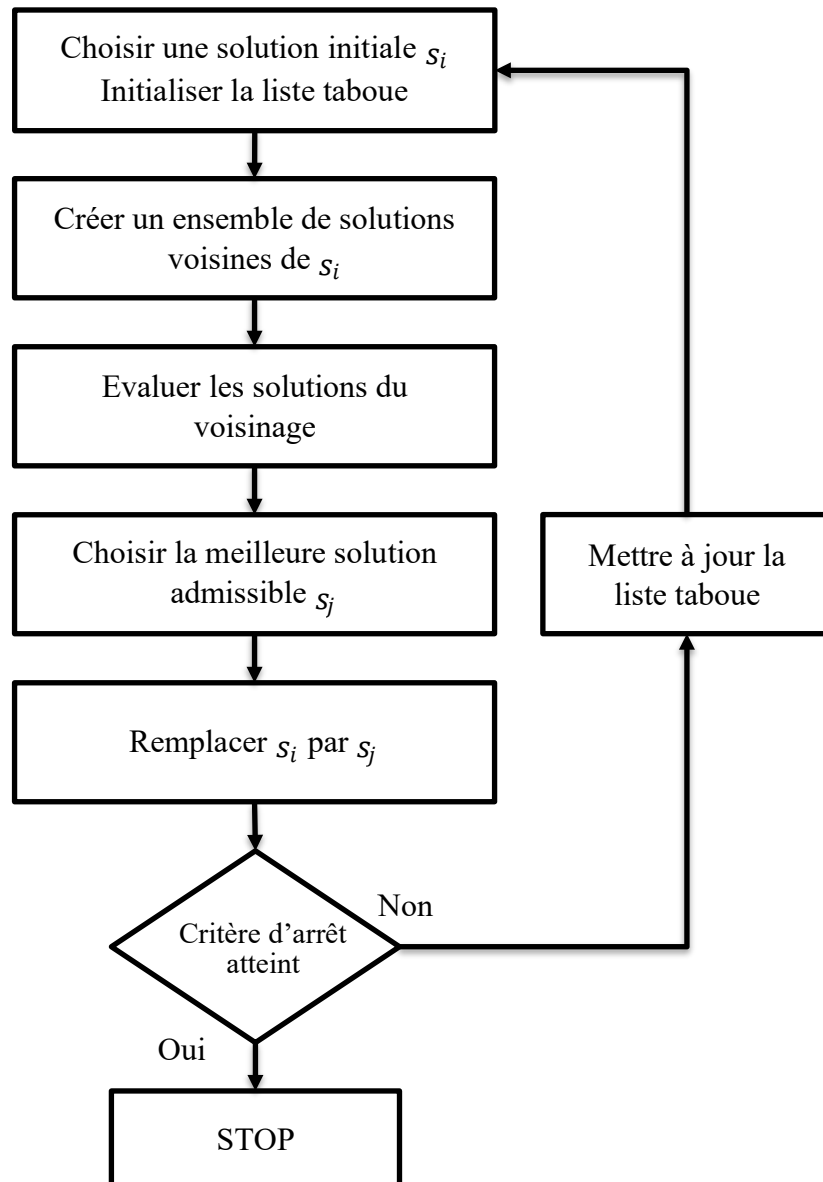


Figure 14 - Principe de fonctionnement de la recherche taboue

1.5.2 Les méthodes évolutives

Ces méthodes s'inspirent des principes de l'évolution, elles ont pour but de faire évoluer un ensemble de solutions appelé population permettant atteindre des nouvelles solutions, qu'en a déjà présentés dans la population précédente, afin d'optimiser une fonction prédéfinie et parmi les méthodes il y a (AG) algorithme génétique.

1.5.2.1 Algorithme Génétique

L'Algorithme Génétique (AG) est une des méthodes d'optimisation inspirer des principes de la sélection naturelle et de la génétique. On utilise cet algorithme pour résoudre des problèmes complexes, comme les problèmes d'ordonnancement [11].

Parmi ces principales caractéristiques de cet algorithme sont :

Dans un AG :

Fonctionnement de l'algorithme génétique

Le processus d'un AG suit plusieurs étapes principales, illustrées dans la **Figure-16** :

Génération de la population initiale

La population initiale est généralement créée de manière aléatoire ou via des heuristiques ou méta-heuristiques. Elle doit être suffisamment diversifiée pour garantir une exploration efficace de l'espace de solutions.

- **Sélection** : La sélection est basée sur la valeur d'aptitude et la concentration pour augmenter la diversité de la population et pour éviter la convergence prématurée

Croisement et mutation.

- **Croisement (crossover)** : permet d'échanger les séquences opérations entre les machines pour générer des descendant plus diversifier.
- **Mutation** : consiste à modifier aléatoirement un ou plusieurs gènes d'un individu. La mutation joue un rôle crucial. Elle utilise les propriétés de chemin critique pour ajuster les séquences des opérations pour éviter les solutions locales optimale.
- **Remplacement** : Les individus créent par croisement et/ou mutation, n'en peux pas décider de la façon d'intégrer ces nouveaux individus et s'ils remplacent ou non des individus précédent.

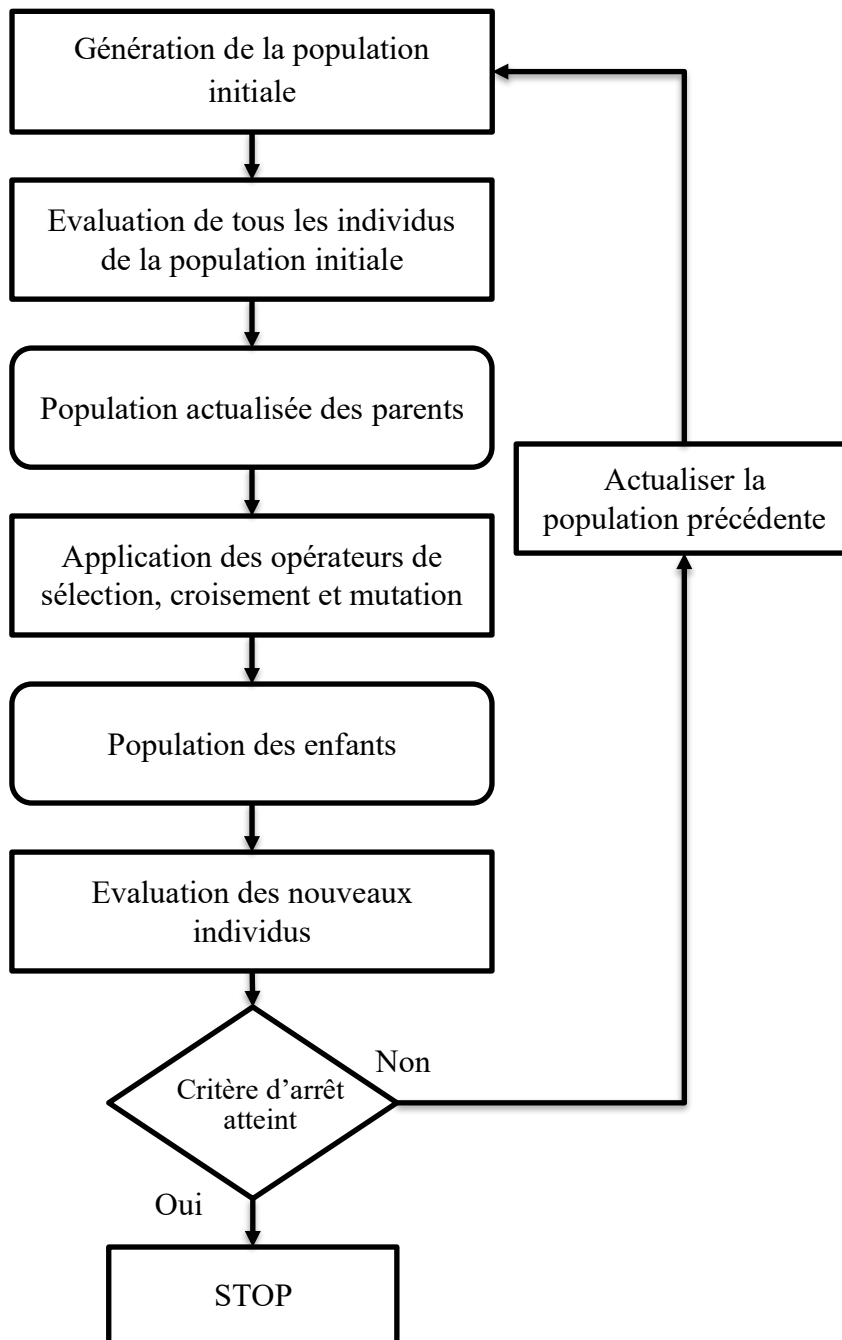


Figure 15 - Fonctionnement de l'algorithme génétique

1.6 Conclusion

Ce chapitre a présenté les concepts fondamentaux de l'ordonnancement dans les systèmes de production, en abordant les tâches, les ressources, les contraintes et les objectifs. Nous avons également exploré les différentes classes d'ordonnancement et les types d'ateliers, tout en examinant les méthodes de résolution classiques. Ces bases théoriques sont essentielles pour comprendre l'adaptation du Galaxy-based Search Algorithm (GbSA) au problème SMTWT, qui sera développée dans les chapitres suivants.

CHAPITRE 2 : Le Galaxy-Based Search Algorithm (GbSA)

2.1 Introduction

Le Galaxy-based Search Algorithm (GbSA) est un algorithme métaheuristique innovant inspiré des mouvements des galaxies, combinant l'exploration et l'exploitation pour résoudre des problèmes d'optimisation complexes. Dans ce chapitre, nous présentons le GbSA au problème SMTWT, pour comprendre le fonctionnement GBSA en exploitant ses mécanismes clés comme le SpiralChaoticMove et la recherche locale puis nous proposerons une adaptation pour le problème job shop.

2.2 Galaxy-based Search Algorithm (GbSA)

GbSA est un algorithme d'optimisation métaheuristique basé sur la dynamique des spirales des galaxies, et il est utilisé pour donner des solutions à des problèmes d'optimisation complexes. GbSA distingue sa facilité d'implémentation et sa capacité d'éviter les pièges optimaux locaux, et sa convergence rapide. [30]

2.3 Procédures du Galaxy-based Search Algorithm (GbSA)

Dans GBSA on distingue 3 procédures : [30]

- **Initialisation**

Généralisation aléatoire d'un ensemble de solutions initiales dans l'espace dimensionnel d'un problème

- **Mouvement chaotique spirales**

Est un mécanisme qui explore l'espace pour trouver la solution optimale incluse en utilisant un mouvement qu'il s'appelle un **mouvement en spirale** renforcé par une variable chaotique générée par *nextchaos()*. Cela permet de maintenir une grande diversité et d'éviter les pièges des optimaux locaux

- **Recherche locale (Local Search)**

Recherche locale c'est une étape utilisée pour affiner les solutions après une exploration globale par le mouvement chaotique spirale

Adaptation de Galaxy-Based Search Algorithm (GbSA) pour ce problème SMTWT

2.3.1 SMTWT (Single Machine Total Weighted Tardiness)

Le problème SMTWT (Single Machine Total Weighted Tardiness) est un problème d'ordonnancement en recherche opérationnelle. Il consiste d'un ensemble de tâches sur une seule machine.

les auteurs [22] [8] [28] en L'objectif c'est de minimiser la somme des retards pondérés des tâches en utilisant l'algorithme d'optimisation galaxy based search (GbSA).

2.3.2 Structure de L'algorithme

- La procédure GbSA principale

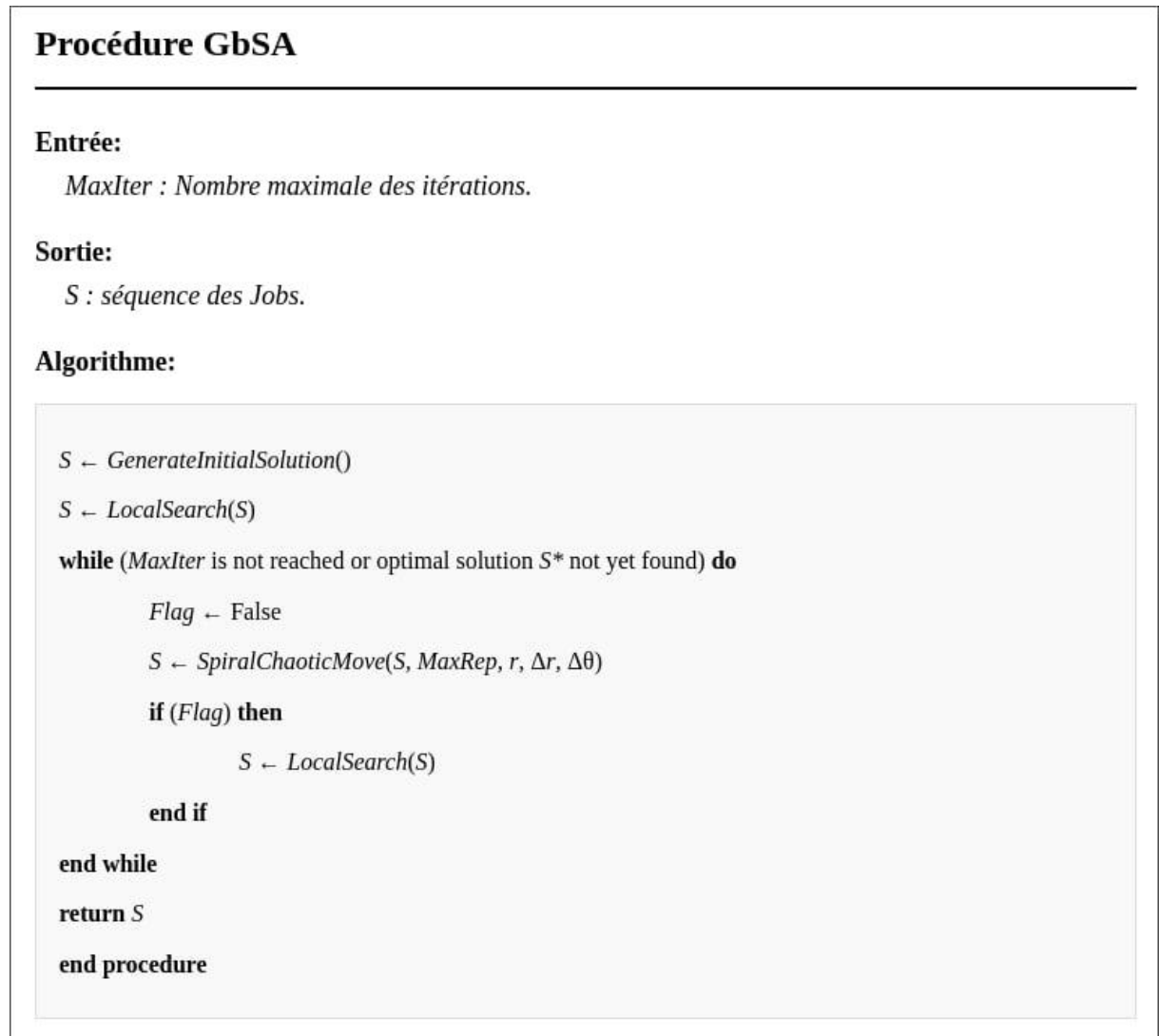


Figure 16 - La procédure GbSA principale

- Solution initiale

Dans l'adaptation SMTWT la solution initiale est générée en utilisant six règles de répartition différentes

- EDD (Earliest Due Date)
- WMDD (date d'échéance modifiée pondérée)
- MDR (règle de répartition mixte)
- Règles H1, H2 et H3

Pour chaque règle l'algorithme GBSA exécute une fois la totalité et 6 fois, chaque fois avec une solution initiale déférente, et en prend la meilleure solution pour initialiser la recherche.

- **Recherche Locale**

La fonction locale search appliquer pour explorer les solutions voisines et doit strictement améliorer (celles qui minimisent le TWT) par rapport à la meilleure solution de l'itération en cours sont acceptées, c'est à dire que la solution doit être mise à jour lorsqu'en trouve une autre meilleure solution que les précédentes (Shah-Hosseini, 2011b) [30].

La procédure de recherche locale recherche la meilleure solution la plus proche avec des pas de taille réduite

- **Insérer** : Retirer un travail de sa position et l'insérer ailleurs.
- **Échanger** : échanger les positions de deux tâches sélectionnées de manière aléatoire.
- **Insertion d'un bloc de deux emplois** : Déplacer un bloc de deux tâches consécutives vers une autre position.
- **Insertion d'un bloc de trois emplois** : Déplacer un bloc de trois travaux consécutifs vers une autre position.
- **Inversion de bloc** : inverser un bloc de 4 à 10 emplois et l'insérer ailleurs.

Au cours de la recherche locale, tous ces voisinages sont explorés afin d'améliorer la solution actuelle.

- **La procédure SpiralChaoticMove**

Le mouvement chaotique en spirale est un mécanisme clé de l'algorithme de recherche basé sur la galaxie (GbSA), conçu pour explorer l'espace des solutions de manière globale et éviter les optima locaux. Inspiré par la dynamique des bras galactiques, ce mouvement combine une trajectoire en spirale avec une variable chaotique générée par la carte logistique $x(m+1) = \lambda \times x(m) \times (1-x(m))$ avec $\lambda=4$. À chaque itération, le rayon r et les angles θ_i sont ajustés via des pas Δr et $\Delta \theta$, permettant une recherche progressive et diversifiée autour de la solution courante. Si une solution améliorée est trouvée (réduisant le retard total pondéré, TWT), elle est retenue et

une recherche locale est déclenchée pour affiner les résultats. Ce mécanisme, analogue à un opérateur de mutation dans les algorithmes génétiques, garantit une exploration robuste et une convergence efficace vers des solutions optimales ou quasi-optimales.

Procédure SpiralChaoticMove

Entrée:

$S, MaxRep, r, \Delta, \Delta\theta.$

Sortie:

$\hat{S}$ séquence des Jobs.

Algorithme:

```
 $\hat{S} \leftarrow S$ 
repeat  $i = 0$  to  $n - 1$ 
     $\theta_i \leftarrow (-1 + 2 \times NextChaos()) \times \pi$ 
end repeat
while ( $rep < MaxRep$ ) do
    repeat  $i = 0$  to  $n - 1$ 
         $SNext \leftarrow (Si + NextChaos() \times r \times \cos(\theta_i)) \bmod n$ 
        if  $SNext < 0$  then
             $SNext \leftarrow SNext + n$ 
        end if
         $Swap(\hat{S}, Si, SNext)$ 
    end repeat
    if  $M(\hat{S}) < M(S)$ 
         $S \leftarrow \hat{S}$ 
         $Flag \leftarrow true$ 
        Go to end procedure
    end if
    repeat  $i = 0$  to  $n - 1$ 
         $SNext \leftarrow (Si - NextChaos() \times r \times \cos(\theta_i)) \bmod n$ 
        if  $SNext < 0$  then
             $SNext \leftarrow SNext + n$ 
```

```

        end if
        Swap( $\hat{S}$ ,  $S_i$ ,  $S_{Next}$ )
    end repeat
    if  $M(\hat{S}) < M(S)$ 
         $S \leftarrow \hat{S}$ 
         $Flag \leftarrow true$ 
        Go to end procedure
    end if
     $r \leftarrow r + \Delta r$ 
    repeat  $i = 0$  to  $n - 1$ 
         $\theta_i \leftarrow \theta_i + \Delta \theta$ 
        if  $\theta_i \geq \pi$  then
             $\theta_i \leftarrow -\pi$ 
        end if
    end repeat
     $rep \leftarrow rep + 1$ 
end while
return  $S$ 
end procedure

```

Figure 17 - La procédure SpiralChaoticMove

2.4 Adaptation de Galaxy-Based Search Algorithm (GbSA) sur notre problème d'atelier (JSSP)

3. Problème Job Shop JSSP

Nous allons implémenter l'algorithme de Galaxy-based Search pour résoudre notre problème d'ordonnancement d'atelier Job Shop JSSP, dont l'objectif est de trouver l'ordre optimal d'exécution de toutes ces opérations pour minimiser le temps maximal de fin d'exécution.

3.1.1 Structure de l'algorithme

- **Initialisation**

L'implémentation initialise plusieurs paramètres critiques : le nombre maximum d'itérations (2000), le nombre maximum de répétitions pour les mouvements spiraux (2000), le rayon initial du spiral (2.0), l'incrément du rayon (0.3), l'incrément de l'angle (0.05), et la valeur initiale de la fonction chaotique (0.4). Ces paramètres contrôlent respectivement la durée de la recherche, l'intensité de l'exploration locale, et la dynamique du mouvement spiral.

- **Algorithme principale**

Maintenant, l'algorithme principal débute par la génération d'une solution initiale suivie d'une phase de recherche locale pour établir une base de qualité. La boucle principale itère jusqu'au nombre maximum d'itérations défini en utilisant le mouvement spirale chaotique. La fonction chaotique **NextChaos()** est toujours utilisée sous cette forme : « $x0 = 4 * x0 * (1 - x0)$ »

Pour chaque opération dans la séquence, l'algorithme détermine le temps de début comme le maximum entre le temps de disponibilité de la machine requise et le temps de fin de l'opération précédente du même travail, et le **Makespan** final est le maximum des temps de fin de toutes les machines. À chaque itération, l'algorithme applique le mouvement spiral chaotique. Si ce mouvement produit une amélioration, une nouvelle phase de recherche locale est déclenchée pour exploiter la région prometteuse découverte. À la fin, l'algorithme retourne la meilleure solution après la complétion de toutes les itérations.

- **Génération de la solution initiale**

La génération de solutions initiales maintient une liste d'opérations disponibles, où une opération devient disponible uniquement après la programmation de toutes les opérations précédentes de son travail. Pour construire ces solutions, différentes règles d'ordonnancement sont utilisées :

- Séquence aléatoire valide des opérations (Random).

Procédure Random (Génération Aléatoire)

Entrée:

Jobs : Ensemble des travaux avec leurs opérations.

Sortie:

Sequence : Séquence d'opérations générée aléatoirement.

Algorithme:

```

Sequence ← ∅
OperationsDone ← [0] * NumJobs
AvailableOperations ← ∅

for job_id = 0 to NumJobs - 1 do
    if |Jobs[job_id]| > 0 then
        AvailableOperations ← AvailableOperations ∪ {(job_id, 0)}
    end if
end for

while AvailableOperations ≠ ∅ do
    // Sélection aléatoire d'une opération disponible
    randomIndex ← Random(0, |AvailableOperations| - 1)
    (job_id, op_id) ← Pop(AvailableOperations, randomIndex)
    Sequence ← Sequence ∪ {(job_id, op_id)}
    OperationsDone[job_id] ← OperationsDone[job_id] + 1

    if OperationsDone[job_id] < |Jobs[job_id]| then
        AvailableOperations ← AvailableOperations ∪ {(job_id, OperationsDone[job_id])}
    end if
end while

return Sequence

end procedure

```

Figure 18 - Procédure Random (Génération Aléatoire)

- SPT (Shortest Processing Time).

Procédure SPT (Shortest Processing Time)

Entrée:

Jobs : Ensemble des travaux avec leurs opérations et temps de traitement.

Sortie:

Sequence : Séquence d'opérations ordonnée par temps de traitement croissant.

Algorithme:

```

Sequence ← ∅
OperationsDone ← [0] * NumJobs
AvailableOperations ← ∅

for job_id = 0 to NumJobs - 1 do
    if |Jobs[job_id]| > 0 then
        AvailableOperations ← AvailableOperations ∪ {(job_id, 0)}
    end if
end for

while AvailableOperations ≠ ∅ do
    // Trier par temps de traitement (plus court en premier)
    Sort(AvailableOperations, key=ProcessingTime, ascending=True)
    (job_id, op_id) ← Pop(AvailableOperations, 0)
    Sequence ← Sequence ∪ {(job_id, op_id)}
    OperationsDone[job_id] ← OperationsDone[job_id] + 1

    if OperationsDone[job_id] < |Jobs[job_id]| then
        AvailableOperations ← AvailableOperations ∪ {(job_id, OperationsDone[job_id])}
    end if
end while

return Sequence

end procedure

```

Figure 19 - Procédure SPT (shortest processing Time)

- LPT (Longest Processing Time).

Procédure LPT (Longest Processing Time)

Entrée:

Jobs : Ensemble des travaux avec leurs opérations et temps de traitement.

Sortie:

Sequence : Séquence d'opérations ordonnée par temps de traitement décroissant.

Algorithme:

```

Sequence ← ∅
OperationsDone ← [0] * NumJobs
AvailableOperations ← ∅

for job_id = 0 to NumJobs - 1 do
    if |Jobs[job_id]| > 0 then
        AvailableOperations ← AvailableOperations ∪ {(job_id, 0)}
    end if
end for

while AvailableOperations ≠ ∅ do
    // Trier par temps de traitement (plus long en premier)
    Sort(AvailableOperations, key=ProcessingTime, descending=True)
    (job_id, op_id) ← Pop(AvailableOperations, 0)
    Sequence ← Sequence ∪ {(job_id, op_id)}
    OperationsDone[job_id] ← OperationsDone[job_id] + 1

    if OperationsDone[job_id] < |Jobs[job_id]| then
        AvailableOperations ← AvailableOperations ∪ {(job_id, OperationsDone[job_id])}
    end if
end while

return Sequence

end procedure

```

Figure 20 - Procédure LPT (Longest Processing Time).

- MOR (Most Operations Remaining).

Procédure MOR (Most Operations Remaining)

Entrée:

Jobs : Ensemble des travaux avec leurs opérations.

Sortie:

Sequence : Séquence d'opérations ordonnée par nombre d'opérations restantes décroissant.

Algorithme:

```

Sequence ← ∅
OperationsDone ← [0] * NumJobs
AvailableOperations ← ∅

for job_id = 0 to NumJobs - 1 do
    if |Jobs[job_id]| > 0 then
        AvailableOperations ← AvailableOperations ∪ {(job_id, 0)}
    end if
end for

while AvailableOperations ≠ ∅ do
    // Trier par nombre d'opérations restantes (plus grand en premier)
    Sort(AvailableOperations, key=RemainingOperations, descending=True)
    // où RemainingOperations(x) = |Jobs[x[0]]| - OperationsDone[x[0]]
    (job_id, op_id) ← Pop(AvailableOperations, 0)
    Sequence ← Sequence ∪ {(job_id, op_id)}
    OperationsDone[job_id] ← OperationsDone[job_id] + 1

    if OperationsDone[job_id] < |Jobs[job_id]| then
        AvailableOperations ← AvailableOperations ∪ {(job_id, OperationsDone[job_id])}
    end if
end while

return Sequence

end procedure

```

Figure 21 - Procédure MOR (Most Operations Remaining).

On assure que chaque opération ne peut commencer qu'après la fin de l'opération précédente du même travail.

- **Mouvement spiral chaotique**

Ce mouvement constitue le mécanisme d'exploration principal de l'algorithme. Chaque position dans la séquence se voit attribuer un angle initial généré aléatoirement. À chaque itération,

l'algorithme calcule de nouvelles positions selon la formule basée sur le rayon courant et l'angle associé à chaque élément.

Le processus teste les échanges dans les deux directions du mouvement spiral, et voici les deux fonctions de ces mouvements :

Premier mouvement avec l'addition

$S_{Next} \leftarrow (S_i + NextChaos() * r * \cos(\theta_i)) \% n$

Deuxième mouvement avec la soustraction :

$S_{Next} \leftarrow (S_i - NextChaos() * r * \cos(\theta_i)) \% n$

Si aucune amélioration n'est détectée, le rayon augmente selon l'incrément défini, et tous les angles sont mis à jour. La bidirectionnalité du mouvement double les opportunités d'amélioration à chaque itération.

- **Recherche locale**

La recherche locale améliore itérativement les solutions existantes en explorant son voisinage local. Nous avons repris l'algorithme original et l'avons modifié pour l'adapter à notre problème **JSSP** (voir figure 19). Pour chaque composant de la solution, notre algorithme de recherche locale modifié applique des structures de voisinage à la solution initiale via la fonction **ApplyNeighborhoodStructures()** pour explorer différentes configurations. Ensuite, il génère deux solutions candidates (une supérieure et une inférieure) en ajoutant et soustrayant un pas de recherche adaptatif, et il utilise la fonction génératrice de nombres chaotiques **NextChaos()** pour introduire une perturbation contrôlée et éviter les minima locaux. Le paramètre α s'ajuste dynamiquement selon la qualité des solutions trouvées : il augmente modérément (0.01) quand une meilleure solution est découverte, ou plus fortement (0.05) en cas de stagnation.

Procédure LocalSearch

Entrée :

S : solution actuelle avec L composants tels que S_i désigne le ième composant de la solution S.

Sortie :

SNext : sortie de la recherche locale.

Algorithme :

```
ΔS ← ΔSmax × NextChaos()
S ← ApplyNeighborhoodStructures(S)

Repeat for i = 1 to L
  α ← 1, k ← 0

  While k < kMax
    SLi ← Si - α × ΔS
    SUi ← Si + α × ΔS

    If (m(SU) < m(S)) then
      ✓ Si ← SUi
      ✓ SLi ← SUi
      ✓ α ← α + 0.01 × NextChaos()
      ✓ k ← k + 1

    Else If (m(SL) < m(S)) then
      ✓ Si ← SLi
      ✓ SUi ← SLi
      ✓ α ← α + 0.01 × NextChaos()
      ✓ k ← k + 1

    Else
      ✓ α ← α + 0.05 × NextChaos()
      ✓ k ← k + 1

  End while

  SLi ← Si, SUi ← Si

End repeat

SNext ← S

End procedure
```

Figure 22 - Procédure Local Search

Les types de voisinage utilisés sont :

Voisinage d'insertion :

Il retire un travail de sa position et l'insère ailleurs dans la position d'autre travail en testant toutes les positions possibles d'insertion.

Procédure Voisinage_Insertion

Entrée: séquence S.

Sortie: meilleure_séquence.

Algorithme:

```
meilleure_séq ← S
meilleur_makespan ← makespan(S)
L ← longueur(S)
  Pour i = 0 à L-1
    Pour j = 0 à L-1 (j ≠ i)
      nouvelle_séq ← copie(S)
      job ← retirer(nouvelle_séq[i])
      insérer(job, position j)
      makespan ← calculer_makespan(nouvelle_séq)
      makespan < meilleur_makespan ?
      Oui: meilleur_makespan ← makespan
      meilleure_séq ← nouvelle_séq
  retourner meilleure_séq
```

Figure 23 - Procédure Voisinage Insertion

Voisinage d'échange :

Ce voisinage échange deux travaux sélectionnés aléatoirement en testant aussi tous les échanges possibles.

Procédure Voisinage_Échange

Entrée: séquence S.

Sortie: meilleure_séquence.

Algorithme:

```
meilleure_séq ← S
meilleur_makespan ← makespan(S)
L ← longueur(S)
  Pour i = 0 à L-1
    Pour j = i+1 à L-1
      nouvelle_séq ← copie(S)
      échanger nouvelle_séq[i] ↔ nouvelle_séq[j]
      makespan ← calculer_makespan(nouvelle_séq)
      makespan < meilleur_makespan ?
      Oui: meilleur_makespan ← makespan
      meilleure_séq ← nouvelle_séq
  retourner meilleure_séq
```

Figure 24 - Procédure Voisinage Échange

Insertion de bloc de deux travaux :

Déplace un bloc de deux travaux consécutifs vers une autre position en maintenant l'ordre relatif des travaux dans le bloc.

Procédure Voisinage_Insertion_Bloc_2_Jobs

Entrée: séquence S.

Sortie: meilleure_séquence.

Algorithme:

```
meilleure_séq ← S
meilleur_makespan ← makespan(S)
L ← longueur(S)
  Pour i = 0 à L-2
    bloc ← S[i:i+2]
    séq_temp ← S sans bloc
    Pour j = 0 à L-2
      nouvelle_séq ← copie(séq_temp)
      insérer bloc à position j
      makespan ← calculer_makespan(nouvelle_séq)
      makespan < meilleur_makespan ?
    Oui: meilleur_makespan ← makespan
    meilleure_séq ← nouvelle_séq
  retourner meilleure_séq
```

Figure 25 - Procédure Voisinage_Insertion_Bloc_2_Jobs

Insertion de bloc de trois travaux :

Similaire au précédent mais avec des blocs de trois travaux.

Procédure Voisinage_Insertion_Bloc_3_Jobs

Entrée: séquence S.

Sortie: meilleure_séquence.

Algorithme:

```
L ← longueur(S)
L < 3 ?
Oui: retourner S
Non: meilleure_séq ← S
meilleur_makespan ← makespan(S)
  Pour i = 0 à L-3
    bloc ← S[i:i+3]
    séq_temp ← S sans bloc
    Pour j = 0 à L-3
      nouvelle_séq ← copie(séq_temp)
      insérer bloc à position j
      makespan ← calculer_makespan(nouvelle_séq)
      makespan < meilleur_makespan ?
    Oui: meilleur_makespan ← makespan
    meilleure_séq ← nouvelle_séq
  retourner meilleure_séq
```

Figure 26 - Procédure Voisinage_Insertion_Bloc_3_Jobs

Inversion de bloc :

Inverse l'ordre d'un bloc de 4 à 10 travaux et l'insère inverser à une nouvelle position.

Procédure Voisinage_Inversion_Bloc_Aléatoire

Entrée: séquence S.

Sortie: meilleure_séquence.

Algorithme:

```
meilleure_séq ← S
meilleur_makespan ← makespan(S)
L ← longueur(S)
taille_min ← min(4, L)
  taille_min < 2 ?
Oui: retourner meilleure_séq
Non: taille_max ← min(10, L)
taille_bloc ← aléatoire(taille_min, taille_max)
  L ≤ taille_bloc ?
Oui: nouvelle_séq ← inverser(S)
  si makespan(nouvelle_séq) < meilleur_makespan
  alors retourner nouvelle_séq
  sinon retourner meilleure_séq
Non: Pour i = 0 à L-taille_bloc
  bloc ← S[i:i+taille_bloc]
  inverser(bloc)
  nouvelle_séq ← S[:i] + bloc + S[i+taille_bloc:]
  makespan ← calculer_makespan(nouvelle_séq)
  makespan < meilleur_makespan ?
Oui: meilleur_makespan ← makespan
  meilleure_séq ← nouvelle_séq
retourner meilleure_séq
```

Figure 27 - Procédure Voisinage_Inversion_Bloc_Aléatoire

- **Résultats**

Enfin, l'algorithme génère un ordonnancement détaillé spécifiant les temps de début et de fin de chaque opération, ainsi que l'assignation des machines. Cette information permet de calculer le taux d'utilisation de chaque machine, défini comme le rapport entre le temps de traitement effectif et le **Makespan** total avec cette formule :

$$\text{Utilisation} = (\text{temps occupé} / \text{meilleur Makespan}) * 100$$

Aussi, l'algorithme représente cet ordonnancement graphiquement par le diagramme de Gantt, où chaque travail est identifié par une couleur distincte et chaque barre horizontale indique la durée et le minutage d'une opération.

3.2 Conclusion

Dans ce chapitre nous avons présenté le GBSA (Galaxy Based Search) et ses procédures. Par la suite, nous avons présenté une adaptation sur un problème SMTWT pour comprendre le fonctionnement algorithme GBSA, et puis notre propre adaptation sur notre problème Job Shop.

CHAPITRE 3 : Implémentation et Résultats

4.1 Introduction

Ce chapitre est consacré à la partie implémentation et de notre interface qui s'appuie sur l'adaptation proposer de notre problème job shop présenté dans le chapitre précédent. Nous allons présenter dans ce qui suit les différents outils et langage utilisés lors de l'implémentation puis nous présentons les résultats.

4.2 Les outils

- **Python [31]** : est le langage de programmation open source le plus employé par les informaticiens. Ce langage s'est propulsé en tête de la gestion d'infrastructure, d'analyse de données ou dans le domaine du développement de logiciels. En effet, parmi ses qualités, Python permet notamment aux développeurs de se concentrer sur ce qu'ils font plutôt que sur la manière dont ils le font. Il a libéré les développeurs des contraintes de formes qui occupaient leur temps avec les langages plus anciens. Ainsi, développer du code avec Python est plus rapide qu'avec d'autres langages.
- **Qt Framework [32]** : Qt est un Framework de développement logiciel complet qui permet de créer des applications multi-plateformes avec une interface graphique (GUI) sophistiquée
- **PyQt[33]** : Une Librairie Qt spécialement adaptée pour le langage Python. C'est un outil puissant qui permet de créer des applications avec des interfaces graphiques sophistiquées en utilisant la simplicité de Python
- **Qt Creator [34]** : Un environnement de développement intégré (IDE) complet qui permet de créer des applications Qt de manière efficace. C'est l'outil principal pour développer des applications avec le Framework Qt.
- **Zed Editor [35]** : Zed est un éditeur de code de nouvelle génération source ouverte entièrement écrit avec le langage de programmation Rust conçu pour une collaboration performante avec les humains et l'IA.

Présentation Générale d'application

Cette application constitue un système d'aide à la décision pour la résolution du problème d'ordonnancement d'ateliers (Job Shop Scheduling Problem - JSSP) à travers l'implémentation d'algorithme Galaxy-based Search Algorithm (GBSA). L'architecture logicielle s'appuie sur le langage de programmation Python pour le fonctionnement d'algorithme, et le framework QT pour offrir une interface utilisateur graphique moderne et intuitive, permettant aux utilisateurs de définir, résoudre et visualiser des problèmes d'ordonnancement complexes. Le système intègre des fonctionnalités avancées de visualisation, notamment la génération automatique de diagrammes de Gantt interactifs, ainsi qu'un ensemble complet d'outils d'analyse des performances.

L'interface utilisateur a été conçue selon une approche modulaire, organisant les fonctionnalités en trois onglets distincts correspondant aux différentes phases du processus de résolution : **la définition du problème, l'affichage des résultats, et la visualisation graphique.**

4.3 Interface de Saisie des Données

L'onglet de saisie du problème ("Saisie du problème") offre une interface complète pour la définition des paramètres d'ordonnancement. La section de définition du problème permet de spécifier le nombre de jobs et le nombre de machines à travers des composants de saisie numérique dédiés. Le système propose une méthode de chargement des données avec l'importation depuis des fichiers externes aux formats JSON ou texte, et propose aussi l'utilisation d'un problème d'exemple prédéfini de 3x3 classique.

La section de définition des jobs s'adapte automatiquement au nombre de jobs spécifié. Chaque job dispose de sa propre table de saisie permettant de définir le nombre d'opérations et, pour chaque opération, la machine assignée ainsi que le temps de traitement correspondant. L'interface intègre des zones de défilement pour gérer efficacement les problèmes de grande taille.

Les paramètres algorithmiques sont configurables à travers une section dédiée permettant de spécifier le nombre maximal d'itérations de 10 à 10000. Enfin, le processus est lancé via un bouton de résolution qui déclenche l'exécution de l'algorithme en arrière-plan, accompagné d'une barre de progression et d'une zone de texte affichant les messages de progression en temps réel.

Saisie du problème
Résultats
Visualisation Graphique

Définition du problème

Nombre de Jobs: 3
Nombre de Machines: 3

Charger le problème d'exemple
Charger à partir d'un fichier

Définition des Jobs

Job 1

Nombre d'opérations: 3

	Machine	mps de traitement
1	0	3
2	1	3

Job 2

Nombre d'opérations: 3

	Machine	mps de traitement
1	0	1
2	2	5

Job 3

Nombre d'opérations: 3

	Machine	mps de traitement
1	1	3
2	0	2

Paramètres d'algorithme

Nombre maximal d'itérations: 100

Résoudre le problème

Iteration en cours: 90/100
Iteration en cours: 100/100
Algorithme terminé après 100 itérations. Meilleur makespan finale: 10
Temps d'exécution total: 11.58 secondes

Figure 28 Interface de Saisie des Données

4.4 Interface d'Affichage des Résultats

L'onglet des résultats ("Résultats") présente la solution optimale obtenue. La section de résumé affiche les métriques principales : le makespan (temps de complétion total), le temps d'exécution de l'algorithme, le nombre total d'opérations, et l'utilisation moyenne des machines. Le tableau d'utilisation des machines offre les pourcentages d'utilisation de chaque machine avec une précision de deux décimales, et les colonnes s'ajustent automatiquement pour accommoder les différents nombres de machines.

La table de planning détaillé constitue l'élément central de l'analyse, présentant chaque opération avec ses caractéristiques : identifiant du job, numéro d'opération, machine assignée, temps de début, temps de fin, et durée.

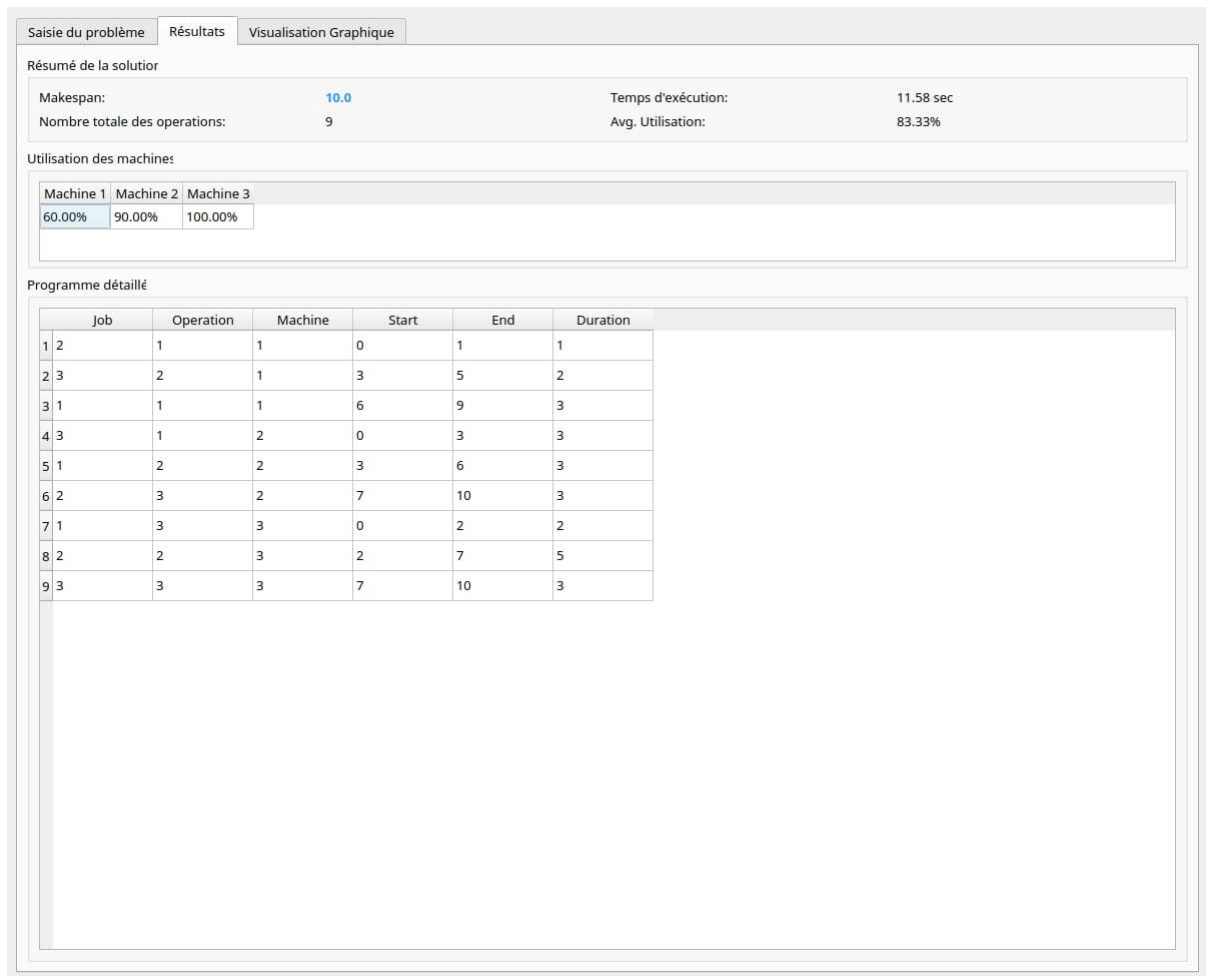


Figure 29 - Interface d'Affichage des Résultats

4.5 Interface de Visualisation Graphique

L'onglet de diagramme de Gantt ("Visualisation Graphique") propose une visualisation interactive et flexible de la solution d'ordonnancement. Les contrôles de dimensionnement permettent d'ajuster dynamiquement la largeur de 8 à 50 unités et la hauteur de 4 à 30 unités du graphique, avec une mise à jour en temps réel de l'affichage.

Le diagramme de Gantt intègre un codage couleur automatique assignant une couleur distincte à chaque job en utilisant la palette de couleurs de matplotlib. Chaque opération est représentée par un bloc rectangulaire positionné selon son temps de début et sa durée, avec une étiquette identifiant le job et l'opération correspondante. Les pourcentages d'utilisation des machines sont affichés à droite de chaque ligne de machine.

L'interface de visualisation intègre la barre d'outils **Matplotlib** complète, offrant des fonctionnalités avancées de zoom, de panoramique, et de sauvegarde, et permet de naviguer efficacement dans les diagrammes de grande taille.

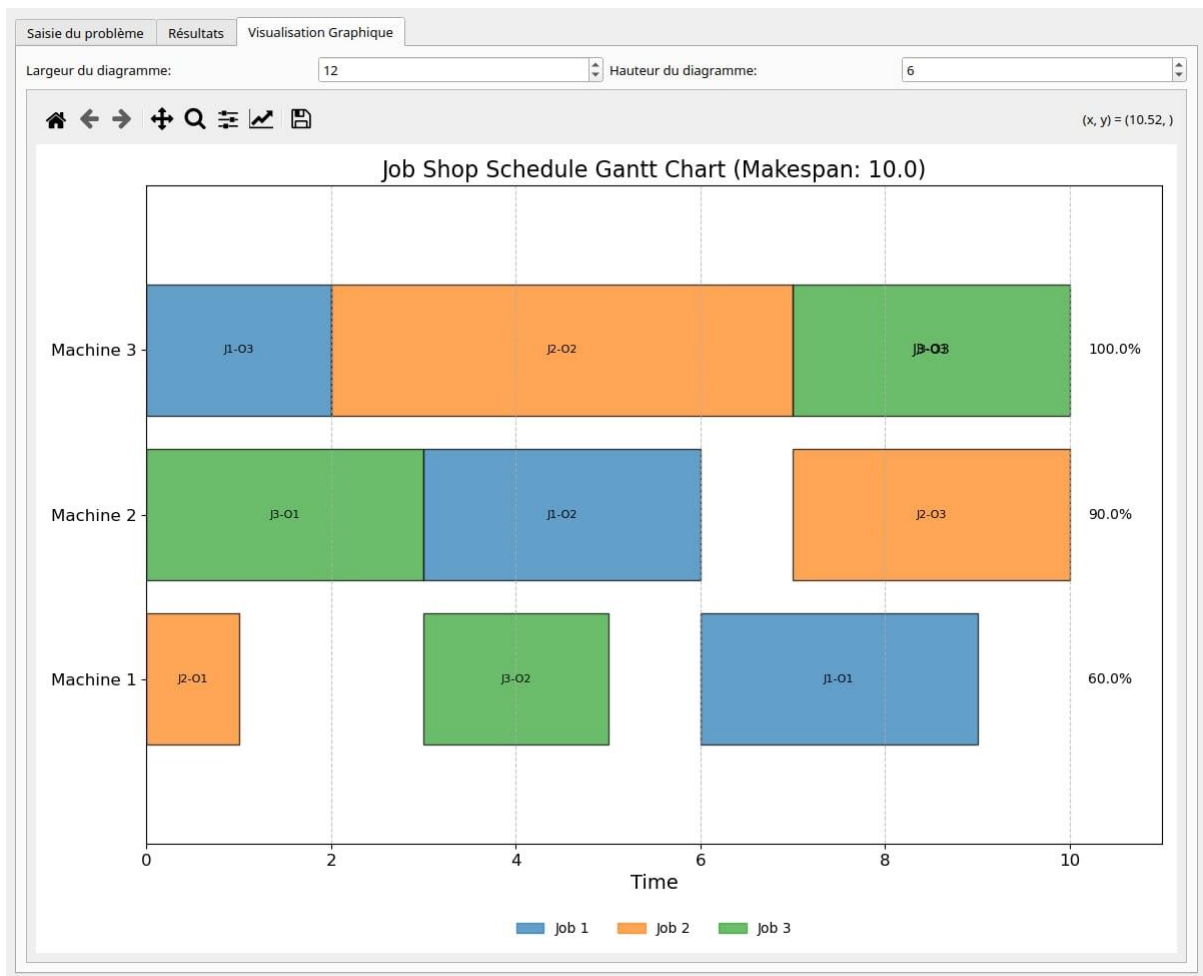


Figure 30 Interface de Visualisation Graphique

Figure 31 - Interface de la visualisation graphique

4.6 Résultats des problèmes d'ordonnancement d'ateliers (JSSP)

D'abord, on a appliqué notre l'algorithme sur 70 problèmes pris d'OR-Library (OR-Lib par J. E. Beasley) Job Shop Scheduling Benchmarks, et on a utilisé de la même source des instances de référence standard pour les problèmes d'ordonnancement d'atelier pour comparer les résultats, et ces instances sont largement utilisées en recherche opérationnelle depuis 1993, et il s'appelle : ta01 à ta80 (regroupées sous ta01-05, ta06-10, etc..). Ces instances varient en taille de 15 Jobs $\times$ 15 machines (petite échelle) à 50 Jobs $\times$ 20 machines (grande échelle). Pour chaque instance on a utilisé 2000 itérations du l'algorithme GbSA avec une seule solution dans chaque population.

En essaie de trouver des solutions avec le minimum Makespan (temps total de réalisation) sous des contraintes strictes : chaque travail doit être traité sur chaque machine exactement une fois, sans qu'aucune machine ne traite plusieurs travaux simultanément. On a fait l'execution de l'application qui est implementé en Python dans un laptop avec un i5-6300U 2.4GHz processeur

deux threads et une RAM de 8GB, et le système d'exploitation utilisé c'est Arch Linux une distribution de Linux.

On a calculé l'écart qui représente la différence en pourcentage entre nos solutions et les benchmarks de référence avec cette formule :

$$\text{Écart (\%)} = (\text{Résultat de l'algorithme} - \text{Meilleure solution}) / \text{Meilleure solution} \times 100$$

4.6.1 Instances 15×15

Instance	Makespan	Écart (%)	Meilleure solution
JSSP_15x15_1	1332	8,20%	1231
JSSP_15x15_2	1373	10,35%	1244
JSSP_15x15_3	1340	10,01%	1218
JSSP_15x15_4	1401	19,23%	1175
JSSP_15x15_5	1445	18,05%	1224
JSSP_15x15_6	1445	16,72%	1238
JSSP_15x15_7	1418	15,56%	1227
JSSP_15x15_8	1371	12,65%	1217
JSSP_15x15_9	1492	17,11%	1274
JSSP_15x15_10	1321	6,44%	1241

Figure 32 - Instances 15x15

4.6.2 Instances 20×15

Instance	Makespan	Écart (%)	Meilleure solution
JSSP_20x15_1	1630	23,20%	1323
JSSP_20x15_2	1665	23,24%	1351
JSSP_20x15_3	1639	27,84%	1282
JSSP_20x15_4	1512	12,41%	1345
JSSP_20x15_5	1580	21,16%	1304
JSSP_20x15_6	1615	23,84%	1304
JSSP_20x15_7	1688	15,45%	1462
JSSP_20x15_8	1794	31,04%	1369
JSSP_20x15_9	1605	23,08%	1304
JSSP_20x15_10	1662	26,01%	1318

Figure 33 - Instances 20x15

4.6.3 Instances 20×20

Instance	Makespan	Écart (%)	Meilleure solution
JSSP_20x20_1	1948	23,83%	1573
JSSP_20x20_2	1995	29,37%	1542
JSSP_20x20_3	1871	26,93%	1474
JSSP_20x20_4	1946	21,17%	1606
JSSP_20x20_5	1969	29,71%	1518
JSSP_20x20_6	2026	30,03%	1558
JSSP_20x20_7	2141	32,40%	1617
JSSP_20x20_8	2076	30,48%	1591
JSSP_20x20_9	1928	26,42%	1525
JSSP_20x20_10	1932	30,10%	1485

Figure 34 - Instances 20x20

4.6.4 Instances 30×15

Instance	Makespan	Écart (%)	Meilleure solution
JSSP_30x15_1	2088	18,36%	1764
JSSP_30x15_2	2306	29,98%	1774
JSSP_30x15_3	2266	27,44%	1778
JSSP_30x15_4	2244	22,75%	1828
JSSP_30x15_5	2192	9,21%	2007
JSSP_30x15_6	2421	33,09%	1819
JSSP_30x15_7	2226	25,69%	1771
JSSP_30x15_8	2127	27,13%	1673
JSSP_30x15_9	2207	22,95%	1795
JSSP_30x15_10	2085	27,83%	1631

Figure 35 - Instances 30x15

4.6.5 Instances 30×20

Instance	Makespan	Écart (%)	Meilleure solution
JSSP_30x20_1	2681	30,02%	1876
JSSP_30x20_2	2532	26,34%	1867
JSSP_30x20_3	2524	23,32%	1809
JSSP_30x20_4	2683	28,17%	1927
JSSP_30x20_5	2566	22,17%	1997
JSSP_30x20_6	2576	24,68%	1940
JSSP_30x20_7	2523	29,09%	1789
JSSP_30x20_8	2523	24,21%	1912
JSSP_30x20_9	2585	25,91%	1915
JSSP_30x20_10	2420	25,33%	1807

Figure 36 - Instances 30x20

4.6.6 Instances 50×15

Instance	Makespan	Écart (%)	Meilleure solution
JSSP_50x15_1	3306	19,78%	2760
JSSP_50x15_2	3409	23,69%	2756
JSSP_50x15_3	3051	12,29%	2717
JSSP_50x15_4	3222	13,49%	2839
JSSP_50x15_5	3392	26,61%	2679
JSSP_50x15_6	3529	26,89%	2781
JSSP_50x15_7	3353	13,93%	2943
JSSP_50x15_8	3406	18,05%	2885
JSSP_50x15_9	3319	25,00%	2655
JSSP_50x15_10	3110	14,21%	2723

Figure 37 - Instances 50x15

4.6.7 Instances 50×20

Instance	Makespan	Écart (%)	Meilleure solution
JSSP_50x20_1	3633	26,67%	2868
JSSP_50x20_2	3711	29,34%	2869
JSSP_50x20_3	3473	26,06%	2755
JSSP_50x20_4	3476	28,64%	2702
JSSP_50x20_5	3570	31,00%	2725
JSSP_50x20_6	3642	28,01%	2845
JSSP_50x20_7	3763	33,20%	2825
JSSP_50x20_8	3621	30,06%	2784
JSSP_50x20_9	3748	22,04%	3071
JSSP_50x20_10	3873	29,31%	2995

Figure 38 - Instances 50x20

3.4.8 Analyse des résultats:

Les instances 15×15 offrent les meilleurs résultats avec un écart moyen de 13,43%. La performance est stable et homogène, avec le meilleur cas à 6,44% (JSSP_15x15_10) et le pire à 19,23% (JSSP_15x15_4).

Les instances 20×15 présentent une performance modérée (écart moyen 22,64%) mais avec une forte variabilité. JSSP_20x15_4 se distingue à 12,41% tandis que JSSP_20x15_8 atteint 31,04%.

Les instances 20×20 s'avèrent difficiles avec un écart moyen de 28,04%. Le format carré pose plus de défis, avec des résultats généralement entre 25-32%.

La catégorie 30×15 montre la plus grande variabilité (écart moyen 24,44%). Les résultats vont d'excellent (9,21% pour JSSP_30x15_5) à difficile (33,09% pour JSSP_30x15_6).

Les instances 30×20 sont plus cohérentes avec un écart moyen de 25,92%. Les résultats restent groupés entre 22-30% sans cas exceptionnels.

Étonnamment, les instances 50×15 affichent de bonnes performances (écart moyen 19,39%) malgré leur taille. L'algorithme gère bien l'augmentation des tâches avec peu de machines. Les instances 50×20 constituent le plus grand défi avec l'écart moyen le plus élevé (28,30%). Ces grandes instances avec nombreuses machines représentent la limite de l'algorithme. L'analyse révèle que la performance se dégrade plus avec l'augmentation des machines qu'avec celle des tâches.

4.7 Conclusion

Dans ce dernier chapitre on a détaillé l'implémentation du GbSA pour résoudre le JSSP en utilisant des outils comme Python, Qt Framework, PyQt, Qt Creator et Zed Editor. Les résultats, illustrés par des diagrammes de Gantt, montrent que l'algorithme propose des solutions viables bien qu'encore perfectibles.

Conclusion générale

Ce mémoire s'inscrit dans le cadre du projet de fin d'études en vue de l'obtention du diplôme de Master.

Ce projet a présenté une adaptation innovante du Galaxy-based Search Algorithm (GbSA) pour résoudre le problème complexe d'ordonnancement Job Shop Scheduling (JSSP). Inspiré par les mouvements des galaxies, le GbSA a été choisi pour sa capacité à équilibrer exploration et exploitation dans des espaces de recherche vastes et complexes. Les résultats obtenus, bien que perfectibles, démontrent le potentiel de cette approche métaheuristique pour optimiser les systèmes de production.

Les contributions principales de ce travail incluent :

L'adaptation du GbSA au JSSP, avec des mécanismes améliorés pour gérer les contraintes spécifiques de ce problème NP-difficile.

L'implémentation pratique d'une interface utilisateur intuitive, développée avec Python et Qt, permettant de visualiser les solutions via des diagrammes de Gantt et d'analyser les performances.

Une évaluation expérimentale mettant en lumière les forces et les limites de l'algorithme proposé, ouvrant la voie à des améliorations futures.

Malgré des résultats encore non compétitifs par rapport aux méthodes existantes, cette recherche offre des perspectives prometteuses pour l'optimisation des systèmes de production. Les travaux futurs pourraient explorer des hybridations avec d'autres métaheuristiques ou l'intégration de techniques d'apprentissage automatique pour renforcer l'efficacité du GbSA. La réalisation de ce projet a été l'occasion d'enrichir considérablement nos connaissances, de renforcer nos compétences méthodologiques et de nous initier concrètement au travail de recherche

Bibliographie

- [1] Pinedo, M. (2016). Scheduling: Theory, Algorithms, and Systems (5th ed.). Springer.
- [2] Garey, M. R., & Johnson, D. S. (1979). Computers and Intractability: A Guide to

- the Theory of NP-Completeness. W.H. Freeman.
- [3] Jain, A. S., & Meeran, S. (1999). Deterministic Job-Shop Scheduling: Past, Present and Future. *European Journal of Operational Research*, 113(2), 390–434.
 - [4] Zhang, R., et al. (2020). Real-World Job Shop Scheduling: A Survey. *International Journal of Production Research*, 58(15), 4431–4467.
 - [5] Lawler, E. L., et al. (1993). Sequencing and Scheduling: Algorithms and Complexity. *Handbooks in Operations Research and Management Science*, 4, 445–522.
 - [6] Brucker, P. (2007). *Scheduling Algorithms* (5th ed.). Springer.
 - [7] Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation*. Wiley.
 - [8] Shah-Hosseini, H. (2019). Galaxy-Based Search Algorithm: A Novel Metaheuristic Inspired by Galactic Motion. *Soft Computing*, 23(12), 4361–4378.
 - [9] Karami, H., & Anaraki, M. V. (2021). Improved GbSA for Combinatorial Optimization: Applications to Job Shop Scheduling. *Applied Soft Computing*, 105, 107245.
 - [10] Glover, F., & Kochenberger, G. (2003). *Handbook of Metaheuristics*. Springer.
 - [11] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.
 - [12] Taillard, E. (1993). Benchmarks for Basic Scheduling Problems. *European Journal of Operational Research*, 64(2), 278–285.
 - [13] Framinan, J. M., et al. (2014). A Survey of Scheduling in Dynamic Job Shops. *International Journal of Production Research*, 52(11), 3137–3158.
 - [14] Ouelhadj, D., & Petrovic, S. (2009). A Survey of Dynamic Scheduling in Manufacturing Systems. *Journal of Scheduling*, 12(4), 417–431.
 - [15] Zhang, G., et al. (2022). Metaheuristics for Industry 4.0: A Review. *Computers & Industrial Engineering*, 168, 108123.
 - [16] Baker, K. R., & Trietsch, D. (2009). *Principles of Sequencing and Scheduling*. Wiley.
 - [17] Tangour, C., & Toumi, S. (2015). Classification des contraintes en ordonnancement : une revue. *Journal of Operational Research*, 45(2), 89–104.
 - [18] Rajendran, C., & Holthaus, O. (1999). A comparative study of dispatching rules in dynamic flowshops and jobshops. *European Journal of Operational Research*, 116(1), 156–170.
 - [19] Bai, D., et al. (2012). A hybrid algorithm for flexible open shop scheduling. *Computers & Industrial Engineering*, 63(1), 62–71.

- [20] Kirkpatrick, S., et al. (1983). Optimization by simulated annealing. *Science*, 220(4598), 671–680.
- [21] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5), 533–549.
- [22] Zahmani, M. H. (2020). An adaptation of the galaxy-based search algorithm for solving the single machine total weighted tardiness problem*. *International Journal of Manufacturing Research*.
- [23] Kanet, J. J., & Li, X.(2004). A weighted modified due date rule for sequencing to minimize weighted tardiness. *Journal of Scheduling*.
- [24] Yin, A., & Wang, J. (2013). Mixed dispatch rule for single machine total weighted tardiness problem*. *Journal of Applied Sciences*.
- [25] Yoon, S. H., & Lee, I. S.(2011). New constructive heuristics for the total weighted tardiness problem*. *Journal of the Operational Research Society*.
- [26] Chung, T., Fu, Q., Liao, C., & Liu, Y.(2017).Multiple-variable neighborhood search for the single-machine total weighted tardiness problem. *Engineering Optimization*.
- [27] Fatih Tasgetiren, M., Liang, Y.-C., Sevkli, M., & Geneyilmaz, G.(2006). Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem. *International Journal of Production Research*.
- [28] Subramanian, A., Battarra, M., & Potts, C. N.(2014). An iterated local search heuristic for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times. *International Journal of Production Research*.
- [29] Yang J, Sun L. Clonal selection based memetic algorithm for job shop scheduling problems. *Journal of Bionic Engineering* 2008;5:111-9.
- [30] H. Shah-Hosseini, "Principal Components Analysis by the Galaxy-Based Search Algorithm: A Novel Metaheuristic for Continuous Optimization", *International Journal of Computational Science and Engineering* (2011).
- [31] : <https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1445304-python-definition-et-utilisation-de-ce-langage-informatique/>
- [32] : <https://www.sitepoint.com/developing-cross-platform-applications-qt/>
- [33] : <https://www.pythonguis.com/pyside6-tutorial/>
- [34]: <https://scythe-studio.com/en/blog/what-is-qt-framework-and-how-to-create-gui-with-it>
- [35] : <https://zed.dev/>

[36]:http://mistic.heigvd.ch/taillard/problemes.dir/ordonnancement.dir/jobshop.dir/best_lb_up.txt

الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي

جامعة عبد الحميد بن باديس - مستغانم -
كلية العلوم الدقيقة والإعلام الآلي

تصريح شرفي خاص بالالتزام بقواعد النزاهة العلمية
لإنجاز البحث

أنا الممضي أدناه،
الطالب(ة): *صابر أحمد أمين*
الجامعي: *202032017674*
الحامل لبطاقة التعريف الوطنية رقم: *118225014* والصادرة بتاريخ:
..... *2020/08/04*
عن *مستغانم*
المسجل ب: *رياضيات وعلوم آلي*
شعبة: *علوم آلي*
..... *هندسة أنظمة المعلومات*
والمكلف بإنجاز مذكرة ماستر بعنوان:
..... *Optimisation de systèmes de production : adaptation*
..... *du « Galaxy based Search Algorithm » pour*
..... *le Job Shop*
أصرح بشرفي أنني ألتزم بمراعاة المعايير العلمية والمنهجية ومعايير
الأخلاقيات العلمية والنزاهة الأكاديمية المطلوبة في إنجاز البحث ،
وأتحمل المسؤولية الشخصية عن كل المحتوى المتضمن في البحث المذكور
أعلاه .

التاريخ: *2023/06/23* م

إمضاء المعني

الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي

جامعة عبد الحميد بن باديس-مستغانم-
كلية العلوم الدقيقة والإعلام الآلي

تصريح شرقي خاص بالالتزام بقواعد النزاهة العلمية
لإنجاز البحث

أنا الممضي أدناه،

الطالب(ة): راشد رشيد رقم التسجيل الجامعي: 191937050611

الحامل لبطاقة التعريف الوطنية رقم: 419988728 والصادرة بتاريخ: 0210112023

عن بلدية: مستغانم مستغانم

المسجل ب: كلية العلوم الدقيقة والإعلام الآلي / قسم: نظم المعلومات والإعلام الآلي

شعبة: (الإعلام الآلي) / التخصص: هندسة أنظمة المعلومات

والمكلف بإنجاز مذكرة ماستر بعنوان:

optimisation du system de production : Adaptation du « Galaxy
based Search Algorithm » pour le Job Shop

أصرح بشرقي أنني ألتزم بمراعاة المعايير العلمية والمنهجية ومعايير الأخلاقيات العلمية والنزاهة الأكاديمية
المطلوبة في إنجاز البحث ، وأتحمل المسؤولية الشخصية عن كل المحتوى المتضمن في البحث المذكور أعلاه .

التاريخ: 28/06/2024

إمضاء المعني