الجمهورية الجزائرية الديمقراطية الشعبية

وزارة التعليم العالي والبحث العلمي

جامعة سعيدة د. مولاي لطاهر

كلية الرياضيات و الإعلام الآلي و

الاتصالات السلكية و اللاسلكية

قسم: الإعلام الآلي

Mémoire de Master en informatique

Spécialité : Réseaux Informatique et Systèmes Répartis (RISR)

Thème

TS for performance evaluation of non identical server Scheduling system

▪ **Présenté par :**

BOUACHRIA Keltouma

▪ **Dirigé par**

Dr. MEKOUR Mansour

Année universitaire 2024-2025

الإهداء و الشكر

تسابقت الكلمات و تزاحمت العبارات لتنظم عقد الشكر الذي لا يستحقه إلا من غرسوا معي بذور نجاح و هاهو اليوم الذي احصد فيه انجازاتي بفضل الله تعالى ،

إليك يا أبي رحمه الله عليك يا من كان له قدم السبق في ركب العلم و التعليم،

إلى القلب المعطاء والصدر الحنون أمي الحبيبة

لروح أخي الغائب جسدا و الحاضر روحا ،

إلى من شد الله بهم عضدي فكانوا خير معين إخواني وأخواتي

إليك يا أستاذي الفاضل مكور منصور الذي لم يدخر جهدا و لا وقتا إليك أهدي عبارات الشكر و التقدير

كما أرفع عبارات الشكر و الامتنان لكل من قدم يد العون من أهل و أقارب و أصدقاء

لكل شكر قصيدة و لكل مقام مقال و لكل مناسبة خاتمة

هنيئا لي بهذا النجاح

Table des matières

Introduction Générale :

Dans un contexte où les systèmes informatiques deviennent de plus en plus complexes, la gestion efficace des ressources et la planification des tâches représentent des enjeux majeurs pour garantir la performance, la disponibilité et la qualité de service. Que ce soit dans le domaine industriel, logistique ou dans les infrastructures informatiques distribuées, l'ordonnancement des tâches ou des commandes clients constitue une problématique critique. Cette problématique consiste à affecter un ensemble de tâches à des ressources limitées, tout en respectant diverses contraintes telles que les délais, les priorités ou la capacité des machines.

Le problème d'ordonnancement de type Job Shop (Job Shop Problem – JSP) est un modèle classique de cette problématique. Il est reconnu pour sa complexité computationnelle et sa difficulté à être résolu de manière exacte, notamment dans les cas à grande échelle. Pour faire face à ces limites, les métaheuristiques se présentent comme des approches puissantes, capables de fournir des solutions satisfaisantes dans des délais raisonnables.

Parmi ces méthodes, la recherche tabou (Tabu Search) s'impose comme une technique d'optimisation efficace pour explorer l'espace des solutions tout en évitant les cycles de recherche et les minima locaux. Elle repose sur une stratégie de mémoire dynamique, qui guide la recherche vers des zones prometteuses de l'espace de solutions.

Dans ce travail, nous nous intéressons à l'adaptation de la recherche tabou à un cas pratique d'ordonnancement de commandes clients sur des serveurs hétérogènes. L'objectif est de développer une solution capable de simuler différentes configurations de charges et de ressources, puis d'évaluer l'efficacité de l'approche proposée à travers une interface graphique intégrant des outils de visualisation et d'analyse.

Ce mémoire structuré de trois chapitres dont nous vous présentons une brève description comme suite : Dans le premier chapitre, nous présente les fondements théoriques de la recherche tabou.

. Dans le deuxième chapitre, traite de la modélisation des commandes clients dans un cadre Job Shop et détaille l'adaptation de la recherche tabou à ce problème. pour le résoudre Le dernier chapitre, décrit l'implémentation de l'application développée, la simulation des cas d'étude et l'analyse des résultats expérimentaux obtenus

Chapitre 1

Recherche de tabou

1 - Introduction :

Les méthodes de recherche locale, telles que la recherche tabou, reposent sur un principe fondamental : elles commencent par une solution initiale et cherchent à améliorer cette solution en se déplaçant vers des solutions voisines par des ajustements successifs. L'ensemble des solutions accessibles à partir d'une solution donnée est appelé le voisinage de cette solution. La recherche tabou, introduite par Fred W. Glover en 1986, utilise un mécanisme de mémoire pour éviter de revenir sur des solutions précédemment explorées, ce qui aide à échapper aux minima locaux. Cette méthode est particulièrement efficace pour résoudre des problèmes d'optimisation complexes, car elle permet d'explorer l'espace des solutions de manière plus stratégique. Les algorithmes de recherche locale, comme le recuit simulé et la recherche tabou, sont souvent utilisés dans des applications pratiques telles que la planification, le routage et d'autres problèmes combinatoires. Ils sont appréciés pour leur capacité à trouver des solutions de qualité dans des délais raisonnables, même lorsque les méthodes de recherche systématiques échouent à traiter des instances de grande taille

2- Méthode de recherche de tabou :

2-1 Principe :

L'approche de la recherche tabou repose sur l'idée d'explorer le voisinage d'une solution actuelle, notée x^n , afin de sélectionner la meilleure solution possible, désignée par x^***[1]**. Il est important de comprendre que cette exploration peut parfois entraîner une augmentation de la valeur de la fonction à minimiser, notamment lorsque toutes les solutions voisines présentent des valeurs supérieures à celle de la solution actuelle. Ce mécanisme est crucial pour sortir d'un minimum local.**[2]**

Cependant, un risque subsiste : il est possible de retomber dans le même minimum local lors de l'étape suivante. Pour éviter ce phénomène, il est nécessaire d'implémenter un mécanisme qui empêche le retour à des solutions déjà explorées. Cela se fait par l'introduction d'une mémoire dans l'algorithme, qui conserve les solutions

précédemment visitées dans une liste, souvent appelée liste tabou. La taille de cette liste est un paramètre ajustable de l'heuristique et doit contenir des solutions complètes. Dans certains cas, cela peut nécessiter de stocker une grande quantité d'informations.**[3]**

Pour contourner cette difficulté, il est possible de ne conserver en mémoire que les mouvements effectués, associés à la valeur de la fonction à minimiser. Dans ce cas, il est envisageable que les listes tabous écartent des solutions non rencontrées. Pour remédier à cela, on peut mettre en place un système permettant de négliger le statut tabou de certaines solutions si cela entraîne un avantage suffisant. Cela se fait à l'aide de critères d'aspiration, tels que l'acceptation d'une solution tabou si elle améliore la valeur de la fonction objectif par rapport à toutes les solutions précédemment rencontrées, ou si cette amélioration est d'au moins 1 %. [4]

Les paramètres les plus critiques à définir dans la méthode de recherche tabou incluent la mémoire, la longueur de la liste tabou (notée LLL), le nombre de points à explorer autour du voisinage (noté MMM), ainsi que les critères d'aspiration, d'intensification et de diversification. À chaque itération, l'algorithme examine complètement le voisinage de la solution courante pour déterminer la meilleure option à suivre.

2-2 Mémoire :

L'algorithme maintient en mémoire un ensemble fini d'états récemment visités (mémoire explicite), ainsi qu'une liste d'états potentiellement interdits (mouvements tabous). Cette double conservation permet :

1. D'éviter les cycles en empêchant le retour immédiat aux solutions précédentes
2. De guider l'exploration vers des régions inédites de l'espace de recherche.[5]

2-3 Liste de Tabou :

La recherche tabou utilise une liste tabou pour représenter la mémoire des solutions précédemment explorées, ce qui permet de diriger l'exploration vers des régions non visitées. Cette liste est gérée comme une structure circulaire : lorsque de nouvelles solutions sont ajoutées, les plus anciennes sont éliminées. Cela peut s'avérer coûteux en termes de mémoire, car il faut conserver une quantité d'informations significative.

Le rôle principal de la liste tabou est d'interdire les mouvements cycliques, empêchant ainsi l'algorithme de revenir sur des solutions déjà explorées. La longueur de cette liste doit être soigneusement choisie, car une liste trop courte pourrait ne pas empêcher les cycles, tandis qu'une liste trop longue pourrait rendre l'algorithme moins efficace. En général, la valeur moyenne des solutions visitées tend à augmenter proportionnellement à la taille de la liste tabou, ce qui peut améliorer la qualité des solutions trouvées au fil du temps. [6]

3 - Critères d'aspiration :

Les critères d'aspiration permettent de lever les restrictions imposées par les mouvements tabous lorsqu'ils conduisent à des solutions de qualité. Voici les principaux critères d'aspiration :

- **Atteinte d'un nombre maximal d'itérations** : Par exemple, le processus peut s'arrêter après 1000 itérations.
- **Absence d'amélioration** : Si aucune amélioration n'est constatée pendant un certain nombre d'itérations, comme 100 itérations sans progrès, cela peut déclencher l'arrêt.
- **Atteinte d'une solution satisfaisante** : Le processus peut également se terminer lorsque le temps d'exécution total atteint un seuil prédéfini, inférieur à une valeur cible.

Ces critères aident à déterminer quand il est approprié de mettre fin à la recherche pour optimiser l'efficacité de l'algorithme.[7]

4 - Intensification :

L'intensification est une stratégie qui vise à concentrer la recherche sur une zone spécifique de l'espace des solutions considérée comme prometteuse. Cette méthode repose sur l'idée intuitive que si une solution particulièrement intéressante est découverte lors d'une exploration rapide, il est probable qu'un examen plus approfondi dans cette zone puisse révéler des solutions encore meilleures. **[8]**

5 – Diversification :

Pour éviter qu'une vaste zone de l'espace de recherche reste inexplorée, il est essentiel de diversifier les approches de recherche en effectuant plusieurs lancements aléatoires. La méthode la plus simple consiste à réaliser plusieurs de ces lancements aléatoires. Une autre approche, qui garantit l'exploration des zones non visitées, consiste à appliquer une pénalité sur les mouvements ou solutions qui ont été fréquemment explorés. Cette pénalité est conçue pour encourager l'éloignement et l'évitement des régions déjà explorées.

Il est également possible d'imposer une pénalité sur les mouvements souvent réalisés tout au long du processus de recherche. Pendant cette phase de diversification, il est important de noter que les solutions visitées ne doivent pas nécessairement être réalisables.

6 – Algorithme de base e la recherche de tabou :

Initialisation :

Identification d'une solution initiale, s_0

Création d'une liste tabou vide,

On pose :

Meilleure solution = solution,

définir une condition d'arrêt

Répéter :

if Valeur de la solution > valeur de la meilleure solution

then.

Poser meilleure solution = solution

if la condition d'arrêt n'est pas satisfaite **then begin**

Ajouter la solution à la liste tabou

if la liste tabou est pleine **then**

Supprimer les anciennes solutions de la liste tabou

Trouver une nouvelle solution par des transformations de la solution

if aucune solution trouvée **or**

if aucune nouvelle solution meilleure trouvée pour une longue période **then**

Générer aléatoirement une nouvelle solution

if la liste tabou ne contient pas la nouvelle solution générée **then**

Poser solution = nouvelle solution

End.

Algorithme de base de la recherche tabou est donné par la Figure 1 [5].

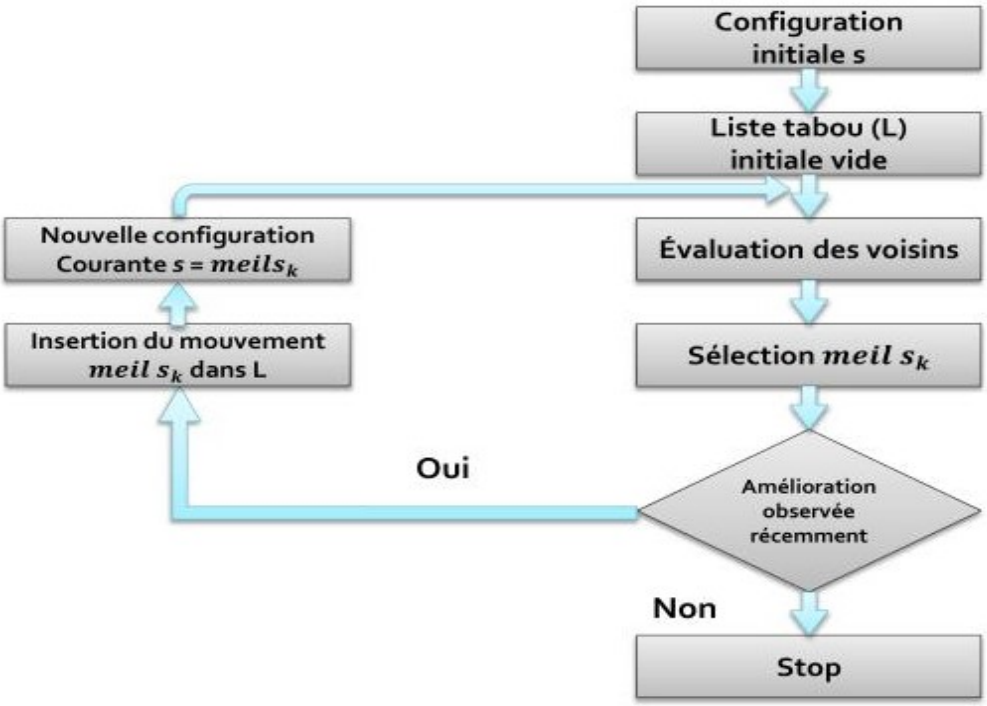


Figure 1 : Algorithme de base de la recherche tabou

s :solution

meil s_k : meilleur solution de $k_{ième}$ itération

7 - Conclusion :

Ce premier chapitre présent la recherche tabou, une métaheuristique puissante conçue pour résoudre des problèmes complexes en évitant les minima locaux grâce à sa mémoire adaptative. Il détaille ses principes fondamentaux, tels que la liste tabou, les critères d'aspiration, et les stratégies d'intensification et de diversification. Son algorithme, basé sur la gestion du voisinage et l'évaluation, est illustré pour montrer son fonctionnement.

La recherche tabou est ainsi présentée comme une méthode flexible et efficace, particulièrement adaptée à l'ordonnancement, notamment pour le problème du Job Shop, abordé dans le chapitre suivant.

Chapitre 2

Ordonnancement des Commandes Clients : Modélisation Job Shop et Approche Tabou

1 - Introduction :

L'ordonnancement des commandes clients est un enjeu crucial dans les systèmes de production modernes, où il s'agit d'organiser efficacement l'exécution des tâches pour optimiser l'utilisation des ressources tout en respectant des contraintes telles que les délais, les priorités et les ressources. Ce problème est similaire à celui du Job Shop, connu pour sa complexité et sa nature NP-difficile. Dans ce contexte, chaque commande est traitée comme une tâche assignée à une ressource unique, ce qui simplifie le modèle tout en restant réaliste.

Pour aborder ce défi, nous avons adapté l'algorithme de recherche tabou, une méthode d'optimisation efficace, à notre cas d'utilisation. Ce chapitre se concentre sur la modélisation des commandes clients, les contraintes associées à leur traitement, et les ajustements nécessaires à la méthode tabou pour répondre aux exigences spécifiques de ce domaine.

2- Modélisation des commandes clients :

2-1 Définition et caractéristiques d'une commande client

Une commande client représente une demande précise faite par un utilisateur ou un client, qui nécessite l'exécution d'un ensemble d'opérations dans un délai défini. Dans un environnement de traitement automatisé ou distribué, chaque commande est considérée comme une tâche indépendante, qui doit être assignée à une ressource, comme un serveur, pour son traitement.

Les caractéristiques principales d'une commande client incluent la date à laquelle elle a été soumise, le temps estimé pour son traitement, les ressources nécessaires et, parfois, son niveau de priorité. La variété et le volume de ces commandes exigent une planification efficace pour garantir la performance du système et satisfaire les attentes des clients.

2-2 Contraintes associées (délais, ressources, priorités)

L'ordonnancement des commandes clients est soumis à plusieurs contraintes opérationnelles essentielles pour garantir un fonctionnement optimal du système.

La première contrainte concerne les délais : certaines commandes doivent être traitées avant une échéance précise, ce qui impose une gestion rigoureuse du temps. La seconde contrainte est liée aux ressources disponibles, notamment la capacité des serveurs ou des machines à exécuter les tâches, ces ressources pouvant être limitées ou partagées. Enfin, les priorités jouent un rôle crucial, car certaines commandes peuvent être considérées comme plus urgentes ou stratégiques que d'autres, nécessitant ainsi un traitement prioritaire.

Ces contraintes combinées rendent le problème d'ordonnancement complexe et justifient le recours à des méthodes d'optimisation avancées.

2-3 Modélisation des commandes sous forme de jobs

Dans un système de planification automatisée, une commande client peut être modélisée comme un job, c'est-à-dire une unité de travail composée d'un ensemble ordonné de tâches à exécuter. Chaque tâche représente une opération spécifique à réaliser sur une ressource déterminée (par exemple un serveur ou une machine), selon une séquence définie.

Cette représentation permet d'intégrer diverses informations clés :

- un **identifiant unique** pour chaque job,
- une **durée estimée** pour chaque tâche,
- une **ressource cible** par tâche,
- et des **contraintes associées** (priorité, échéances, interdépendances, etc.).

Cette représentation formelle permet d'abstraire chaque commande comme un ensemble structuré d'opérations réparties dans le temps et sur les ressources disponibles. Elle reflète fidèlement les exigences opérationnelles des systèmes informatiques complexes, dans lesquels l'ordonnancement des tâches a un impact direct sur les performances globales.

Ce modèle rejoint naturellement le problème de Job Shop (JSP), un cadre théorique bien établi dans le domaine de l'optimisation combinatoire. Dans ce problème, plusieurs jobs (commandes) doivent être planifiés sur un ensemble limité de ressources, chaque job étant composé de tâches devant être exécutées dans un ordre précis, et chaque ressource ne pouvant traiter qu'une tâche à la fois.

Cette modélisation permet de formaliser rigoureusement le problème d'ordonnancement, et de le résoudre à l'aide de méthodes d'optimisation combinatoire, comme la recherche tabou, en vue d'améliorer les performances globales du système. (temps de traitement, respect des délais, équilibre des charges, etc.).

3 – Présentation du problème job shop :

3-1 Atelier à cheminement multiple (Job Shop) :

Dans un atelier de type job shop, chaque tâche suit un itinéraire spécifique qui lui est propre. En d'autres termes, les tâches ne passent pas par les mêmes machines ni dans le même ordre. Ce mode d'organisation est typique des systèmes de production par lots, où l'on fabrique une diversité de produits. Il s'applique particulièrement aux unités de production disposant d'équipements polyvalents, utilisés selon des séquences différentes en fonction des besoins de chaque produit. **[9]**

Si un système de production où chaque tâche suit un ordre fixe de machines, avec une seule machine par opération on dit que l'organisation est un Job Shop simple . Si, au contraire, introduit plusieurs machines pour une même tache on l'appelle Job Shop hybride comme la montre dans la Figure 2 .

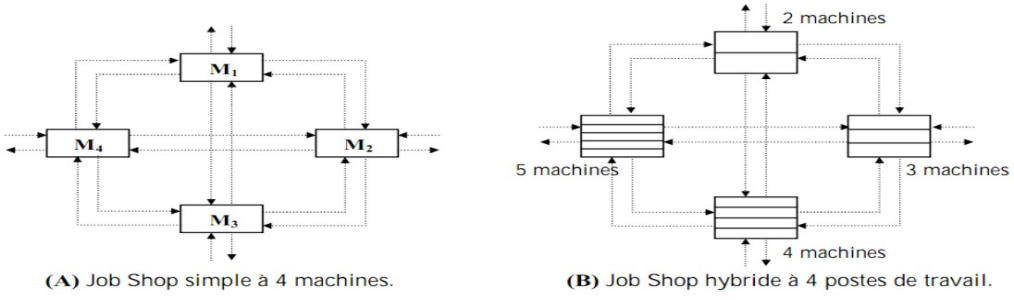


Figure 2 : Exemple de Job Shop simple et hybride

3-2 Types de Job Shop :

3-2-1 Job shop Classique (Simple) :

Le problème d'ordonnancement de type Job Shop est l'un des plus étudiés dans le domaine de l'ordonnancement en atelier. Il existe de nombreuses variantes de ce problème, ce qui rend difficile l'adoption d'une définition unique dans la littérature. Nous nous intéressons ici à la formulation la plus générale du Job Shop simple.

Ce problème consiste à planifier l'exécution Il consiste exécuter un ensemble de n jobs, notés J={J1,J2,…,Jn}, sur un ensemble de m machines, notées M={M1,M2,…,Mm}. Chaque job Ji est constitué d'une séquence ordonnée d'opérations Oi,1,Oi,2,…,Oi,n , pour lesquelles des contraintes de précédence sont établies. À chaque opération Oi,j est associé un temps de traitement pij, représentant sa durée d'exécution , qui doivent être réalisées dans un ordre précis, chaque opération devant être exécutée sur une machine spécifique tout en cherchant à atteindre certains objectifs (comme minimiser le temps total, les retards, etc.).

Le problème est soumis à plusieurs contraintes usuelles, notamment :

- Les machines fonctionnent indépendamment les unes des autres.
- Une machine ne peut traiter qu'une seule opération à la fois.
- Les machines sont supposées disponibles pendant toute la durée de l'ordonnancement.
- Les jobs sont également indépendants entre eux : aucun ordre de priorité n'est imposé entre les différents jobs.

La figure (3) représente un problème type de job shop classique composé de 3 jobs et 4 machines, les gammes opératoires sont les suivantes:

J1 :M1-M2-M4. J2 :M2-M3-M1. **J3** :M4-M3-M2.

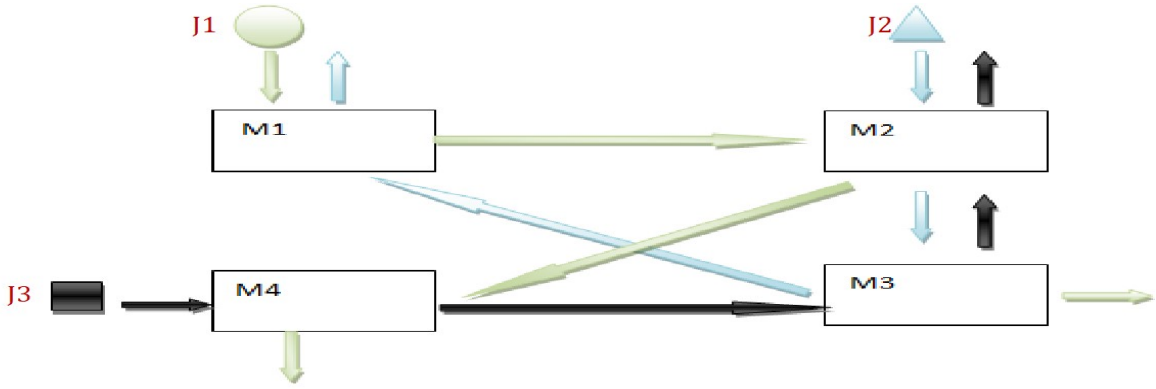


Figure (3): job shop classique à 3 jobs et 4 machines.

3-2-2 Job shop Flexible (Hybride) :

Le job shop flexible est une extension du modèle job shop classique. Sa particularité essentielle réside dans le fait que chaque opération peut être exécutée sur plusieurs machines. Dans ce modèle, les machines qui effectuent la même opération sont groupées dans un même étage. Il s'ensuit qu'il offre plus de flexibilité par rapport au job shop classique grâce à la polyvalence de ces machines. Toutefois, cela induit une complexité supplémentaire due à la nécessité de la détermination des affectations adéquates avant d'établir l'ordre de passage des différentes opérations sur les machines. Un exemple de ce problème à m étages et trois machines maximum par étage, est donné dans la Figure(4). [10]

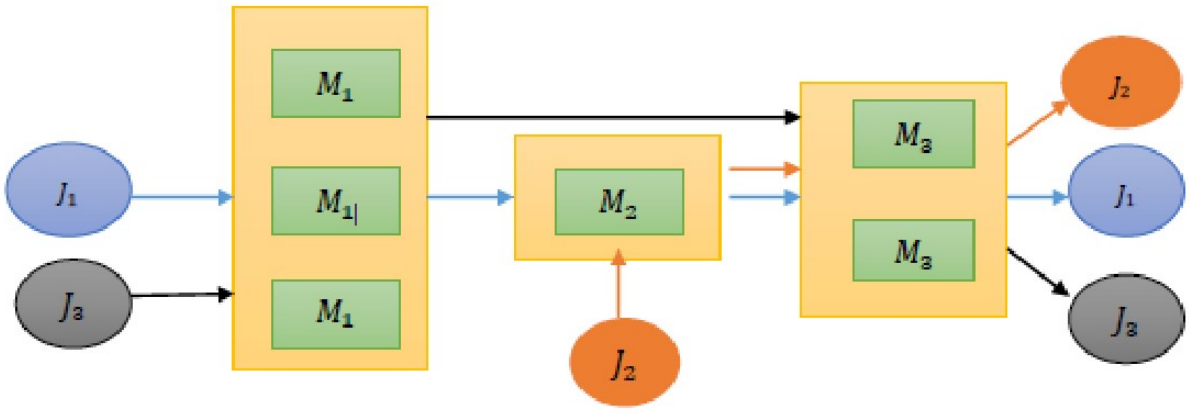


Figure (4): Représentation d'un système de type Job-shop flexible à deux étages

Une remarque importante concernant les problèmes de type jobshop hybride est que le nombre de machines peut varier d'un étage à l'autre, tout comme les performances des machines, qui ne sont pas nécessairement identiques au sein d'un même étage. Ces performances permettent de classer les systèmes avec machines parallèles en trois catégories :

- **Machines identiques** : La durée d'exécution d'une tâche est la même sur toutes les machines.
- **Machines uniformes** : La durée d'exécution d'une tâche est similaire sur toutes les machines, mais peut légèrement varier.
- **Machines indépendantes** : La durée d'exécution d'une opération dépend spécifiquement de la machine sur laquelle elle est réalisée.

3-3 Les caractéristiques clés du Job Shop:

- **Tâches et machines** : Chaque job est composé d'une séquence de tâches qui doivent être exécutées sur des machines spécifiques.
- **Contraintes** : Les contraintes incluent des conditions de précédence (une tâche doit être terminée avant qu'une autre puisse commencer) et des contraintes d'absence de chevauchement (une machine ne peut traiter qu'une seule opération à la fois.) disponibilité continue (toutes les machines sont supposées disponibles dès le début et restent opérationnelles pendant toute la durée de l'ordonnancement), Indépendance des jobs (aucun ordre de priorité ou de dépendance n'est imposé entre les différents jobs).
- Le Job Shop est considéré comme un problème NP-difficile, ce qui signifie qu'il est difficile de trouver une solution optimale dans un temps raisonnable, surtout pour des instances de grande taille.

4- Principes Généraux de la Recherche Tabou :[1]

La recherche tabou est une méthode métaheuristique qui améliore les solutions en explorant les options voisines tout en évitant les cycles grâce à une mémoire des mouvements interdits, connue sous le nom de liste tabou. Lorsqu'elle est appliquée au

problème d'ordonnancement de type job shop, elle commence par un ordonnancement initial qui est admissible et évolue en modifiant les opérations.

Des critères d'aspiration sont parfois utilisés pour permettre de contourner ces interdictions si une solution prometteuse est identifiée. L'algorithme s'appuie également sur des stratégies de diversification à long terme, tout en respectant les contraintes d'ordonnancement des tâches sur les machines.

5- Voisinage d'une solution :

Dans la méthode de recherche tabou appliquée au problème d'ordonnancement de type job shop, le concept de voisinage est crucial. À partir d'une solution s, un ensemble de solutions voisines, noté N(s), est généré par l'application d'opérateurs de transformation définis par des heuristiques locales. L'objectif est de sélectionner une nouvelle solution s' appartenant à N(s) qui minimise la fonction objectif, tout en respectant certaines contraintes, notamment l'exclusion des solutions tabous.

Lorsque l'évaluation de l'ensemble complet du voisinage est coûteuse en temps de calcul ou en mémoire, surtout dans le cas de grands espaces de recherche, il est souvent plus efficace de considérer un sous-ensemble aléatoire $N'(s) \subset N(s)$. Cette stratégie permet de réduire les ressources utilisées tout en maintenant une exploration diversifiée. Pour éviter les retours cycliques vers des solutions déjà explorées, une liste tabou est mise en place. Cette liste sert de mémoire à court terme, enregistrant les derniers mouvements effectués ou les attributs des solutions récemment visitées.

Cependant, pour ne pas bloquer indéfiniment l'accès à des solutions pratiquement intéressantes, un critère d'aspiration peut être appliqué, permettant de lever temporairement une interdiction tabou si la solution considérée améliore significativement la meilleure solution connue. Cette gestion conjointe du voisinage, de la mémoire tabou et du critère d'aspiration permet à l'algorithme d'explorer efficacement l'espace de recherche, en alternant entre phases d'intensification et de diversification. . [1] [3][12]

6- Structure générale de l'algorithme de recherche tabou : [11]

L'algorithme représenté caractérise l'implantation de base de la technique de Recherche tabou :

Identification d'une Solution initiale
Création d'une Liste Tabou vide
Poser Meilleur Solution = Solution
Définir les Conditions d'Arrêts
Faire = Faux
Répéter
Si *valeur de la Solution<valeur de la Meilleure Solution*
Alors
Meilleure Solution = Solution
Si *la Condition d'Arrêt n'est pas satisfaite*
Alors
Ajouter la solution à la Liste Tabou
Si *la Liste Tabou est pleine*
Alors
Supprimer les anciennes solutions de la Liste Tabou
Fin
Sinon
Condition d'Arrêt est satisfaite
Faire = Vrai
Fin
Sinon
Trouver une Nouvelle Solution par des transformations de la solution
Si *aucune Nouvelle Meilleure Solution trouvée pour une longue période*
Alors
Générer aléatoirement une Nouvelle Solution
Si *la liste tabou ne contient pas la Nouvelle Solution*
Alors
Solution = Nouvelle Solution
Fin
Fin
Jusqu'à Faire=Vrai

Pour traduire l'algorithme de base de la recherche tabou, nous avons élaboré un programme écrit en langage PYTHON dont les paramètres de base qui interviennent dans l'algorithme sont :

- **Ndiv** nombre de diversifications**,**
- **L** longueur de la liste tabou**,**
- **Imax** nombre maximal d'itérations**,**
- **M** nombre de points de recherche autour du voisinage**.**

7 - Fonction d'évaluation dans le Job shop :

Dans le problème d'ordonnancement de type Job Shop Scheduling Problem (JSSP), la fonction d'évaluation vise principalement à mesurer la qualité d'une solution, c'est-à-dire l'ordonnancement des opérations sur les machines. La métrique la plus couramment utilisée pour cela est le makespan, noté C_{max}, qui représente le temps total nécessaire pour achever toutes les opérations de tous les jobs.

Considérons un ensemble de n jobs J={J1,J2,…,Jn} à traiter sur m machines M={M1,M2,…,Mm}. Chaque job Ji est composé d'une séquence ordonnée de n_i opérations $O_i,1,O_i,2,…,O_i,n_i$, qui doivent être exécutées dans un ordre strict.

Le temps de complétion d'une opération Oi,j est noté Ci,j , ce qui correspond au moment où cette opération est terminée. Le temps total de complétion du job Ji est donc donné par le temps de fin de sa dernière opération $O_{i,ni}$.

La fonction objectif à minimiser est donc définie par :

$$\mathbf{F = C_{max} = max_{1\leq i\leq n} \; C_{i,ni}}$$

Cette fonction reflète le makespan, ou la durée totale de traitement de l'ensemble des jobs.[13]

Pour garantir la validité des ordonnancements, plusieurs contraintes doivent être prises en compte :

- **Contraintes de précédence au sein d'un même job** : chaque opération doit commencer après la fin de l'opération précédente,
- **Contraintes d'exclusivité machine** : Chaque machine ne peut traiter qu'une seule opération à la fois, ce qui implique que les intervalles de temps alloués aux différentes opérations assignées à une même machine doivent être disjoints.

L'optimisation consiste donc à déterminer un ordonnancement qui respecte ces contraintes tout en minimisant la fonction C_{max} . Cette fonction d'évaluation est utilisée dans la recherche tabou pour comparer et sélectionner les solutions voisines lors des itérations successives.

8- Conclusion :

Ce chapitre explore en profondeur la problématique de l'ordonnancement des commandes clients dans un environnement multi-ressources, en utilisant le modèle Job Shop Scheduling Problem (JSSP). Chaque commande est modélisée comme un job avec des tâches successives sur des serveurs non identiques, ce qui met en évidence la complexité du planning, notamment en raison des contraintes de précédence, ressources et délais.

Pour résoudre cette complexité, la recherche tabou est présentée comme une méthode efficace pour explorer l'espace des solutions tout en évitant les minima locaux, grâce à ses mécanismes de mémoire, d'aspiration, d'intensification et de diversification.

L'intégration de cette méthode dans le contexte des serveurs hétérogènes offre une approche flexible, performante et adaptable, ouvrant la voie à l'implémentation concrète, à l'expérimentation via une interface utilisateur, et à l'analyse des résultats dans le prochain chapitre.

Chapitre 3
Implémentation

1 - Introduction :

Ce chapitre fournit une analyse approfondie de l'interface graphique et de l'algorithme de recherche tabou utilisé dans la planification des serveurs non identiques. L'objectif principal est de proposer une solution interactive capable d'exécuter l'algorithme d'optimisation et d'en afficher les performances de manière intuitive.

2- Environnement de développement

Langage utilisé : Python 3

Bibliothèques principales :

Tkinter : pour la création de l'interface graphique.

Matplotlib : pour la génération des graphiques.

Json : pour le stockage et le transport des données.

numpy, random, threading : pour la logique et l'optimisation.

IDE utilisé : PyCharm .

Matérielle : PC DELL RAM 8 Go DD : 250 SDD .

3- Architecture de l'application

L'application développée repose sur une seule classe principale nommée RechercheTabouApp. Cette classe comprend toutes les tâches nécessaires pour le bon fonctionnement du système, l'architecture est décomposée en plusieurs modules logiques :

3-1 L'interface utilisateur :

Développée avec Tkinter, permet à l'utilisateur de configurer les paramètres de l'algorithme, comme le nombre de tâches, le nombre de serveurs, le nombre d'itérations et la taille de la liste tabou. Elle offre également des visualisations pour consulter les résultats, notamment un tableau d'ordonnancement ainsi que des graphiques illustrant l'évolution du coût et du temps d'exécution.

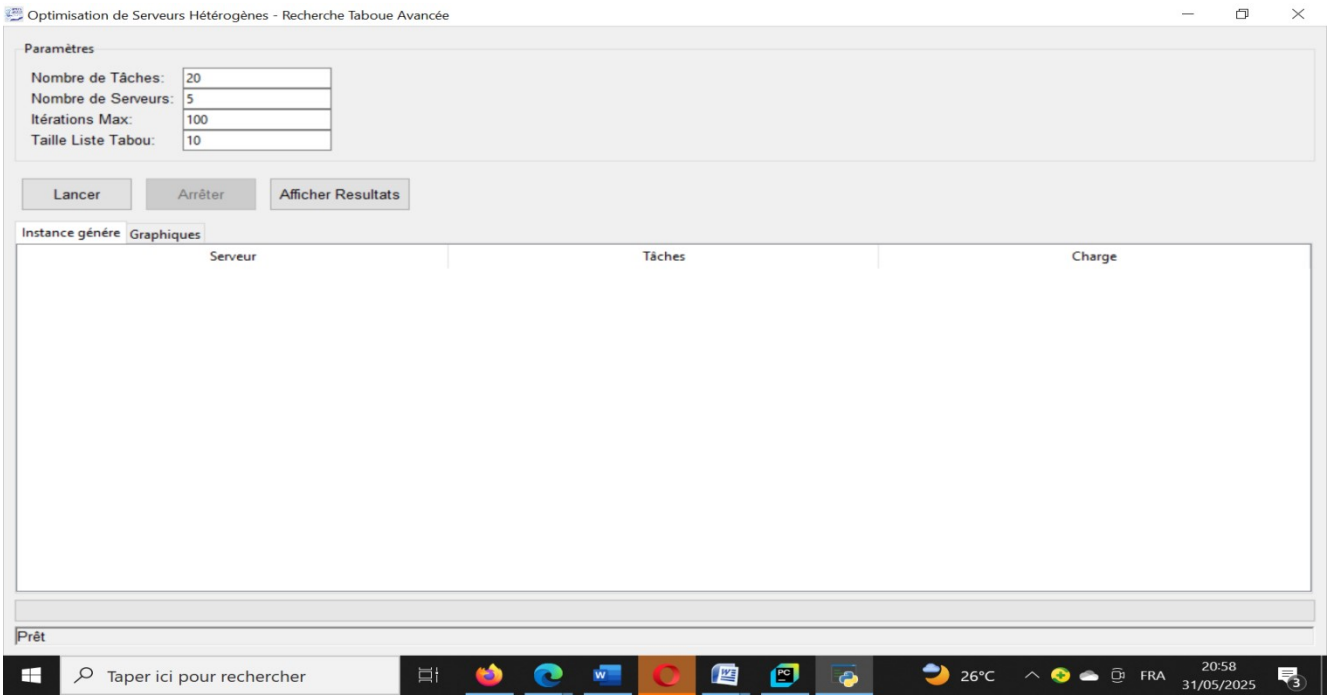


Figure (5) : Interface Initiale.

3-2 Le noyau de l'algorithme de recherche tabou:

L'implémentation regroupe les fonctions clés de la recherche tabou, notamment la génération de la solution initiale, le calcul du makespan, la création de voisins, la gestion de la liste tabou et la logique centrale de l'algorithme. Un mécanisme de cache (self.cache_cout) optimise les performances en évitant les recalculs inutiles. L'affichage des résultats permet d'évaluer le comportement de l'algorithme sur divers scénarios, tandis que l'intégration de Matplotlib dans l'interface Tkinter assure une visualisation graphique claire de l'évolution du coût et du temps d'exécution.

3-3 Principes Clés de l'Implémentation :

- **Génération de la solution initiale** : La fonction generer_solution_initiale crée une solution aléatoire admissible, affectant uniformément les tâches aux serveurs. Elle constitue le point de départ de la recherche.
- **Évaluation des solutions** : Le coût d'une solution est calculé via la fonction calculer_cout, basée sur le **makespan** (temps d'achèvement maximal). Un cache est utilisé pour éviter les recalculs.

- **Exploration du voisinage** : La fonction generer_voisins modifie l'affectation d'une tâche pour explorer de nouvelles solutions voisines, enrichissant la recherche locale.
- **Mécanisme de recherche tabou** : Implémentée dans recherche_tabou, cette méthode gère la solution courante, la meilleure solution trouvée, et la liste tabou. Un critère d'aspiration est implicitement appliqué pour accepter les solutions tabous améliorantes.
- **Contrôle et visualisation** : Les fonctions start_algorithm et stop_algorithm gèrent l'exécution en thread pour préserver la réactivité de l'interface. update_results et update_graphs assurent l'affichage des résultats et des graphiques d'évolution du coût via Matplotlib.
- **Module de Afficher resultats** : Le système de benchmark (run_benchmark, display_benchmark_results, etc.) permet de comparer automatiquement les performances de l'algorithme sous différents paramètres. Les résultats sont présentés sous forme de tableaux et graphiques répartis dans plusieurs onglets.

3-4 Optimisations et Stratégies Avancées Implémentées :

- **Mémorisation des coûts** : Un mécanisme de cache permet d'éviter les recalcules inutiles en stockant les coûts des solutions déjà évaluées, ce qui accélère sensiblement l'exécution.
- **Génération aléatoire des voisins** : Les voisins sont créés par des modifications aléatoires de l'affectation des tâches, avec un nombre fixe de 20 voisins générés par itération.
- **Suivi des solutions tabous** : Une liste tabou conserve les dernières solutions explorées (sous forme de tuples), empêchant ainsi les retours vers des états déjà visités.
- **Règle d'aspiration implicite** : Même si elle n'est pas explicitement nommée, une solution tabou est acceptée si elle surpasse la meilleure solution connue en termes de coût.

- **Benchmarking comparatif** : Une phase de test permet d'évaluer l'impact des différents paramètres sur les performances globales de l'algorithme (qualité des résultats et temps d'exécution).

En parallèle, plusieurs défis techniques ont été surmontés :

- **Affichage graphique avec Tkinter** : L'intégration de Matplotlib dans Tkinter a nécessité l'utilisation de composants spécifiques (comme FigureCanvasTkAgg) pour une visualisation fluide.
- **Maintien de la réactivité de l'interface** : Pour éviter les blocages pendant l'exécution, l'algorithme s'exécute dans un thread séparé, assurant une bonne interactivité.
- **Encodage des solutions** : Les solutions sont encodées sous forme de listes puis converties en tuples immuables pour leur compatibilité avec la liste tabou.

3-5 Simulation pour tester l'ordonnancement des commandes clients :

Pour vérifier l'efficacité de notre algorithme de recherche tabou dans la gestion de serveurs hétérogènes, nous avons développé un simulateur intégré à l'application. Voici comment il fonctionne :

3-5-1 Génération des données de test :

- Le système crée automatiquement un ensemble de commandes clients (tâches)
- Chaque tâche a un temps d'exécution différent selon le serveur utilisé

3-5-2 Paramètres configurables :

- Nombre de commandes à traiter
- Nombre de serveurs disponibles
- Nombre maximum d'essais pour l'algorithme
- Taille de la "liste tabou" (mémoire des solutions déjà testées)

3-5-3 Fonctionnement :

- Les temps de traitement sont générés aléatoirement dans un tableau
- Chaque valeur représente le temps qu'une tâche prend sur un serveur donné
- L'algorithme cherche la répartition qui minimise le temps total (makespan)

3-5-3 Résultats visuels :

Affichage de la meilleure répartition trouvée (quelles tâches sur quels serveurs)

Deux graphiques :

- Progression de la solution au fil des essais
- Temps de calcul accumulé

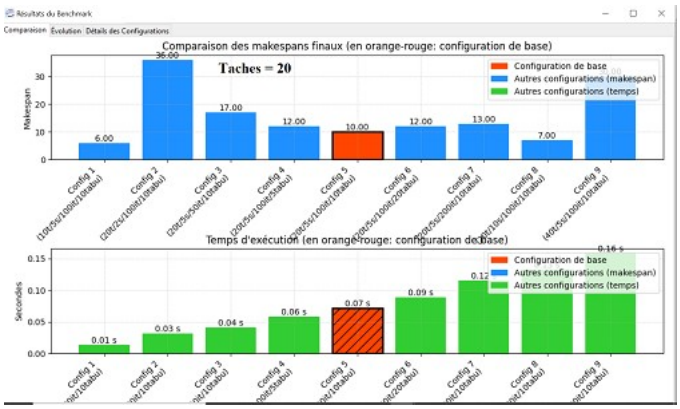
4 - Etude de Cas :

L'étude de cas permet d'analyser une situation réelle afin de comprendre un problème, d'examiner les solutions apportées et d'en tirer des enseignements utiles pour des cas similaires.

4-1 Influence du Nombre de Tâches

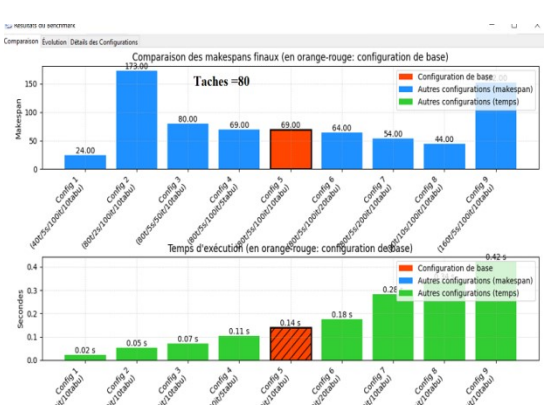
1-Configuration :

- Nombre de serveurs : Fixe (par exemple, 5)
- Nombre d'itérations : Fixe (par exemple, 100)
- Taille de la liste tabou : Fixe (par exemple, 10)
- Nombre de tâches : Faire varier (20, 80)



Comparaison des makespans finaux et temps d'cxecution

Nombre de taches =20



Comparaison des makespans finaux et temps d'cxecution

Nombre de taches =80

2 - Analyses :

- Makespan : Augmentation marquée avec 80 tâches, en raison d'une charge de travail plus élevée.

- Temps d'exécution : Plus long avec 80 tâches, l'espace de recherche étant plus vaste.
- Configurations efficaces :
- *20 tâches* : Configuration 8 (100 itérations / taille tabou 10) donne les meilleurs résultats.
- *80 tâches* : Configuration 1 (100 itérations / taille tabou 5) a le makespan le plus bas, mais reste élevé. La configuration 8 reste compétitive avec plus d'itérations.
- *Configuration 2* (itérations élevées / petite liste) donne de mauvais résultats dans les deux cas.

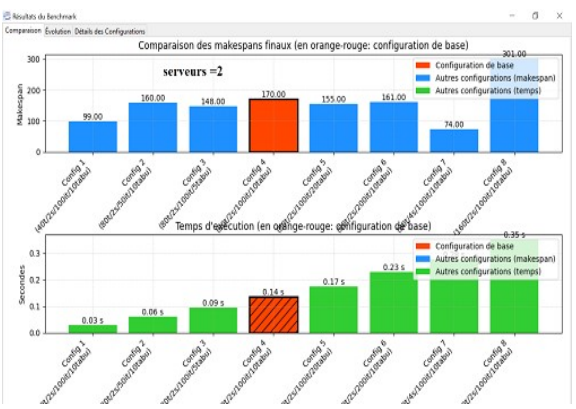
3 – Conclusion :

Ces deux études de cas dotent bien l'importance du nombre de tâches comme facteur déterminant de la performance de l'algorithme de recherche tabou dans cette situation d'ordonnancement. Ils soulignent également la nécessité d'une exploration attentive de l'espace des paramètres pour trouver une configuration efficace.

4-2 Influence du Nombre de Serveurs :

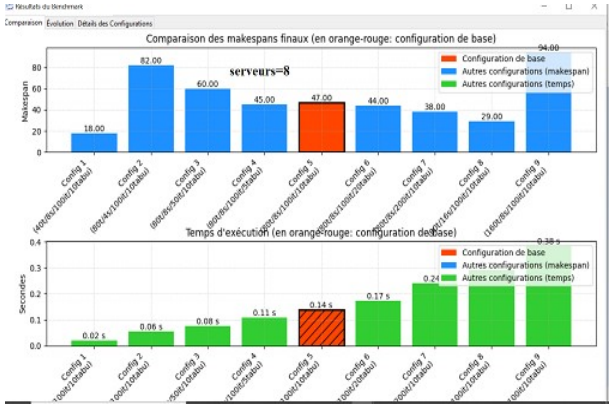
1- Configuration :

- Nombre d'itérations : Fixe (par exemple, 100)
- Taille de la liste tabou : Fixe (par exemple, 10)
- Nombre de tâches : Fixe (par exemple, 80)
- Nombre de serveurs : varier (2, 8)



Comparaison des makespans finaux et temps d'cxecution

Nombre de serveurs =2



Comparaison des makespans finaux et temps d'cxecution

Nombre de serveurs =8

2 – Analyse :

- **Makespan** : L'augmentation du nombre de serveurs améliore nettement le makespan, surtout avec 100 itérations.
- **Temps d'exécution** : Plus long avec 8 serveurs, en raison d'un espace de recherche plus complexe.
- **Taille de la liste tabou** : Son efficacité dépend du nombre de serveurs ; une taille de 8 a donné de bons résultats avec 8 serveurs.
- **Compromis** : 8 serveurs offrent de meilleures solutions mais à un coût temporel plus élevé ; 2 serveurs sont plus rapides mais moins efficaces sur la qualité des solutions.

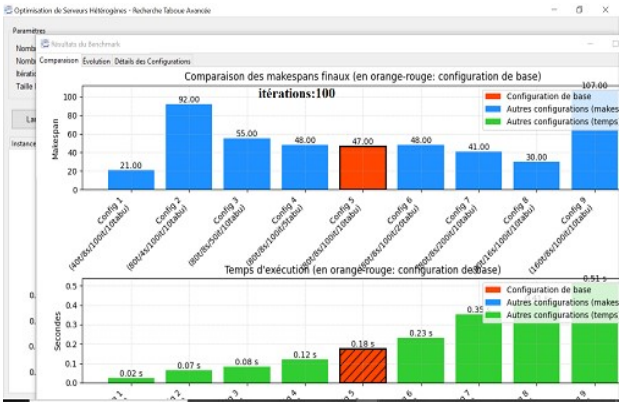
3 - Conclusion :

L'augmentation du nombre de serveurs de 2 à 8 a généralement permis d'obtenir de meilleurs ordonnancements (makespans plus faibles) au prix d'un temps de calcul plus important. Le choix du nombre de serveurs doit donc tenir compte des priorités spécifiques du problème (minimisation du temps d'achèvement vs. contraintes de temps de calcul et de coûts des ressources).

4-3 Influence du Nombre d'Itérations :

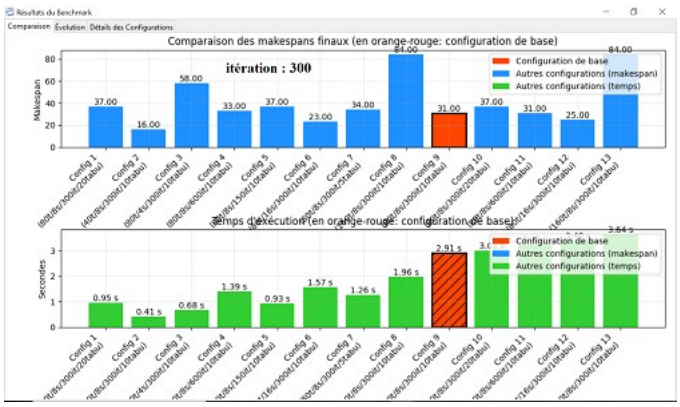
1- Configuration :

- Taille de la liste tabou : Fixe (par exemple, 10)
- Nombre de tâches : Fixe (par exemple, 80)
- Nombre de serveurs : Fixe (par exemple , 8)
- Nombre d'itérations : varier (100, 300)



Comparaison des makespans finaux et temps d'cxecution

Nombre d'iteration = 100



Comparaison des makespans finaux et temps d'cxecution

Nombre d'iteration = 300

2- Analyses :

- Makespan : Passer de 100 à 300 itérations améliore la qualité des solutions.
- Temps d'exécution : Augmentation significative, proportionnelle au nombre d'itérations.
- Paramètres associés : Avec plus d'itérations, l'algorithme devient plus stable, même avec des configurations sous-optimales. Les grandes tailles de liste tabou deviennent plus efficaces.
- Convergence : Plus d'itérations permettent une exploration plus poussée, réduisant le risque de convergence prématurée.
- Compromis : Il faut équilibrer qualité de solution et coût en temps selon les ressources disponibles.

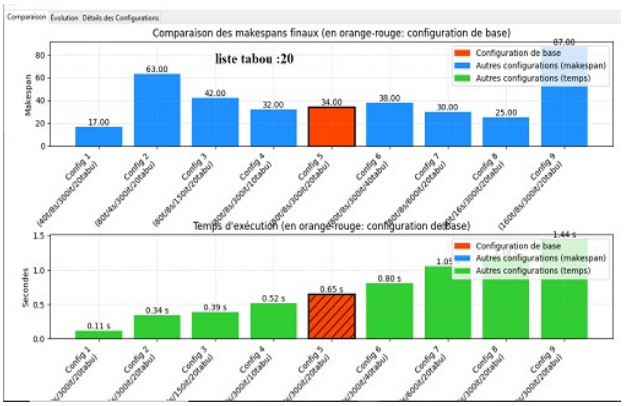
3- Conclusion :

L'augmentation du nombre d'itérations à 300 a permis d'obtenir de meilleurs makespans pour toutes les tâches, confirmant l'intérêt d'un temps de recherche maintenu. Cette amélioration s'accompagne d'un coût en temps de calcul nettement plus élevé.

4-4 Influence de la taille de liste de Tabou :

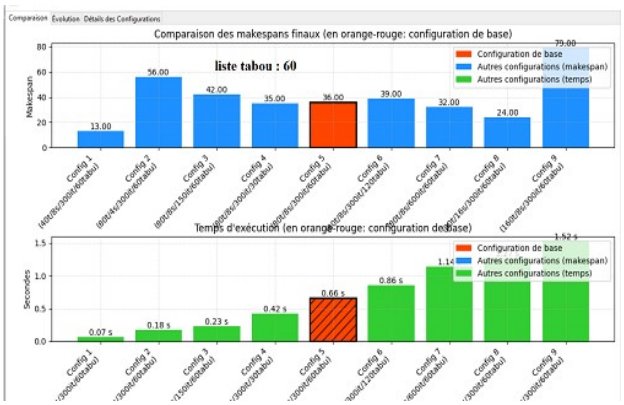
1-Configuration :

- Nombre de tâches : Fixe (par exemple, 80)
- Nombre de serveurs : Fixe (par exemple , 8)
- Nombre d'itérations : Fixer (par exemple, 300)
- Taille de la liste tabou : Varier (20,60)



Comparaison des makespans finaux et temps d'cxecution

Nombre de liste de tabou= 20



Comparaison des makespans finaux et temps d'cxecution

Nombre de liste de tabou = 60

2- Analyse :

- Tendances Générales : La taille 20 offre des résultats aussi bons, voire meilleurs, que la taille 60, qui n'apporte pas d'amélioration claire sur le makespan.
- Variabilité Stochastique : Les performances varient selon les exécutions ; les résultats doivent être interprétés avec circonspection et idéalement moyennés.
- Exploration et Exploitation : Une grande taille (60) peut diminuer l'exploration utile, tandis qu'une taille plus modéré (20) favorise une meilleure dynamique de recherche.

3 - Conclusion :

Une taille de liste tabou de 20 s'est révélée globalement plus efficace que celle de 60, offrant de meilleurs ou équivalents makespans sans surcharger la recherche. Une taille trop grande peut limiter inutilement l'exploration. Le choix optimal reste dépendant du problème et doit être ajusté expérimentalement.

4-5 Analyses du résultats :

Les performances dépendent fortement de la qualité de la solution initiale, de la taille de la liste tabou, et du nombre de voisins générés à chaque itération. Une phase de benchmark a permis d'identifier des configurations optimales de paramètres, garantissant un bon équilibre entre efficacité et rapidité.

Les courbes d'évolution du coût au fil des itérations confirment une amélioration rapide suivie d'une phase de stabilisation. Enfin, les résultats sont reproductibles et robustes, ce qui témoigne de la fiabilité de l'approche dans un cadre pratique.

5 - Conclusion :

En conclusion, cette recherche met en avant que l'efficacité de la recherche tabou dépend principalement d'une gestion des itérations et des serveurs, tandis que la liste tabou fonctionne comme un ajustement précis. Une stratégie rigoureuse, expérimentant diverses combinaisons, facilite l'atteinte d'un compromis idéal entre la qualité des solutions et les ressources de calcul. Ces constatations fournissent des orientations spécifiques pour l'application de la méthode dans des situations concrètes, où le temps et la performance doivent être minutieusement évalués.

Conclusion générale

L'optimisation de l'ordonnancement des tâches sur des serveurs hétérogènes est un enjeu majeur pour améliorer les performances des systèmes informatiques modernes. Ce mémoire propose une solution utilisant la recherche tabou, une méthode efficace pour résoudre des problèmes complexes.

Nous avons d'abord expliqué les principes de la recherche tabou, en décrivant ses mécanismes de mémoire, de génération de voisinage, et ses stratégies d'intensification et de diversification. Cela a permis de comprendre pourquoi cette méthode est souvent plus performante que les approches exactes, surtout en termes de temps de calcul et de capacité à s'adapter à des problèmes de grande taille.

Ensuite, nous avons modélisé le problème d'ordonnancement des commandes clients en simplifiant le Job Shop Problem (JSSP). Chaque commande est considérée comme une tâche à attribuer à un serveur, avec des coûts de traitement simulés. Cette simplification a facilité l'intégration de l'algorithme de recherche tabou, qui a été ajusté pour tenir compte des contraintes spécifiques, comme les délais, les capacités des serveurs et l'équilibre de charge.

L'application développée propose une interface simple pour que l'utilisateur puisse configurer des paramètres clés (nombre de tâches, de serveurs, d'itérations, taille de la liste tabou), lancer des simulations et analyser les résultats à l'aide de tableaux et de graphiques. Cela a permis de valider l'efficacité de l'approche et sa capacité à trouver des solutions proches de l'optimum.

Les résultats expérimentaux montrent que l'algorithme peut s'adapter à différentes configurations et fournir des solutions fiables en un temps raisonnable. Cela confirme l'utilité de la recherche tabou comme outil d'aide à la décision pour des problèmes d'ordonnancement dans des situations réelles.

Enfin, ce travail ouvre plusieurs pistes pour l'avenir, comme :

* L'intégration d'autres critères d'optimisation (coût énergétique, satisfaction client),

* L'évolution vers une version dynamique du problème (avec des commandes arrivant en temps réel),

* La comparaison avec d'autres méthodes (algorithmes génétiques, recuit simulé, etc.).

Ainsi, cette étude montre l'intérêt des approches métaheuristiques, et en particulier de la recherche tabou, pour résoudre des problèmes d'ordonnancement complexes avec un fort impact pratique.

Bibliogphie

[1] : Glover, F. (1989). *Future Paths for Integer Programming and Links to Artificial Intelligence*. Computers & Operations Research, 13(5), 533-549.

[2]: Hansen, P. (1986). *The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming*. Congress on Numerical Methods in Combinatorial Optimization, Capri.

[3]: Glover, F., & Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers.

[4]: Nicolas SIMONS « Optimisation des Architectures Numériques : Formulation du "Design Flow" sous forme d'un problème d'aide à la décision Multi-critères » ULB 2006

[5]: Johann Dréo — Patrick Siarry, « Métaheuristiques pour l'optimisation et auto organisation dans les systèmes biologiques », (LERISS,A 412), 2004

[6] : L .Thomas « Algorithmes génétiques et composites conducteurs », *Rapport de projet de fin d'études* 2006

[7] : Talbi, E.-G. (2009)**.**Metaheuristics: From Design to Implementation.Wiley.

[8] : Christophe Duhamel « Un Cadre Formel pour les Méthodes par Amélioration Itérative :Application à deux problèmes d'Optimisation dans les Réseaux – », Thèse de Doctorat 22

[9]:Duvivier D. « Etude de l'hybridation des méta-heuristique, Application à un problème d'ordonnancement de type Job-shop », thèse de doctorat, Université de littoral cote d'opale,LIL, calais 2000.

[10]: Driss Imen « Analyse D'un Système Job Shop Aspect ordonnancement», Thèse de Doctorat Université de Batna 2, 2016.

[11]: S. Pothiya, P. Tantaswadi, S. Runggeratigul, "Solving the Economic Dispatch ProblemUsing Multiple Tabu Search Algorithm", The first Conference of International Conference on Systems and Signals (ICSS) I-Shou University, Kaohsiung, Taiwan, April 28-29, 2005.

[12]: Gendreau, M., & Potvin, J.-Y. (2010). *Handbook of Metaheuristics*. Springer.

[13]: Pinedo, M. (2016). *Scheduling: Theory, Algorithms, and Systems* (5th ed.). Springer.

ملخـــص

يتناول هذا البحث تقييم الأداء في نظام جدولة يعمل بخوادم غير متماثلة.

ولحل هذه المشكلة المعقدة، تم اقتراح مقاربة تعتمد على خوارزمية البحث التابو(Tabu Search) ، وهي استدلالات عليا (Métaheuristique) تستخدم ذاكرة تكيّفية واستراتيجيات فعالة للاستكشاف.

تم نمذجة النظام وفق بيئة من نوع وظائف المتجر(Job Shop).

وتُظهر النتائج أن هذه الطريقة تُحسِّن بشكل كبير من كفاءة النظام، وجودة الجدولة، والأداء العام.

Abstract

This thesis addresses the performance evaluation of a scheduling system with non-identical servers.

To solve this complex problem, an approach based on Tabou Search is proposed.

This metaheuristic used adaptive memory and effective exploration strategies.

The system is designed using a Job Shop model. The results show that this method significantly improves efficiency, scheduling quality, and overall system performance.

Résumé

Ce mémoire traite de l'évaluation des performances dans un système d'ordonnancement à serveurs non identiques.

Pour résoudre ce problème complexe, une approche basée sur la recherche tabou (Tabu Search) est proposée.

Cette métaheuristique utilise une mémoire adaptative et des stratégies d'exploration efficaces.

Les résultats montrent que cette méthode améliore significativement l'efficacité, la qualité de l'ordonnancement et la performance globale du système.