F.S.D .... N°D'ordre : .....

Université Saâd DAHLAB de Blida
Faculté des Sciences
Département d'informatique

Mémoire présenté par :

M^elle^ Fatima Zahra DJIROUN.

M^elle^ Habiba ALILI.

En vue d'obtenir le diplôme de Master

Domaine : Mathématique et Informatique.

Filière : Informatique.

Spécialité : Informatique.

Option : Ingénierie des logiciels.

Sujet :

Mise en œuvre multi-agents d'une approche par recherche d'harmonies pour la résolution des problèmes d'ordonnancement flexibles

Organisme d'accueil : Centre de Développement des Technologies Avancées (CDTA)

Soutenue le : 01 /07/2012 devant le jury composé de:

Mr. GAHAM Mehdi	Promoteur.
Mr. HADJ YAHYA Wahid	Président.
Mr. OULD AISSA	Examinateur 1.
M^elle^. MEZZI Melyara	Examinatrice2.

2011/2012

Remerciements

Avant tout, nous remercions ALLAH Le tout Puissant pour nous avoir donné la force, le courage et l'abnégation d'accomplir ce travail, qui a vu le jour grâce à Lui, ALHAMDOLILAH…

Nos vifs et sincères remerciements
S'adressent aussi à notre promoteur et encadreur
Mr GAHAM Mehdi
Qu'on a eu la chance de rencontrer et travailler avec, et qui a bien voulu nous confier ce travail riche en expériences et nous guider dans chaque étape de sa consécration.
Nous apprécions la spontanéité avec laquelle il a accepté de nous encadrer.
Son sérieux, sa compétence et son sens du devoir nous ont énormément marquées.
Ses encouragements inlassables et patience méritent toute notre gratitude.
Veuillez trouver ici l'expression de notre respect, notre estime et notre considération.

Nous remercions également
Les membres du jury,
De nous avoir fait l'honneur de juger cette thèse.
Veuillez accepter l'expression de notre vive gratitude et notre parfaite déférence.

Ainsi que
Melle MEGHATRIA Farida
D'avoir accepté d'évaluer la partie théorique de notre travail malgré ses obligations professionnelles.
Veuillez accepter l'expression de notre admiration profonde.

Enfin,
A toute personne qui a contribué de prés ou de loin, d'une manière directe ou indirecte à l'élaboration de ce travail de fin d'étude.

…

Toutes les lettres ne sauraient trouver les mots qu'il faut …
Tous les mots ne sauraient exprimer la gratitude, l'amour, le respect, la reconnaissance …
Aussi, c'est tout simplement que …

Je dédie cette thèse …

A mon très cher père et ma très chère maman…

Tous les mots du monde ne sauraient exprimer l'immense amour que je vous porte, ni la profonde gratitude que je vous témoigne pour tous les efforts et les sacrifices que vous n'avez jamais cessé de consentir pour mon instruction et mon bien-être.
Grâce à votre bienveillance, à votre encouragement et à votre générosité, j'ai pu terminer mes études dans l'enthousiasme.
J'espère avoir répondu aux espoirs que vous avez fondés en moi.
Toutes les encres du monde ne me suffisent pour vous exprimer mon immense gratitude.
Que ce travail puisse être le résultat de vos efforts et de vos sacrifices et que Dieu tout Puissant vous garde et vous procure santé, bonheur et longue vie.

A mon cher mari Halim …

Tu m as été d'un grand réconfort, et tu m as fortement soutenue dans les moments difficiles.
Grace à ton amour et ta compréhension j'ai pu terminer mes études dans l'enthousiasme et la joie.
Que dieu te garde et te protège …
Merci profondément …

A mes Chers Frères et mes Chères Sœurs …

Ahmed, Mohamed, Zahra, Hassan, Ismaël, Tayeb et Oussama.

En témoignage de l'attachement, de l'amour et de l'affection que je porte pour Vous. Je vous dédie ce travail avec tous mes vœux de bonheur, de santé et de Réussite.

A ma belle famille…

Ces quelques lignes ne sauraient suffire pour vous exprimer mon profond amour et l'immense reconnaissance pour tout l'encouragement et l'amoure dont vous avez fait preuve durant toutes mes études.
Je vous rends hommage par ce modeste travail en guise de ma reconnaissance éternelle et de mon infini amour.

A mon binôme Fatima Zahra Djiroun…

Nous avons passé de très agréables moments ensemble, Je vous dédie ce travail avec tous mes vœux de bonheur, de santé et de réussite.

A Tous les membres de ma famille, petits et grands…
A Toutes mes amies ….
A Tous qui me connaissent et qui m'aiment de prés ou de loin…

Biba Alili

Toutes les lettres ne sauraient trouver les mots qu'il faut ...
Tous les mots ne sauraient exprimer la gratitude, l'amour, le respect, la reconnaissance ...
Aussi, c'est tout simplement que ...

Je dédie cette thèse ...

A mes Très Chers Parents...

Chère *Mama*, cher *Papa*, vous représentez pour moi le symbole de la bonté par excellence, la source de tendresse et l'exemple du dévouement, vous n'avez pas cessé de m'encourager et de prier pour moi.
Vos prières et bénédictions m'ont été d'un grand secours pour mener à bien mes études.
Aucune dédicace ne saurait être assez éloquente pour exprimer ce que vous méritez, pour tous les sacrifices que vous n'avez cessé de me donner depuis ma naissance, durant mon enfance et même à l'âge adulte.
Que ce travail, que je vous dédie, en témoignage de mon profond amour, reflète ma profonde affection et ma grande reconnaissance.
Puisse Dieu, le tout Puissant, vous préserver et vous accorder santé, longue vie et bonheur pour que vous demeuriez le flambeau illuminant mon chemin.

A mes Chers Grands Parents...

Je vous remercie pour tout le soutien exemplaire et l'amour exceptionnel que vous me portez depuis mon enfance et j'espère que votre bénédiction m'accompagnera toujours.

A la mémoire de mon Grand Père et de ma Tante...

Le destin ne nous a pas laissé le temps pour jouir de ce bonheur ensemble et de vous exprimer tout mon respect.
Puisse Dieu tout Puissant vous accorder sa clémence, sa miséricorde et vous accueillir dans son vaste paradis.

A mes Très Chères Sœurs et mes Très Chers Frères...

Sarah, Mounira, Imene, Nour El Houda, Mohamed et Abd El Djalil, vous êtes merveilleux, des étoiles brillantes, vous êtes la joie et le bonheur. Que Dieu vous garde et vous réalise tous vos rêves et souhaits.

A Tous mes Oncles, mes Tantes, mes Cousins et mes Cousines...

En témoignage de l'attachement, de l'amour et de l'affection que je porte pour vous. Je vous dédie ce travail avec tous mes vœux de bonheur, de santé et de réussite.

A Tous mes Professeurs...

Mm *KHALFAL*, melle *BOUDIALI* et Mr *HADJ SADOK*
Je saisis cette occasion pour vous exprimer ma profonde gratitude tout en vous témoignant mes respects et ma considération.

A mon cher binôme Habiba Alili,

Je suis très contente de t'avoir connue et travaillé avec toi durant cet année, tu étais plus qu'une sœur pour moi, que dieu te réalise tes vœux et te protège ...

A Toutes mes Amies et Collègues de promotion et d'études...

A tous ceux ou celles qui me sont chers et que j'ai omis involontairement de citer, je vous dédie ce modeste travail...

A Tous qui me connaissent et qui m'aiment de prés ou de loin...

Fazin Djiroun

ملخص:

نهتم في هذه الاطروحة بدراسة مسائل ترتيب الاشغال ذات النمط المتعدد المرن (Job Shop Flexible) مع الاخذ بعين الاعتبار المعايير التالية : تقليص كتلة عمل الورشة (le makespan), تقليص كتلة عمل جميع الالات في الورشة (la charge totale), وتقليص كتلة عمل الالة الواحدة في الورشة (la charge critique). المسالة المعروفة بأنها NP Difficile نقترح لحلها خوارزمية البحث عن التناغم (Harmony Search) بطريقة التوازي. لتطبيقها نستعمل تكنولوجية تعدد العملاء (Multi agents) و Jade كأرضية تطوير. بعد الطور التجريبي لتحديد عوامل الطريقة بالتسلسل وعوامل الطريقة بالتوازي, اجريت مجموعة من الفحوص الاثباتية على مجموعة من اشهر ال benchmarks. النتائج بينت بوضوح تفوق الطريقة المتبعة بالمقارنة مع مختلف الطرق الموجودة.

الكلمات المفاتيح:

مسائل ترتيب الأشغال, السير المتعدد, خوارزمية, البحث عن التناغم, تكنولوجية تعدد العملاء.

RESUME

Nous nous intéresserons dans ce travail aux problèmes d'ordonnancement de type job-shop flexible, avec comme critère la minimisation du Makespan, la minimisation de la charge totale et la minimisation de la charge critique. Le problème étant connu NP-Difficile nous proposons pour sa résolution une version parallèle de la méthode « Harmony Search », mise *en œuvre en utilisant la technologie multi-agent et la plateforme de développement JADE.* Apres une phase de détermination expérimentale des operateurs de l'approche séquentielle, et des paramètres de l'approche parallèle, un ensemble de tests de validation a été réalisé sur les benchmarks les plus connus. Les résultats montrent clairement la supériorité de la méthode relativement aux différentes approches existantes dans la littérature.

Mots clés :

Problème d'ordonnancement, Job Shop Flexible, méta-heuristique parallèle, Harmony Search, technologie multi agents.

ABSTRACT

We are interested in the work by the resolution of the Flexible Job Shop Scheduling Problems (FJSP), with the following criteria: minimization of makespan time, minimization of the critical workload and minimization of the total workload. As the problem is known to be NP hard, we propose a parallel version of the Harmony Search method for its resolution and we use multi agent technology and the JADE development platform for implementing the approach. After an experimental phase for the determination of the operators of the sequential approach, and the parameters of the parallel approach, a set of validation tests have been conducted on some benchmark instances. The results show clearly the superiority of the method relatively to the existing different approaches in literature.

Key words:

Scheduling problems, Flexible Job Shop, parallel Meta-heuristic, Harmony Search, multi agent technology.

Sommaire

Chapitre IV : Expérimentations, synthèses et validation.

Listes des figures et tableaux

1/ Liste des figures :

Chapitre I : L'ordonnancement dans les ateliers de production.

Chapitre II : Approches méta-heuristique parallèles : concepts et technologie.

Chapitre III : Application de la méta-heuristique Harmony Search parallèle aux FJSl

Chapitre IV : Expérimentations, synthèses et validation.

2/La liste des tableaux :

Chapitre I : L'ordonnancement dans les ateliers de production.

Chapitre III : Application de la méta-heuristique Harmony Search parallèle aux FJSP.

Chapitre IV : Expérimentations, synthèses et validation.

INTRODUCTION GENERALE

Accroître la productivité en réduisant les coûts est, actuellement, un objectif majeur pour toutes les entreprises industrielles, surtout que l'environnement où elles évoluent est devenu dynamique, fortement concurrentiel et incertain. En effet, la mondialisation de la production ainsi que l'ouverture des marchés internationaux ont poussé les industriels à se diriger vers des systèmes de fabrication très complexes.

Garantir l'efficacité des systèmes de fabrication ne peut être assuré que par une bonne gestion des ateliers de production. Ce domaine a pris une certaine valeur ces dernières années, vu qu'il possède un champ d'action très vaste et couvre de nombreuses activités. L'optimisation de la planification et de l'ordonnancement des taches est en effet l'une de ces activités qui jouent un rôle primordial en termes de réduction des coûts et des délais. Elle peut être vue comme un problème énorme qui a attiré l'attention de différents chercheurs et fait part de diverses études dernièrement.

Les problèmes d'ordonnancement sont très variés, et ils sont classés en plusieurs familles. Lorsqu'une tâche est composée de plusieurs opérations nécessitant des ressources différentes, on parle de problèmes d'atelier. Le problème d'ordonnancement de type job shop flexible est l'un des problèmes d'atelier les plus compliqués et les plus difficiles à résoudre vu qu'il appartient à la classe NP difficile. La résolution optimale des problèmes d'ordonnancement de type Job Shop Flexible (JSF), s'avère dans certains cas impossible à cause de leur caractère fortement combinatoire. Ainsi, les approches méta-heuristiques sont souvent utilisées pour la résolution même sub-optimale de ces derniers.

Dans ce mémoire, nous nous intéressons particulièrement à la résolution des problèmes d'ordonnancement de type Job Shop Flexible en adaptant une récente méta-heuristique, soit l' « Harmony Search ». En vu d'augmenter ces performances, nous présentons aussi une version parallèle de cette approche implémentée en utilisant la technologie Multi-agent et la plateforme de développement JADE.

Le premier chapitre de ce mémoire est une introduction au monde de la gestion de la production, traitant le problème d'ordonnancent de type Job Shop Flexible et donnant une vue globale sur ses concepts de base.

Dans le deuxième chapitre, nous aborderons la présentation des différentes méthodes de résolution du JSF, puis nous nous focaliserons sur notre méta-heuristique en montrant les différents modèles parallèles qu'elle peut suivre. Puis nous introduirons la technologie qui permet de les implémenter.

Le troisième chapitre, quand à lui, détaille l'implémentation séquentielle et parallèle de notre méta-heuristique en utilisant des exemples illustratifs.

Enfin, le quatrième chapitre consistera en une suite d'expérimentations et de séries de tests de notre approche, faisant l'objet d'analyses et de discussions afin de ressortir certaines conclusions importantes permettant de valider notre implémentation.

Chapitre I

L'ordonnancement dans les ateliers de production.

I. Introduction :

Parmi les problèmes d'optimisation rencontrés ces dernières années, le problème d'ordonnancement est l'un des problèmes combinatoires les plus connus. Il occupe une place très importante dans le secteur industriel, particulièrement dans les systèmes de production des entreprises vu qu'il vise à améliorer son efficacité en terme de coûts de production et de délais de livraison.

Nous consacrons ce chapitre pour aborder les concepts relatifs aux problèmes d'ordonnancement, et notamment les problèmes d'ordonnancement d'ateliers de production. Nous parlerons aussi des différents modèles d'ateliers existant et des différentes approches de résolution connues.

II. La production:

II.1. Définition :

Dans les ateliers manufacturiers, la production est le processus conduisant à la création de bien et de services en transformant des ressources existantes (matières premières ou des produits semi-finis) en des produits finis grâce à des moyens de production (hommes et machines). Cette transformation peut porter sur la forme du produit, sa structure, son apparence, etc. **[5]**

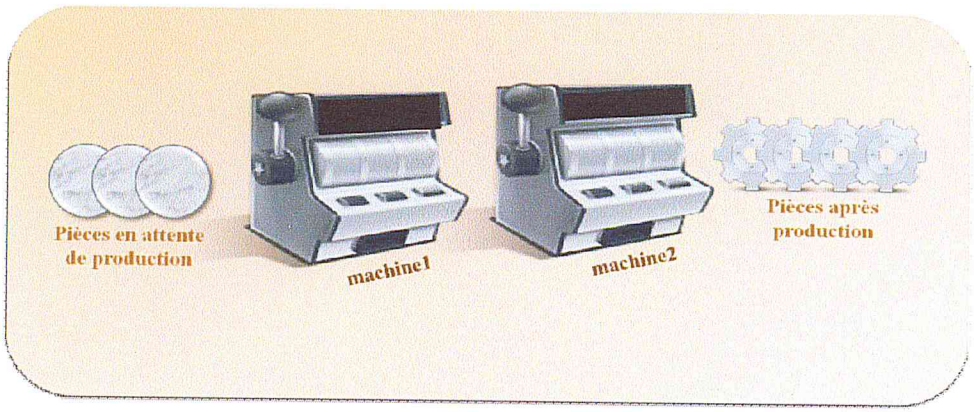


Figure 1 : La production dans l'atelier.

II.2. Gestion de production :

La gestion de production se réfère à toutes les activités liées à la conception et à la planification de la production dans l'atelier ; elle permet d'assurer une organisation efficace de la production afin de fabriquer les produits en quantités et temps voulus compte tenu des moyens (humains ou technologiques) disponibles. **[5]**

III. L'ordonnancement dans l'atelier :

III.1. Définition :

L'ordonnancement est une fonction de la gestion de production qui joue un rôle privilégié. Elle s'inscrit dans des niveaux de décision à la fois tactiques et opérationnels. Dans cette fonction il s'agit de contrôler à court ou moyen terme l'activité d'un ensemble de ressources disponibles en quantité limitée en gérant les conflits d'utilisation que pose la réalisation d'un ensemble d'activités sur un horizon de temps généralement imposé. **[5]**

III.2. Le problème d'ordonnancement : [17]

Le problème d'ordonnancement consiste à déterminer une planification du fonctionnement d'un système industriel de production. Autrement dit, il consiste à gérer l'affectation de ressources aux tâches au cours du temps, tout en satisfaisant au mieux un ensemble de critères et en respectant certaines contraintes.

Le résultat du processus de résolution d'un problème d'ordonnancement est un calendrier précis de tâches à réaliser qui comporte trois importantes caractéristiques :

- L'affectation, qui attribue les ressources nécessaires aux tâches,
- Le séquencement, qui indique l'ordre de passage des tâches sur les ressources,
- Le datage, qui indique les temps de début et de fin d'exécution des tâches sur les ressources.

La représentation des résultats du problème d'ordonnancement peut être faite par plusieurs méthodes, la plus utilisée est la représentation par le diagramme de GANTT.

Diagramme de Gantt :

Le diagramme de Gantt est un outil permettant de représenter la planification des taches nécessaires à la réalisation d'un projet et le suivi des différentes opérations mises en œuvre et leurs réajustements. Il a été élaboré par Henry Gantt en 1917.

Le diagramme de Gantt (Figure 2) présente en arrangé la liste des opérations notées Ti, à exécuter par les machines notées Mj, et en abscisse l'échelle du temps, de telle sorte qu'une opération est représentée par un segment dont la longueur est proportionnelle à sa durée d'exécution. **[15]**

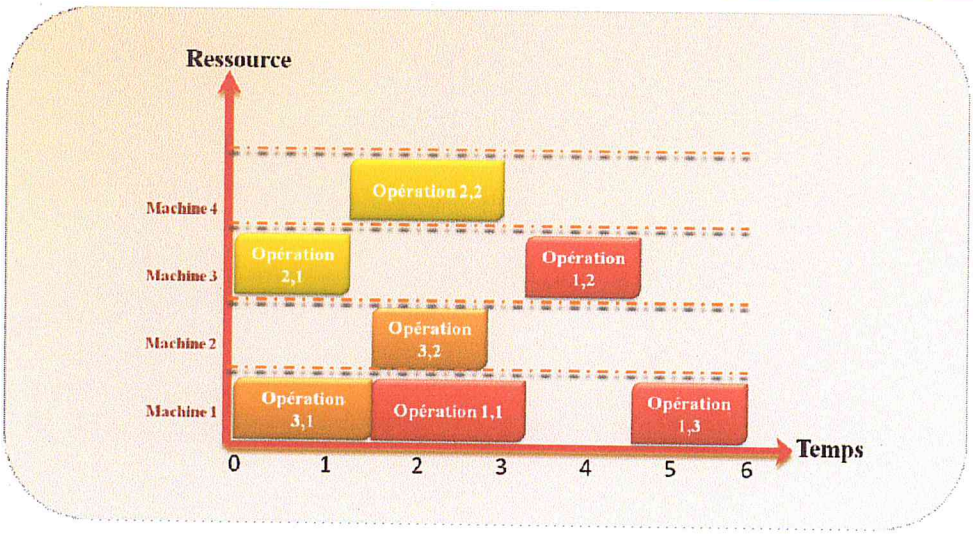


Figure 2 : Exemple de représentation du résultat de la résolution du problème d'ordonnancement par le diagramme de Gantt.

III.3. Concepts de base du problème :

Classiquement pour énoncer un problème d'ordonnancement il faut fixer à priori les caractéristiques des taches, les liens entre elles qui représentent les contraintes, les caractéristiques des ressources et les critères d'optimisation.

III.3.1. Tâche:

Une tâche (Figure 3) appelée aussi un job et qui représente une pièce (ou un produit), est une entité élémentaire localisée dans le temps, par une date de début et une date 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. **[21]**

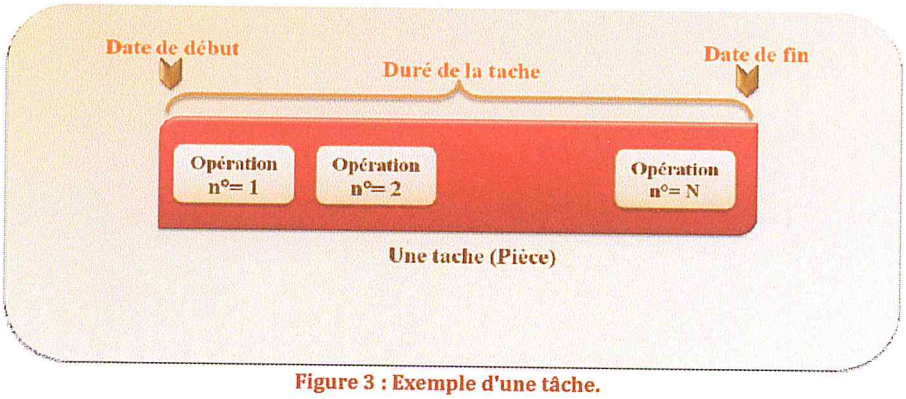


Figure 3 : Exemple d'une tâche.

Dans certains problèmes, les tâches peuvent être exécutées par morceaux, l'entrelacement des différents morceaux permettant de laisser, le moins possible les ressources inactives. Dans d'autres, au contraire, on ne peut pas interrompre une

tâche une fois commencée. On parle alors respectivement de tâches préemptives et non préemptives.

III.3.2. Ressource:

Une ressource est un moyen technique ou humain utilisé pour réaliser une tâche. Lorsque la ressource n'est plus disponible après son affectation à une tache, on parle alors de ressource consommable (argent, matières premières, etc.).Si au contraire, la ressource est encore disponible après chaque allocation à une tache, on parle donc de ressource renouvelable (machines, personnel, etc.), cette dernière peut être disjonctive[1] ou cumulative[2].

Qu'elle soit renouvelable ou consommable, la disponibilité d'une ressource peut varier au cours du temps. **[18]**

III.3.3. Contrainte : [21]

Une contrainte exprime des restrictions sur les valeurs que peuvent prendre conjointement des variables représentant des relations reliant les taches et les ressources.

Suivant la disponibilité des ressources et suivant l'évolution temporelle, plusieurs types de contraintes peuvent être distingués :

- **Les contraintes de ressources :**

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

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

- **Les contraintes temporelles :** elles intègrent :

- Les contraintes de temps allouées issues généralement d'impératifs de gestion. Elles sont relatives aux dates limitées des taches (délais de livraison, disponibilités des approvisionnements) ou à la durée totale d'un projet ;

[1] **Les ressources disjonctives :** qui ne peuvent exécuter qu'une seule tâche à la fois.
[2] **Les ressources cumulatives :** qui peuvent être utilisées par plusieurs tâches simultanément mais en nombre limité.

- Les contraintes d'antériorité et plus généralement les contraintes de cohérence technologique qui décrivent le positionnement relatif de certaines taches par rapport à d'autres ;
- Les contraintes de calendrier liées au respect d'horaire de travail, etc.

III.3.4. Critère : [18]

Un critère correspond à des exigences qualitatives et quantitatives permettant d'évaluer l'ordonnancement établi. Les critères dépendant d'une application donnée sont très nombreux; lorsque une même application doit respecter plusieurs critères on parle alors d'un problème d'ordonnancement multiobjectifs (ou multicritères), dans le cas contraire, le problème est monobjectif. Le choix de la solution la plus satisfaisante dépend du .ou des critères préalablement définis qui peuvent être classés suivant trois types :

- **Les critères usuels :** sont :
  - La durée totale de l'ordonnancement, le makespan.
  - Le respect des dates au plus tard.
  - La minimisation d'un coût, etc.
- **Les critères réguliers :** constituent des fonctions décroissantes des dates d'achèvement des opérations. Dont on peut citer les deux exemples suivants :
  - La minimisation des dates d'achèvement des actions ;
  - La minimisation des retards sur les dates d'achèvement des actions.
- **Les critères irréguliers :** sont des critères non réguliers c.-à-d. qui ne sont pas des fonctions monotones des dates de fin d'exécution des opérations, telles que :
  - La minimisation des risques (encours) ;
  - La minimisation du coût de stockage des matières premières ;
  - L'équilibrage des charges des machines.

IV. Classification des problèmes d'ordonnancement :

Une classification des problèmes d'ordonnancement dans un atelier peut s'opérer selon le nombre de machines qui existent et leur ordre d'utilisation pour fabriquer un produit, qui dépend de la nature de l'atelier considéré.

Un atelier est caractérisé par le nombre de machines qu'il contient et par son type. Selon la définition, nous pouvons distinguer cinq catégories de problèmes d'ordonnancement, avec des extensions possibles pour chacune d'elles.

IV.1. Atelier à machine unique :

Ce sont des ateliers où l'ensemble de tâches à réaliser est fait par une seule machine (Figure 4). Les tâches sont alors composées d'une seule opération qui nécessite la même machine. **[15]**

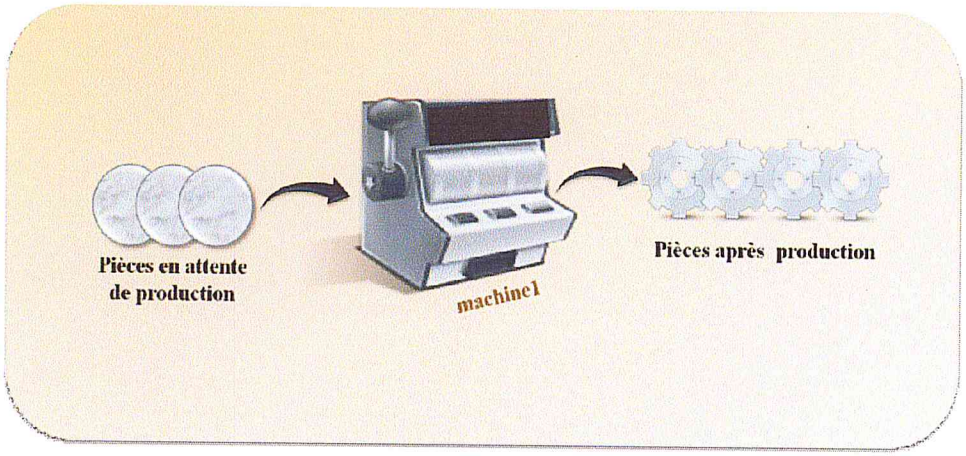


Figure 4: Atelier à machine unique.

IV.2. Atelier à machines parallèles :

Dans ce cas, l'atelier dispose d'un ensemble de machines identiques pour réaliser les travaux (Figure 5). Chaque travail se compose d'une seule opération et exige une seule machine. L'ordonnancement s'effectue en deux phases : la première consiste à affecter les travaux aux machines et la deuxième à établir la séquence de réalisation sur chaque machine. **[21]**

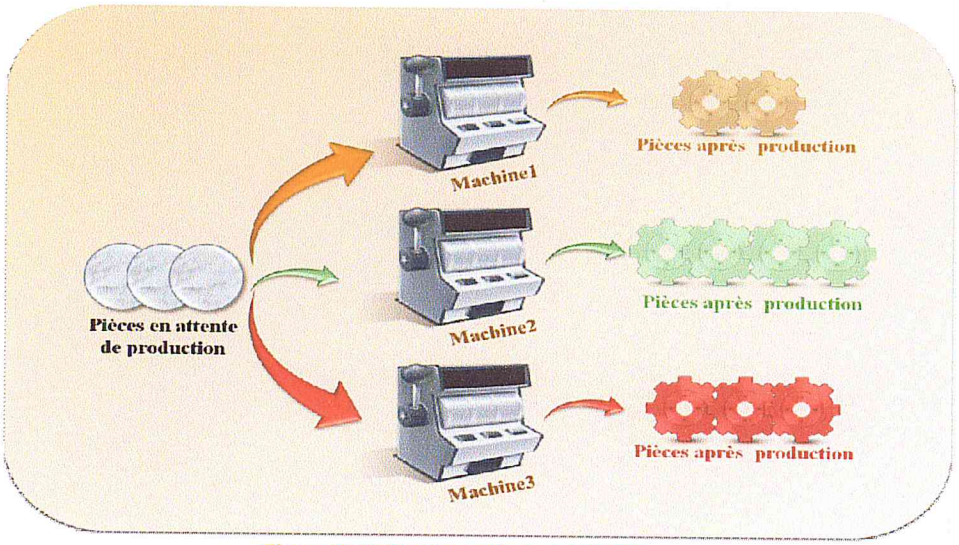


Figure 5: Atelier à machines parallèles.

IV.3. Ateliers à cheminements multiples :

Ce sont des ateliers de type Job Shop. Dans ce cas les opérations élémentaires d'une même tache (produit) suivent une séquence d'exécution bien déterminée, mais cette séquence est variable selon la tache à exécuter, chaque job possède donc une gamme spécifique (Figure 6). **[15]**

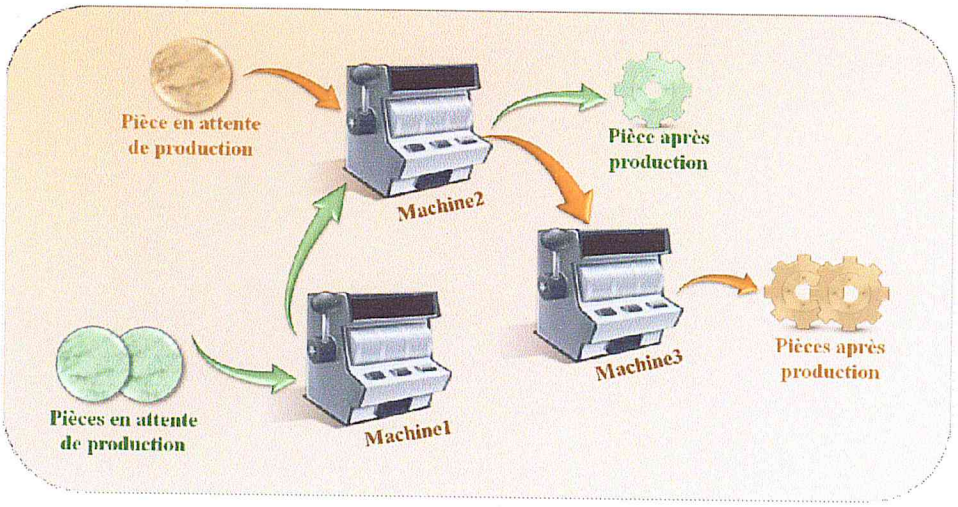


Figure 6 : Atelier à cheminements multiples.

IV.4. Ateliers à cheminement unique :

Appelés également ateliers de type Flow Shop, 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 toutes les machines dans le même ordre (Figure 7). **[15]**

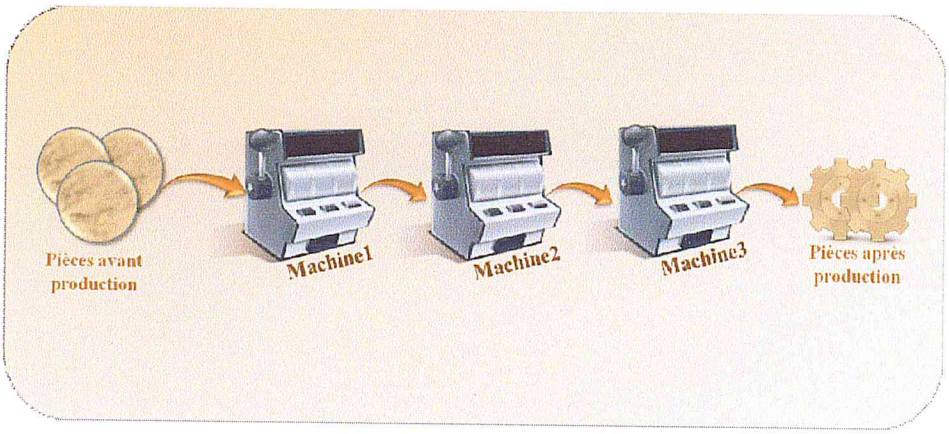


Figure 7 : Atelier à cheminement unique.

Atelier de type Flow shop hybrid:

Une organisation de type flow shop hybride à k étages est une généralisation du flow shop traditionnel pour lequel chaque travail doit subir k opérations différentes telles que chaque opération doit se faire sur une unique machine choisie parmi un

ensemble, appelé *étage*, de machines parallèles dédiées à cette opération (Figure 8). Une machine ne peut appartenir qu'à un seul étage. A la différence du flow shop classique, pour résoudre ce type de problème, il ne s'agit pas de déterminer seulement une date de début d'exécution pour chaque opération, mais aussi une affectation à une machine donnée. **[15]**

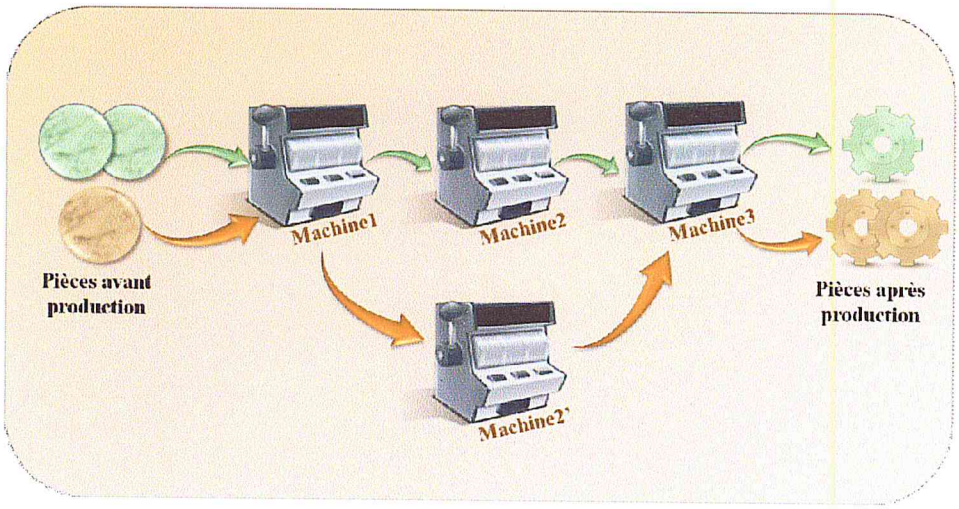


Figure 8 : Atelier de type Flow Shop Hybride.

IV.5. Ateliers à cheminements libres : Ce type d'atelier, qui est nommé Open Shop, est moins contraint que celui de type flow-shop ou de type job-shop. Ainsi, l'ordre des opérations n'est pas fixé à priori, c.-à-d. l'exécution d'une tache donnée n'impose pas des contraintes d'antériorité ou de précédence entre ses opérations (Figure 9). Comparé aux autres modèles d'ateliers, l'open-shop n'est pas couramment utilisé dans les entreprises. **[15]**

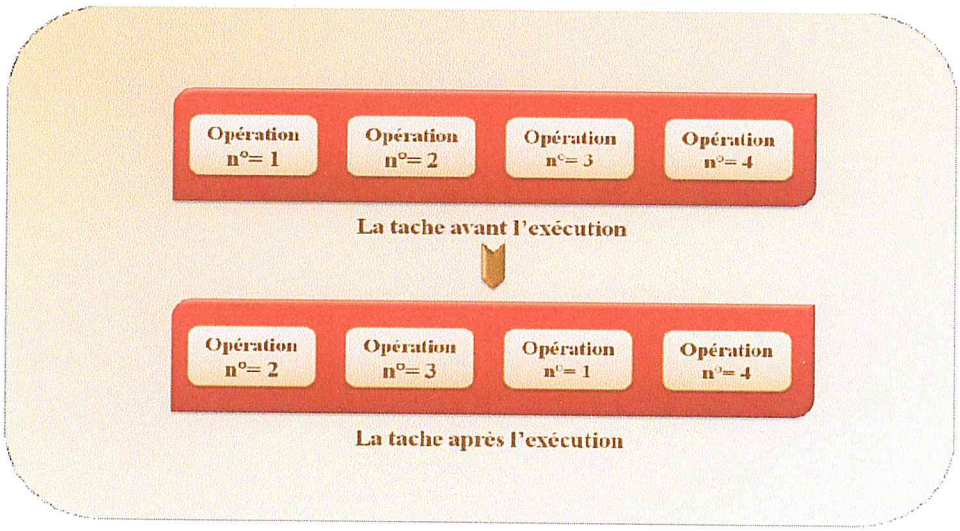


Figure 9 : Atelier à cheminement libre.

V. Atelier de type Job Shop Flexible :

V.1. Définition :

Un atelier de type job shop flexible est une extension du type d'atelier à cheminements multiples classique. Les problèmes d'ordonnancement d'ateliers de ce type sont connus dans la littérature comme étant les problèmes les plus difficiles à résoudre. En effet, la résolution de ces problèmes présente une difficulté supplémentaire dans la mesure où plusieurs machines sont potentiellement capables de réaliser la même opération qui possède différentes durées de traitement dépendant de la ressource utilisée (Figure 10). Les contraintes relatives à ce type de problème sont de type précédence, temporel ou disjonctif. **[4]**

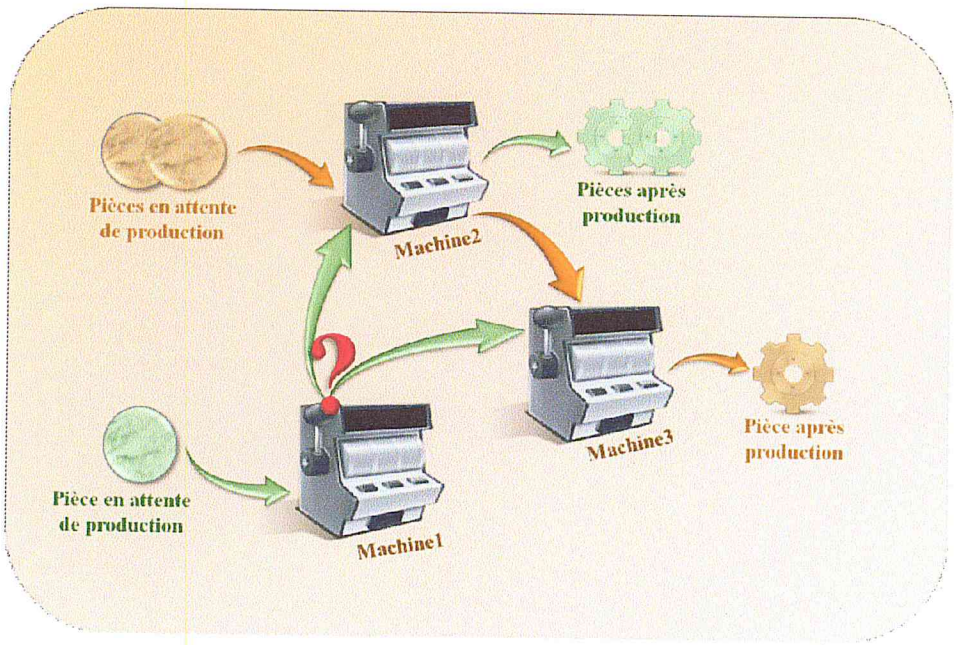


Figure 10 : Atelier de type Job Shop Flexible.

V.2. Formulation du FJSP : [18]

Les problèmes d'ateliers FJSP (Flexible Job shop Scheduling Problem) peuvent être formulés comme suit :

- Soit un ensemble de n produits indépendants à réaliser sur m machine M_k, $k = 1,2 \ldots m$.
- Chaque produit P_i est constitué d'une séquence de l_i opérations O_{ij}, $j = 1, 2, \ldots l_i$, à exécuter selon un ordre bien défini.
- L'exécution de chaque opération $O_{i\ j}$ d'un job P_i nécessite une ressource sélectionnée à partir d'un ensemble de machines disponibles.
- Chaque machine ne peut réaliser qu'une seule opération à la fois.

- L'assignation d'une opération O_{ij} à une machine M_k entraîne l'occupation de cette dernière durant tout le temps d'exécution de l'opération, noté D_{ijk}.
- la préemption n'est pas autorisée.

Le FJSP présente deux difficultés principales :

- la première est relative à l'assignation de chaque opération O_{ij} à une machine M_k.
- la seconde correspond au calcul des temps de début TD_{ij} et des temps de fin TF_{ij} de chaque opération O_{ij}.

V.3. Exemple d'énonciation du FJSP :

Pour simplifier la présentation du problème job shop flexible, nous avons conçu une simple instance de celui-ci. Le tableau ci-dessous, contient trois taches (jobs) et trois machines, chaque tache possède un nombre fixe d'opérations. Les opérations sont représentées par les lignes du tableau et les colonnes correspondent aux machines. Chaque cellule indique le temps d'exécution de l'opération sur la machine correspondante. Le « - » signifie que l'opération indiquée ne peut pas être exécutée sur la machine correspondante.

		Machine 1	Machine 2	Machine 3
Job n°=1	O_{11}	10	-	5
	O_{12}	10	6	-
	O_{13}	-	20	-
Job n°=2	O_{21}	10	-	10
	O_{22}	5	-	-
	O_{23}	10	6	-
	O_{24}	-	-	7
Job n°=3	O_{31}	5	10	-
	O_{32}	-	12	20

Tableau 1 : Exemple d'énonciation d'un problème FJSP.

VI. Méthodes de résolution des problèmes d'ordonnancement : [14]

Bien que le problème d'ordonnancement soit souvent facile à définir, il est généralement l'un des problèmes d'optimisation combinatoire les plus difficiles à résoudre. En fait, il appartient à la classe des problèmes NP-difficiles et ne possède donc pas à ce jour de solution algorithmique efficace valable pour toutes les instances.

Résoudre un problème d'ordonnancement consiste à minimiser (ou maximiser) la fonction d'évaluation par rapport à un ou plusieurs critères, en respectant certaines contraintes sur un ensemble fini de possibilités, autrement dit, à trouver une bonne solution, si possible optimale, entre un nombre fini de choix. La solution optimale du problème d'ordonnancement est représentée par une séquence d'exécution des tâches sur les machines.

Etant donné l'importance de ce problème, de nombreuses méthodes de résolution ont été développées en recherche opérationnelle (RO) et en intelligence artificielle (IA). Ces méthodes peuvent être classées sommairement en deux grandes catégories:

VI.1. Méthodes exactes : [18]

Appelées aussi complètes, sont des méthodes qui garantissent la complétude de la résolution. Le principe essentiel d'une méthode exacte consiste généralement à énumérer, souvent de manière implicite, l'ensemble des solutions de l'espace de recherche.

Pour améliorer l'énumération des solutions, une telle méthode dispose de techniques pour détecter le plus tôt possible les échecs (calculs de bornes) et orienter les différents choix.

Parmi les méthodes exactes, on trouve la plupart des méthodes traditionnelles (développées depuis une trentaine d'années) telles les techniques de séparation et évaluation progressive (SEP).

Les méthodes exactes ont permis de trouver des solutions optimales pour des problèmes de taille raisonnable mais comme le temps de calcul nécessaire pour trouver une solution risque d'augmenter exponentiellement avec la taille du problème, alors ces méthodes rencontrent généralement des difficultés face aux applications de taille importante.

VI.2. Méthodes approchées :

Inversement aux méthodes exactes, les méthodes approchées, dites aussi d'approximation, sont des méthodes incomplètes qui perdent de la complétude pour gagner en efficacité. Elles constituent une alternative très importante pour traiter les problèmes d'optimisation combinatoire de grande taille.

Plusieurs méthodes approchées ont étés développées pour la résolution des problèmes d'ordonnancement. Selon le principe utilisé, elles peuvent êtres rassemblées en deux groupes : [21]

- **Heuristiques :** ce sont des méthodes simples, rapides et destinées à des problèmes particuliers. Elles se basent sur des règles simplifiées pour optimiser un ou plusieurs critères. Le principe général de cette catégorie de méthodes est d'intégrer des stratégies de décision pour construire une solution proche de celle optimale tout en cherchant à avoir un temps de calcul raisonnable. [7]

- **Méta-heuristiques :** ce sont des méthodes plus puissantes et se placent à un niveau plus général encore, elles interviennent dans toutes les situations où l'ingénieur ne connaît pas d'heuristique efficace pour résoudre un problème donné, ou lorsqu'il estime qu'il ne dispose pas du temps nécessaire pour en déterminer une. [21]

VII. Conclusion :

Dans ce chapitre nous avons présenté les principales notions et concepts du problème d'ordonnancement d'atelier. L'un de ses modèles les plus étudiés est le problème de type Job Shop Flexible qu'on a abordé d'une manière détaillée. Nous avons aussi présenté les différentes méthodes existantes qui permettent la résolution de ce type de problème.

Chapitre II

Approches méta-heuristique parallèles : concepts et technologie.

I. Introduction :

Compte tenu des difficultés rencontrées par les méthodes exactes, la plupart des spécialistes de l'optimisation combinatoire ont orienté leurs recherches vers le développement de nouvelles approches permettant l'obtention de solutions de bonne qualité tout en réduisant les temps de calcul. Ces dernières sont des méthodes approchées qui portent généralement le nom de méta-heuristiques. Dans ce chapitre nous allons présenter les approches les plus utilisées.

Vu que le degré de difficulté n'est pas lié seulement à la dimension de l'espace de recherche, mais il est aussi limité par les capacités des ressources matérielles disponibles. Le parallélisme constitue une arme efficace pour lutter contre cette explosion combinatoire. Nous allons aussi présenter dans ce chapitre les méthodes d'implémentation parallèle de méta-heuristiques et la technologie d'implémentation multi agent JADE.

II. Méta heuristique :

II.1. Définition :

Le mot méta heuristique est la combinaison de deux termes grecs ; « méta » qui signifie « au-delà » comprendre ici « à un plus haut niveau » et heuristique qui vient du verbe « heuriskein » et qui signifie trouver. Les méta-heuristiques sont donc des méthodes de recherche générales, dédiées aux problèmes d'optimisation difficile. Elles sont, en général, présentées sous forme de concepts. **[14] , [20]**

En 2006, le réseau Metaheuristics (metaheuristics.org) définit les méta-heuristiques comme « un ensemble de concepts utilisés pour définir des méthodes heuristiques, pouvant être appliquées à une grande variété de problèmes. On peut voir les méta-heuristiques comme une « boîte à outils » algorithmique, utilisable pour résoudre différents problèmes d'optimisation, et ne nécessitant que peu de modifications pour qu'elle puisse s'adapter à un problème particulier ». Elles ont donc pour objectif de pouvoir être programmées et testées rapidement sur un problème donné. **[6]**

II.2. Classification des méta-heuristiques :

Il existe un grand nombre de méta-heuristiques différentes, cependant elles peuvent êtres classées en deux grandes classes : **[6]**

II.2.1. Approches trajectoires :

Dites aussi à solution unique. Ce sont des algorithmes de recherche locale qui peuvent être résumés comme étant des procédures de recherche itérative qui, à partir d'une première solution réalisable, l'améliorent progressivement en appliquant une série de modifications (ou mouvements) locales. A chaque itération, la recherche s'oriente vers une nouvelle solution réalisable qui diffère légèrement de la solution courante en remplaçant celle-ci par la meilleure trouvée. Dans cette approche on peut trouver plusieurs méthodes telles que le recuit simulé et la recherche Tabou. **[13]**

II.2.2. Approches évolutionnaires (ou à population) :

Contrairement aux méthodes de recherche locale qui font intervenir une solution unique, les méthodes évolutives manipulent un groupe de solutions admissibles à chacune des étapes du processus de recherche. L'idée centrale consiste à utiliser régulièrement les propriétés collectives d'un ensemble de solutions distinguables, appelé population, dans le but de guider efficacement la recherche vers de bonnes solutions dans l'espace de recherche. Parmi les méthodes évolutionnistes, nous citons les algorithmes génétiques et les algorithmes de colonies de fourmis. **[13]**

III. Exemples de méta-heuristiques :

III.1. Le recuit simulé : [20]

III.1.1. Origine et principe :

Inspiré du recuit physique, ce processus est utilisé en métallurgie pour améliorer la qualité d'un solide et cherche un état d'énergie minimale qui correspond à une structure stable du solide. Ainsi, pour qu'un métal retrouve une structure proche du cristal parfait, on porte celui-ci à une température élevée, puis on le laisse refroidir lentement de manière à ce que les atomes aient le temps de s'ordonner régulièrement. **[9]**

L'algorithme du recuit simulé permet de résoudre les problèmes de minima locaux. En effet, une nouvelle solution de coût supérieur à celui de la solution courante ne sera pas forcément rejetée, son acceptation sera déterminée aléatoirement en tenant compte de la différence entre les coûts ainsi que du facteur température T. Ce paramètre, sert à prendre en compte le fait que plus le processus d'optimisation est avancé, moins on est près à accepter une solution plus coûteuse.

Par contre, l'acceptation de solutions fortement coûteuses permet, au début, de mieux explorer l'espace des solutions possibles et ainsi, d'accroître les chances d'approcher le minimum global. [13]

III.1.2. Algorithme : [6]

Poser $T \leftarrow T0$

Répéter :

Choisir aléatoirement $s' \in N(s)$

Générer un nombre réel aléatoire r dans [0,1]

Si $r < e^{\frac{f(s)-f(s')}{T}}$ alors Poser $s \leftarrow s'$

Mettre à jour T

Jusqu'à ce que le critère de terminaison soit satisfait

Fin

III.2. L'algorithme génétique : [6]

III.2.1. Origine et principe :

Les algorithmes génétiques sont des algorithmes d'optimisation s'appuyant sur des techniques dérivées de la génétique et de la théorie de l'évolution naturelle. En effet ces algorithmes adoptent une sorte d'évolution artificielle analogue à l'évolution naturelle telle que décrite par la théorie. Le vocabulaire utilisé dans cet algorithme est directement inspiré de celui de la théorie de l'évolution et de la génétique. Nous parlerons donc d'*individus*, pour parler de solutions. L'ensemble des individus formera une *population*, que nous ferons évoluer pendant une certaine succession d'itérations appelées *générations*, jusqu'à ce qu'un critère d'arrêt soit vérifié. Pour passer d'une génération à une autre, on soumettra la population à des *opérateurs de sélection*, de *croisement* et de *mutation* qui permettront de transformer la population, de façon à favoriser l'émergence de meilleurs individus.

Sélection : La sélection permet d'identifier statistiquement les meilleurs individus d'une population et d'éliminer les mauvais.

Croisement : Le croisement a pour but d'enrichir la diversité de la population en manipulant la structure des chromosomes. Classiquement, les croisements sont envisagés avec deux parents et génèrent deux enfants.

Mutation : L'opérateur de mutation apporte aux algorithmes génétiques la propriété d'exploration de l'espace de recherche. Cette propriété indique que l'algorithme génétique sera susceptible d'atteindre tous les points de l'espace d'état, sans pour autant les parcourir tous dans le processus de résolution.

III.2.2. Fonctionnement général : [8]

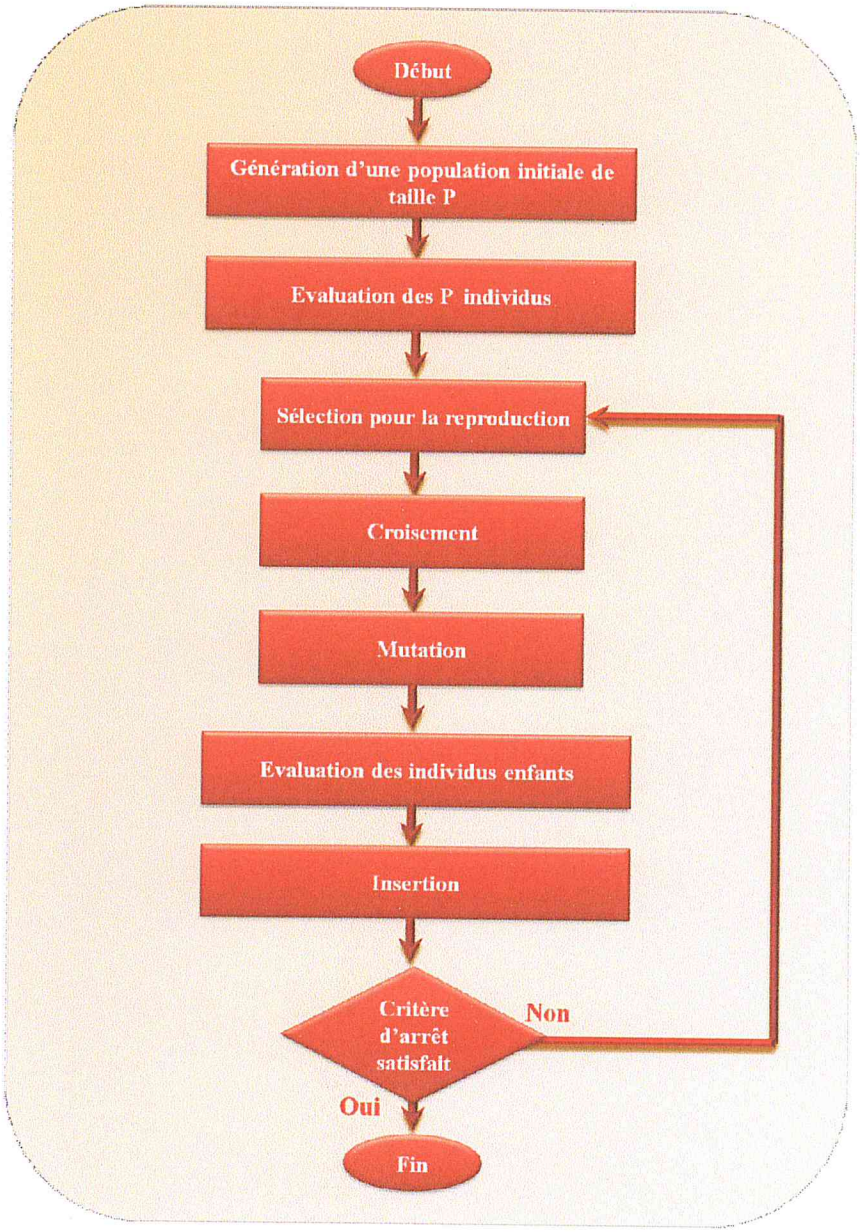


Figure 1 : Fonctionnement général de l'algorithme génétique.

IV. La méthode Harmony Search :

IV.1. Origine et définition :

Cette nouvelle méthode a été développée par Zong Woo Geem, s'inspirant du processus d'harmonisation et d'improvisation musicale dans un orchestre. L'harmonie de musique est une combinaison de bruits considérés comme une satisfaction d'un point de vue esthétique. Les exécutions musicales cherchent à trouver l'harmonie agréable (un état parfait) déterminée par une norme esthétique, juste comme le processus d'optimisation de trouver une solution optimale globale d'une fonction objective.

Ce nouvel algorithme HS est basé sur les processus d'exécution musicale qui se produisent quand un musicien cherche un meilleur état d'harmonies, comme pendant l'improvisation dans le jazz. Dans l'improvisation de musique, chaque joueur retentit n'importe quel lancement dans la marge possible, en suivant l'une de ces trois règles :

- Règle 1 : jouant un lancement de sa mémoire ;
- Règle 2 : jouant un lancement adjacent d'un lancement de sa mémoire ;
- Règle 3 : jouant un lancement totalement aléatoire de la gamme possible.

Par analogie, le processus d'optimisation dans la méthode HS utilise ces trois règles pour la génération d'une nouvelle solution. Ici la mémoire est un ensemble de solutions générées aléatoirement au début. Ainsi le processus génère à chaque itération une nouvelle solution en utilisant ou bien des valeurs de la mémoire, ou des valeurs modifiées de la mémoire, ou des valeurs totalement aléatoires. Une règle de remplacement permet la mise à jour de la mémoire en vue de garder les meilleures solutions et de les utiliser ensuite. **[19]**

Figure 2 : Improvisation musicale.

IV.2. Algorithme : [18]

L'algorithme de cette méthode peut être résumé comme ci-dessous:

Début

Générer une mémoire d'harmonies initiale HM de solutions de taille |HM| = n.

Fixer un Taux d'acceptation *HMCR* □ (0, 1).

Fixer un Taux d'ajustement *PAR* □ (0, 1).

Tant que (critère d'arrêt n'est pas attient) ***faire***

  Pour (j=1 à TH) ***faire*** *//(TH = la taille de l'harmonie)*

    Sélectionner une harmonie X dans HM ;

    Déterminer *hmcr* □ (0,1) ; // probabilité d'acceptation,

    Si (*hmcr* < *HMCR*) ***alors***

      Déterminer *par* □ (0,1) ; // probabilité d'ajustement,

      Si (*par* < *PAR*) ***alors***

        Ajuster la j^{eme} valeur de X et l'insérer dans X' // (tq X' la nouvelle H)

      Sinon Insérer directement la j^{eme} valeur de X dans X'

    Sinon Générer un nombre aléatoire valide et insérer le dans X'

  Fin Pour.

  Si *X'* vérifier les critères d'optimisations **alors** MAJ la mémoire d'harmonies

Fin tant que.

Fin.

IV.3. Principe : selon le concept expliqué ci-dessus, l'algorithme de HS comprend les cinq étapes suivantes : **[18]**

IV.3.1. Initialisation des paramètres du HS : les paramètres de l'algorithme « Harmony Search » qui sont exigés pour résoudre un problème d'optimisation sont spécifiés dans cette étape :

- La capacité de la mémoire d'harmonies,
- Le taux d'acceptation des lancements,
- Le taux d'ajustement des lancements,
- Critère d'arrêt (nombre maximum des recherches).

IV.3.2. Initialisation de la mémoire d'harmonies: dans cette étape la mémoire d'harmonies est remplie par des euphonies tirées aléatoirement.

IV.3.3. Improvisation d'une nouvelle harmonie : dans l'étape (3) une nouvelle harmonie est produite en se basant sur les trois règles mentionnées précédemment.

IV.3.4. Mise à jour de la mémoire d'harmonies : dans cette étape ; si la nouvelle harmonie est meilleure que la plus mauvaise de la mémoire, elle sera incluse dedans et la plus mauvaise existante est exclue de la mémoire.

IV.3.5. Vérification du critère d'arrêt : les calculs sont terminés lorsque le critère d'arrêt est atteint ; sinon les étapes 3 et 4 sont répétées.

V. Modèles parallèles pour les méta-heuristiques évolutionnaires : [16]

Les méta-heuristiques apportent des solutions approchées aux problèmes d'optimisation combinatoire pour des instances de grande taille, et demandent un calcul scientifique intensif. Toutefois, leur efficacité est limitée par les capacités des ressources disponibles. Ainsi, l'utilisation des modèles parallèles appropriés diminuent le temps de calcul et améliorent la qualité des solutions obtenues. **[10]**

Afin de réaliser ce concept, différents modèles sont adaptés ; les plus utilisés sont cités ci-dessous :

V.1. Modèle insulaire : [2]

Le modèle insulaire se base sur le concept des îles. Chacune des îles, se composent d'un ensemble de solutions différent, les différentes îles sont évoluées simultanément et de manière isolée, en effectuant des migrations de solutions entres elles pour apporter des nouveautés sur les résultats obtenus (Figure 3).

Ainsi, dans ce modèle, chaque processeur fait évoluer un ensemble de solutions, exécutant séquentiellement toutes les phases de l'algorithme implémenté. Après un nombre donné d'itérations (modèle synchrone) ou bien lorsqu'un certain critère est vérifié (modèle asynchrone), telle que la convergence d'une île, a lieu une migration : chaque processeur communique des solutions de son île aux autres processeurs, et en reçoit de la même façon. Les solutions reçues sont intégrées dans l'ensemble de solutions courant, ce qui permet une meilleure diversification au sein des espaces de recherche, et une exécution plus robuste minimisant la déviation en terme de qualité d'une exécution à une autre.

En effet, afin d'appliquer ce processus, un critère de décision de migration, soit aveugle ou intelligent, est choisi. La migration aveugle peut être périodique (intervient sur chaque algorithme après une période fixée) ou probabiliste (intervient à chaque itération avec une probabilité fixée). A l'inverse, la migration intelligente est dirigée par des critères d'amélioration de la qualité des solutions.

En outre, une topologie des échanges d'individus doit être mise en œuvre. Divers travaux ont été menés pour étudier l'impact de la topologie sur la qualité des résultats obtenus. Il ressort de ces études que les graphes cycliques sont préférables, les modèles en anneau sont d'ailleurs largement utilisés.

Le nombre des émigrants définis comme étant un nombre fixe, une variable ou un pourcentage de la population est sélectionné selon une politique bien précise. Cette politique de sélection des émigrants indique pour chaque île de manière élitiste ou aléatoire, les éléments à migrer. En effet, la stratégie aléatoire ne garantit pas la sélection des meilleurs éléments. Par contre, la stratégie élitiste tente de sélectionner les meilleurs éléments de l'île. Par ailleurs, ces émigrants seront intégrés dans les îles et tenteront de remplacer les anciens éléments de manière, aussi, aléatoire ou élitiste.

Par conséquent, nous pouvons conclure que les cinq paramètres permettant une définition complète de la politique de migration d'individus entre les îles, sont : le critère de décision de migration, la topologie des échanges d'individus, le nombre des émigrants et la stratégie de leur sélection et enfin la politique d'intégration des immigrants.

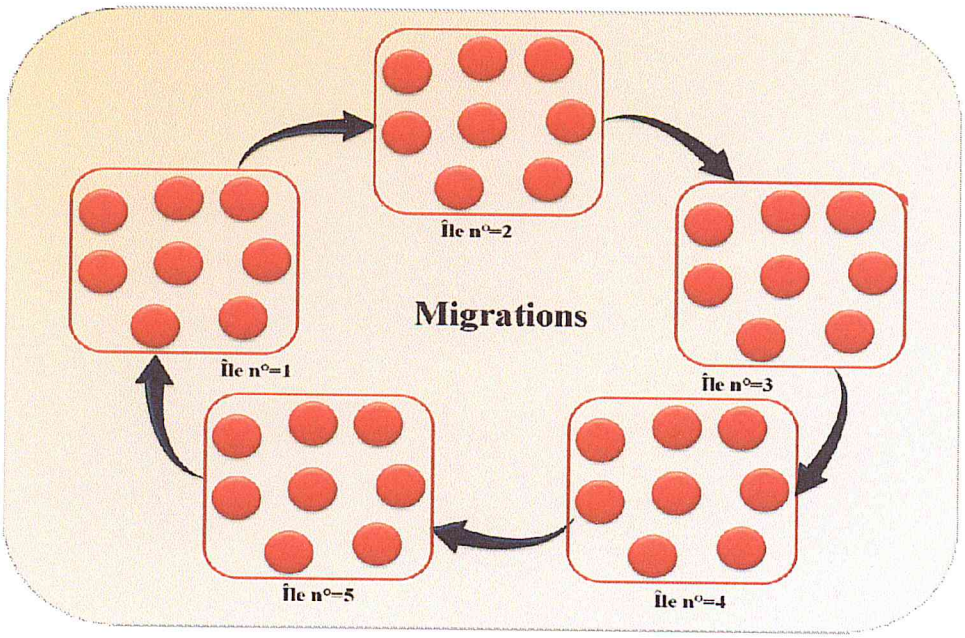


Figure 3 : Modèle insulaire avec une topologie en anneau.

V.2. Modèle parallèle centralisé : [21], [2]

Ce modèle de parallélisme, qui est de type maître/esclaves, a pour caractéristique de conserver le comportement séquentiel de la méthode en ne distribuant que les phases coûteuses. Le processeur maître assure les phases de sélection et de remplacement. A chaque itération, le maître distribue les éléments sélectionnés à évaluer aux processeurs esclaves. Chaque esclave réalise les phases d'évaluation et retourne les solutions évaluées au processeur maître qui procède au remplacement de l'ensemble de solutions courant (Figure 4).

Selon l'ordre d'exécution de la phase d'évaluation par rapport au reste de l'algorithme implémenté, le modèle a deux modes : synchrone et asynchrone.

Dans le mode synchrone, le processus maître gère l'évolution de la population, exécute en séquence les phases de sélection, transformation et remplacement. A chaque itération, il distribue l'ensemble des nouvelles solutions générées entre différents agents évaluateurs (esclaves). Après la collecte des résultats, le processus d'évolution se poursuit à nouveau.

Dans le mode asynchrone, la phase d'évaluation n'est pas synchronisée avec le reste de l'algorithme implémenté. Le processus maître n'attend pas le retour de toutes les évaluations pour exécuter les différentes phases.

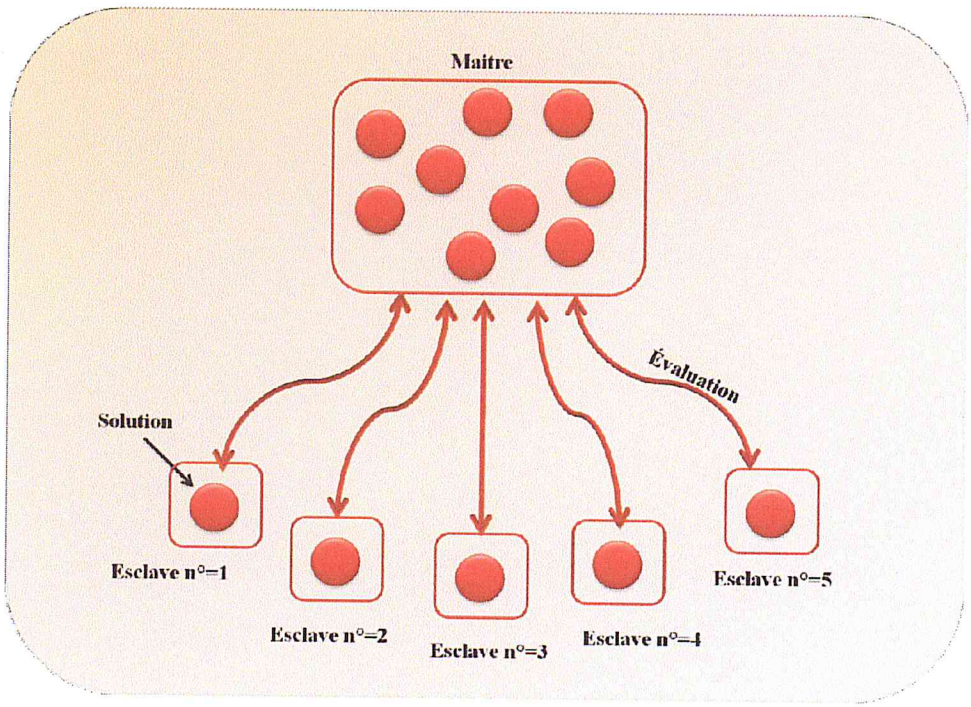


Figure 4 : Modèle maitre/esclave.

VI. Développement Multi-Agents et plateforme JADE :

Les méta-heuristiques parallèles nécessitent pour leurs implémentations des technologies garantissant le concept du parallélisme ; parmi ces technologies, nous trouvons les systèmes multi agent et la plateforme de développement JADE. Nous définissons, dans ce qui suit, les concepts relatifs à l'implémentation multi agent des méta-heuristiques parallèles.

VI.1. Système Multi-Agents : [21]

Un Système Multi-Agents (SMA), est une structure composée d'un environnement et d'un ensemble d'agents artificiels. Ces derniers sont capables d'agir sur l'environnement et, de collaborer entre eux et / ou avec des agents extérieurs. La collaboration se fait généralement par des messages. Les interactions entre les agents et leur environnement, ou entre eux, mène à un comportement propice qui leur permet de réaliser leurs propres buts et, par-là même, atteindre le but global.

VI.2. Agent : [21]

Selon Ferber « un agent est une entité autonome, réelle ou abstraite, qui est capable d'agir sur elle-même et sur son environnement, qui, dans un univers multi-agents, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de ses connaissances et des interactions avec les autres agents ».

Autrement dit : un agent qualifie tout composant logiciel, autonome, situé, communicant, réactif, proactif, ayant des buts et des compétences et offrant des services.

VI.3. Interactions entre agents : [21]

L'interaction est une notion importante dans les systèmes multi-agents. Elle est définie comme étant la mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions réciproques dont les conséquences exercent en retour une influence sur le comportement futur des agents. Selon les agents et les systèmes nous distinguons différentes formes d'interactions, les plus connues sont citées ci-après :

VI.3.1. Coopération :

La coopération peut être considérée comme une attitude intentionnelle adoptée par les agents qui décident de travailler ensemble. Dans ce cas les agents s'engagent

dans une action après avoir identifié et adopté un but commun considéré comme un élément essentiel de l'activité sociale.

D'une autre façon on peut dire que plusieurs agents coopèrent, ou encore qu'ils sont dans une situation de coopération, si l'une des deux conditions est vérifiée :

- ✓ L'ajout d'un nouvel agent permet d'accroître les performances du groupe.
- ✓ L'action des agents sert à éviter ou à résoudre des conflits potentiels ou actuels.

VI.3.2. Communication :

La communication est un moyen ou une méthode d'interaction entre agents, c'est l'un des points fondamentaux dans l'étude des systèmes multi agents. Les agents peuvent interagir soit en accomplissant des actions linguistiques (en communiquant entre eux), soit en accomplissant des actions non linguistiques qui modifient leur environnement. En communicant, les agents peuvent échanger des informations et coordonner leurs activités.

Dans les SMA deux stratégies principales ont été utilisées pour la communication entre agents : soit par échange ou envoi de messages directement, soit par partage d'informations.

La communication par partage d'information :

Dans ce modèle, tous les messages transitent par un système qui les redistribue ensuite. La communication est donc centralisée et les agents ne communiquent pas directement entre eux. Le système central est constitué d'un « blackboard » et d'un superviseur.

Le « blackboard » centralise tous les messages et les agents peuvent y accéder pour retirer les informations nécessaires. Cette méthode était essentiellement utilisée au début de l'histoire des systèmes multi-agents. Aujourd'hui on utilise généralement la méthode suivante.

La communication par envoi de messages :

Dans ce type, le mécanisme par transmission de messages suppose une communication directe entre les agents, puisqu'il n'existe pas de modules intermédiaires entre eux, il y'aura un envoi direct et explicite du message au destinataire.

VII. Plateforme JADE :

VII.1. Définition : [21]

JADE est une plateforme de développement des systèmes multi-agents qui permet aux développeurs de concevoir et réaliser des applications multi agent plus rapidement et sans devoir avoir la nécessité d'être familier avec les différents concepts théoriques des SMA.

Elle permet l'implémentation et l'exécution des systèmes multi-agents conformes aux normes FIPA (Foundation for Intelligent Physical Agents). Elle est implémentée en Java et fournit deux composantes de base : Un « runtime » agents compatible FIPA et un paquet logiciel pour le développement des agents Java.

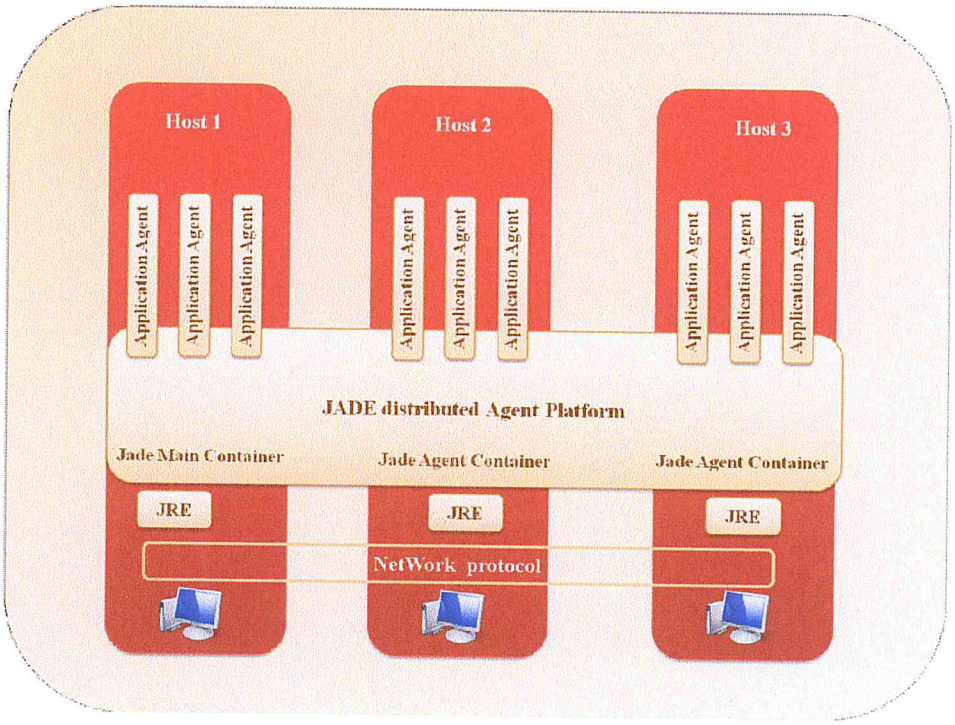


Figure 5 : Architecture de la plateforme JADE.

VII.2. Conteneur et plateforme: [20], [21]

Chaque instance de l'environnement d'exécution de JADE est appelée « conteneur » (container) car elle peut contenir plusieurs agents. L'ensemble des conteneurs actifs est appelé «plateforme». Cette dernière peut être distribuée sur plusieurs hôtes (Figure 6).

Dans chaque plateforme, un conteneur spécial appelé conteneur principal (main container) doit toujours être en activité, les autres conteneurs s'enregistrent auprès de celui-ci à leurs démarrages.

Chaque processus java est un conteneur d'agents qui fournit un environnement d'exécution complet pour l'exécution des agents et leur permet de s'exécuter en parallèle sur le même hôte. Chaque conteneur d'agents est un environnement multi-threads composé d'un thread d'exécution pour chaque agent.

La plateforme possède des agents prédéfinis. Les plus importants sont le Remote Monitoring Agent (RMA), le Directory Facilitator (DF) et l'Agent Management System (AMS)).

- L'agent RMA sert d'administrateur pour les agents. Avec son interface graphique (GUI), il est possible de gérer à distance (créer, supprimer et migrer) le statut des agents sur le même hôte, ou bien sur un hôte éloigné à condition qu'un conteneur d'agents s'exécute déjà sur cet hôte. L'agent RMA permet également la création et la suppression des conteneurs et la fermeture de la plateforme.

- L'agent DF enregistre les descriptions et les services des agents et maintient des pages jaunes de services.

- L'agent AMS est chargé du contrôle de l'accès des agents à la plateforme.

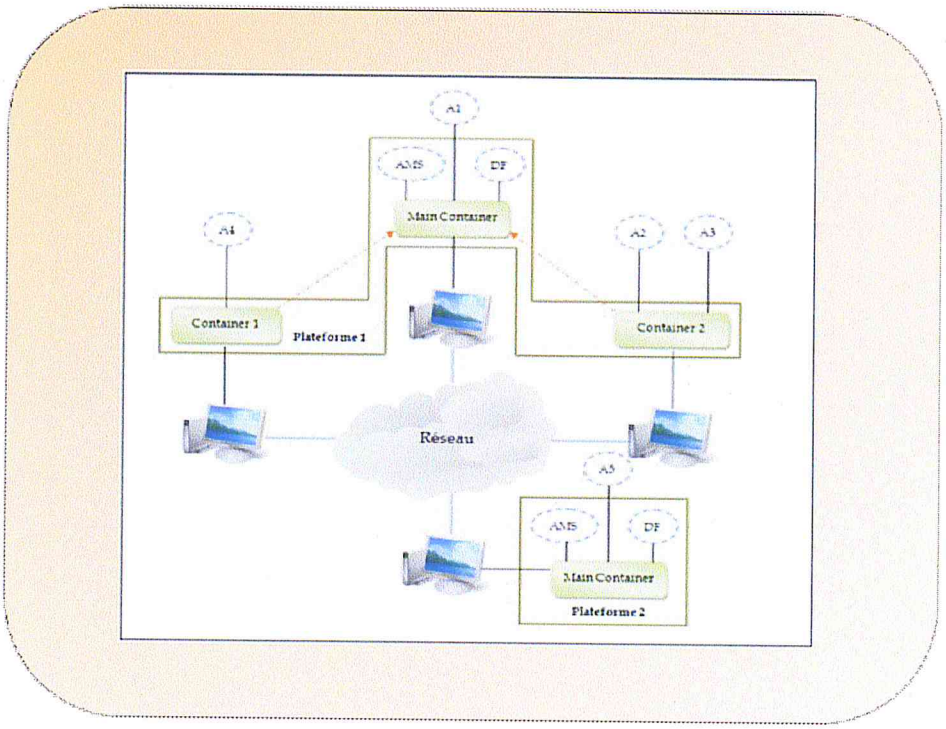


Figure 6 : Relation entre plateforme, conteneur principal et conteneur secondaire.

VII .3. Manipulation des agents JADE : [10]

Un agent selon Jade possède un cycle de vie, il peut avoir un ou plusieurs comportements (Behaviours). Il communique en utilisant des messages de type Agent Communication Language (ACL) et rend des services. Il est identifié de manière globale par un nom unique (l'AgentIdentifier ou AID). Chaque agent peut joindre ou quitter librement la plateforme et rentrer en contact avec chacun des autres agents.

La plateforme Jade permet à l'utilisateur de programmer des agents selon les besoins. Pour cela, elle fournit un ensemble de méthodes prédéfinies permettant de faciliter leurs manipulations. Les plus utilisées sont :

- **setup()**, elle permet d'initialiser l'agent, en particulier créer et activer les comportements initiaux ;
- **addBehaviour (Behaviour)** et **removeBehaviour (Behaviour)**, permettant d'ajouter à l'agent, ou de lui supprimer, des comportements ;
- **send (ACLMessage)**, qui permet d'envoyer un message ACL à un ou plusieurs destinataires ;
- **receive()**,qui permet de lire le message suivant dans la boite aux lettres de l'agent.

VII.4. Comportement des agents JADE : [21]

Un comportement (Behaviour) représente une tâche qu'un agent doit effectuer. Chaque comportement possède deux méthodes : la méthode action () qui définit les opérations à exécuter par l'agent, et la méthode done () qui renvoie un booléen spécifiant si le comportement est terminé et doit être supprimé de la file des comportements à exécuter par l'agent. On distingue principalement trois types de Behaviours :

- Le « OneShot behaviours » qui se termine immédiatement et dont la méthode action () est exécutée seulement une fois. La méthode done () retourne «Vrai».
- Le « Cyclic behaviours » qui ne se termine jamais et dont la méthode action() exécute les mêmes opérations chaque fois qu'elle est appelée. La méthode done() retourne «Faux».

Behaviours génériques « Generic behaviours » qui exécutent différentes opérations dépendamment d'un état ; Ils se terminent quand une certaine condition est vérifiée.

VII.5. Echanges de messages dans JADE : [21]

L'un des dispositifs les plus importants que les agents JADE fournissent est la capacité de communiquer. Le paradigme de communication adopté est le mode asynchrone. Chaque agent a une sorte de boîte aux lettres (la file d'attente de messages de l'agent). A chaque fois qu'un message arrive, il est ajouté à la queue de la file et l'agent commence par traiter le premier message arrivé.

Les messages échangés par les agents JADE obéissent au format spécifié par le langage ACL lequel a été défini par la norme FIPA pour l'interopérabilité des agents. Ce format inclut différents champs. Les plus importants sont: L'émetteur du message ; la liste des destinataires ; le contenu ; l'intention de la communication (appelée aussi « performative »).

VII.5.1. Envoi de message :

L'envoi d'un message consiste à créer un objet ACL-Message et appeler la méthode send () de la classe « Agent ».

Le code suivant correspond à l'envoi d'un message à l'agent « Agent1 ».

```
ACLMessage msg = new ACLMessage (ACLMessage.INFORM) ;
msg.addReceiver (new AID ("Agent1", AID.ISLOCALNAME)) ;
msg.setContent ("Salut");
send (msg);
```

VII.5.2. Réception de message :

La lecture des messages de la file d'attente de message est accomplie par la méthode receive () de la classe « agent ». Cette méthode renvoie le premier message dans la file d'attente (en le retirant de la file) ou NULL si la file d'attente est vide.

Remarque: lorsqu'on spécifie une Template, la méthode «receive()» ne retourne que le premier message correspondant à ce Template en ignorant les autres messages qui ne lui correspondent pas.

L'exemple suivant montre comment se fait la réception et la lecture du message par l'agent.

```
Public void action() {
MessageTemplate msgTemplat = MessageTemplate.MatchPerformative(ACLMessage. INFORM);
ACLMessage messagRecu = receive(msgTemplat);
If (msg != null){
  //Message reçu: effectuer les traitements sur le message qui a un performative (INFORM)
String text = msg.getContent() ;}
}
```

VIII .Conclusion:

Dans ce chapitre, nous avons présenté les méta-heuristiques et donné des exemples de celles qui sont les plus connues et utilisées actuellement, puis nous nous sommes intéressés à la méta-heuristique « Harmony Search » que nous avons détaillé. Nous avons aussi montré les différents modèles pouvant correspondre à son implémentation parallèle, et présenté la technologie permettant d'assurer ce parallélisme.

Dans le prochain chapitre, nous allons expliquer, à l'aide d'exemples simples illustratifs, le fonctionnement de l'algorithme « Harmony Search », puis les différentes étapes de son implémentation parallèle.

Chapitre III

Application de la méta-heuristique Harmony Search parallèle aux FJSP.

I. Introduction :

La méthode « Harmony Search » (HS) est une méta-heuristique relativement récente qui s'inspire du phénomène d'improvisation musicale. Initialement, elle a été conçue dans le cadre de la résolution des problèmes d'optimisations dans des applications technologiques. Dans notre application, nous l'avons invoqué pour la résolution du problème d'ordonnancement Job Shop Flexible. Comme pour toute méta-heuristique, la résolution de tel problème est limitée par les capacités de ressources matérielles disponibles, nous avons pensé à introduire le concept du parallélisme en adoptant le modèle insulaire.

Dans ce chapitre, nous allons donner un aperçu sur la démarche suivie pour l'implémentation de l'algorithme « Harmony Search » parallèle, en se basant sur le modèle du Job Shop Flexible avec deux machines et deux produits (job) : multi-Agents.

II. Adaptation de la méta-heuristique « Harmony Search » :

L'implémentation de toute méta-heuristique nécessite la détermination d'un codage efficace et d'une méthode adaptée pour l'évaluation des solutions.

Tout au long de cette partie et afin de clarifier l'étude de l'algorithme de la recherche d'harmonies, nous allons adopter l'exemple présenté dans le tableau ci-après :

	Operations	M1	M2
Job 1	$O_{1,1}$	4	10
	$O_{1,2}$	8	11
Job 2	$O_{2,1}$	6	-
	$O_{2,2}$	8	10
	$O_{2,3}$	5	3

Tableau 4 : Exemple d'un problème FJSP.

L'exemple qui est décrit dans le tableau représente une instance du problème Job Shop Flexible :

- La première pièce (produit) dispose de deux opérations, la première ($O_{1,1}$) peut s'exécuter sur la machine M1 avec un temps de 4 et sur la machine M2 avec un temps de 10, et la deuxième ($O_{1,2}$) peut être traitée par la machine M1 avec une durée de 8 et sur la machine M2 avec une durée de 11.

La deuxième pièce possède trois opérations, la première ($O_{2,1}$) peut être effectuée par une seule machine seulement qui est M1avec un temps de 6, la deuxième peut être réalisée par la machine M1 avec une durée de (8) et par la machine M2 avec une durée de (10) et la troisième et dernière opération peut être achevée par la machine M1 avec un temps de (5) et par la machine M2 avec un temps de (3).

II.1. Codage d'une harmonie :

Pour le codage des solutions, nous avons choisi, d'adopter un modèle de codage très utilisé dans la littérature qui se base sur deux composants : Machine Selection (appelé MS) et Operation Sequence (appelé OS). L'union de ces deux derniers permettra de donner la structure globale de l'harmonie (Tableau 2).

Machine Selection (MS)					Operation Sequence (OS)				
2	1	1	2	1	2	1	2	2	1

Tableau 2 : Exemple sur la structure d'une harmonie.

II.1.1. La partie Machine Selection (MS) :

Cette partie est représentée par un vecteur d'entiers qui est de taille équivalente au nombre d'opérations (Tableau 3), chaque case de celui-ci est destinée à la $i^{ème}$ opération pour lui indiquer son choix machine, par exemple : la première case qui est de deux (2) signifie que la première opération de la première pièce, va être effectuée par la deuxième machine choisie parmi l'ensemble de machines proposées pour cette dernière.

$O_{1,1}$	$O_{1,2}$	$O_{2,1}$	$O_{2,2}$	$O_{2,3}$
2	1	1	2	1

Tableau 3 : Exemple de la partie MS.

Remarque : la deuxième machine choisie n'est pas nécessairement la machine d'ordre deux (2), mais plutôt le deuxième choix machine de cette opération.

II.1.2. La partie Operation Sequence (OS):

De même, cette partie est représentée par un vecteur d'entiers de même taille que le vecteur MS (Tableau 4), elle permet de donner un ordre d'exécutions de toutes les opérations à réaliser, les opérations correspondantes au même job sont indiquées par le même numéro et interprétées en fonction de la séquence de leurs apparitions,

par exemple la troisième case qui contient la valeur deux (2) signifie que la troisième opération à exécuter sera la deuxième opération de la deuxième pièce.

$O_{2,1}$	$O_{1,1}$	$O_{2,2}$	$O_{2,3}$	$O_{1,2}$
2	1	2	2	1

Tableau 4 : Exemple de la partie OS.

II.2. Evaluation d'une harmonie :

Dite aussi la phase de décodage, elle représente l'étape la plus importante dans toute méta-heuristique vu qu'elle permet de mesurer les performances de chaque harmonie générée. Pour cela une fonction appelée fonction d'évaluation est utilisée pour évaluer la capacité de l'harmonie en associant à chacune un poids (valeur) appelé fitness. La fonction d'évaluation dépend du problème traité et se différencie d'un problème à l'autre. Dans notre cas traitant le problème d'ordonnancement FJSP qui est de nature multiobjectifs, la fonction d'évaluation respecte trois critères, le « makespan », la « charge critique » et la « charge totale ».

- **Le makespan :** représente la date d'achèvement de touts les jobs.
- **La charge critique :** représente la durée d'exécution totale d'une machine donnée.
- **La charge totale :** représente la somme de toutes les durées de toutes les machines.

Afin de donner une compréhension et une évaluation à l'harmonie, une démarche doit être procédée comme suit :

- Pour chaque opération de la partie OS dans leurs ordres d'apparition ;
- Récupérer son affectation machine (c.à.d. le choix machine de cette opération) en accédant à la partie MS.
- A partir des données du problème, déterminer le temps d'exécution de cette opération sur cette machine.

Ces trois premières étapes se résument dans la Figure1.

- Parcourir la machine courante pour déterminer s'il existe un temps d'inoccupation, de durée convenable avec le temps d'exécution de l'opération

courante et garantissant les contraintes de disjonction et de précédence, et l'insérer en diminuant le temps d'inoccupation de la machine.

- Dans le cas contraire, c.à.d. il n'y a pas de durées inoccupées dans la machine, l'insertion se fait à la fin de la machine en respectant toujours les contraintes de disjonction et de précédence.

- Après avoir construit l'ordonnancement correspondant à cette solution, récupérer les valeurs du Makespan de la charge totale de l'atelier et de la charge critique.

- Calculer la fitness finale de la solution en attribuant des facteurs pour chacun des critères précédents.

La Figure 2 représente le diagramme de Gantt de la solution.

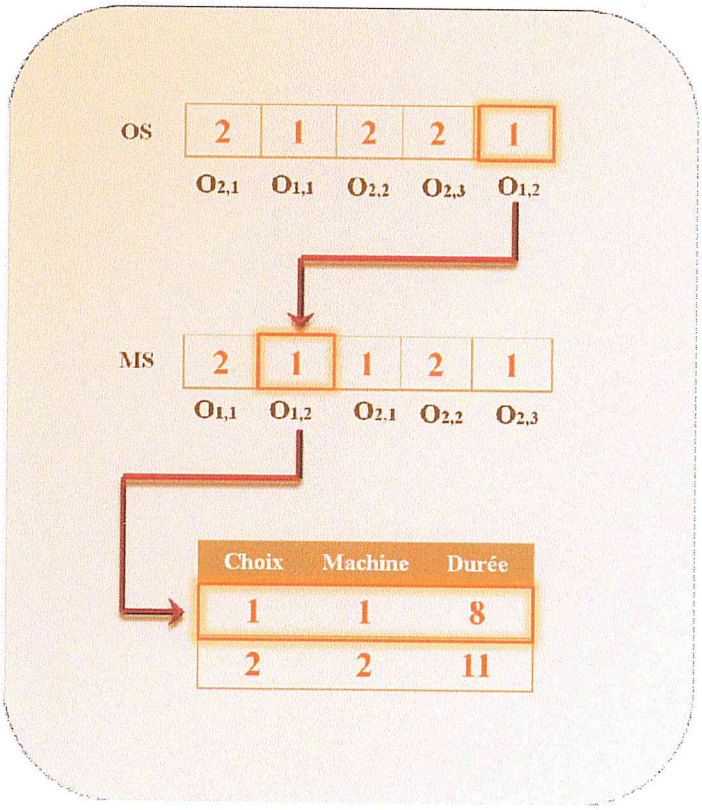


Figure 1: Exemple résumant les trois premières étapes.

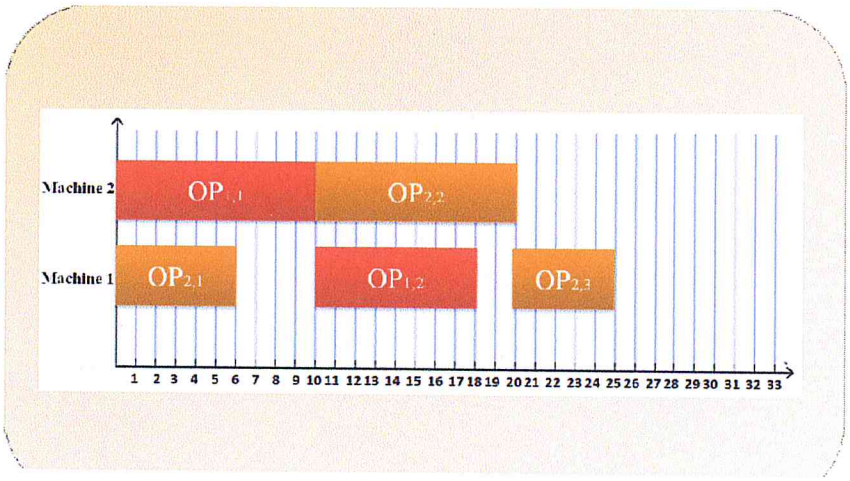


Figure 2: Résultat du FJSP représenté par le diagramme de Gantt.

Remarque : L'ancienne version (3.A) de la méthode d'évaluation de la méta-heuristique « Harmony Search » consistait à faire un placement simple des opérations, sans prendre en considération le temps d'inoccupation de la machine, pour cela nous avons pensé, afin d'améliorer les performances des résultats trouvées, d'adopter une nouvelle technique (3.B) permettant de rechercher les zones de machines libres et de les occuper. Cette nouvelle technique a permis effectivement de minimiser le temps d'inoccupation de la machine et d'apporter de bons résultats.

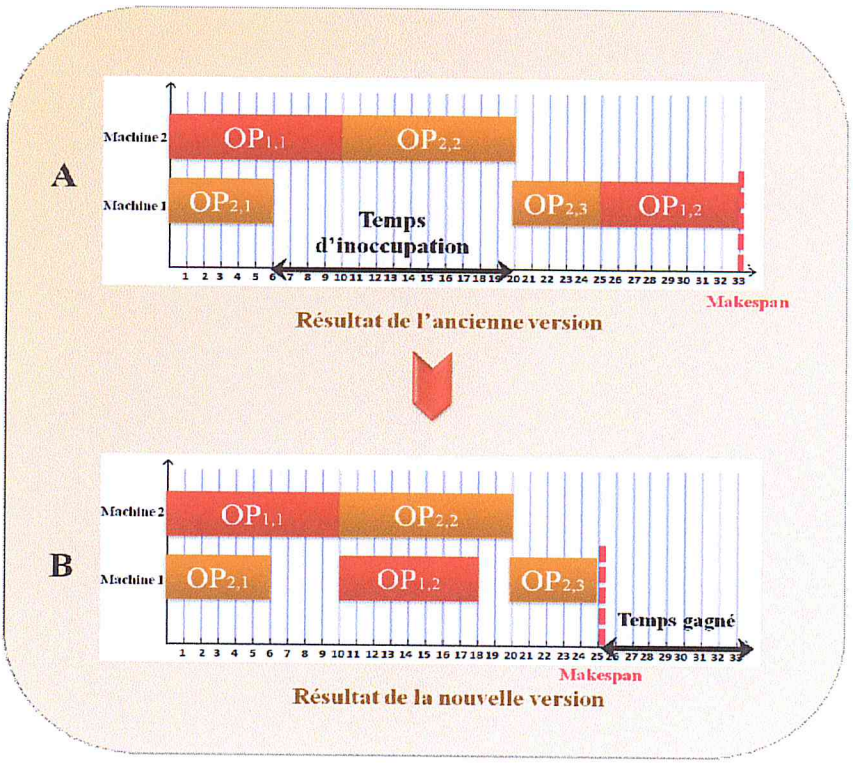


Figure 3: Comparaison entre les résultats de l'ancienne et la nouvelle version.

La figure 3 présentée ci-dessus montre les résultats des deux types d'évaluations et la différence entre eux. Le premier type d'évaluation qui est utilisé dans l'ancienne version (3.A) a donné un résultat de (33) comme valeur de makespan avec une durée très importante non occupée par la machine. Tandis que la deuxième technique d'évaluation utilisée dans la nouvelle version (3.B) a permis de réduire le temps d'inoccupation de la machine en améliorant la valeur de makespan grâce à l'insertion de la dernière opération dans l'espace inoccupé.

II.3. Etapes de l'algorithme Harmony Search:

Les principales étapes de l'algorithme « Harmony Search » sont résumées dans l'organigramme proposé dans la figure ci-dessous (figure 3).

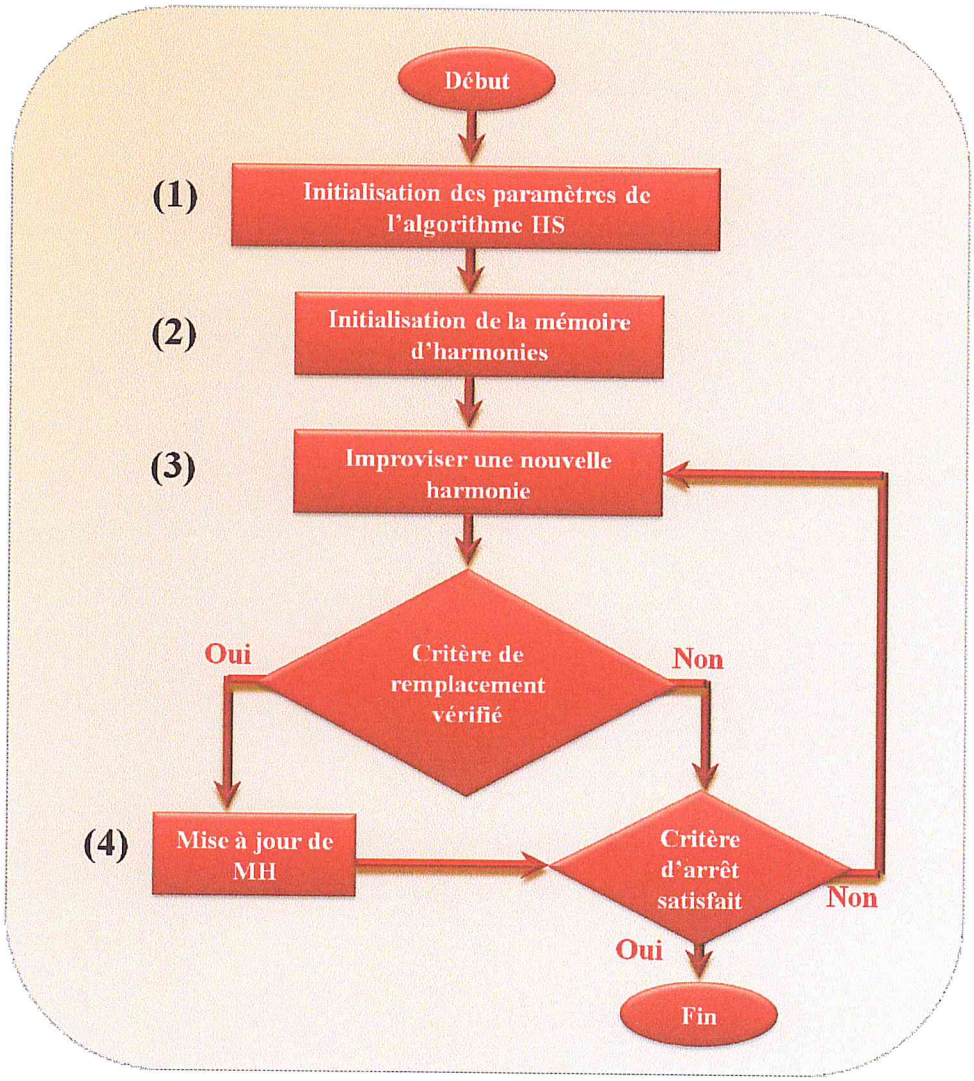


Figure 4: Procédure d'optimisation de l'algorithme HS.

II.3.1. Initialisation des paramètres de l'algorithme :

L'initialisation des paramètres joue un rôle très important dans l'implémentation de l'algorithme, car elle permet d'influencer de manière très sensible les résultats obtenus. Les paramètres de l'algorithme « Harmony Search » qui sont exigés pour résoudre le problème d'ordonnancement JSF, sont spécifiés ci-dessous :

Taille de la mémoire d'harmonies :

Cet élément représente le nombre d'harmonies présentes à chaque itération. Une mémoire de très petite taille ne pourra pas évoluer de manière satisfaisante, car elle ne peut pas explorer tout l'espace de recherche et par la suite, elle peut ne pas avoir les bonnes solutions. Dans le cas où la mémoire d'harmonies aura une grande taille, le temps de calcul sera accru très rapidement.

Nombre des itérations :

Ce paramètre doit être suffisant pour permettre une exploration correcte et efficace de l'espace de recherche, mais pas trop grand pour ne pas devenir pénalisant sur le plan temps de calcul.

Taux d'acceptation :

Ce paramètre est utilisé pour vérifier la première règle de la méta-heuristique HS. En effet si ce taux est très élevé, la probabilité d'accepter une solution à partir de la mémoire sera augmentée. Dans le cas contraire, c.à.d. le taux d'acceptation est trop faible, ça risque de refuser toutes les harmonies de la mémoire.

Taux d'ajustement :

Ce taux correspond à la deuxième règle de la méta-heuristique, utilisé essentiellement pour échapper aux minima locaux et améliorer les solutions obtenues. Il doit demeurer assez faible afin de ne pas perturber les informations des harmonies de la mémoire.

II.3.2. Initialisation de la mémoire d'harmonies :

L'initialisation de la mémoire d'harmonies constitue le point de départ de l'algorithme de la recherche d'harmonies; il est généralement admis que l'efficacité ultérieure de l'algorithme est étroitement liée à la qualité de la mémoire d'harmonies.

En effet, il est intuitivement préférable d'avoir une mémoire d'harmonies constituant un échantillon représentatif de toutes les solutions qui existent dans l'espace de recherche du problème.

Dans notre implémentation, la mémoire d'harmonies est remplie par des harmonies générées aléatoirement.

Le remplissage aléatoire présente l'avantage de proposer une mémoire variée, assurant un bon recouvrement de l'espace de recherche. Il permet de générer une mémoire acceptable, quand aucune information n'est a priori disponible sur la localisation de l'optimum.

II.3.3. Improvisation d'une nouvelle harmonie :

L'improvisation est l'étape la plus importante dans la méta-heuristique « Harmony Search » puisque elle permet la génération d'une nouvelle harmonie en utilisant les différents operateurs de l'approche. L'algorithme présenté ci-après montre en générale le principe de l'improvisation.

```
Pour i = 0 à taille de l'harmonie
    Déterminer une probabilité d'acceptation hmcr
    Si (hmcr < HMCR)
        Récupérer la i^eme valeur val de l'harmonie
        Déterminer une probabilité d'ajustement par
            Si (par < PAR)
                Ajuster la valeur val
            Sinon accepter la valeur val
    Sinon déterminer une valeur aléatoire val
    Insérer val dans la nouvelle harmonie
Fin Pour.
```

Vu que notre harmonie se compose de deux parties différentes, partie MS et partie OS, nous avons implémenté deux types d'improvisations, une pour la partie MS et l'autre pour la partie OS.

Les deux types d'improvisation se basent sur l'opérateur d'ajustement, qui est à son tour implémenté de deux façons distinctes, une pour la partie MS, et l'autre pour la partie OS.

La différence entre la partie MS et la partie OS apparait dans le parcours et l'ajustement ; pour la partie MS, nous suivons un schéma de parcours et d'ajustement classiques et qui ressemble beaucoup au schéma général de l'improvisation, tandis que le schéma suivi pour la partie OS est implémenté d'une manière totalement différente, assurant d'éviter le risque d'avoir des harmonies irréalisables.

Les deux types d'ajustement seront expliqués dans ce qui suit, et la figure présentées ci-dessous résume la différence entre les deux parties : MS et OS.

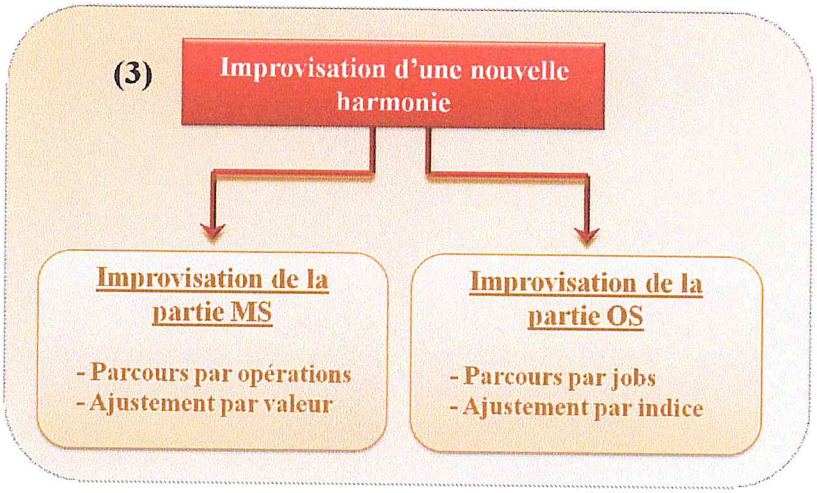


Figure 5: La différence entre les deux types d'improvisation.

Improvisation de la partie MS :

L'improvisation de la partie MS utilise le modèle d'ajustement par valeur suivant un parcours par opération, elle est résumée dans le pseudo code suivant :

```
Pour i=0 à nombre opérations
    Récupérer une harmonie aléatoire à partir de la mémoire d'harmonies
    Récupérer val la i^eme valeur du vecteur MS de l'harmonie choisie
    Générer une valeur hmcr entre 0 et 1
    Si hmcr < HMCR
        Générer une valeur par entre 0 et 1
        Si par < PAR
            val = val_ajustée   //Ajuster la valeur du choix machine de l'opération.
        Sinon val = val         //Accepter la valeur du choix machine.
    Sinon val = val_aléatoire   // Déterminer un choix machine aléatoire.
    Insérer val dans le vecteur nouv_MS
Fin Pour
```

Les étapes de l'improvisation de la partie MS sont détaillées dans l'organigramme suivant :

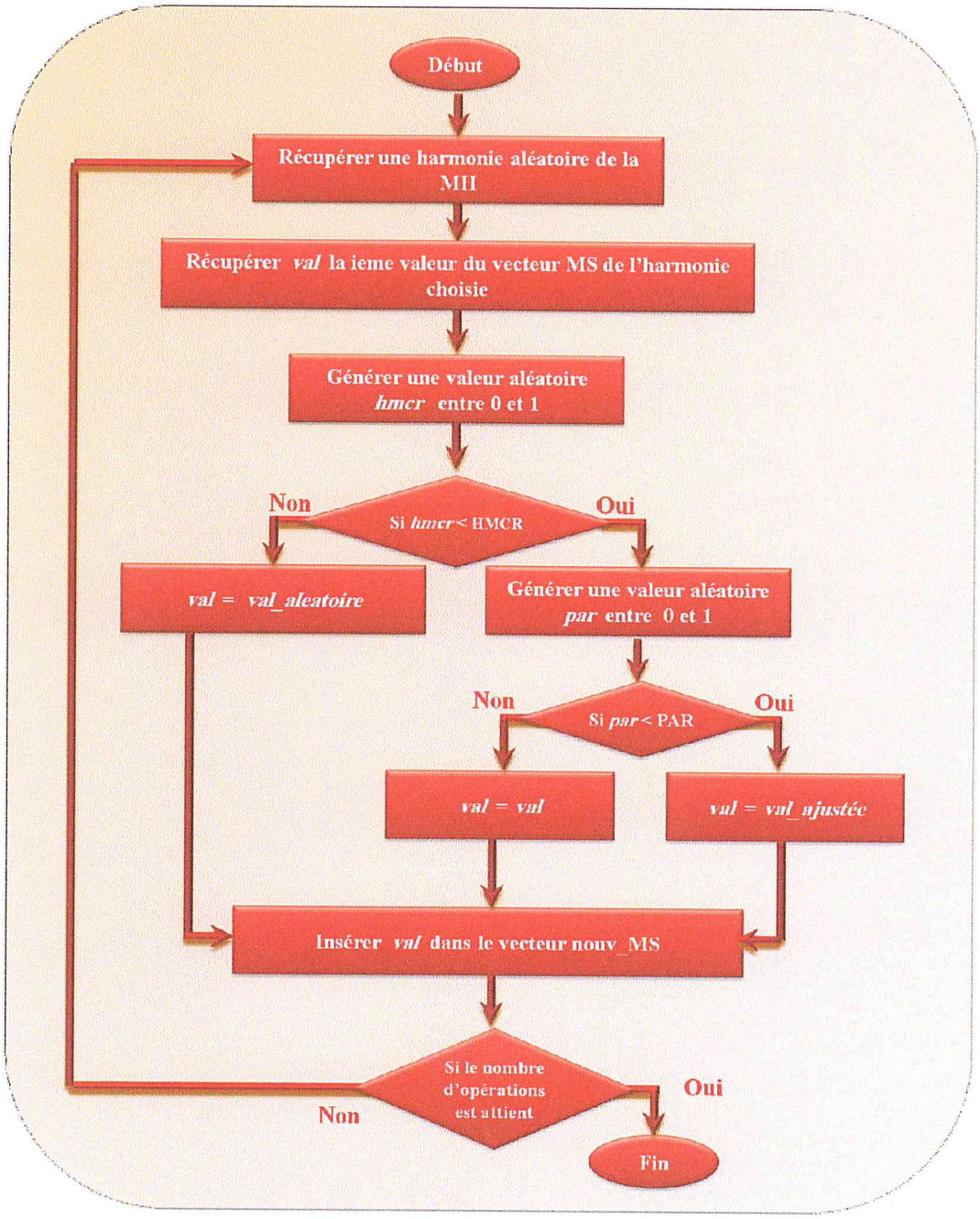


Figure 6: Procédure de l'improvisation de la partie MS.

Ajustement par valeur :

C'est le type d'ajustement qui est destiné à la partie MS de l'harmonie, il consiste à modifier la valeur du choix machine d'une opération donnée en l'incrémentant ou en la décrémentant d'une unité, de telle sorte que la nouvelle valeur doit être prise entre un (1) et le nombre maximum du choix machine de cet opération.

L'organigramme suivant montre la stratégie d'ajustement par valeur utilisé dans notre algorithme de recherche d'harmonies.

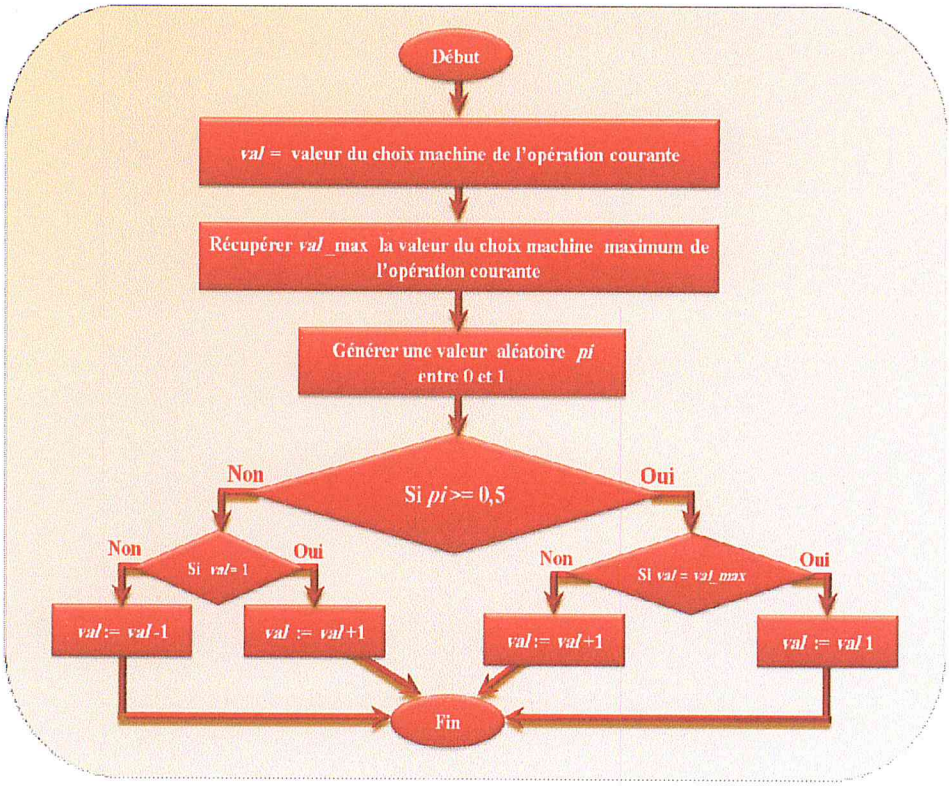


Figure 7: Procédure d'ajustement par valeur.

Exemple 01 : Cet exemple explique la procédure d'ajustement par valeur.

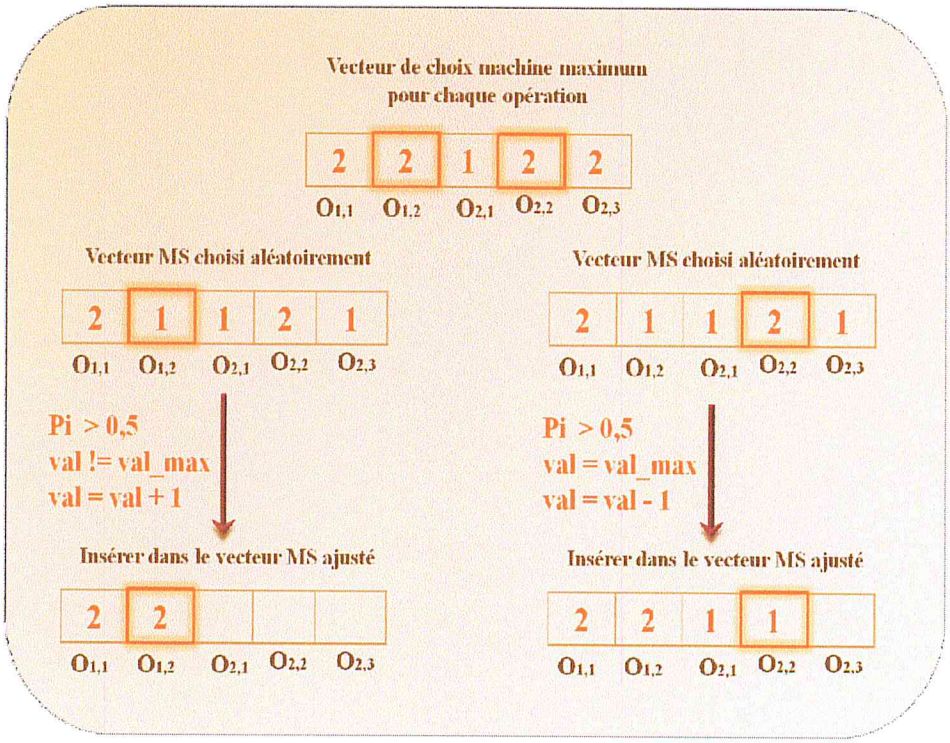


Figure 8 : Exemple d'ajustement par valeur.

Exemple 02: Dans cet exemple, nous allons expliquer la démarche de l'improvisation de la partie MS et ses résultats, pour cela nous avons initialisé les paramètres d'acceptation et d'ajustement comme suit :

$HMCR = 0.97$

$PAR = 0.005$

Suivant l'exemple pris au départ, nous considérons que le choix machine maximum pour chaque opération est tel que décrit dans le vecteur suivant (Tableau 5):

$O_{1,1}$	$O_{1,2}$	$O_{2,1}$	$O_{2,2}$	$O_{2,3}$
2	2	1	2	2

Tableau 5: Vecteur de choix machine maximum pour chaque opération.

Pour chaque étape, nous récupérons une harmonie d'une manière aléatoire à partir de la mémoire d'harmonies, puis nous déterminons une valeur « *hmcr* » pour l'acceptation et une valeur « *par* » pour l'ajustement et une valeur « *pi* » pour déterminer le signe d'ajustement. Nous répétons le processus autant de fois que le nombre d'opérations qui existent.

Etape 1: i = 0

hmcr = 0.92 < **0.97**
par = 0.1 > **0.005**
Acceptation sans ajustement

2	1	1	2	1

Vecteur MS récupéré aléatoirement

2				

Vecteur nouv_MS

Etape 2 : i = 1

hmcr = 0.96 < **0.97** ;
par = 0.004 < **0.005** ;
pi = 0.6 > **0.5** ;
Acceptation avec ajustement.

2	1	1	2	2

Vecteur MS récupéré aléatoirement

2	2			

Vecteur nouv_MS

Etape 3 : i = 2

Acceptation directe car l'opération courante dispose d'un seul choix machine.

1	2	1	1	2

Vecteur MS récupéré aléatoirement

2	2	1		

Vecteur nouv_MS

Etape 4 : i = 3

hmcr = 0.93 < **0.97** ;
par = 0.006 >**0.005**;
Acceptation sans ajustement.

1	2	1	2	1

Vecteur MS récupéré aléatoirement

2	2	1	2	

Vecteur nouv_MS

Etape 5 : i = 4

hmcr = 0.98 > **0.97** ;
Génération d'une valeur aléatoire.

2	2	1	1	1

Vecteur MS récupéré aléatoirement

2	2	1	2	2

Vecteur nouv_MS

Le résultat de toutes les étapes de l'improvisation sera le vecteur MS improvisé qui est présenté ci-après (Tableau 6):

2	2	1	2	2

Tableau 6 : Vecteur MS improvisé.

Improvisation de la partie OS :

L'improvisation de la partie OS est totalement différente de celle de la partie MS, car ses valeurs sont liées ce qui provoque un risque d'avoir une harmonie (solution) irréalisable si nous suivons le même principe d'improvisation utilisé pour la partie MS. Pour cela nous avons utilisé, pour la partie OS, un schéma d'improvisation basé sur un parcours par job utilisant un autre modèle d'ajustement qui est l'ajustement par indice.

Les différentes étapes de l'improvisation de la partie OS sont détaillées dans le pseudo code suivant :

//Etape 1 : Extraction des vecteurs har_OS.

Tant que le nombre de pièces n'est pas atteint

1. Déterminer aléatoirement un numéro de pièce en faisant un tirage sans remise

2. Choisir une harmonie d'une manière aléatoire à partir de la mémoire d'harmonies

3. Ajouter une copie de l'harmonie au vecteur har_OS // vecteur de vecteur OS, sa taille en nombre de jobs

4. Accéder au vecteur OS et remettre toute valeur différente à la valeur de la pièce choisie à nul

Fin TQ

//Etape 2 : construction du vecteur har_OS_adj.

Tant Que la fin du vecteur har_OS n'est pas atteinte

Déterminer une valeur *hmcr* entre 0 et 1

Si *hmcr* < HMCR

Déterminer une valeur *par* entre 0 et 1

Si *par* < PAR

Ajuster par indice le vecteur OS //appeler la méthode «ajuster ()»

L'ajouter au vecteur har_OS_adj

Sinon ajouter le vecteur OS au vecteur har_OS_adj sans ajustement

Sinon générer une harmonie aléatoirement

Répéter 3 et 4

L'ajouter au vecteur har_OS_adj

Fin TQ

//Etape 3 : Détermination du vecteur OS ajusté.

Tant Que i<= le nombre d'opération

Tant Que j<= la taille du vecteur har_OS_adj

Récupérer la valeur de l'opération à la position (i, j)

Fin Tant Que

Fin Tant Que

Si le nombre d'opération est à 1, l'insérer directement à nouv_OS

Sinon insérer les opérations d'une manière aléatoire à nouv_OS

Ajustement par indice :

Ce modèle d'ajustement est consacré pour la partie OS de l'harmonie. Il consiste à ajuster aléatoirement l'indice de la première opération dans le vecteur OS, de telle sorte que sa nouvelle position prenne celle de la première case contenant un nul.

La technique utilisée dans notre algorithme de recherche d'harmonies est expliquée dans l'organigramme suivant :

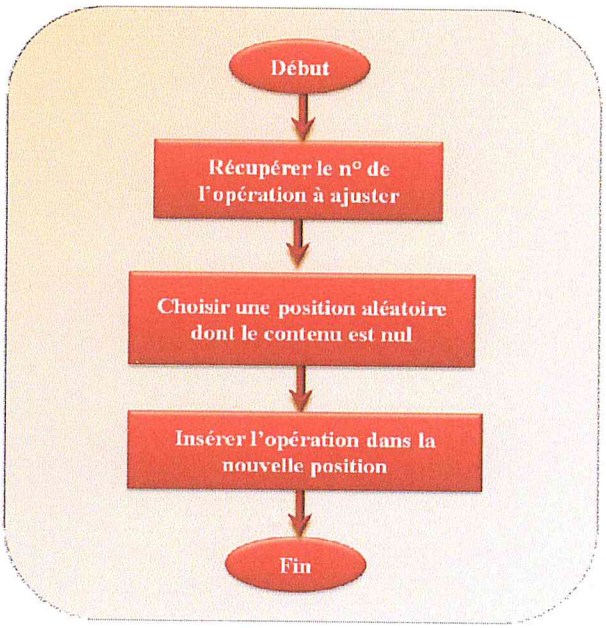


Figure 9 : Procédure d'ajustement par indice.

Exemple 03: Cet exemple explique la procédure d'ajustement par indice.

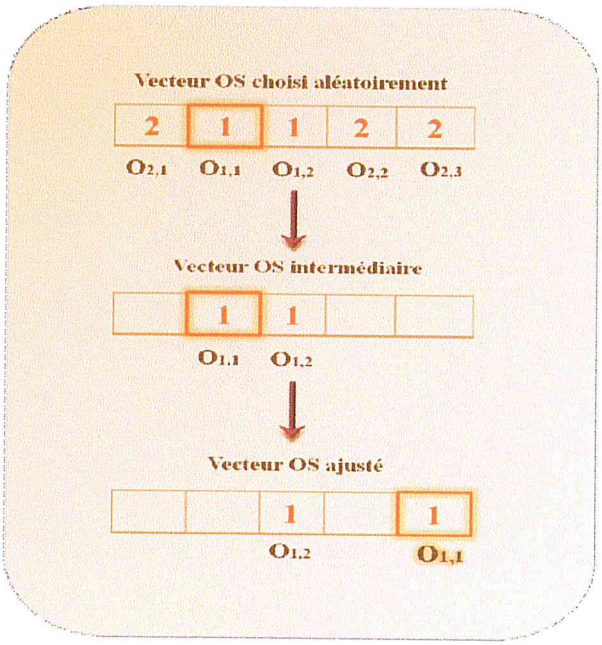


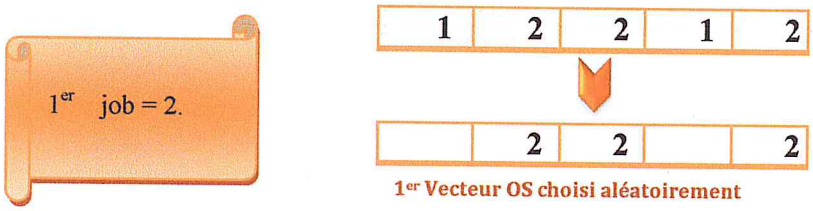
Figure 10 : Exemple d'ajustement par indice.

Exemple 04: Dans cet exemple, nous allons expliquer la démarche de l'improvisation de la partie OS et ses résultats, en prenant les mêmes valeurs de paramètres d'acceptation et d'ajustement initialisés dans l'exemple précédent et qui sont :

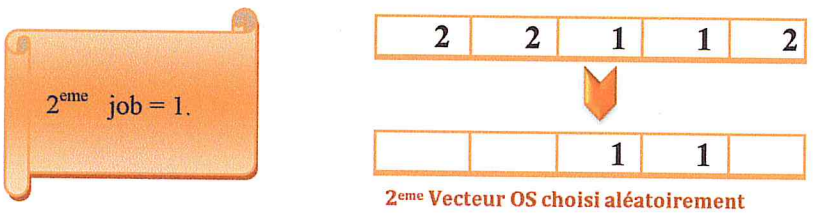
$HMCR = 0.97$

$PAR = 0.005$

Etape 1 : Dans cette étape, nous récupérons des harmonies aléatoires à partir de la mémoire d'harmonies en nombre de pièces qui existent dans l'atelier, et nous faisons correspondre chaque harmonie à une pièce choisie aléatoirement de telle sorte que son vecteur OS ne contiendra que les opérations de celle-ci.



1er Vecteur OS choisi aléatoirement



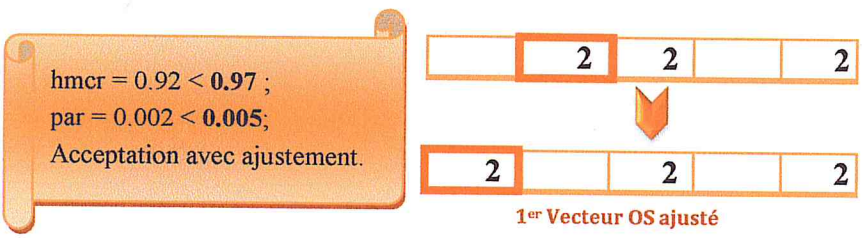
2eme Vecteur OS choisi aléatoirement

Le résultat de la première étape se résume dans le vecteur har_OS suivant:

har_OS				
	2	2		2
		1	1	

Tableau 7 : Vecteur har_OS.

Etape 2 : Pour chaque vecteur OS appartenant au vecteur har_OS, déterminer une valeur *hmcr* pour l'acceptation et une valeur *par* pour l'ajustement.



1er Vecteur OS ajusté

hmcr = 0.98 > **0.97**;
Génération d'une nouvelle harmonie aléatoire.

1	1	2	2	2

↓

1	1			

2eme Vecteur OS ajusté

Le résultat de cette étape sera mis dans le vecteur har_OS_adj comme il est montré ci- dessous :

har_OS_adj				
2		2		2
1	1			

Tableau8 : Vecteur har_OS_adj.

Etape 3 : Dans cette étape l'algorithme fait parcourir le vecteur har_OS_adj pour construire le nouveau vecteur OS improvisé, le parcours sera expliqué dans les sous phases suivantes :

3.1 : i = 0, j = **0** / i =1, j = **0.**

Insertion aléatoire ;

1ere opération aléatoire = 1 ;

2eme opération aléatoire = 2.

har_OS_adj				
2		2		2
1	1			

↓

1	2			

Vecteur OS improvisé

3.2 : i = 0, j = **1** / i =1, j = **1.**

Insertion directe de l'opération.

har_OS_adj				
2		2		2
1	1			

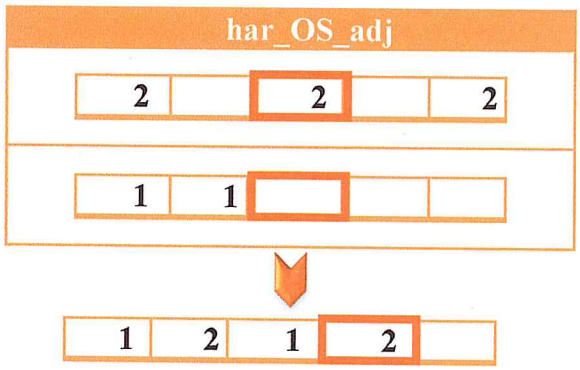
↓

1	2	1		

Vecteur OS improvisé

3.3 : i = 0, j = 2 / i =1, j = 2.

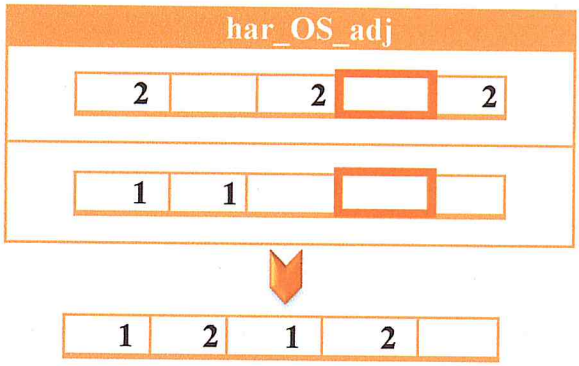
Insertion directe de l'opération.



Vecteur OS improvisé

3.4 : i = 0, j = 3 / i =1, j = 3.

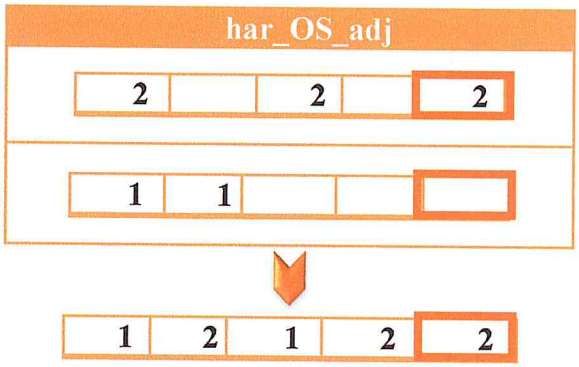
Il n'y a pas d'opérations à insérer.



Vecteur OS improvisé

3.5 : i = 0, j = 4 / i =1, j = 4.

Insertion directe de l'opération.



Vecteur OS improvisé

Le résultat final de l'improvisation de la partie OS sera le vecteur OS improvisé qui est présenté ci-après (Tableau 9) :

1	2	1	2	2

Tableau 9 : Vecteur OS improvisé.

II.3.4. Mettre à jour la mémoire d'harmonies :

Apres avoir terminé l'improvisation de notre nouvelle harmonie avec ses deux parties MS et OS, une mise à jour de la mémoire d'harmonies sera nécessaire.

Dans notre algorithme nous avons implémenté deux façons de mise à jour, la mise à jour systématique qui représente l'operateur de remplacement, elle suit le principe suivant : la plus mauvaise harmonie dans la MH en termes de valeur de la fonction objectif (fitness) sera toujours remplacée par la nouvelle harmonie improvisée.

La mise à jour conditionnelle qui suit ce principe : Si la nouvelle harmonie est meilleure que la plus mauvaise harmonie de la MH, en termes de valeur de fonction objectif (fitness), cette dernière est insérée dans la MH et la plus mauvaise harmonie sera retirée de la MH.

II.3.5. Vérification du critère d'arrêt :

Dans l'implémentation de notre algorithme de recherche d'harmonies, le critère d'arrêt est vérifié lorsque le nombre maximum d'itérations est atteint.

Remarque En plus de l'operateur d'ajustement, il existe d'autres operateurs tels que l'operateur de remplacement modifié (décrit dans la partie **II.3.4.**), l'operateur de sélection (décrit dans la partie **II.3.6.**), et l'operateur de mutation intelligente (décrit dans la partie **II.3.7.**), qui peuvent être rajoutés comme ils peuvent être enlevés.

II.3.6. Operateur de sélection

C'est un operateur issu des algorithmes évolutionnaires et n'appartient pas aux operateurs de base de la méthode « Harmony Search ». Utilisé afin de tester l'amélioration de la méthode, il pourra être rajouté dans la phase d'improvisation pour récupérer une harmonie de la mémoire d'harmonies au lieu de tirer une aléatoire.

La méthode de sélection implémentée, dans cette partie, est la sélection par tournois qui permet de favoriser au cours du temps les harmonies les mieux adaptées en terme de fitness. Son principe est le suivant :

a. Sélectionner deux harmonies aléatoires, H1 et H2 ;
b. Récupérer la meilleure harmonie en terme de fitness.

II.3.7. Opérateur de mutation intelligente

C'est un operateur issu des algorithmes évolutionnaires. Il est spécifique au problème d'ordonnancement de type Job Shop Flexible et destiné pour réduire la

charge critique. Cet operateur est utilisé après l'étape de l'improvisation comme le montre la Figure 8. Son principe est expliqué dans les étapes suivantes :

a. Déterminer la machine la plus chargée dans le système.

b. Tirer aléatoirement, à partir de la machine la plus chargée, une opération qui possède plusieurs choix machine.

c. Affecter cette opération à une autre machine sélectionnée aléatoirement parmi ses choix de machine.

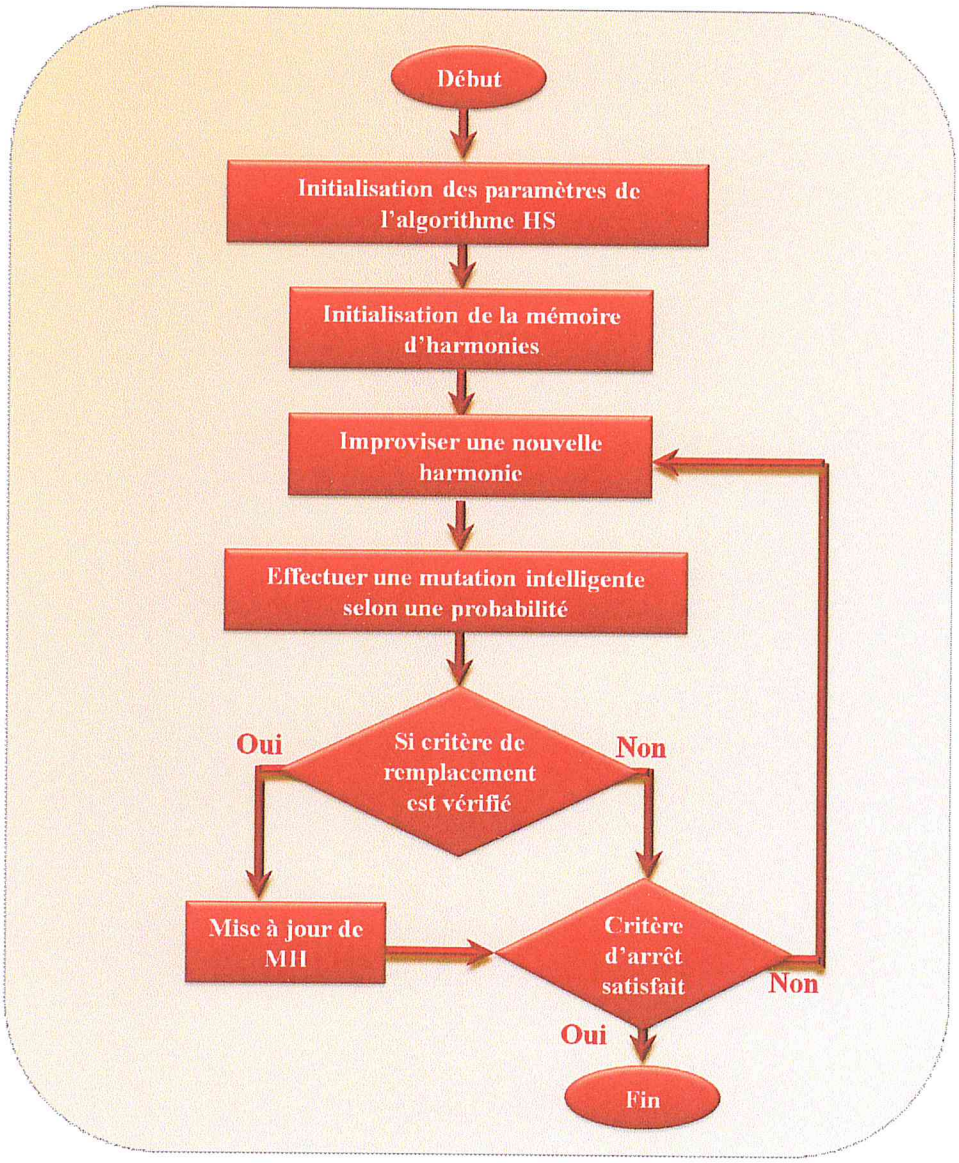


Figure 11 : Procédure de la méthode Harmony Search en utilisant l'operateur de la mutation intelligente.

III. Implémentation parallèle de la méta-heuristique Harmony Search :

III.1. Description générale du système :

Le système multi agents que nous proposons pour l'implémentation parallèle de notre approche se compose de deux classes d'agents :

La première classe d'agents, dispose d'un seul agent, qui s'appelle « *agent lanceur* ». Cet agent est responsable du lancement de tout le système de calcul en envoyant à tous les agents de l'autre classe, les paramètres de l'algorithme à exécuter (Figure 12).

La deuxième classe contient les agents Ordonnanceurs. Chaque agent exécute le processus de résolution, qui est l'algorithme « Harmony Search » (Figure 13).

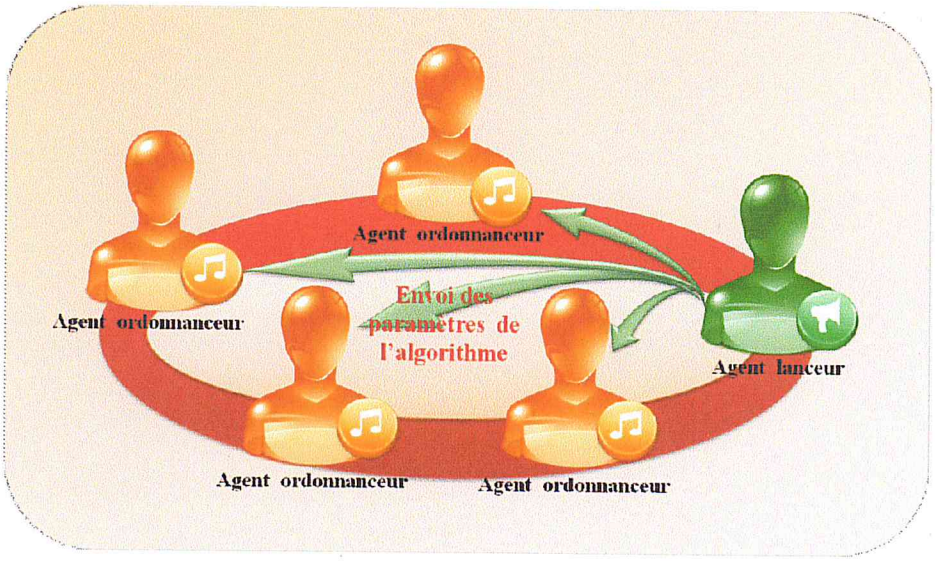


Figure 12 : Envoi des paramètres de l'algorithme aux agents Ordonnanceurs.

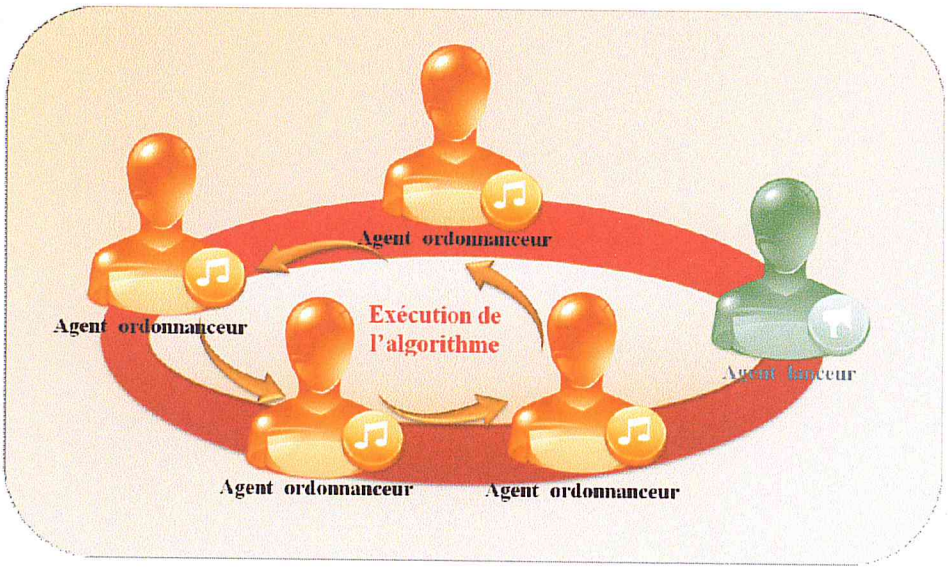


Figure 13 : Exécution de l'algorithme par les agents Ordonnanceurs.

III.2. Description du modèle insulaire :

Dans notre travail, le système multi-agents que nous avons implémenté suit le modèle insulaire (Chapitre 2, Figure 3) qui est le modèle le plus utilisé pour le développement des méta-heuristiques évolutionnaires parallèles. Ce modèle repose sur le principe des iles qui sont définies comme étant des ensembles de solutions dispersés sur plusieurs hôtes (agents), de telle façon que chaque hôte exécute son propre algorithme. La migration est utilisée pour l'échange de solutions entre les îles (Figure 14).

Pour la communication des solutions, nous avons choisi d'adopter le modèle synchrone qui permet l'échange des meilleures harmonies entre les hôtes voisins, après un certain nombre d'itérations, ce qui permet une migration périodique selon un critère de décision aveugle.

Chaque hôte possède un voisin prédéfini, vu que la topologie que nous avons suivie est une typologie en anneau. Pour qu'ils puissent s'échanger les solutions, nous avons utilisé une politique de sélection et de remplacement d'harmonies selon une stratégie élitiste garantissant la sélection des meilleures harmonies.

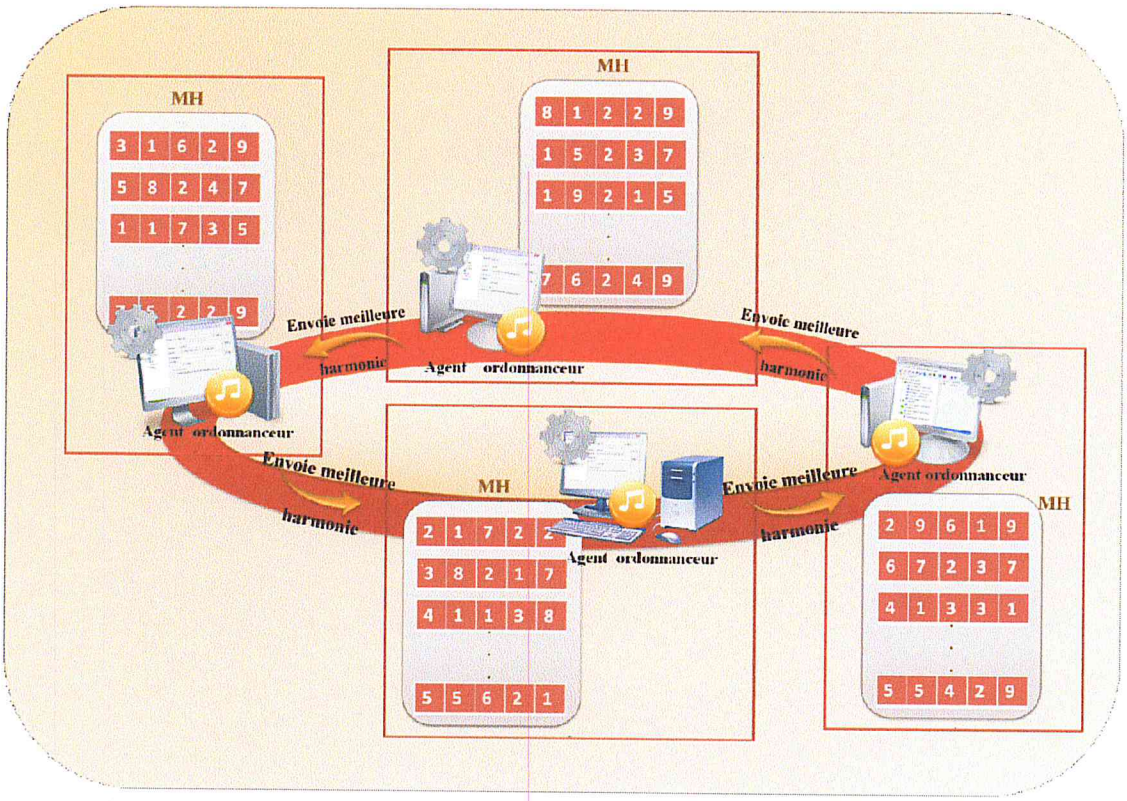


Figure 14 : Description du modèle insulaire.

III.3. Types d'agents :

Chaque agent composant notre système possède ses propres connaissances statiques et dynamiques et son propre comportement. De plus, chaque agent possède des accointances qui sont les agents voisins.

Dans la suite de cette partie, nous détaillons les différentes taches de chaque type d'agents.

III.3.1. Agent lanceur :

Il lance l'exécution de tous les agents lancés de l'autre classe en leur envoyant tous les paramètres nécessaires, sous forme d'objet de type « *Parametre* ». Pour cela il accède au DF (Directory Facilitor) afin de récupérer la liste des agents Ordonnanceurs. Ces derniers s'enregistrent au préalable auprès du DF.

III.3.2. Agent Ordonnanceur :

Chaque agent de ce type exécute son propre algorithme « Harmony Search », pour cela il doit s'enregistrer auprès du DF afin qu'il puisse récupérer les paramètres

de l'algorithme envoyés par l'agent lanceur (Le problème à résoudre, les paramètres d'exécution, le voisin direct).

Durant l'exécution de l'algorithme, les agents Ordonnanceurs communiquent leurs solutions au sein d'une topologie en anneau unidirectionnel. En effet l'échange de meilleures harmonies se fait après un nombre fixe d'itérations (taux de migration). L'agent Ordonnanceur code la meilleure harmonie en un message de type chaine de caractères.

D'un autre coté, l'agent Ordonnanceur accède selon le taux de migration à sa boite aux lettres pour récupérer les messages reçus de son voisin, les décode en une harmonie et procède à leurs insertions dans sa mémoire d'harmonies, selon la règle de remplacement.

Quand le nombre d'itérations est atteint, chaque agent Ordonnanceur envoie une notification à l'agent lanceur pour l'informer que l'exécution de l'algorithme est terminée.

Tout agent Ordonnanceur dispose d'un ensemble de connaissances statiques et dynamiques, ses connaissances statiques englobent :

- Tous les paramètres de l'algorithme «Harmony Search » (la taille de la MH, le nombre d'itérations, etc.) ;
- Le taux de migration ;
- Toutes les données du problème ;
- L'agent voisin.

Ses connaissances dynamiques sont composées de :

- Le nombre d'itérations effectuées ;
- La solution courante ainsi que son temps de calcul ;
- La meilleure solution rencontrée et son temps de calcul.

III.4. Diagramme de séquence :

Le diagramme de séquence (Figure 12) représente une exécution de l'algorithme parallèle implémenté. Après l'envoi de toutes les notifications des agents Ordonnanceurs, l'agent lanceur peut les relancer autant de fois qu'il est indiqué dans l'interface.

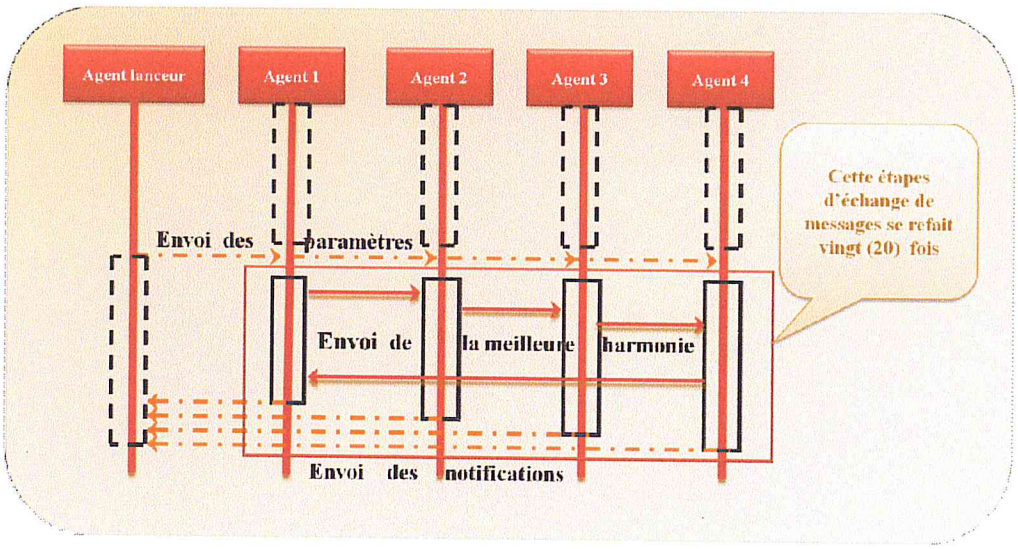


Figure 15 : Diagramme de séquence.

Remarque : L'envoi de la meilleure harmonie qui se fait entre chaque deux voisins, se refait, d'après le nombre d'itérations et le taux de migration que nous avons fixé, vingt (20) fois pour le même essai.

III.5. Diagrammes d'activités :

Vu qu'il y a deux types d'agents, donc deux types de diagrammes d'activités doivent être mis en œuvre.

III.5.1. Diagramme d'activités de l'agent lanceur :

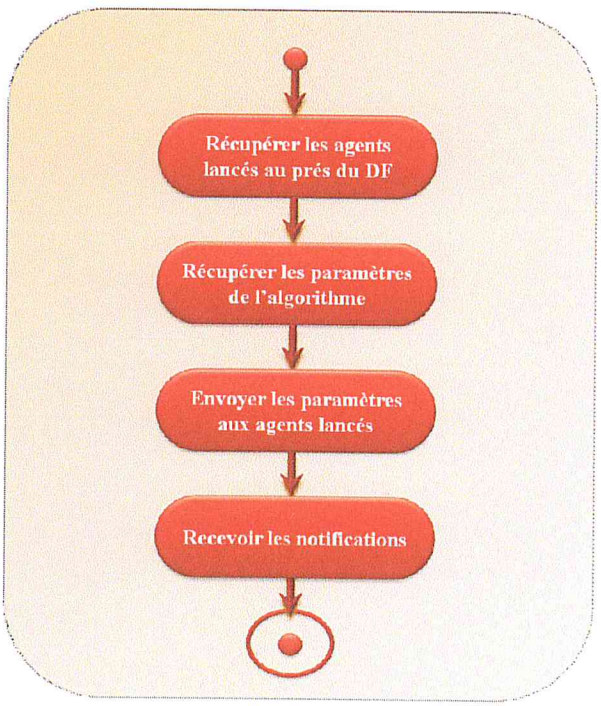


Figure 16 : Diagramme d'activités de l'agent lanceur.

III.5.2. Diagramme d'activités de l'agent Ordonnanceur :

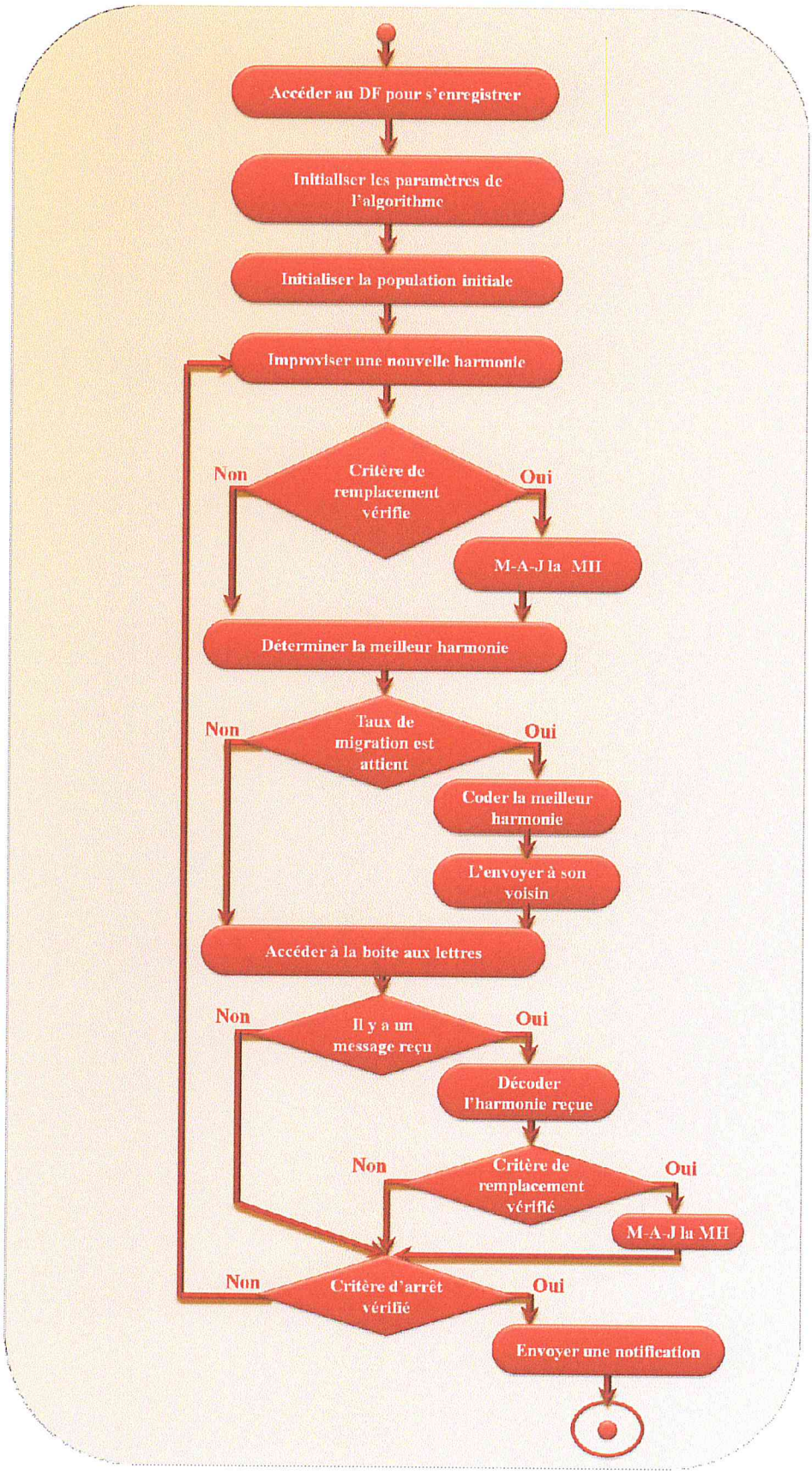


Figure 17 : Diagramme d'activités de l'agent Ordonnanceur.

IV. Conclusion :

Dans ce chapitre, nous avons présenté un algorithme « Harmony Search » parallèle pour l'ordonnancement Job Shop Flexible multicritères. Nous avons aussi explicité ces différentes étapes avec des exemples.

Dans le chapitre suivant, et afin de valider notre approche, nous présenterons une série d'expérimentations permettant de le vérifier et de le comparer à d'autres méthodes développées dans la littérature.

Chapitre IV

Expérimentations, synthèses et validation.

I. Introduction :

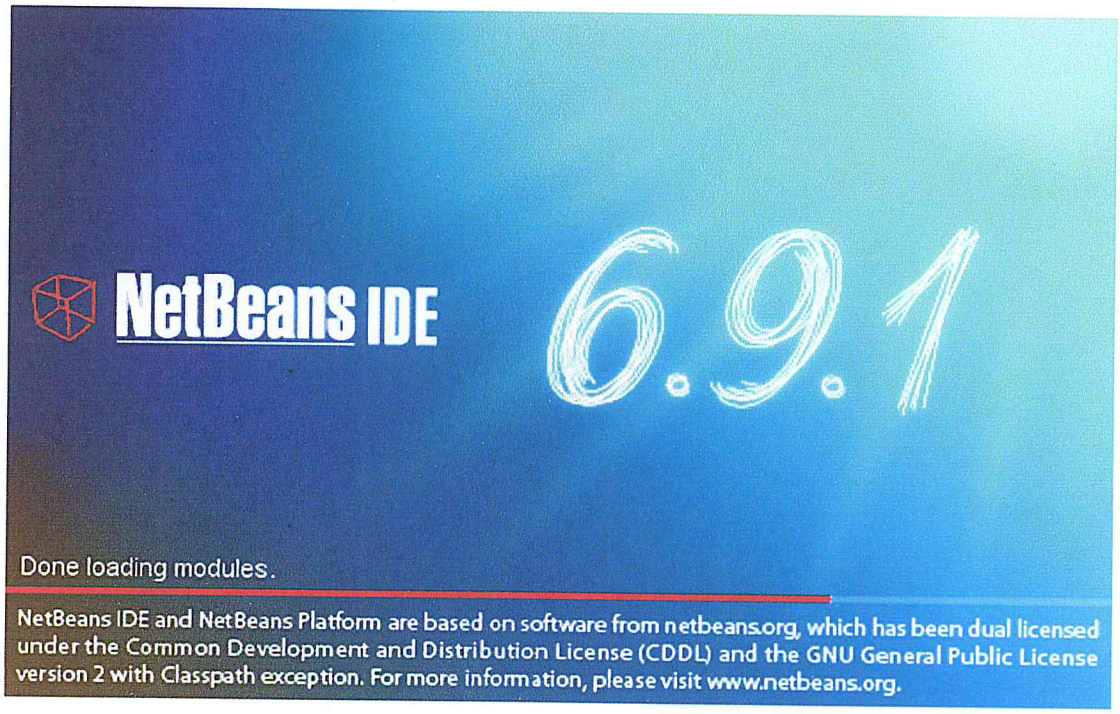
Afin de mieux cerner la portée de notre application sur l'algorithme « Harmony Search » destiné aux problèmes d'ordonnancement de la production de type Job Shop Flexible et la nécessité d'améliorer sa version classique, il convient de faire des expérimentations permettant d'évaluer notre système.

Dans ce chapitre nous allons parler beaucoup plus de l'approche parallèle et ses phases d'implémentation en donnant une synthèse des résultats relatifs aux expérimentations effectuées sur les deux approches développées (séquentielle et parallèle) à l'aide de graphiques et tableaux récapitulatifs.

II. Outils de développement :

II.1. Langage de programmation et éditeur :

Pour implémenter notre application, nous avons utilisé le langage de programmation JAVA sous l'éditeur NetBeans (version 6.9.1)



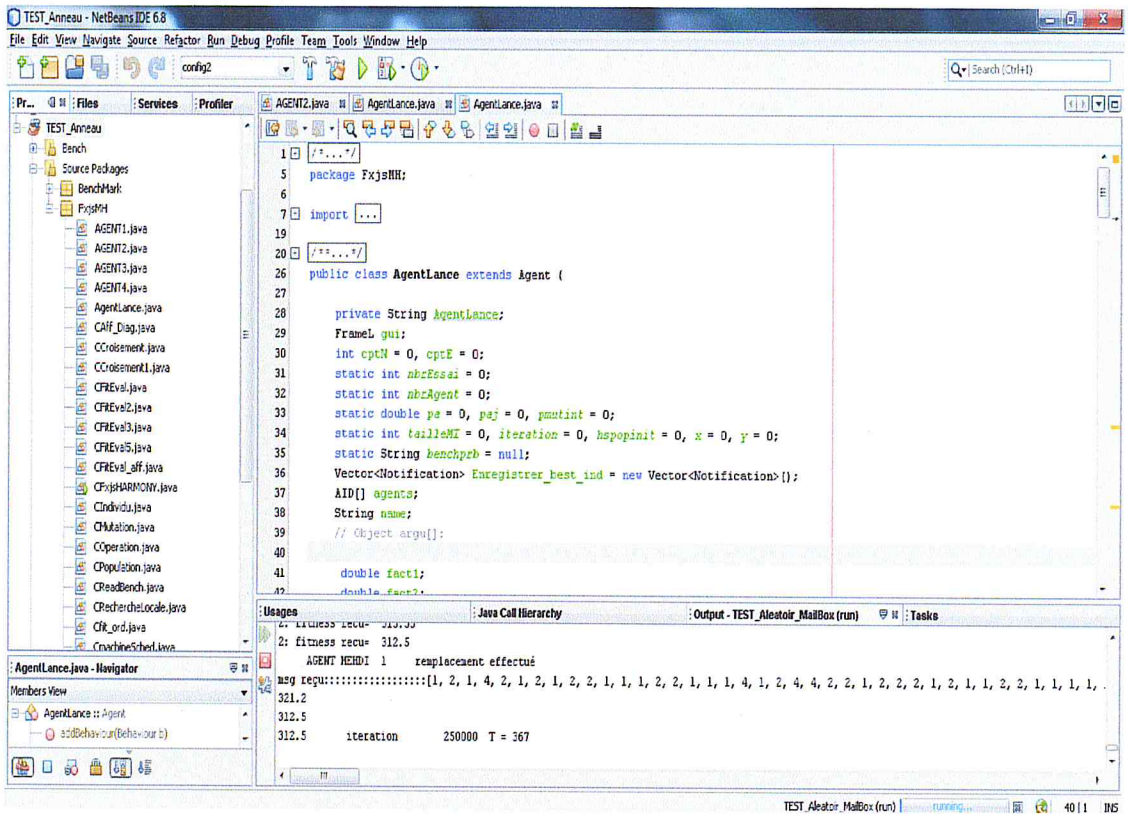


Figure 1 : Interface de l'éditeur NetBeans.

II.2. Plateforme de développement JADE :

JADE qui signifie Java Agent DEveloppent framework est un framework logiciel libre distribué par TILab (Telecom Italia) en Open Source. Il est intégré en java en tant que librairie, comme le montre la figure ci-dessous. **[3]**

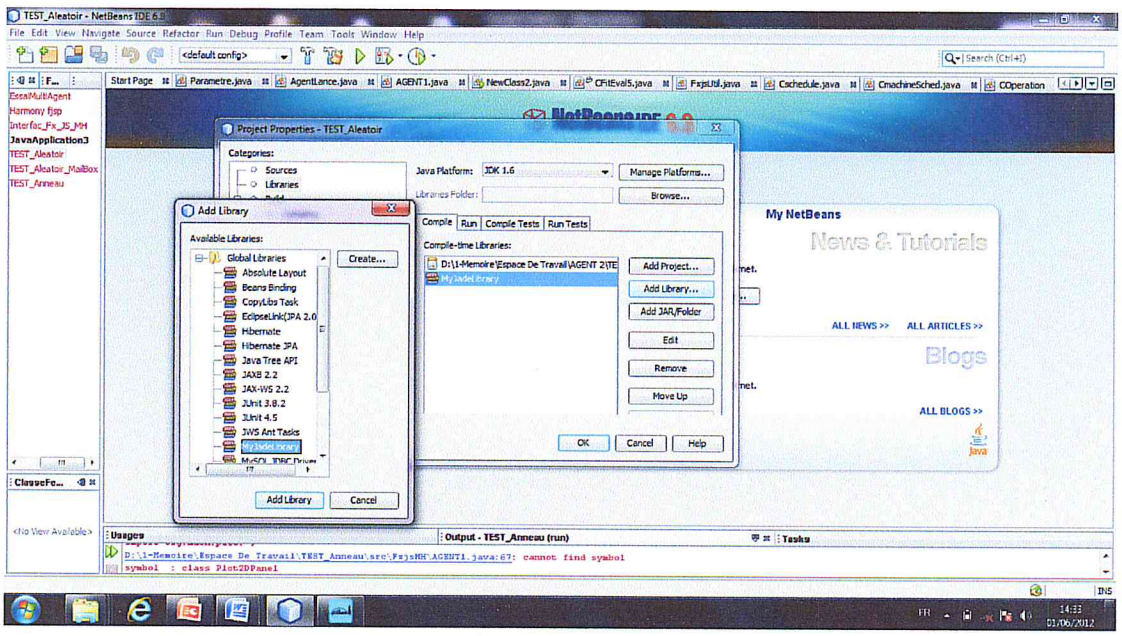


Figure 2 : Intégration de JADE sous NetBeans.

II.3. Benchmark : [18], [21]

En ordonnancement d'ateliers il est souvent très utile d'avoir des Benchmarks pour tester les différents modèles d'ordonnancement et surtout les méthodes de résolution appropriées.

Un Benchmark est un problème-type représentant une instance d'un problème particulier tel que le Job Shop Flexible et servant à son étude. Il est construit par plusieurs auteurs pour expérimenter les performances des approches de résolution, en procédant surtout à des comparaisons sur les mêmes instances.

De grands nombres de benchmarks destinés aux problèmes de type Job Shop Flexible sont proposés par plusieurs auteurs. Certains sont largement employés dans la littérature.

Parmi ces benchmarks, nous avons utilisé les deux classes suivantes :

BRdata : Cette classe est constituée de 10 problèmes décrits par *Brandimarte* (1993). Le nombre de jobs n varie entre 10 et 20, le nombre de machines M varie entre 4 et 15, le nombre d'opérations pour chaque job varie entre 5 et 15.

La moyenne du nombre de machines par opération (nommée aussi flexibilité et notée *flex.*), varie entre 1.43 et 4.10. Les machines considérées dans cette classe sont non-reliées. Cette série de problèmes peut être téléchargée sur le site : http://www.idsia.ch/~monaldo/fjsp.html

Kacem : Cette classe est constituée de 5 problèmes décrits par *Kacem* (2002). Le nombre de jobs n varie entre 4 et 15, le nombre de machines m varie entre 5 et 10 et le nombre d'opérations pour chaque job varie entre 2 et 4.

Un exemple d'une instance est présenté sur la figure 3.

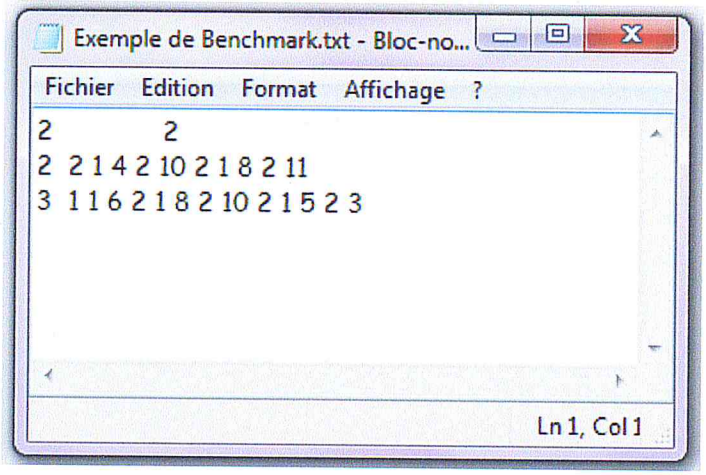


Figure 3 : Exemple de benchmark.

L'interprétation de cet exemple peut être procédée comme suit :

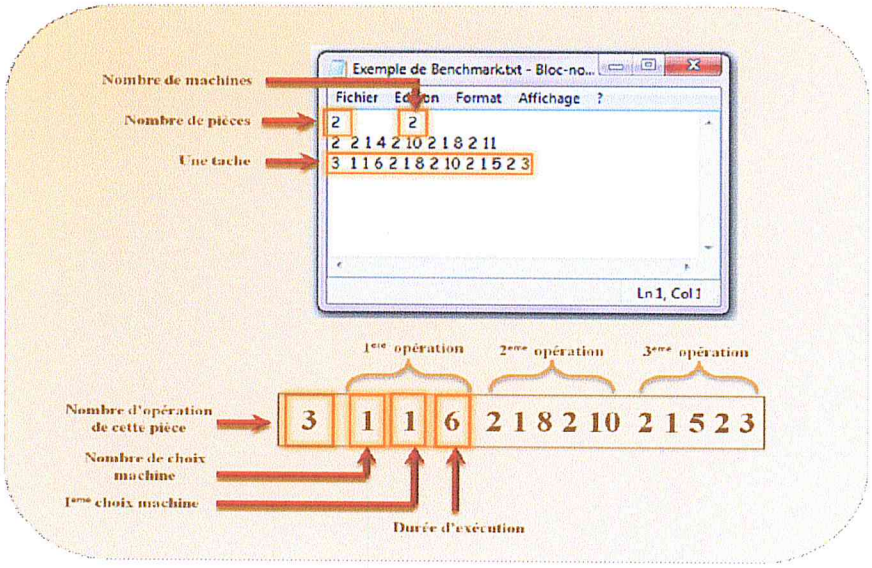


Figure 4 : Interprétation d'un exemple de Benchmark.

III. Description générale du système :

Pour la mise en œuvre de notre système, nous avons eu recours à l'architecture matérielle et logicielle présentée ci-après :

III.1. Architecture matérielle :

La construction de toute application multi agents est basée sur l'utilisation d'un réseau d'ordinateur. Pour cela nous avons conçu un réseau local contenant quatre ordinateurs reliés entre eux par un Switch, chacun d'eux représente un agent Ordonnanceur (Figure 5).

L'agent lanceur est implémenté sur un parmi les quatre ordinateurs du réseau.

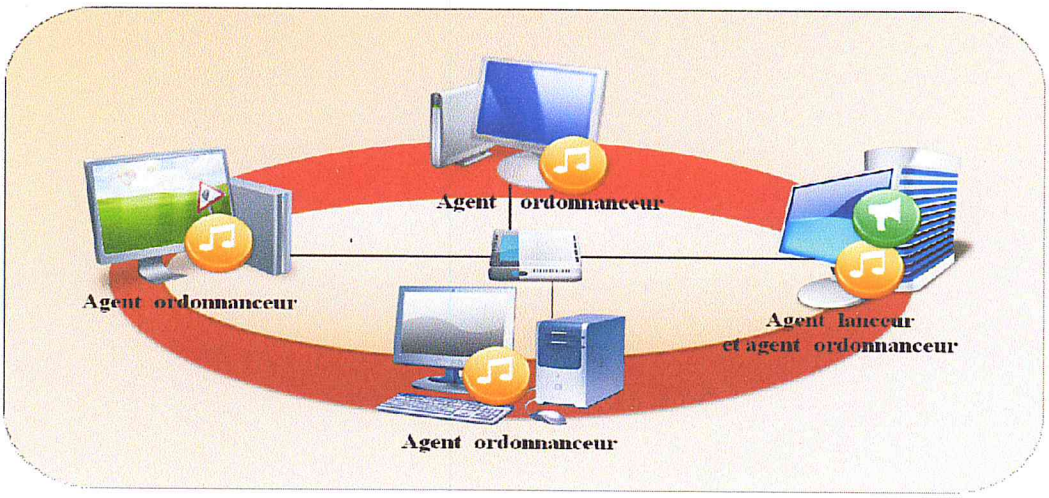


Figure 5 : Représentation de l'architecture matérielle suivie dans notre application.

III.2. Architecture logicielle :

Cette architecture est basée sur l'architecture logicielle de JADE qui est distribuée et hiérarchique au travers d'une plateforme regroupant différents conteneurs.

Dans notre application, nous nous sommes basés sur cette architecture pour implémenter notre système. Notre plateforme multi agents (Figure 6) se compose d'un seul main container et de trois conteneurs secondaires. Le main container est celui qui représente l'agent lanceur. Il contient aussi l'agent Ordonnanceur créé par lui. Les conteneurs secondaires correspondent aux autres agents Ordonnanceurs. Ils rejoignent le conteneur principal après le lancement de leurs agents Ordonnanceurs. La figure suivante représente l'architecture logicielle suivie dans notre implémentation.

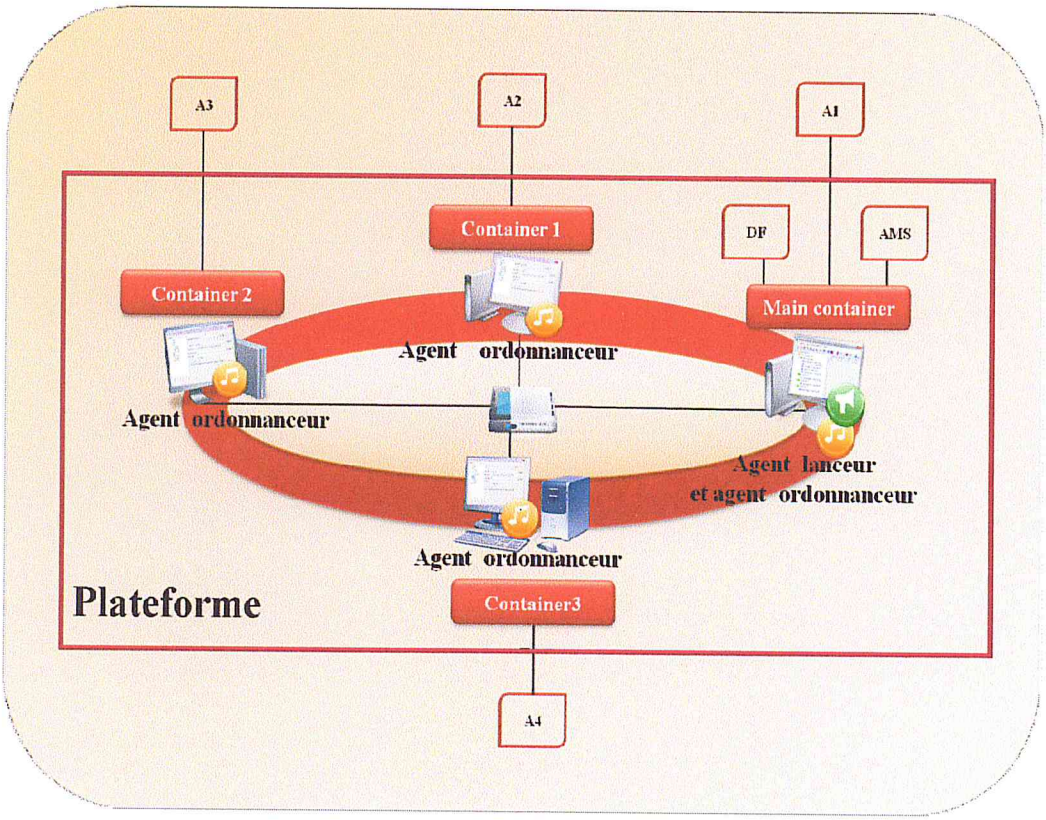


Figure 6 : Représentation de l'architecture logicielle suivie dans notre application.

IV. Application :

IV.1. Fenêtres d'application :

Les fenêtres principales de notre application sont présentées dans les figures suivantes :

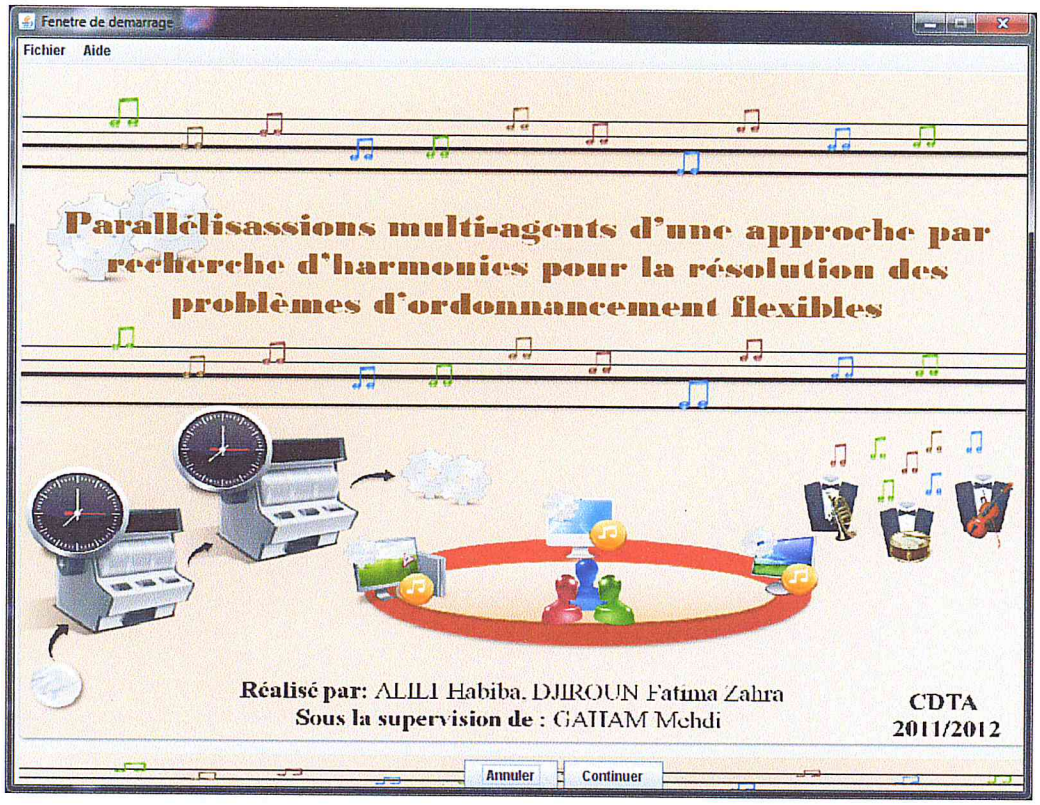


Figure 7 : Fenêtre principale de notre application.

L'interface principale de notre application (voir la Figure 7) contient :

- Un Menu Bar de deux Menus qui sont: **Fichier** et **Aide**.
  - ✓ Le Menu « **Fichier** » contient deux Menus Items, le premier « **Ouvrir** » permet d'accéder à l'interface de saisie des paramètres, et le deuxième « **Exit** » permet de quitter l'application.
  - ✓ Le Menu « **Aide** » contient une description sur l'application réalisée.
- Un bouton principal « **Continuer**» qui nous permet de passer à la 2^{éme} interface (la Figure 8)
- Un autre bouton « **Annuler** » qui permet de quitter l'application.

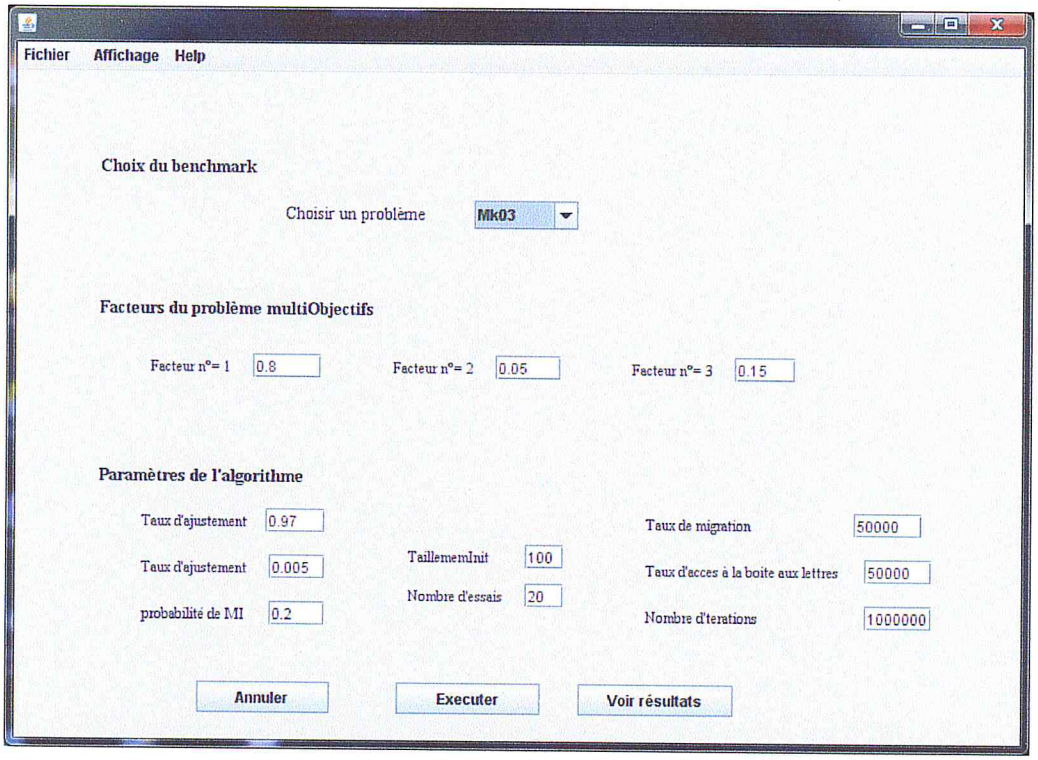


Figure 8 : Fenêtre de saisie des paramètres de l'algorithme « Harmony Search ».

L' interface ci-dessus permet la saisie des paramètres de l'application realisée, elle contient plusieurs zones de text subdivisées en trois parties:

- La première partie contient un combo Box permettant de choisir le benchmark à traiter.
- La deuxième partie contient trois zones de texte permettant de saisir les facteurs de pondération pour l'optimisation multiobjectifs.
- La troisieme partie contient cinq zones de texte ou l'utilisateur pourra introduire les parametres de l'algorithme « Harmony Search ».
- Le bouton « **Executer** » permet de commencer l'exécution de l'algorithme.
- Le bouton « **Voir résultats** » permet la visualisation des resultats numeriques (voir la figure 9).
- Le bouton « **Annuler** » permet d'arrêter l'exécution et de quitter l'application.

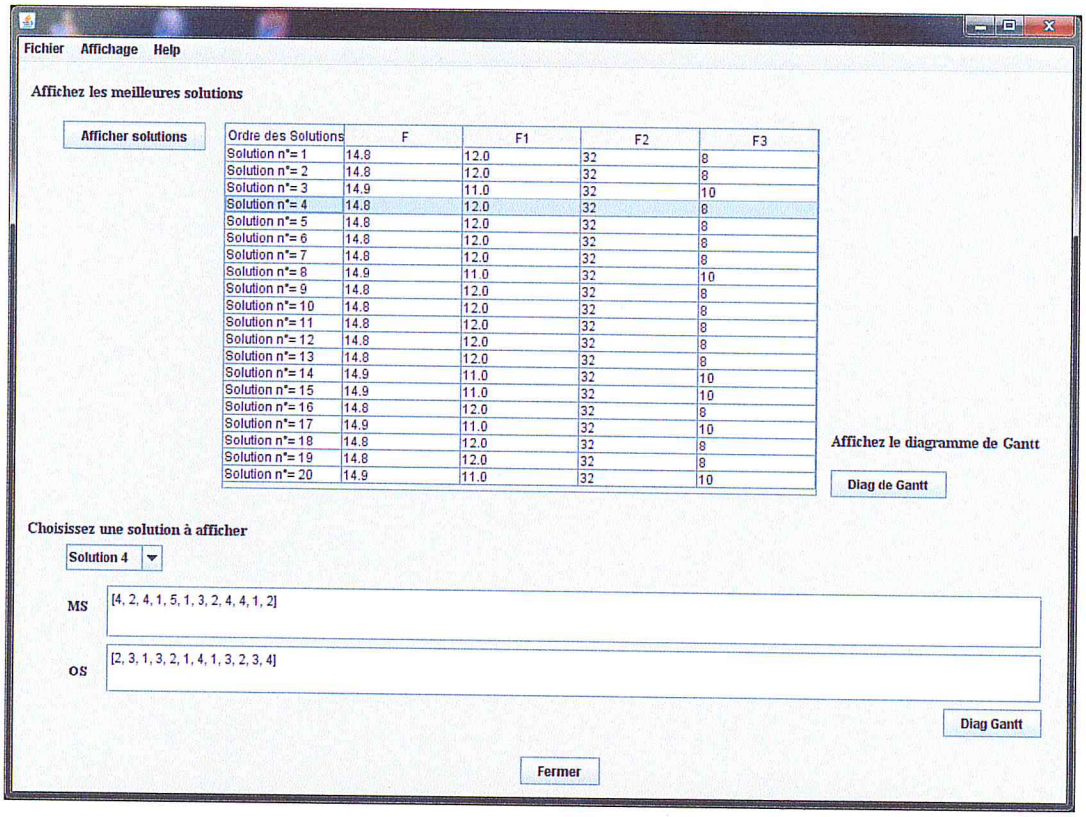


Figure 9 : Fenêtre d'affichage des résultats.

L'interface présentée dans la figure 9 permet d'afficher les résultats de l'exécution de l'algorithme.

- Le bouton « **Afficher solutions** » en haut de l'interface permet d'afficher dans la table à coté, pour chaque essai, la meilleure fitness ainsi que la valeur de chaque critère trouvées entre les différents agents Ordonnanceurs actifs.
- Le bouton « **Diag de Gantt** » en bas à droite de la table, permet d'afficher le diagramme de Gantt de la meilleure solution trouvée pour tous les essais.
- Le combo Box en dessous de la table permet de choisir la solution à afficher dans les zones de texte.
- Les zones de texte « **MS** » et « **OS** » permettent d'afficher respectivement la partie MS et la partie OS de la solution.
- Le bouton « **Diag Gantt** » permet d'afficher le diagramme de Gantt de la solution affichée dans les zones de texte.
- Le bouton « **Fermer** » en bas de l'interface permet de quitter l'application.

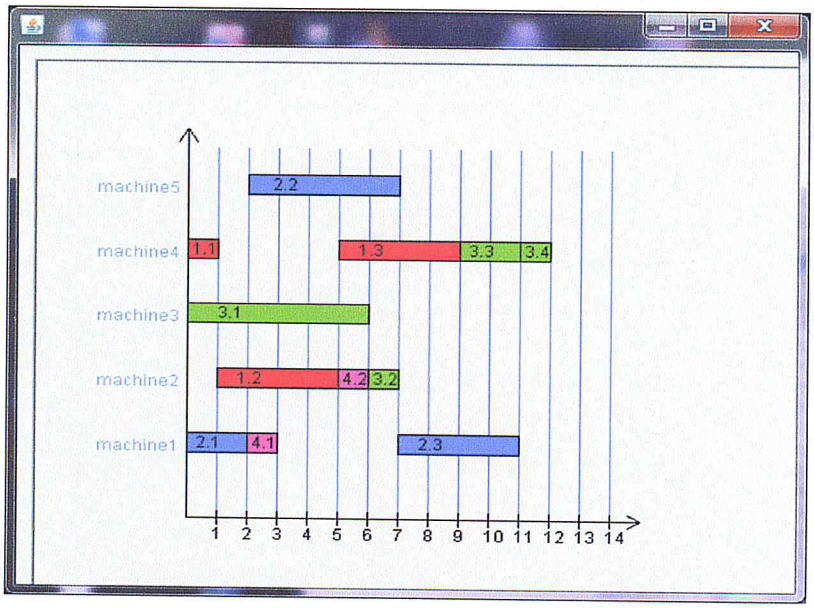


Figure 10 : Fenêtre d'affichage du diagramme de Gantt.

Cette interface (Figure 10) permet d'afficher le diagramme de Gantt.

IV.2. Configuration et lancement :

Afin de lancer notre application, il faut lancer tout d'abord l'agent lanceur et lui affecter l'argument : « *-gui lanceur :FxjsMH.AgentLance* » comme le montre la figure ci-dessous (Figure 11):

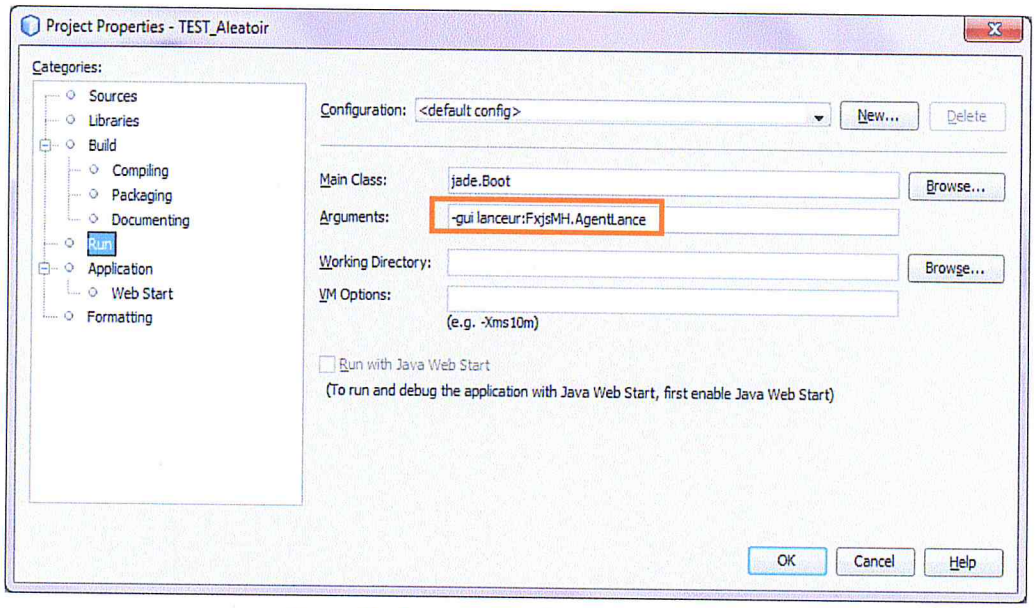


Figure 11 : Lancement de l'agent lanceur.

Tel que :*-gui :* représente l'interface RMA ;

- *Lanceur :* nom attribué à l'agent lanceur ;
- *FxjsMH.AgentLance* : nom du package.nom de la classe.

Juste après le lancement de l'agent lanceur, il faut lancer les agents Ordonnanceurs sur les autres hôtes en initialisant chacun d'eux (figure 12) :

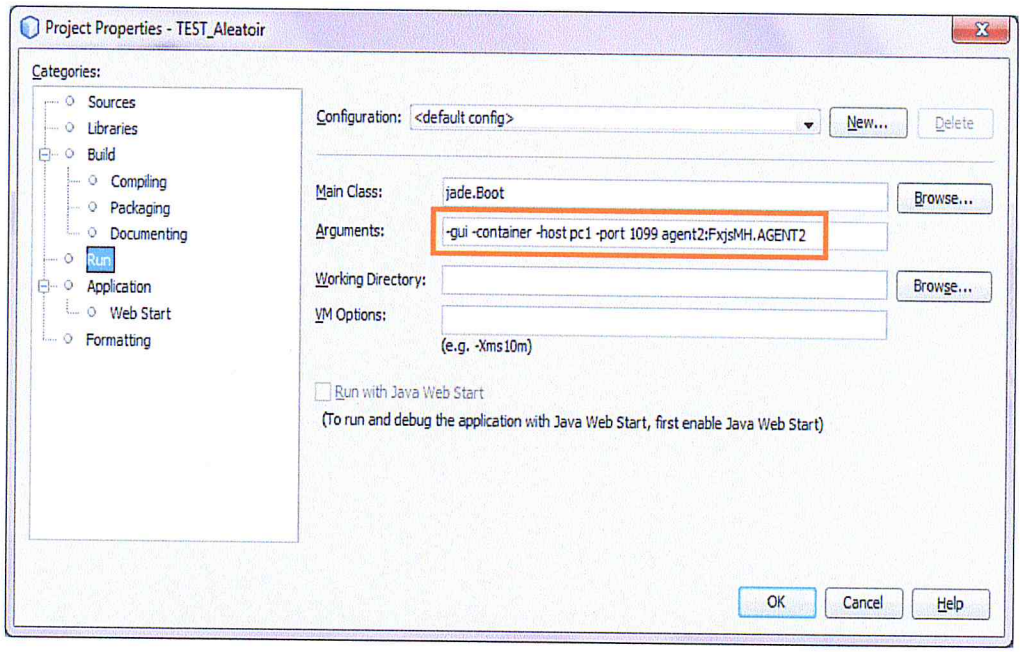


Figure 12 : Lancement de l'agent Ordonnanceur.

Telle que :

- *gui* : représente l'interface RMA ;
- *container* : signifie que c'est un conteneur secondaire ;
- *host pc1* : le nom de l'ordinateur disposant du main container;
- *port 1099* : le numéro de port attribué à JADE ;
- *agent4* : nom attribué à l'agent ;
- *FxjsMH.AGENT4* : nom du package.nom de la classe.

La fenêtre suivante (Figure 13) montre l'interface RMA-GUI, où apparait la plateforme avec les différents conteneurs et agents associés.

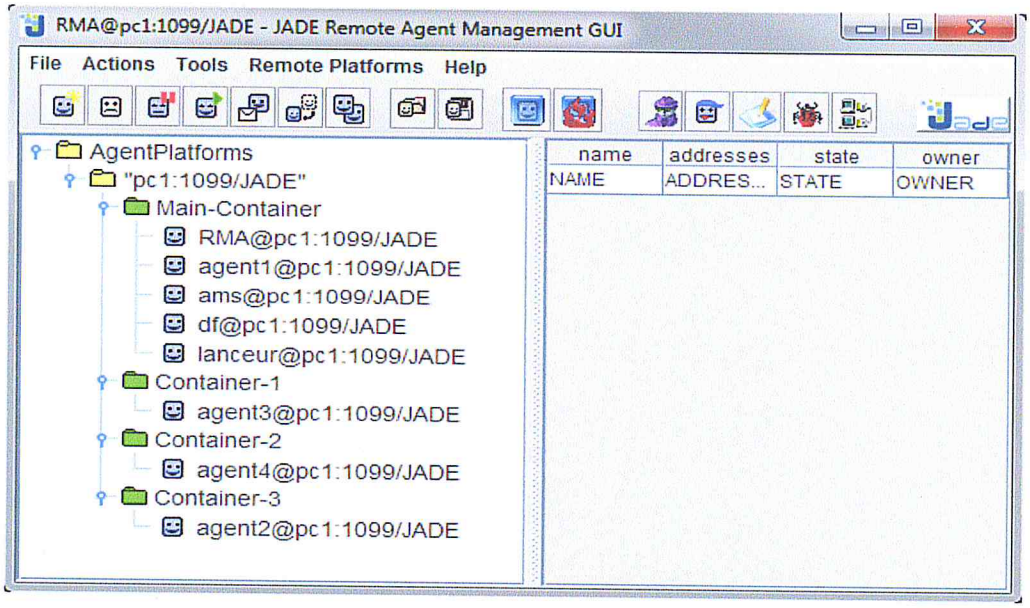


Figure 13 : L'interface GUI.

V. Expérimentations numériques et résultats:

Dans cette partie, nous allons donner une synthèse de tous les résultats relatifs à l'implémentation de notre approche.

Les expérimentations, que nous avons effectué, ont étés réalisées sur deux catégories d'ordinateurs ; des ordinateurs équipés d'un Pentium IV micro processeur Intel ® Core (TM)i3-2100 CPU @ 3.10GHz, avec 6 Go de RAM, fonctionnant sous Windows7 version 2009. Et un ordinateur équipé d'un micro processeur Intel Core (TM) 2DUO CPU T6400 @2.00 GHz, avec 3Go de RAM, fonctionnant sous Windows7 version 2009.

Dans un premier temps, un test de validation de l'algorithme « Harmony Search » séquentiel est réalisé, suivi par des tests comparatifs permettant de déterminer la meilleure combinaison d'opérateurs.

Nous passons à l'implémentation parallèle où nous avons fait un premier test permettant de déterminer le nombre d'agents (ordinateurs) nécessaires. Le deuxième test est destiné à la détermination de taux de migration qui pourra être utilisé par les agents du système.

Apres avoir déterminé le nombre d'agents et le taux de migration nécessaires pour un bon fonctionnement et une bonne efficacité des résultats, le troisième test est consacré à l'évaluation finale de notre approche en utilisant les valeurs de paramètres déduites dans les tests précédents sur les différents Benchmarks choisis au départ.

V.1. Test de validation de la méthode Harmony Search :

Pour pouvoir évaluer les performances de nos modifications (évaluation et improvisation) sur la méta-heuristique « Harmony Search » de base, nous avons réalisé un premier test de validation sur une série de benchmarks de la classe BRdata (de MK01à MK07 sauf MK03) dans le cas multiobjectifs, en comparant ses résultats avec ceux de l'implémentation de la dernière année pour la même méta-heuristique. Nous avons utilisé les paramètres suivants :

Nombre d'itérations = 1.000.000 ;

Taux d'acceptation = 0.97 ;

Taux d'ajustement = 0.005 ;

Taille de la mémoire initiale = 100 ;

Le test de validation est fait vingt fois pour chacun des benchmarks, et a donné les résultats montrés dans le tableau présenté ci-après :

PROBs	HSA (ancienne version)	HSA (notre version)
Mk01	47.35	**45.75**
Mk02	33.1	**32.25**
Mk04	81.1	**76.85**
Mk05	198.6	**197.75**
Mk06	85.5	**78.55**
Mk07	169.45	**167.25**

Tableau 1 : Comparaison entre les résultats de notre version et ceux de la version 2010 /2011.

Discussion :

La méthode arrive sur tous les benchmarks testés à dépasser les résultats de l'ancienne version en atteignant de meilleures valeurs de la fonction objectif. Ce qui signifie que les modifications que nous avons apporté sur l'algorithme de base sont très efficaces.

V.2. Test de détermination de la meilleure combinaison d'operateurs :

Dans le but d'augmenter les performances de notre algorithme, nous avons pensé à lui greffer d'autres opérateurs. Ces opérateurs sont en partie issus des approches évolutionnaires et sont: l'opérateur de sélection, l'opérateur de mutation intelligente et l'opérateur de remplacement modifié.

Dans cette partie de tests (vingt fois pour chacun des benchmarks, dans le cas multiobjectifs), nous avons essayé d'explorer toutes les combinaisons d'opérateurs possibles (tableau 2) sur un échantillon de benchmarks de la même classe que la précédente (MK01, MK02, MK04, MK05, MK06 et MK07).

Version	Combinaison d'operateurs
V1	Algorithme de base que nous avons implémenté
V2	Algo de base + operateur de Remplacement modifié
V3	Algo de base + operateur de Sélection
V4	Algo de base + operateur de Mutation Intelligente (prob = 0.1)
V5	Algo de base + operateur de Mutation Intelligente (prob = 0.2)
V6	Algo de base + Op de Remplacement mod + Op de Sélection
V7	Algo de base + Op de Remplacement mod + Op de MI (prob = 0.1)
V8	Algo de base + Op de Remplacement mod + Op de MI (prob = 0.2)
V9	Algo de base + Op de Sélection + Op de MI (prob = 0.1)
V10	Algo de base + Op de Sélection + Op de MI (prob = 0.2)
V11	Algo B + Op de R mod+ Op de S + Op de MI (prob = 0.1)
V12	Algo B + Op de R mod + Op de S + Op de MI (prob = 0.2)

Tableau 2 : Explication des operateurs de chaque version.

Les tableaux suivants récapitulent tous les résultats de cette série de tests en termes de meilleure fitness (BF) et fitness moyenne (FM) :

MK 01	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12
BF	45.75	45,75	45,75	45,75	45,75	45,75	45,75	45,75	45,75	45,75	45,75	45,75
FM	47,59	47,37	47,62	45,93	45,92	47,43	46,32	45,94	46,21	46,13	46,25	46,11

Tableau 3 : Les valeurs de best fit et fitness moyenne de toutes les versions pour MK01.

MK 02	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12
BF	32.25	32,25	33,05	32,3	32,25	32,3	32,25	32,2	32,25	32,3	33,0	32,25
FM	33,20	33,13	33,41	33,07	32,87	33,22	32,86	33,04	33,26	33,10	33,31	33,28

Tableau 4 : Les valeurs de best fit et fitness moyenne de toutes les versions pour MK02.

MK 04	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12
BF	76.85	76,25	76,3	75,9	76,25	75,6	76,25	75,6	76,15	76,55	76,1	75,75
FM	78,76	78,61	78,66	78,17	78,05	79,18	77,87	77,55	78,59	78,63	78,81	78,00

Tableau 5 : Les valeurs de best fit et fitness moyenne de toutes les versions pour MK04.

MK 05	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12
BF	197.75	198,5	198,5	197,75	197,75	197.75	197,75	197,75	198,5	197,75	197,75	197,75
FM	198,55	198,56	198,59	198,5	198,5	198,55	198,41	198,25	198,58	198,54	198,17	198,37

Tableau 6 : Les valeurs de best fit et fitness moyenne de toutes les versions pour MK05.

MK 06	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12
BF	78.55	78,5	78,8	78,75	78,85	78,8	78,2	79,25	79,3	79,4	79,45	79,1
FM	80,41	80,33	80,58	80,46	80,32	80,58	79,97	80,40	81,18	80,75	80,61	80,81

Tableau 7 : Les valeurs de best fit et fitness moyenne de toutes les versions pour MK06.

MK 07	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12
BF	167.25	167,85	167,4	167,3	167,3	167,3	166,7	167,3	167,3	167,3	166,7	167,3
FM	168,73	168,77	169,27	168,73	168,60	168,77	168,42	168,42	168,77	168,42	168,76	168,44

Tableau 8 : Les valeurs de best fit et fitness moyenne de toutes les versions pour MK07.

Afin de déterminer la meilleure version de l'algorithme, nous avons calculé pour chacune des versions, la moyenne des meilleures fitness de tous les benchmarks en la comparant avec la moyenne des fitness moyennes. Le diagramme présenté ci-après, résume les résultats comparatifs de toutes les versions.

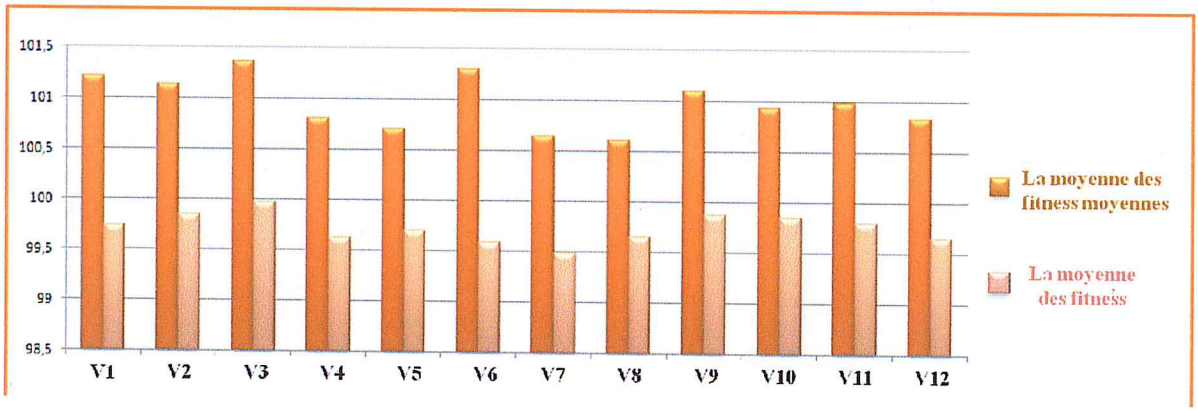


Figure 14 : Graphique représentant la différence entre la moyenne des meilleures fitness et la moyenne des fitness moyennes pour toutes les versions.

Discussion :

D'après les tableaux et le graphique, nous pouvons remarquer que les versions V7 et V8 sont les versions les plus intéressantes, car elles ont donné les meilleures résultats. Ce qui signifie, d'un coté, que l'intégration de l'opérateur de remplacement modifié et de l'opérateur de mutation intelligente avec les probabilités (0.1) et (0.2) permet d'améliorer l'efficacité et la performance de l'algorithme « Harmony Search ». On peut aussi, d'un autre coté, conclure que l'intégration de l'opérateur de sélection n'a pas permis d'améliorer les performances de l'algorithme.

Les résultats de cette série de tests nous ont permis de décider de choisir les versions V7 et V8 pour continuer le processus des tests restants.

V.3. Tests de validation de l'implémentation parallèle de l'Harmony Search :

La deuxième partie de tests a été faite dans le cadre de la validation de l'approche parallèle en déterminant le nombre d'ordinateurs et le taux de migration permettant d'apporter de bons résultats. Pour cela, une première série de tests a été explorée où nous avons choisi d'expérimenter le problème MK04, sur un (1), deux (2), trois (3), et quatre (4) postes, vingt (20) fois pour chacune des configurations. Les paramètres de l'algorithme « Harmony Search » sont les suivants:

- Taille de la mémoire initiale : 100
- Nombre d'itérations : 500000.
- Taux d'acceptation : 0.95
- Taux d'ajustement : 0.005
- Fréquence de migration : 10000

➢ Taux de la mutation intelligente : 0.2

Le tableau suivant donne la moyenne des fitness et la moyenne des temps d'exécution globaux pour chacune des configurations :

Nombre d'agents	Moyenne des fitness	Moyenne des temps globaux
1	78 ,49	101
2	77,33	223,3
3	77,12	223,4
4	76,95	265,45

Tableau 9: Résultats du test du nombre d'ordinateurs.

Pour bien clarifier les résultats de ce test, deux graphiques en courbes ont été réalisés. Le premier montre l'évolution de la moyenne des fitness en fonction du nombre d'ordinateur, et le deuxième représente la moyenne des temps d'exécution en fonction du nombre d'ordinateurs.

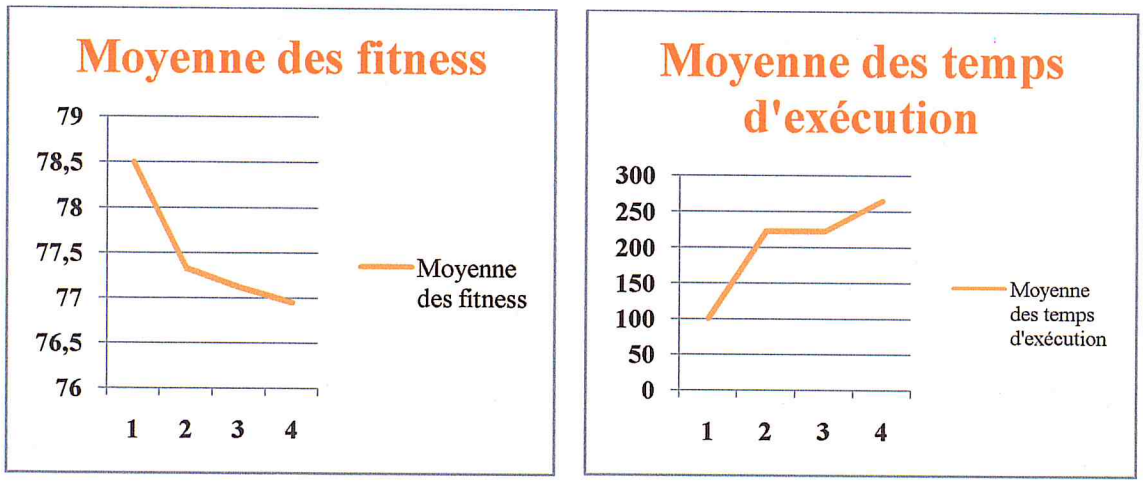


Figure 15 : Graphiques représentants la moyenne des fitness et la moyenne des temps d'exécution pour les quatre configurations.

Nous remarquons, d'après les graphiques, qu'en augmentant le nombre d'ordinateurs (d'agents), la moyenne des fitness diminue. Cependant la moyenne des temps d'exécution augmente, ce qui nous a poussé à explorer une autre série de tests servant à déterminer le taux de migration efficace.

Les tests selon une configuration de 4 ordinateurs sont refaits vingt (20) fois pour chaque taux de migration et sur le même benchmark qui est le MK04. Leurs résultats sont présentés dans le tableau suivant :

Ordre	Taux de migration	Moyenne des fitness	Moyenne des temps globaux
1	10000	76,95	265,45
2	20000	76,77	164,65
3	50000	76,67	165,95

Tableau 10 : Résultats des tests du taux de migration.

Les deux graphiques en courbes présentés ci-après interprètent les résultats contenus dans le tableau précédent, c.-à-d. la moyenne des fitness et la moyenne des temps globaux, en fonction du taux de migration et les synthétise d'une manière très claire permettant de les juger facilement.

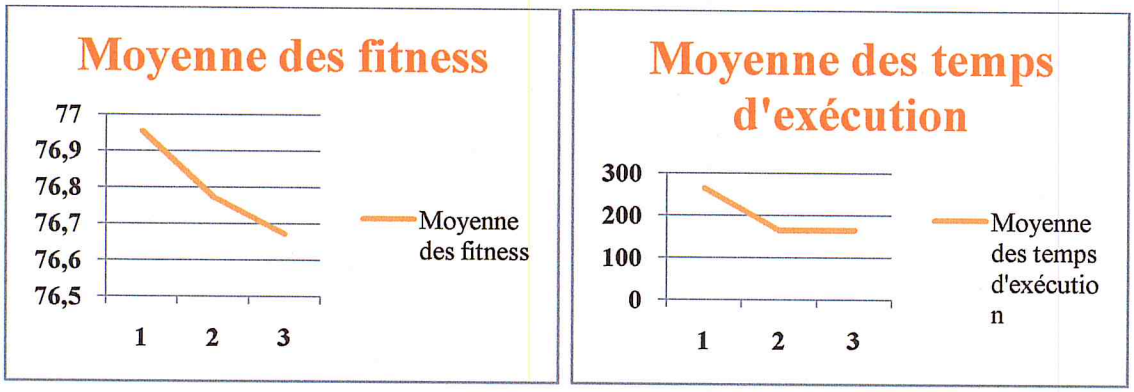


Figure 16 : Graphiques représentants la moyenne des fitness et la moyenne des temps d'exécution pour le test du taux de migration.

Discussion :

D'après les résultats trouvés dans cette phase de test, nous remarquons nettement qu'en diminuant la fréquence de migration, la moyenne des fitness s'améliore ainsi que la moyenne des temps d'exécution, ce qui nous a permis de conclure qu'une meilleure configuration est celle contenant quatre (4) ordinateurs c.-à-d. quatre agents avec un taux de migration de (50000).

VI. Validation de l'implémentation parallèle:

Toute la série des expérimentations faites nous a permis de décider de choisir, afin de réaliser les tests finaux de validation de notre méthode, la version de l'algorithme qui implémente les deux opérateurs : opérateur de remplacement et opérateur de mutation intelligente avec les probabilités : (0.1) et (0.2), la configuration de quatre agents et le taux de migration qui est de 50000.

La validation finale de l'implémentation a été, en fait, réalisée en deux séries de tests, une première utilisant les problèmes de la classe BRdata, et la deuxième effectuée sur ceux de la classe Kacem.

VI.1. Classe BRdata :

La série d'essais présentée dans ce qui suit a été réalisée sur dix (10) problèmes de la classe BRdata (MK01, MK02, MK03, MK04, MK05, MK06, MK07, MK08, MK09, MK10), vingt (20) fois pour chaque problème avec les mêmes paramètres choisis précédemment.

VI.1.1. Comparaison avec les versions séquentielles de l'algorithme :

Nous résumons dans ce tableau les résultats de la série de tests effectuée, en les comparant avec ceux de l'implémentation séquentielle :

Prob	Version V7			Version V8			HS parallèle		
	B_Fit	MoyF	MTG	B_Fit	MoyF	MTG	B_Fit	MoyF	MTG
MK 01	45,75	46,32	93,65	45,75	45,94	98 ,4	**45.75**	45.75	98,05
MK 02	32,25	32,86	95,85	32,2	33,04	100,3	**32,2**	32,50	99,4
MK 04	76,25	77,87	178 ,7	75,6	77,55	187,5	**75,6**	76,67	165,95
MK 05	197,75	198,41	221.8	197,75	198,25	234,35	**197,75**	197,87	313,9
MK 06	78,2	79,97	287,2	79,25	80,40	307,15	**78,0**	79,08	313,93
MK 07	166,7	168,42	226,25	167,3	168,42	235,45	**166,7**	167,59	328,2
MK 09	404,5	405,92	640,9	404,55	405,72	689,05	**404,5**	404,77	699,4
MK 10	302,9	306,65	651,05	302,35	307,29	697,85	**301,85**	306,98	903,35

Tableau 11 : Résultats comparatifs entre PHS et V7 et V8 en termes de best fit, fit moyenne et moyenne du temps global.

Discussion :

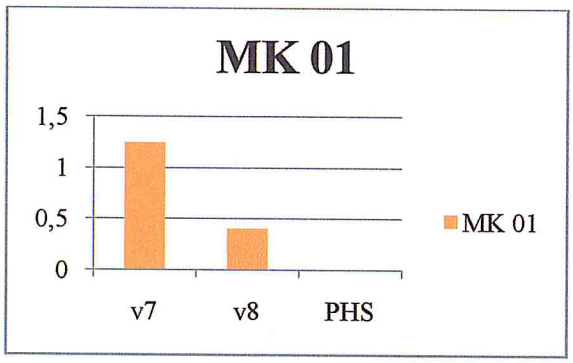
D'après ce tableau nous remarquons que, sur la globalité des benchmarks, la version parallèle arrive à atteindre les meilleures valeurs déjà obtenues, et particulièrement, elle a pu dépasser les versions séquentielles sur deux instances de problèmes (MK06, MK10), ce qui prouve son efficacité.

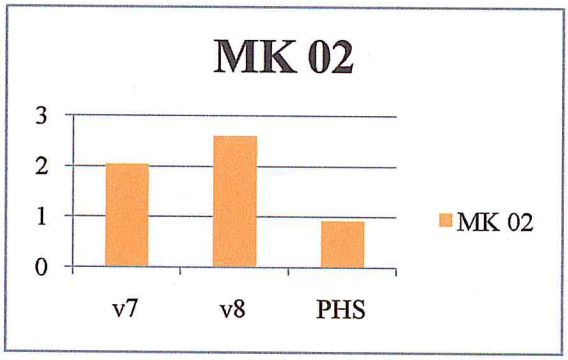
Pour montrer nettement la différence entre les deux versions séquentielles de l'algorithme avec sa version parallèle, nous avons calculé la déviation de la moyenne des fitness en valeur absolue de toutes les versions par rapport aux meilleures valeurs obtenues de la fitness. Les résultats sont montrés dans le tableau suivant :

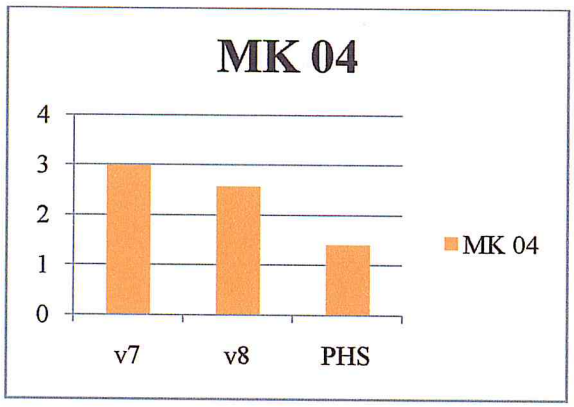
PROBs	V7	V8	PHS
MK 01	1,25	0,41	0
MK 02	2,04	2,60	0,93
MK 04	3	2,57	1,41
MK 05	0,33	0,25	0,06
MK 06	2,52	3,07	1,38
MK 07	1,03	1,03	0,53
MK 09	0,35	0,30	0,06
MK 10	1,59	1,80	1,69

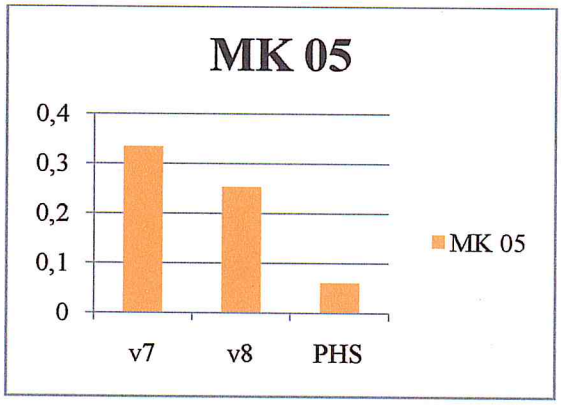
Tableau 12 : Résultats comparatifs entre PHS et V7 et V8 en termes de déviation.

Afin de bien éclaircir les résultats de ces calculs, un histogramme pour chaque version a été élaboré, comme le montre les figures ci-dessous :









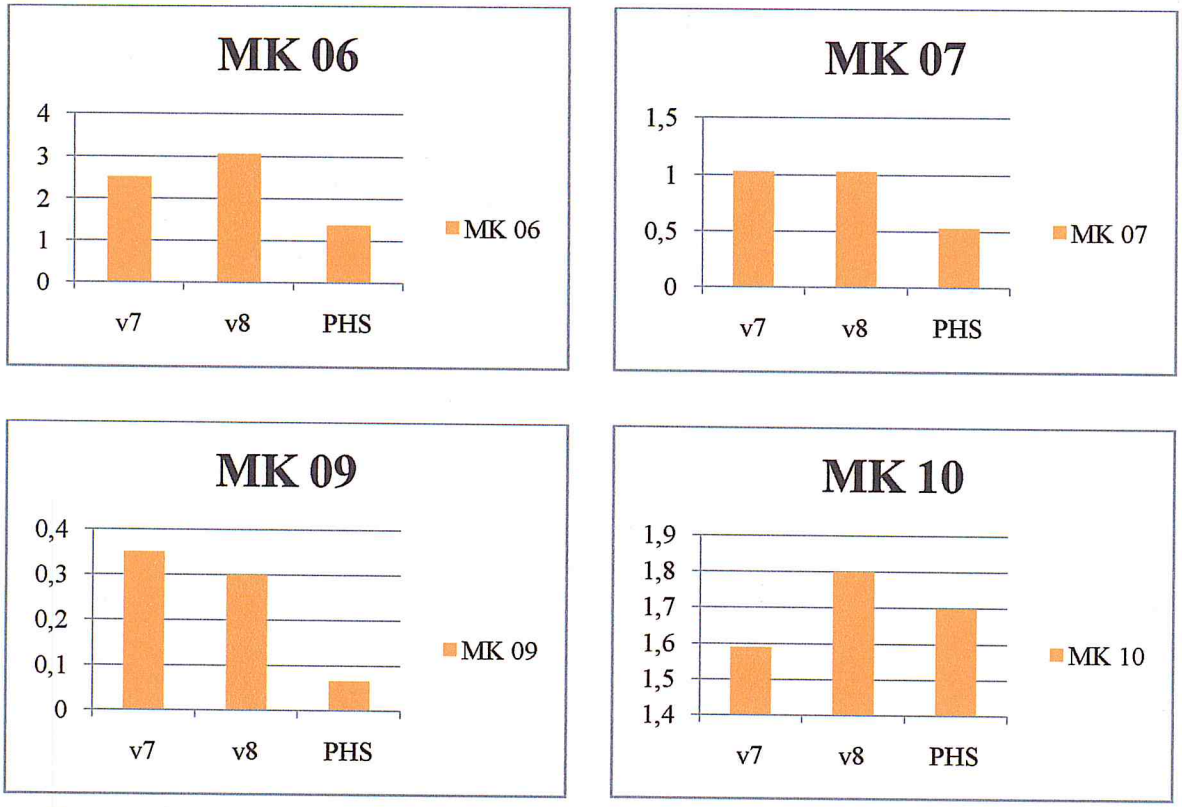


Figure 17 : Graphiques représentants la différence entre la déviation de PHS et celles de V7 et V8 pour tout les benchmarks.

Discussion :

D'après le tableau et les histogrammes présentés plus haut, nous remarquons clairement que pour la grande majorité des benchmarks, les résultats de l'approche parallèle présentent une meilleure stabilité (les valeurs de la fitness moyenne (MoyF)), par rapport à ceux de l'approche séquentielle. Une exception réside dans le benchmark MK10, où la déviation de la version V7 était meilleure par rapport à la version parallèle.

VI.1.2. Comparaison avec d'autres méthodes de la littérature :

Dans le but de valider notre approche parallèle et pour pouvoir montrer son efficacité et son apport, nous allons présenter, dans ce qui suit, une comparaison entre nos résultats et les meilleurs résultats de la littérature.

VI.1.2.1. Comparaison détaillée avec la méthode Xing :

		F	F 1	F 2	F 3	Temps
MK01	Xing	48	42	162	42	286.8
	HSA	45.75	40.0	167	36	< 1
MK02	Xing	34.35	28	155	28	181.2
	HSA	32,2	26.0	150	26	82
MK03	Xing	236.40	204	852	204	1568.4
	HSA	236.3	204.0	850	204	14
MK04	Xing	82.05	68	352	67	1064.4
	HSA	75.6	60.0	372	60	17
MK05	Xing	203.25	177	702	177	495.6
	HSA	197.75	172.0	687	172	22
MK06	Xing	91.60	75	431	67	1127.4
	HSA	78,0	62.0	406	54	203
MK07	Xing	178.35	150	717	150	340.8
	HSA	166.7	139.0	693	139	16
MK08	Xing	623.05	523	2524	523	4060.2
	HSA	623.05	523.0	2524	523	5
MK09	Xing	412.35	311	2374	299	4665.6
	HSA	404.50	307.0	2281	299	459
MK10	Xing	314.20	227	1989	221	7351.2
	HSA	301,85	212.0	2030	205	1597

Tableau 13 : Résultats détaillés comparant notre algorithme avec celui de Xing.

Discussion :

A partir de ce tableau, nous remarquons que les résultats de notre méthode dépassent, sur tous les benchmarks, les résultats de la méthode d'Xing qui utilise une méthode heuristique.

VI.1.2.2. Comparaison détaillée avec la méthode HTSA :

La méthode HTSA qui signifie « Hybrid Tabu Search Algorithm », est une méthode très sophistiquée qui se base sur une recherche tabou globale hybridée avec une recherche locale à voisinage variable. Le système utilise beaucoup de

techniques compliquées pour améliorer l'algorithme de base, par exemple afin de choisir la solution de départ, il met tout un mécanisme pour générer une population initiale, puis l'évalue pour déterminer la solution initiale.

		F	F 1	F 2	F 3	Temps
MK01	HTSA	45.75	40	167	36	2.07
	HSA	**45.75**	**40.0**	**167**	**36**	**<1**
MK02	HTSA	32 .25	26	151	26	16.35
	HSA	**32,2**	**26.0**	**150**	**26**	**82**
MK03	HTSA	236.4	204	852	204	8.32
	HSA	**236.3**	**204.0**	**850**	**204**	**14**
MK04	HTSA	76.25	61	366	61	42 .96
	HSA	**75.6**	**60.0**	**372**	**60**	**17**
MK05	HTSA	197.75	172	687	172	15.34
	HSA	**197.75**	**172.0**	**687**	**172**	**22**
MK06	HTSA	81.2	65	398	62	63.67
	HSA	**78,0**	**62.0**	**406**	**54**	**203**
MK07	HTSA	167.75	140	695	140	36.18
	HSA	**166.7**	**139.0**	**693**	**139**	**16**
MK08	HTSA	623.05	523	2524	523	4.09
	HSA	**623.05**	**523.0**	**2524**	**523**	**5**
MK09	HTSA	407.85	310	2294	301	166.21
	HSA	**404.50**	**307.0**	**2281**	**299**	**459**
MK10	HTSA	305.35	214	2053	210	120.87
	HSA	**301,85**	**212.0**	**2030**	**205**	**1597**

Tableau 14 : Résultats détaillés comparant notre algorithme avec celui de HTSA.

Discussion :

D'après le tableau présenté ci-avant, nous remarquons que notre méthode arrive à dépasser l'HTSA sur sept (7) problèmes, et donne les mêmes résultats en terme de meilleure fitness sur les trois (3) autres.

VI.1.2.3. Comparaison détaillée avec la méthode ABC :

La méthode Artificial Bee Colony (ABC) est une méthode à population qui est aussi très sophistiquée. Elle se base sur des operateurs de croisement et de mutation en utilisant plusieurs stratégies d'hybridations comme pour la génération de la population initiale. Elle utilise aussi une recherche locale.

		F	F 1	F 2	F 3	Temps
MK01	ABC	45.75	40	167	36	3.92
	HSA	45.75	40.0	167	36	<1
MK02	ABC	32.25	26	151	26	10.37
	HSA	32,2	26.0	150	26	82
MK03	ABC	236.30	204	850	204	37.93
	HSA	236.3	204.0	850	204	14
MK04	ABC	76.10	60	382	60	34.34
	HSA	75.6	60.0	372	60	17
MK05	ABC	197.75	172	687	172	18.65
	HSA	197.75	172.0	687	172	22
MK06	ABC	79.85	61	444	59	57.42
	HSA	78,0	62.0	406	54	203
MK07	ABC	166.70	139	693	139	36.91
	HSA	166.7	139.0	693	139	16
MK08	ABC	623.05	523	2524	523	12.94
	HSA	623.05	523.0	2524	523	5
MK09	ABC	407.10	309	2301	299	72.18
	HSA	404.50	307.0	2281	299	459
MK10	ABC	301.35	209	2065	206	110.08
	HSA	301,85	212.0	2030	205	1597

Tableau 15 : Résultats détaillés comparant notre algorithme avec celui de ABC.

Discussion :

Le tableau présenté ci-dessus montre que notre méthode donne, sur cinq (5) problèmes, les mêmes résultats que l'ABC et arrive à le dépasser sur quatre (4) autres problèmes ; sauf le MK10 où notre méthode n'a pas pu améliorer la valeur.

Dans ce qui suit, nous présentons un tableau récapitulatif comparant nos résultats à ceux des trois méthodes précédentes :

PROBs	n*m	Xing et al	HTSA	ABC	HSA (notre version)
Mk01	10*6	48	45.75	45.75	45.75
Mk02	10*6	34.35	32.25	32.25	32,2
Mk03	15*8	236.4	236.4	236.3	236,3
Mk04	15*8	82.05	76.25	76.1	75,6
Mk05	15*4	203.25	197.75	197.75	197,75
Mk06	10*15	91.6	81.2	79.85	78,0
Mk07	20*5	178.35	167.75	166.7	166,7
Mk08	20*10	623.05	623.05	623,05	623,05
Mk09	20*10	412.35	407.85	407.10	404,5
Mk10	20*15	314.2	305.35	301.35	**301,85**

Tableau 16 : Résultats comparant notre algorithme avec celui de Xing, HTSA et ABC sur les MK.

Discussion :

Le tableau comparatif montre nettement que les valeurs d'Xing sont assez loin par rapport à celle de l'HTSA et de l'ABC ; et en même temps les valeurs résultantes de l'algorithme ABC sont meilleures en les comparant à ceux d'Xing et HTSA. Cependant les résultats que nous avons obtenus par notre méthode sont les meilleurs par rapport à toutes les autres méthodes.

VI.2. Classe Kacem :

La série d'essais présentée dans ce qui suit a été réalisée sur cinq (05) problèmes de la classe Kacem (4*5, 8*8, 10*7, 10*10, 15*10), dix (10) fois pour chaque problème avec les mêmes paramètres choisis précédemment.

Les résultats sont résumés dans le tableau présenté ci-dessous (Tableau 18).

PROBs	n*m	Xing et al	HTS	ABC	PHSA (notre version)
kacem04	4*5	14,8	14,8	14,8	14,8
kacem08	8*8	26	26	26	26
Kacem10	10*7	20,9	20,9	20,9	20.9
Kacem10	10*10	13,7	13,6	13,6	13,6
Kacem15	15*10	27	27	27	27

Tableau 17: Résultats comparant notre algorithme avec celui de Xing, HTSA et ABC sur les benchmarks de la classe Kacem.

Discussion :

A partir de ce tableau nous remarquons que toutes les méthodes, y compris notre version, arrivent à avoir les meilleurs résultats de la littérature.

VII. Conclusion :

Dans ce chapitre, nous avons donné une vue globale sur le système que nous avons implémenté, puis nous sommes passés à la présentation de la synthèse des différents résultats obtenus. L'objectif est, d'un coté, d'explorer les performances des stratégies et techniques que nous avons adoptées, et de l'autre coté, de valider notre implémentation de la méta-heuristique « Harmony Search ». Les résultats restent dans leur globalité très bons, comparativement à ceux trouvés dans la littérature.

CONCLUSION GENERALE

Notre travail est une continuité des travaux déjà entamés l'année dernière au sein du laboratoire Systèmes Robotisés de Production du Centre de Développement des Technologies Avancées (CDTA). Ce travail propose une approche Multi-Agents pour l'implémentation de l'algorithme « Harmony Search » afin de résoudre les problèmes d'ordonnancement d'atelier de type Job Shop Flexible multicritères.

Un problème d'ordonnancement de type Job Shop Flexible est l'un des problèmes fréquemment rencontrés dans la gestion des systèmes de production. Il est considéré comme un problème d'ordonnancement extrêmement complexe, et il est classé parmi les problèmes combinatoires difficiles au sens fort. Cette complexité est due à l'explosion combinatoire du nombre de solution qui croît exponentiellement avec la taille du problème.

En réalité, le Job Shop Flexible est une extension du modèle Job Shop classique dont le principe se base sur les opérations réalisées qui suivent un ordre bien déterminé, variant selon la tâche à exécuter. Dans les ateliers de type Job Shop Flexible, l'énoncé du problème reste toujours le même, sauf qu'une particularité essentielle est rajoutée et qui réside dans le fait que plusieurs machines sont potentiellement capables de réaliser la même opération qui possède différentes durées de traitement dépendant de la ressource utilisée. En effet sa résolution présente une difficulté supplémentaire qui est la détermination d'une affectation des opérations aux machines, en plus de la détermination de leur ordre d'exécution.

Notre objectif de la résolution de ces problèmes est de minimiser la durée totale de l'ordonnancement, la charge critique de chaque machine dans l'atelier et la charge totale de toutes les machines de l'atelier.

Pour atteindre cet objectif, nous avons eu recours aux méthodes approchées, vu que les méthodes exactes rencontrent des difficultés énormes avec des problèmes de taille importante. En effet, en plus de leur adaptabilité aux différents problèmes combinatoires, les méta-heuristiques ont l'avantage de ne parcourir qu'une zone de l'espace de solution pour parvenir à une solution de bonne qualité. Ce qui réduit nettement les temps de calcul.

La méthode que nous avons adapté est une récente méta-heuristique qui a été implémentée l'année dernière, c'est la méta-heuristique « Harmony Search » que nous avons repris, en modifiant totalement sa version de base et en l'implémentant en deux approches, séquentielle et parallèle.

Dans un premier temps nous l'avons implémenté d'une manière séquentielle et, d'après les résultats obtenus, qui ont été meilleurs que ceux de l'année dernière, nous avons pu constater que c'est une méta-heuristique très efficace pour la résolution des problèmes d'ordonnancement de type Job Shop Flexible.

Afin d'améliorer les performances de la méthode nous lui avons greffé trois opérateurs (l'opérateur de remplacement, l'opérateur de sélection et l'opérateur de mutation intelligente) en explorant toutes les combinaisons possibles. Les résultats de cette série de tests ont montré que l'opérateur de sélection n'est pas tellement efficace, son injection dans l'algorithme n'a pas apporté d'amélioration de résultats, tandis que l'opérateur de remplacement et celui de mutation intelligente ont nettement amélioré les résultats de l'algorithme, avec indication que l'opérateur de mutation intelligente a donné de bons résultats pour deux valeurs de paramètres (deux probabilités de mutation intelligente).

Dans le but d'absorber la variabilité des paramètres, et pour pouvoir améliorer la qualité des résultats trouvés en termes de stabilité, nous nous sommes dirigés vers l'approche parallèle en utilisant la technologie multi-agents.

Apres une phase de détermination des paramètres de fonctionnement de l'approche parallèle, nous avons effectué une série de tests de validation de l'approche sur les quinze instances du problème job shop flexible les plus utilisées dans la littérature. Sur toutes les instances, les résultats étaient très bons, et sur quatre particulièrement, nous sommes arrivés à de meilleures valeurs que celles trouvées dans la littérature.

Pour conclure, les résultats sont très encourageants, et les solutions obtenues sont largement satisfaisantes, ce qui montre nettement l'efficacité de notre approche.

Par ailleurs, ces travaux ont permis d'ouvrir de nouvelles perspectives pour les études futures :

- Il peut être très intéressant d'adapter d'autres modèles de parallélisme tel que le modèle maître/esclaves qui permet d'améliorer très efficacement le temps d'exécution de l'algorithme.
- Explorer d'autres types de problèmes d'ordonnancement tel que le flow shop et le flow shop hybride.
- Etudier le problème dans le cas monobjectif (monocritères).

BIBLIOGRAPHIES

[1] Kirkpatrick S, Gelatt CD and Vecchi MP « Optimization by simulated annealing ». Science (1983).

[2] MEUNIER Herve «Algorithmes évolutionnaires parallèles pour l'optimisation multi-objectif de réseaux de télécommunications mobiles » Thèse de Doctorat de l' U.S.T.L de l'Université des Sciences et Technologies de Lille ,(2002).

[3] Olivier Fourdrinoy , « Creation d'une librairie de Balises pour l'insertion d'agents JADE dans des pages JSP »,Université d'Atois ,lens (2002)

[4] DUPAS Rémy « Amélioration de performance des systèmes de production : apport des algorithmes évolutionnistes aux problèmes d'ordonnancement cycliques et flexibles » l'Université d'Artois (2004).

[5] AIRZEM Mustapha, EMKIEDECHE Youcef «Méthode exacte basée sur le concept de dominance pour minimiser le nombre de taches en retard, cas d'une ressource », mémoire pour l'obtention d'un diplôme d'ingénieur d'état en recherche Opérationnelle, USTHB promotion (2006).

[6] BAPTISE Autin « Les méta-heuristiques en optimisation combinatoire » Mémoire présenté en vue d'obtenir l'examen probatoire en informatique, Conservatoire nationale des arts et metiers, paris (2006).

[7] Fatma TANGOUR TOUMI « Ordonnancement Dynamique dans les Industries Agroalimentaires » Thèse de Doctorat de l'Université de TUNIS ELMANAR Ecole nationale d'ingénieurs TUNIS (2007).

[8] HERNANE Soumeya, BELKADI Khaled « Deux stratégies parallèles de l'optimisation par colonie de fourmis » (2007).

[9] SEBASTIEN NOEL «Méta-heuristiques Hybrides pour la résolution du problème d'ordonnancement de vôtres dans une chaine d'assemblage automobile », mémoire présenté comme exigence partielle de la maitrise en informatique, Université du Quebec a Chicoutreal et Université du Quebec a Montreal, (2007).

[10] Joseph Schäppi « Extension de MediMAS, Développement et déploiement d'agents JADE sur des supports mobiles » Travail de Bachelor, (2008)

[11] O.Kone, C.Artigues, P.Lopes et M.Mongeau. « Nouvelle formulation du problème d'ordonnancement de projet a moyens limites basée sur les événements ». ROA-DEF'(2008).Clermont –Ferrand.

[12] Giovanni Caire : «JADE TUTORIAL JADE PROGRAMMING FOR BEGINNERS» (2009)

[13] Hela Boukef BenOthman «L'ordonnancement d'ateliers job-shop flexibles et flow-shop en industries pharmaceutiques Optimisation par algorithmes génétiques et essaims particulaires» THÈSE présentée en vue d'obtenir le grade de DOCTEUR en Automatique et Informatique Industrielle, Doctorat délivré conjointement par l'École Centrale de Lille et l'École Nationale d'Ingénieurs de Tunis, (2009).

[14] KONE Oumar « Nouvelles approches pour la résolution du problème d'ordonnancement de projet a moyens limités » Thèse de Doctorat de l'Université de TOULOUSE (2009).

[15] AZEM Sadia «Ordonnancement des System Flexibles de production sous contraintes de disponibilité des ressources.» Thèse de Doctorat de l'Ecole Nationale Supérieure des Mines de Sant –Etienne (2010).

[16] BELKHELLADI kamel «Stratégie d'échange d'information dans un système de calcul distribue pour l'optimisation des problèmes combinatoires» Thèse de Doctorat de l'Université d'Angers (2010).

[17] FERRAH DJahida, BABA Kahina «Approche évolutionnaire pour la résolution du problème d'ordonnancement de type Job Shop Flexible Dynamique » , Université Saad Dahlab promotion (2010).

[18] ADDAD Nora, SAIDANI Monira « Approche méta-heuristique basée sur la méthode de recherche d'harmonique pour la résolution des problèmes d'ordonnancement industriels», mémoire pour l'obtention d'un diplôme Master en informatique. Université Saad Dahlab promotion (2011).

[19] F .Z.BENAYED, M.RAHLI et L.ABDELHAKE-KORIDAK «Optimisation du Dispatching Economique par l'Algorithme Harmony Search» Volume 52 ,numéro 1, (2011).

[20] Tarek CHAARI « Un algorithme génétique pour l'ordonnancement robuste : application au problème du flow shop hybride » Thèse en cotutelle pour obtenir le grade de Docteur de l'Université de Valenciennes et du Hainaut-Cambrésis, Discipline : Automatique, Spécialité : Automatique, Génie Informatique, et le grade de Docteur de la Faculté des Sciences Économiques et de Gestion de Sfax, Discipline : Méthodes Quantitatives (2011).

[21] LEKHAL Meriem, HAMAIDI Naima «Implémentation Parallèle Multi-Agent de méta-heuristique pour la résolution des problèmes d'ordonnancement industriels», mémoire pour l'obtention d'un diplôme d'ingénieur d'état en informatique, Université Saad Dahlab promotion (2011).