الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التـعـليـم العـــالي والبـحث العـلـمـي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
جامعة تلمسان – أبي بكر بلقايد –
Université de Tlemcen – Aboubakr Belkaïd –
كلية التكنولوجيا
Faculté de Technologie
قسم الهندسة الصناعية
Département de Génie Industriel

Mémoire de Projet de fin d'Etudes

Présenté pour l'obtention du **diplôme** de :

Master en : Génie Industriel, **Spécialité :** Chaînes Logistiques

Étude comparative des performances des métaheuristiques pour minimiser le Makespan (Cmax) dans un atelier de type Flowshop

Présenté Par : Aimane SEDOUD

Soutenu le 23/06/2025 devant le jury composé de :

Présidente : Mme. Sihem KOULOUGHLI, Professeur, Université de Tlemcen
Examinatrice : Mme. Nassima KEDDARI, M.C.B, Université de Tlemcen
Examinatrice : Mme. Fatima Zohra BOUMEDIENE, M.C.B, Université de Tlemcen
Encadrant : Mr. Ahmed HASSAM, M.C.B, Université de Tlemcen

Année Universitaire 2024/2025

REMERCIEMENTS

Avec une profonde dévotion, je tiens à exprimer ma gratitude envers Allah le Tout-Puissant pour m'avoir accordé la santé, la patience et la volonté nécessaires à l'accomplissement de ce mémoire.

Je souhaite ensuite remercier sincèrement Mr. Ahmed HASSAM, mon encadrant, pour m'avoir accompagné tout au long de ce projet. Vos efforts, vos conseils et votre soutien m'ont énormément aidé. Grâce à votre encadrement et à votre disponibilité, j'ai pu avancer avec confiance et sérénité dans la réalisation de ce travail. Je vous suis profondément reconnaissant pour la qualité de votre accompagnement.

J'adresse également mes plus remerciements à Mr. Brahim ABBAD, qui m'a soutenu durant ces cinq années d'études universitaires. Votre aide, vos conseils et votre présence ont toujours été pour moi une source de motivation et de réconfort. Je vous remercie sincèrement pour votre patience, votre bienveillance et le temps que vous m'avez consacré. Votre soutien a été d'une grande importance dans mon parcours, et je vous en suis très reconnaissant.

DÉDICACE

Je rends grâce à Allah, Le Tout-Puissant, pour Sa guidance et Sa miséricorde qui m'ont soutenu dans ce parcours.

Ce mémoire est dédié à ma mère, dont l'amour, les prières et le soutien m'ont porté jusqu'ici, avec toute ma tendresse.

À mon père, merci pour tes conseils et ta confiance qui m'ont poussé à l'excellence.

À mes frères, Rami et Abderezzaq, ainsi qu'à mes sœurs, Rihab et Aridj, mes précieux complices, je dédie ce mémoire avec tout mon amour. Votre présence éclaire mon chemin et enrichit ma vie.

À ma grand-mère, Mama Kheira, ta douceur et tes prières m'ont fortifié.

À mes oncles maternels — Brahim, Habib, Moussa, Abdellah, Bilal, Hamza et Moho Lamine, merci pour votre soutien.

À mes collègues, notamment Taha.M, Houssem.C, Ibrahim.M, et notre contact d'urgence Imene.B Merci pour votre camaraderie et tous les bons moments partagés entre stress, rires et souvenirs mémorables.

À tous mes proches et amis, votre soutien a rendu ce mémoire possible, merci du fond du cœur.

LISTES DES ABRÉVIATIONS

Problèmes d'ordonnancement

FSSP Flow Shop Scheduling Problem (problème d'ordonnancement en atelier de flux)

FSP Flow Shop Problem (problème de flux)

PFSP Permutation Flow Shop Problem (problème de flux avec permutation)

FFS Flexible Flow Shop (atelier de flux flexible)

SPT Shortest Processing Time (temps de traitement le plus court)

LPT Longest Processing Time (temps de traitement le plus long)

WSPT Weighted Shortest Processing Time (temps de traitement pondéré le plus court)

EDD Earliest Due Date (date d'échéance la plus proche)

WIP Work In Progress (travaux en cours)

NEH Nawaz-Enscore-Ham (algorithme de Nawaz-Enscore-Ham)

C_{max} Makespan (temps d'exécution totale)

Complexité algorithmique

P Polynomial (polynomial)

NP Non-deterministic Polynomial (polynomial non déterministe)

NP-difficile NP-hard (NP-difficile)

NP-complet NP-complete (NP-complet)

IP Integer Programming (programmation en nombre entier)

Méthodes d'optimisation (métaheuristiques)

VNS Variable Neighborhood Search (recherche à voisinage variable)

GA Genetic Algorithm (algorithme génétique)

PSO Particle Swarm Optimization (optimisation par essaim de particules)

TS Tabu Search (recherche taboue)

SA Simulated Annealing (recuit simulé)

ACO Ant Colony Optimization (optimisation par colonie de fourmis)

DPSO Discrete Particle Swarm Optimization (optimisation par essaim particulaire discret)

Problèmes de routage

VRP Vehicle Routing Problem (problème de tournées de véhicules)

TSP Traveling Salesman Problem (problème du voyageur de commerce)

VSC Vehicle Scheduling (ordonnancement des véhicules)

Opérateurs de croisement (pour algorithmes génétiques)

PMX Partially Matched Crossover (croisement partiellement apparié)

CX Cycle Crossover (croisement par cycle)

PBX Position-Based Crossover (croisement basé sur la position)

ERX Edge Recombination Crossover (croisement par recombinaison d'arêtes)

OX Order Crossover (croisement par ordre)

Autres concepts spécifiques

PCB Printed Circuit Board (circuits imprimés)

LS Local Search (recherche locale)

Abréviations des tableaux

minC_{max} Minimum Makespan (C_{max} minimum)

maxC_{max} Maximum Makespan (C_{max} maximum)

avgC_{max} Average Makespan (C_{max} moyen)

runtime Runtime (temps d'exécution)

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GÉNÉRALE

Dans un contexte industriel caractérisé par une demande toujours plus exigeante en termes de productivité et de qualité, l'optimisation des systèmes de production représente un enjeu crucial. L'ordonnancement des tâches dans les ateliers de type flowshop constitue une problématique centrale, où chaque produit doit traverser une séquence fixe de machines selon un ordre rigoureux. La complexité de ce problème est accentuée par la gestion des capacités de stockage entre les machines et la nécessité de minimiser le temps total de production, connu sous le nom de makespan. Une gestion efficace de ce processus est indispensable pour accroître la performance industrielle tout en maîtrisant les coûts.

Compte tenu de la nature combinatoire et NP-difficile du problème d'ordonnancement flowshop, les méthodes exactes deviennent rapidement impraticables sur des instances de taille moyenne à grande. Les métaheuristiques se positionnent alors comme des approches particulièrement adaptées, capables d'explorer efficacement de vastes espaces de solutions et de converger vers des résultats satisfaisants en un temps raisonnable. Parmi les algorithmes retenus dans cette étude figurent l'algorithme génétique (GA), le recuit simulé (SA), l'optimisation par essaim de particules (PSO) et la recherche tabou (TS), avec une attention particulière portée à des stratégies d'hybridation combinant GA et SA d'une part, ainsi que PSO et TS d'autre part. Ces hybrides permettent de tirer parti des forces complémentaires des méthodes utilisées, améliorant la qualité des solutions et la robustesse des résultats.

Le mémoire présente ainsi une étude comparative de ces approches métaheuristiques appliquées à la minimisation du makespan dans un atelier flowshop. Après une introduction théorique des concepts d'ordonnancement et des techniques d'optimisation, chaque algorithme est modélisé et implémenté en Python. L'évaluation expérimentale repose sur des instances issues de la littérature, analysées selon des critères de performance tels que la qualité des solutions, le temps de calcul et la stabilité des résultats.

Cette démarche permet de mettre en lumière les avantages et les limites des différentes métaheuristiques et de leurs hybrides, fournissant ainsi des recommandations précieuses pour leur application dans des contextes industriels réels. Par cette contribution, ce travail vise à enrichir les connaissances dans le domaine de l'ordonnancement avancé tout en proposant des outils opérationnels pour l'optimisation des systèmes de production.

CHAPITRE 1

THÉORIE SUR L'UTILISATION DES MÉTAHEURISTIQUES DANS L'ORDONNANCEMENT DES ATELIERS FLOWSHOP

1.1 Introduction au problème flowshop

Le problème de flowshop est un concept clé dans le domaine de l'ordonnancement, particulièrement pertinent dans les environnements de fabrication où chaque tâche doit passer par une série d'opérations spécifiques. Dans un atelier flowshop, chaque produit suit un chemin prédéfini, ce qui signifie que toutes les tâches doivent être effectuées dans le même ordre sur une série de machines disposées en séquence. Cette configuration permet souvent une capacité de stockage entre les machines presque illimitée, surtout lorsque les produits sont petits et faciles à stocker.

Cependant, pour des produits plus volumineux, comme des téléviseurs, l'espace de stockage peut être restreint, entraînant des situations de blocage où une machine ne peut pas libérer un produit en raison d'un espace tampon plein. Dans un flowshop, les machines sont généralement organisées de manière à ce que chaque opération soit réalisée sur une machine spécifique. Les tâches sont divisées en opérations distinctes, et chaque opération doit être exécutée dans un ordre strict pour que le produit final soit achevé. Cette structure impose une séquence rigoureuse où chaque opération a un prédécesseur direct et un successeur direct, rendant l'ordonnancement crucial pour minimiser le temps total de production.

Un aspect important du problème des ateliers flowshop est la gestion des capacités de stockage entre les machines. Dans certains cas, la capacité de stockage peut être presque illimitée, ce qui permet d'accumuler des produits en cours de traitement. Cependant, lorsque cette capacité est limitée, cela peut entraîner des situations de blocage où une machine ne peut pas libérer une tâche en raison d'un espace tampon plein. Cela complique davantage l'ordonnancement et nécessite des stratégies efficaces pour éviter les goulots d'étranglement.

Ainsi, le problème de flowshop représente un défi majeur dans l'optimisation des systèmes de production. Il nécessite des approches algorithmiques adaptées pour gérer efficacement les contraintes d'ordonnancement, tout en maximisant l'efficacité et en réduisant le temps total nécessaire pour compléter toutes les tâches. L'étude approfondie de ce problème est essentielle pour améliorer la productivité et la rentabilité dans divers secteurs industriels [1][2][3].

1.1.1 Définition

Le flowshop désigne un modèle de production dans lequel les machines sont organisées en série. Les travaux passent d'une machine initiale à plusieurs machines intermédiaires, puis atteignent la machine finale avant d'être complétés. Chaque tâche d'un travail est divisée en plusieurs opérations, et chaque opération est effectuée sur une machine spécifique. Dans ce modèle, un travail est constitué de plusieurs opérations qui doivent être exécutées dans un ordre précis, chaque opération ayant un prédécesseur direct et un successeur direct, créant ainsi une séquence de production obligatoire pour chaque travail.

Dans un flowshop pur, chaque travail comporte autant d'opérations que de machines (m opérations pour m machines), chaque machine étant dédiée à une opération spécifique. Le modèle du flowshop peut se traduire par un problème complexe d'optimisation combinatoire, connu sous le nom de problème de planification de flowshop (FSSP), où l'objectif est généralement de minimiser des critères tels que le makespan (temps total de production) ou le flowtime total [4][1].

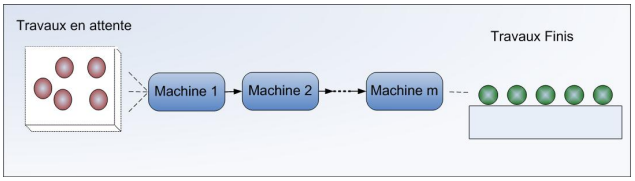

FIG. 1.1 : Représentation schématique d'un flowshop.[5]

1.1.2 Objectif principal

L'ordonnancement en flowshop vise principalement à déterminer une séquence optimale des tâches sur plusieurs machines afin de minimiser le makespan (Cmax), c'est-à-dire le temps d'achèvement maximal de l'ensemble des opérations. L'objectif est d'optimiser l'utilisation des ressources tout en respectant les contraintes de précédence et en minimisant les délais d'exécution.

Selon la complexité du problème, d'autres critères peuvent être pris en compte, notamment la réduction du temps moyen de séjour des tâches, la diminution de la variance des temps de finition pour équilibrer la charge de travail, ainsi que la minimisation du retard total afin de limiter les pénalités liées aux délais non respectés.

Le problème étant NP-complet, il est généralement résolu par des méthodes heuristiques et approchées, comme le recuit simulé, la recherche tabou et les algorithmes génétiques. Pour certains cas particuliers, notamment lorsque les temps de traitement sont identiques ou que la structure du problème suit une organisation en arbre (intree ou outtree), des algorithmes polynomiaux peuvent être appliqués, comme l'algorithme du chemin critique ou l'algorithme de Hu [3].

1.1.3 Types de flowshop

Permutational Flowshop

Permutation Flowshop Problem (PFSP) est un problème d'ordonnancement où un ensemble fini de n jobs doit être traité séquentiellement sur un ensemble fini de m machines. Chaque job suit le même ordre prédéfini de machines, et

tous les jobs sont traités dans une séquence identique sur chaque machine, maintenant ainsi la même permutation tout au long du processus.

Chaque job est composé de *m* opérations, où la k^{eme} opération du job J_i doit être traitée sur la machine M_k. pendant une durée fixe et ininterrompue. Aucun job ne peut être préempté, et chaque machine ne peut traiter qu'un seul job à la fois. L'objectif principal est généralement la minimisation du makespan ($C_{\max}$), représentant le temps total nécessaire pour effectuer toutes les tâches. Étant donné la nature combinatoire du PFSP, des approches heuristiques et métaheuristiques, telles que les algorithmes génétiques (GA), le variable neighborhood search (VNS) et des techniques d'optimisation hybrides, sont souvent employées pour obtenir des solutions proches de l'optimal de manière efficace. [4]

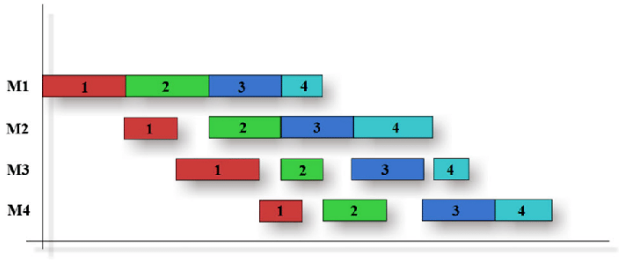

FIG. 1.2 : Ordonnancement d'un flowshop permutationnel.[6]

Flexible Flowshop

Flexible Flowshop (FFS) est un système de production organisé en plusieurs étapes de traitement successives, où chaque étape comprend plusieurs machines identiques fonctionnant en parallèle. Contrairement aux ateliers à flux classique avec une seule machine par étape, la présence de machines parallèles améliore l'efficacité et la flexibilité de la production. La planification dans un FFS consiste à attribuer de manière optimale les tâches aux machines tout en respectant la séquence de traitement requise. Ce type de système est largement utilisé dans des secteurs tels que la fabrication de semi-conducteurs, la production de contenants en verre, l'industrie du câble et l'assemblage de circuits imprimés (PCB), où l'automatisation et des dispositifs d'alimentation standardisés facilitent les opérations .[2]

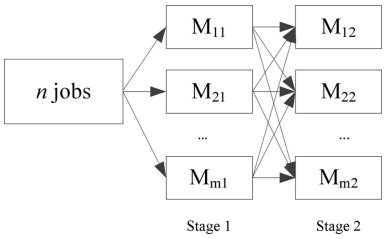

FIG. 1.3 : Ordonnancement d'un flowshop flexible.[7]

1.1.4 Importance du problème dans l'industrie manufacturière

Dans l'industrie manufacturière, la minimisation du makespan (C_{max}) est un objectif clé pour optimiser l'efficacité. Une réduction du makespan permet d'augmenter le taux de production, de diminuer les coûts et d'améliorer l'utilisation des ressources, des éléments essentiels pour maintenir la compétitivité.

Dans les Permutation Flow Shop Scheduling Problem (PFSP), où les jobs doivent être traités dans le même ordre sur plusieurs machines, la minimisation du makespan a un impact direct sur la performance globale de la production. Étant donné que le problème est NP-difficile, les méthodes exactes deviennent inapplicables pour des instances de grande taille. Par conséquent, les chercheurs et industriels se tournent vers des approches heuristiques et métaheuristiques comme les Genetic Algorithms (GA), Tabu Search (TA), Simulated Annealing (SA) et les techniques d'optimisation hybrides, afin d'obtenir des solutions proches de l'optimal de manière efficace.

Une stratégie efficace de minimisation du makespan permet aux entreprises de réduire les délais de production, d'accroître la flexibilité et d'améliorer le taux de satisfaction des commandes, des éléments clés pour des secteurs tels que l'automobile, l'électronique et la fabrication de semi-conducteurs. En intégrant des techniques d'optimisation avancées, les entreprises peuvent trouver un équilibre entre efficacité de production et rentabilité, assurant ainsi des opérations durables et compétitives.[4]

1.2 Complexité et défis

Le problème de la minimisation du makespan dans un système de production en flux est classé comme un problème NP-difficile, ce qui signifie qu'aucun algorithme en temps polynomial ne peut garantir une solution optimale, sauf si P = NP. Cette complexité provient de la nature combinatoire de l'ordonnancement, où le nombre de séquences de tâches possibles croît exponentiellement avec le nombre de tâches et de machines.

L'un des principaux défis réside dans la présence de contraintes de précédence et de temps de préparation dépendants de la séquence, ce qui complique davantage la recherche d'un emploi du temps optimal. Les méthodes exactes traditionnelles, telles que la programmation dynamique ou l'algorithme de branch and bound, deviennent rapidement impraticables pour les grandes instances en raison de la croissance exponentielle du temps de calcul.

Pour surmonter cette difficulté, des approches heuristiques et métaheuristiques, telles que les Algorithmes Génétiques, le Recuit Simulé et la Recherche Tabou, sont utilisées pour obtenir des solutions quasi-optimales dans des délais raisonnables. Cependant, ces méthodes présentent leurs propres défis, notamment le réglage des paramètres, les problèmes de convergence et l'équilibre entre exploration et exploitation afin d'éviter les optima locaux.

De plus, l'intégration des contraintes du monde réel, telles que les pannes de machines, la variabilité des temps de traitement et les limitations de ressources, accroît encore la complexité du problème. La modélisation efficace de ces contraintes tout en maintenant une efficacité computationnelle reste un défi de recherche majeur.

Malgré les progrès des techniques d'optimisation, obtenir un ordonnancement optimal ou quasi-optimal dans un temps raisonnable demeure une tâche ardue, soulignant la nécessité continue de développer des algorithmes améliorés et des approches hybrides combinant plusieurs métaheuristiques.[8]

1.2.1 Le problème est NP-difficile

En ordonnancement de la production, un problème est dit NP-difficile (NP-hard) lorsqu'il est au moins aussi complexe que tout problème appartenant à la classe NP, ce qui signifie que tout problème NP peut y être réduit en temps polynomial. Contrairement aux problèmes NP-complets, un problème NP-difficile ne nécessite pas que ses solutions puissent être vérifiées en temps polynomial.

Les problèmes d'ordonnancement, tels que l'ordonnancement en flow shop, job shop ou open shop, sont souvent NP-difficiles, car ils impliquent l'optimisation de critères tels que le makespan (temps d'achèvement total), le retard moyen, ou la charge équilibrée des machines. La résolution exacte de ces problèmes devient rapidement impraticable à mesure que la taille des instances augmente, rendant les approches de recherche exhaustive inefficaces. Toutefois, des algorithmes exacts améliorés, comme la programmation dynamique ou la branche-et-bound, permettent de résoudre certaines instances, bien que leur complexité reste super-polynomiale. Par ailleurs, le développement d'heuristiques et de métaheuristiques, telles que les algorithmes génétiques, la recherche tabou ou l'optimisation par colonies de fourmis, constitue une alternative efficace pour obtenir des solutions proches de l'optimum dans des délais raisonnables.[9]

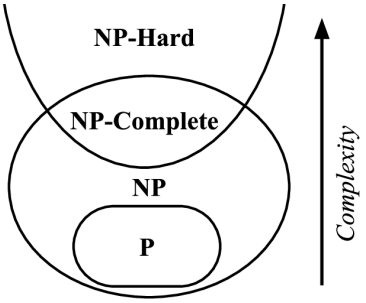

Fig. 1.4 : Classification des Problèmes en Fonction de Leur Complexité Algorithmique.[10]

1.2.2 Détail des difficultés liées à la recherche de solutions optimales

La résolution optimale du problème Flowshop est confrontée à plusieurs défis majeurs. Tout d'abord, la croissance exponentielle du nombre de séquences possibles entraîne une explosion combinatoire, ce qui rend les méthodes exactes inadaptées pour des instances de grande taille [11].

De plus, les contraintes d'ordonnancement, telles que la précédence des tâches et la disponibilité des machines, limitent l'espace de recherche et compliquent la détermination d'un ordonnancement optimal [12]. Enfin, en environnement industriel, les variations des temps d'exécution et des conditions de production rendent difficile l'application rigide des solutions obtenues par des approches exactes[13]. Face à ces obstacles, l'utilisation des métaheuristiques devient essentielle pour obtenir des solutions de qualité en un temps raisonnable [14], [15].

1.3 Résolution par Méthodes Classiques

Il existe plusieurs méthodes d'ordonnancement de production permettant d'optimiser l'affectation des ressources et la planification des tâches. Toutefois, ces méthodes peuvent rapidement être limitées par la complexité computationnelle, notamment pour les problèmes de grande taille. Pour surmonter ces contraintes, les métaheuristiques constituent

une alternative efficace, en explorant l'espace de recherche de manière approximative mais plus rapide.[16]

Méthodes exactes

Les méthodes exactes garantissent l'obtention d'une solution optimale et la preuve de son optimalité pour toute instance finie d'un problème donné. Parmi ces méthodes, la programmation en nombres entiers (IP) constitue une approche largement exploitée, avec des techniques telles que la séparation et évaluation (branch-and-bound), la séparation et coupes (branch-and-cut), ainsi que la programmation dynamique. [17]

Programmation linéaire

La programmation linéaire est une méthode d'optimisation mathématique permettant de résoudre des problèmes où l'on cherche à maximiser ou minimiser une fonction objective linéaire, tout en respectant un ensemble de contraintes également linéaires. Un problème de programmation linéaire se formule généralement sous la forme suivante :

$$\text{Maximiser (ou Minimiser) } Z = c_1x_1 + c_2x_2 + \cdots + c_nx_n \tag{1.1}$$

sous les contraintes :

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1 \tag{1.2}$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leq b_2 \tag{1.3}$$

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m \tag{1.4}$$

$$x_1, x_2, \ldots, x_n \geq 0 \tag{1.5}$$

où :

Z est la fonction objective à optimiser (maximisation du profit, minimisation des coûts, etc.),

$x_1, x_2 ... x_n$ sont les variables de décision,

$c_1, c_2 ... c_n$ représente les coefficients des contraintes,

a_{ij} sont les ressources ou limites imposées.

Cette approche est largement utilisée dans divers domaines tels que la gestion de la chaîne d'approvisionnement, la planification de production et l'ordonnancement, où elle permet de prendre des décisions optimales en fonction de ressources limitées.[18][19]

Méthodes basées sur la théorie des graphes

Les méthodes basées sur la théorie des graphes permettent de modéliser les problèmes d'ordonnancement en représentant les tâches, les ressources et leurs interdépendances sous forme de graphes orientés. Chaque sommet du graphe correspond à une opération, tandis que les arêtes indiquent les relations de précédence ou les contraintes entre les tâches.

Ces méthodes sont particulièrement adaptées aux environnements complexes et dynamiques comme la fabrication

en cloud, où la flexibilité et l'adaptabilité des ressources sont essentielles. Elles facilitent l'optimisation des décisions d'ordonnancement en prenant en compte les contraintes du système, tout en assurant une gestion efficace des ressources disponibles.[20]

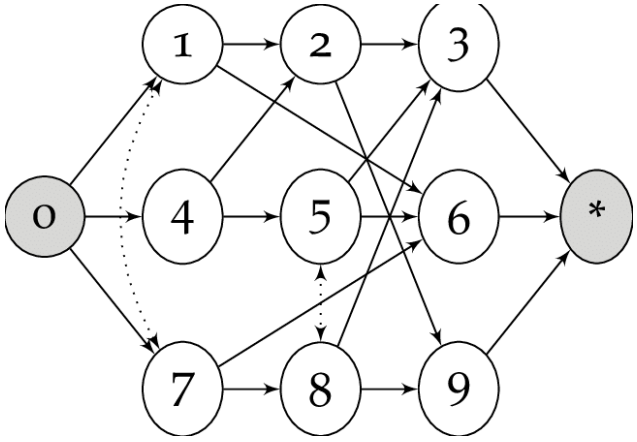

FIG. 1.5 : Ordonnancemnt basé sur la théorie des graphes.[21]

Règles de priorité

Les règles de priorité sont des critères décisionnels largement utilisés en ordonnancement pour déterminer l'ordre de traitement des tâches. Elles visent à optimiser divers objectifs, tels que la réduction du temps de séjour moyen, la diminution du retard, le contrôle des niveaux de travail en cours (WIP), ainsi que la réduction du temps total d'exécution (makespan,C_{max}).[22]

SPT (Shortest Processing Time) : Cette règle donne la priorité aux tâches ayant le plus court temps de traitement en premier. Elle est efficace pour minimiser le temps moyen de séjour des tâches dans le système.[12]

LPT (Longest Processing Time) : Cette règle ordonne les tâches du plus long au plus court temps de traitement. Elle est souvent utilisée dans les systèmes où l'équilibrage des charges entre les machines est important.[23]

WSPT (Weighted Shortest Processing Time) : Variante de SPT qui pondère les tâches en fonction d'un poids W_i , représentant par exemple une priorité ou une pénalité pour retard. L'ordre de traitement est donné par le ratio P_i/W_i , où P_i est le temps de traitement.[24]

EDD (Earliest Due Date) : Cette règle classe les tâches en fonction de leur date d'échéance, la plus proche étant traitée en premier. Elle est efficace pour minimiser le retard maximum.[25]

LS (Least Slack Time First) : Priorise les tâches avec le moins de marge de manœuvre

($Slack = Date\ D'echeance$ - $temps\ de\ traitement\ restant$).[26]

Branch-and-Bound

La méthode Branch and Bound est une approche largement employée pour résoudre des problèmes d'optimisation combinatoire. Elle repose sur deux mécanismes fondamentaux : la décomposition en sous-problèmes et l'estimation de bornes. Le premier consiste à diviser un problème complexe en plusieurs sous-problèmes plus simples, tandis que le second vise à établir une borne inférieure pour la meilleure solution possible de chaque sous-problème, réduisant ainsi l'espace de recherche à explorer.

Le processus de décomposition segmente le problème initial en plusieurs sous-problèmes qui respectent trois conditions essentielles :

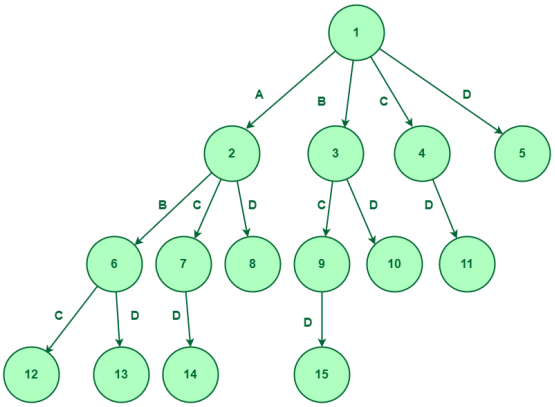

FIG. 1.6 : diagramme Branch and Bound.[27]

Ils sont exclusifs les uns par rapport aux autres et, ensemble, couvrent l'ensemble du problème de départ. Ils représentent des solutions partielles du problème global. Ils sont plus restreints en taille que le problème initial. Chaque sous-problème peut être divisé à son tour, formant ainsi une structure arborescente. Par exemple, dans un problème d'ordonnancement sur une machine unique avec n tâches, le problème initial $P(0)$ peut être divisé en n sous-problèmes $P(1)$, $P(2)$, ..., $P(n)$ en assignant une tâche fixe en dernière position. À chaque niveau de l'arbre, de nouvelles tâches sont placées progressivement, et si l'énumération se poursuit jusqu'au bout, on obtiendrait $n!$ solutions distinctes. Toutefois, une telle approche devient impraticable pour des problèmes de grande envergure en raison de sa complexité exponentielle.

Pour éviter une exploration exhaustive, l'évaluation des bornes intervient en identifiant et éliminant les branches non prometteuses. Plus précisément, une borne inférieure est calculée pour chaque sous-problème. Si une solution complète est trouvée avec une valeur Z pour la fonction objectif, et qu'un sous-problème possède une borne inférieure b supérieure à $Z (b > Z)$, alors ce sous-problème ainsi que ses dérivés peuvent être ignorés. Ce principe, appelé élagage, réduit significativement l'arbre de recherche et améliore l'efficacité du processus.

Afin d'orienter la recherche, une solution intermédiaire (ou trial solution) peut être utilisée. Cette solution réalisable complète peut être obtenue par une approche heuristique ou en explorant rapidement certaines branches de l'arbre.

L'algorithme Branch and Bound maintient une liste active des sous-problèmes à explorer et sélectionne en priorité celui ayant la borne inférieure la plus faible. Le processus se poursuit jusqu'à ce qu'aucune solution possible ne puisse surpasser la meilleure solution identifiée.

Ce cadre méthodologique est particulièrement utile dans les problèmes d'ordonnancement, de séquencement et d'optimisation combinatoire, car il permet de trouver des solutions optimales tout en évitant une recherche exhaustive, ce qui en fait une approche efficace et rigoureuse.[1]

1.3.1 Limites pour des instances complexes

L'optimisation des problèmes d'ordonnancement de type Flowshop repose souvent sur des approches exactes telles que la programmation linéaire ou les méthodes arborescentes comme Branch-and-Bound. Bien que ces techniques garantissent des solutions optimales, elles deviennent inefficaces lorsque la taille des instances augmente considérablement. Cette limitation est due à la complexité combinatoire du problème, qui engendre une explosion exponentielle du nombre de configurations possibles à explorer [12]

Taille

La difficulté principale des méthodes exactes réside dans la croissance exponentielle du nombre de solutions possibles en fonction du nombre de tâches et de machines. Pour un problème comportant n tâches à répartir sur m machines, l'espace de recherche est de l'ordre de $(n!)^m$, ce qui rend la résolution exacte impraticable au-delà d'une certaine échelle.[11]Cette contrainte de dimensionnement explique pourquoi les instances de grande taille nécessitent des approches heuristiques et métaheuristiques pour produire des solutions de qualité en un temps raisonnable.

Temps de calcul

L'un des principaux défis des méthodes exactes réside dans le temps nécessaire pour explorer l'ensemble des solutions possibles. Pour des instances complexes, la résolution peut nécessiter plusieurs heures, voire plusieurs jours, ce qui est peu compatible avec les exigences opérationnelles de l'industrie, où des décisions doivent être prises rapidement.[13]Cette contrainte temporelle motive le recours aux métaheuristiques, qui permettent d'obtenir des solutions acceptables en une fraction du temps requis par les approches classiques.

1.3.2 Avantages des métaheuristiques

Face aux limites des méthodes exactes, les métaheuristiques se sont imposées comme une alternative efficace pour l'optimisation des problèmes complexes. Elles permettent d'obtenir des solutions proches de l'optimalité en un temps réduit, tout en conservant une flexibilité d'adaptation à divers contextes industriels[14]

Flexibilité

L'un des atouts majeurs des métaheuristiques réside dans leur caractère générique. Contrairement aux méthodes exactes, qui nécessitent une formulation spécifique pour chaque problème, les métaheuristiques peuvent être adaptées à une large gamme de problèmes d'optimisation, avec seulement quelques ajustements de paramètres.[28] Par exemple, un algorithme génétique peut être appliqué aussi bien à un problème Flowshop qu'à un problème de routage en logistique, moyennant une adaptation des fonctions d'évaluation et des opérateurs de recherche.

Efficacité

Les métaheuristiques offrent un compromis entre la qualité des solutions et le temps de calcul. Bien qu'elles ne garantissent pas l'optimalité, elles parviennent à fournir des résultats compétitifs en des temps bien inférieurs à ceux requis par les approches exactes. Des études ont démontré que des méthodes comme la recherche tabou ou le recuit simulé permettent d'atteindre des solutions proches de l'optimalité avec un temps de calcul maîtrisé[29]

Adaptabilité

Les métaheuristiques présentent également l'avantage d'être modulables et combinables entre elles pour améliorer leurs performances. Par exemple, l'optimisation par colonies de fourmis (ACO) peut être couplée à un algorithme génétique afin d'explorer plus efficacement l'espace de recherche et d'éviter la stagnation dans des optima locaux[30].

Cette capacité d'adaptation leur permet d'être utilisées pour une large variété de problèmes industriels.

1.4 Travaux antérieurs

L'ordonnancement des tâches en atelier de type Flowshop est un problème d'optimisation combinatoire qui a fait l'objet de nombreuses recherches, notamment avec l'émergence des métaheuristiques comme alternative aux méthodes exactes. Cette section présente un aperçu des études marquantes sur l'application des métaheuristiques aux problèmes de Flowshop et propose une synthèse des résultats obtenus.

1.4.1 Études marquantes sur l'utilisation des métaheuristiques pour les Flowshop

Les métaheuristiques ont été largement étudiées pour résoudre le problème Flowshop, notamment en raison de leur capacité à fournir des solutions de bonne qualité en un temps raisonnable. Parmi les travaux les plus notables, Nawaz, Enscore et Ham (1983)[31] ont introduit la règle NEH, qui reste l'une des heuristiques les plus performantes pour l'ordonnancement Flowshop, en fournissant une solution initiale efficace pour de nombreuses métaheuristiques. Leur approche a inspiré des améliorations basées sur des stratégies d'exploration et d'intensification, notamment en combinaison avec des algorithmes comme la recherche tabou (Taillard, 1990)[32] et le recuit simulé (van Laarhoven et al., 1992)[33].

L'algorithme génétique (Genetic Algorithm, GA) a été appliqué avec succès au problème Flowshop par Ruiz et Stützle (2007)[15], qui ont démontré que des opérateurs de croisement spécifiques permettent d'améliorer la qualité des solutions. De même, Liao et al. (2012)[34] ont développé une variante du recuit simulé intégrant une adaptation dynamique de la température, ce qui a permis d'améliorer la robustesse des solutions obtenues.

L'optimisation par essaims particulaires (Particle Swarm Optimization, PSO) a également été explorée pour ce problème. Zhang et al. (2007)[35] ont montré que l'utilisation d'une approche hybride combinant PSO et une recherche locale permet d'atteindre des solutions compétitives par rapport aux algorithmes évolutionnaires classiques. De plus, l'optimisation par colonies de fourmis (Ant Colony Optimization, ACO) a été étudiée par Rajendran et Ziegler (2004)[36], qui ont démontré son efficacité pour des instances de taille moyenne, bien que son temps de convergence puisse être un facteur limitant pour les instances de grande échelle.

L'intégration des métaheuristiques a également suscité un intérêt croissant. Par exemple, Zobolas, Tarantilis et Ioannou (2009)[37] ont développé une approche hybride combinant la recherche tabou et l'algorithme de recuit simulé, montrant qu'une combinaison judicieuse des forces de plusieurs méthodes permet d'améliorer la qualité des solutions et de réduire le temps de calcul.

1.4.2 Conclusion et synthèse des résultats existants

Les études sur les métaheuristiques appliquées au Flowshop mettent en évidence plusieurs tendances. Tout d'abord, les algorithmes évolutifs comme les algorithmes génétiques et l'optimisation par essaims particulaires offrent des solutions de qualité grâce à leur capacité d'exploration globale, mais peuvent nécessiter des mécanismes d'intensification pour affiner les solutions finales [15], [35].

Ensuite, les approches basées sur la recherche locale, telles que la recherche tabou et le recuit simulé, sont efficaces pour affiner les solutions existantes, mais peuvent souffrir d'un piégeage dans des optima locaux si les paramètres ne sont pas bien ajustés[32], [33]. C'est pourquoi des stratégies hybrides combinant exploration et intensification se sont révélées particulièrement prometteuses[37].

Un autre constat majeur est que les performances des métaheuristiques dépendent fortement des paramètres utilisés. Par exemple, les paramètres de refroidissement dans le recuit simulé ou les coefficients d'inertie dans PSO influencent considérablement la qualité des solutions obtenues[34][35]. L'optimisation adaptative des paramètres est donc une direction de recherche clé pour améliorer encore l'efficacité des algorithmes [36].

Enfin, les approches hybrides, combinant plusieurs métaheuristiques ou intégrant des méthodes exactes dans une première phase de recherche, ont montré leur potentiel pour améliorer la robustesse et la convergence des solutions[4], [15]. Ces résultats suggèrent que l'avenir des recherches sur l'ordonnancement Flowshop réside dans l'exploration d'approches adaptatives et hybrides, exploitant les forces complémentaires des différentes métaheuristiques.

CHAPITRE 2

ÉTUDE DÉTAILLÉE SUR LES MÉTAHEURISTIQUES ÉTUDIÉES

2.1 Introduction

L'optimisation des systèmes de production constitue un enjeu majeur dans l'industrie, notamment pour l'ordonnancement des tâches dans les ateliers de type Flowshop.

La minimisation du Makespan est un objectif clé, car elle permet d'améliorer l'efficacité globale, de réduire les coûts opérationnels et d'optimiser l'utilisation des ressources disponibles. Cependant, en raison de la complexité combinatoire de ce problème, il est nécessaire d'adopter des approches d'optimisation performantes capables de fournir des solutions de haute qualité dans un temps raisonnable [38].

Les métaheuristiques se sont imposées comme des outils incontournables dans ce domaine. Elles reposent sur des principes d'exploration et d'exploitation de l'espace des solutions pour converger vers des configurations optimales. Ces algorithmes sont largement utilisés pour résoudre des problèmes d'ordonnancement en raison de leur flexibilité, de leur capacité d'adaptation et de leur efficacité sur des instances de grande taille [39].

Dans ce chapitre, nous nous intéressons à cinq métaheuristiques majeures, qui ont démontré leur pertinence pour l'optimisation des problèmes de type Flowshop :

L'algorithme génétique (GA) : inspiré de la sélection naturelle, il repose sur des mécanismes de croisement et de mutation pour améliorer progressivement la qualité des solutions. L'optimisation par essaim de particules

Optimisation par essaim de particules (PSO) : basée sur la coopération entre particules, elle simule le comportement collectif d'un groupe d'individus pour rechercher un optimum.

Recherche Tabou (TS) : cette approche repose sur une mémoire adaptative pour éviter les cycles et améliorer la diversité des solutions explorées.

Recuit Simulé (SA) : inspiré des principes thermodynamiques du refroidissement des métaux, il permet d'échapper aux optima locaux en acceptant temporairement des solutions sous-optimales. Ces métaheuristiques seront décrites en détail dans les sections suivantes, en mettant en évidence leurs principes fondamentaux, leurs mécanismes d'exploration et leurs applications dans le cadre du Flowshop Scheduling Problem.

2.2 Tableau Comparatif des Métaheuristiques utilisées

Après avoir mené une brève étude sur les différentes métaheuristiques sélectionnées dans le cadre de ce travail, nous proposons dans le tableau suivant une comparaison synthétique de leurs principales caractéristiques. Cette comparaison vise à mettre en évidence leurs mécanismes d'exploration, leurs paramètres fondamentaux, ainsi que leurs avantages et limitations dans le contexte de l'ordonnancement en flow shop. Elle permet ainsi d'obtenir une vision globale des atouts de chaque méthode avant leur mise en œuvre pratique.

Critères	**GA**	**PSO**	**TS**	**SA**
Type de Recherche	Populationnelle	Populationnelle	Locale	Locale
Taille de la Population	Grande	Moyenne	N/A	N/A
Exploration vs Exploitation	Équilibrée	Exploration	Exploitation	Exploitation
Temps de Calcul	Élevé	Moyen	Faible	Moyen
Robustesse	Élevée	Moyenne	Élevée	Moyenne
Facilité de mise en œuvre	Moyenne	Facile	Moyenne	Moyenne

TAB. 2.1 : Comparaison des différentes métaheuristiques utilisées selon plusieurs critères

2.3 Présentation des algorithmes

2.3.1 L'algorithme génétique (GA)

Les algorithmes génétiques (GA) font partie des métaheuristiques inspirées des processus de sélection naturelle introduits par Darwin. Ils sont largement utilisés pour résoudre des problèmes d'optimisation combinatoire, notamment les problèmes d'ordonnancement en environnement Flowshop. Le principe fondamental du GA repose sur la création, l'évaluation et l'évolution d'une population de solutions candidates en appliquant des opérateurs inspirés de la biologie évolutive, tels que la sélection, le croisement et la mutation (Goldberg, 1989).

Processus

L'algorithme génétique (GA) est une méthode d'optimisation inspirée du processus de sélection naturelle, où une population de solutions évolue au fil des générations en appliquant des principes biologiques artificiels tels que la sélection, le croisement et la mutation [40]. Chaque individu de la population représente une solution potentielle et est évalué selon une fonction de fitness, qui mesure sa qualité par rapport à l'objectif du problème. Les individus les plus performants sont favorisés dans le processus de reproduction afin d'améliorer progressivement la population au cours

des itérations[41]. Le croisement permet de combiner les caractéristiques de plusieurs individus, tandis que la mutation introduit des variations aléatoires afin d'éviter la stagnation dans des optima locaux et d'assurer une exploration efficace de l'espace de recherche [42]. Ce processus itératif se poursuit jusqu'à ce qu'un critère d'arrêt soit atteint, tel qu'un nombre maximal d'itérations ou l'absence d'amélioration significative de la fitness. Grâce à sa flexibilité et son efficacité dans l'exploration des solutions, le GA est largement utilisé pour des problèmes complexes en optimisation combinatoire, notamment dans le domaine de l'ordonnancement et de la logistique [14].

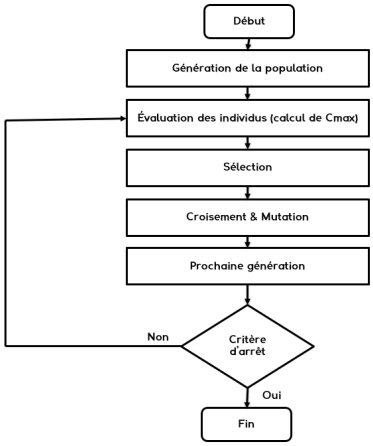

FIG. 2.1 : processus de l'algorithme génétique.[43]

Étapes clés

L'algorithme génétique (GA) suit un processus itératif reposant sur plusieurs étapes essentielles qui assurent l'évolution progressive des solutions vers un optimum. Ces étapes comprennent l'initialisation, la sélection, le croisement, la mutation et le remplacement. Chaque phase joue un rôle fondamental dans l'équilibre entre l'exploitation des solutions prometteuses et l'exploration de nouvelles configurations[42].

1. Initialisation de la Population La première étape consiste à générer une population initiale composée de N individus, chacun représentant une solution candidate au problème d'optimisation. Ces individus sont généralement codés sous forme de chaînes binaires, de permutations ou de vecteurs réels, selon la nature du problème [40] . L'initialisation peut être réalisée de manière aléatoire afin de garantir une diversité génétique suffisante ou bien être guidée par des heuristiques pour améliorer la qualité des solutions initiales.

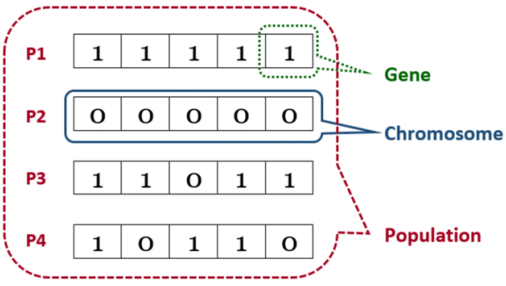

FIG. 2.2 : l'étape d'initialisation dans l'algorithme génétique.[44]

2. Évaluation et Fonction de Fitness Chaque individu est évalué à l'aide d'une fonction de fitness, qui mesure sa qualité par rapport à l'objectif du problème. Dans un problème d'ordonnancement Flowshop, par exemple, la fitness peut être définie par la minimisation du makespan (temps total d'exécution) [14]. Cette évaluation permet de sélectionner les individus les plus performants pour les étapes suivantes.

3. Sélection des Parents L'étape de sélection vise à choisir les individus les plus adaptés pour participer à la reproduction. Plusieurs méthodes de sélection sont couramment utilisées [41] :

Roulette Wheel Selection : La probabilité de sélection est proportionnelle à la valeur de fitness de l'individu.

Tournoi : Un groupe aléatoire d'individus est formé, et le meilleur du groupe est sélectionné.

Sélection élitiste : Une fraction des meilleurs individus est conservée d'une génération à l'autre pour garantir une amélioration continue des solutions.

4. Croisement (Crossover) Le croisement permet de combiner les caractéristiques de deux parents pour générer une nouvelle descendance, favorisant ainsi l'exploration de l'espace de recherche. Plusieurs opérateurs de croisement existent selon le type de représentation des solutions [42] :

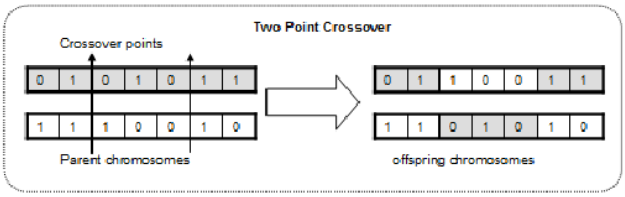

FIG. 2.3 : l'étape de croisement dans l'algorithme génétique.[44]

One-point crossover : Une coupure est réalisée à un point aléatoire du chromosome, et les segments sont échangés entre les parents.

Two-point crossover : Deux points de coupure sont définis, ce qui permet un mélange plus riche des gènes.

Order Crossover (OX) : Spécifique aux problèmes de permutation, il conserve la structure des séquences.

5. Mutation La mutation introduit une variation aléatoire dans la population pour éviter la convergence prématurée vers un optimum local. Elle consiste à modifier un ou plusieurs gènes d'un individu avec une probabilité faible. Dans un problème d'ordonnancement, une mutation peut consister à échanger deux tâches dans une séquence[14] .

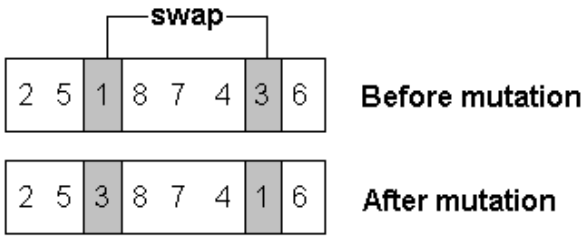

FIG. 2.4 : l'étape de mutation dans l'algorithme génétique.[44]

6. Remplacement et Critère d'Arrêt Une nouvelle population est formée en sélectionnant les meilleurs individus parmi les parents et les enfants. Ce processus est répété jusqu'à l'atteinte d'un critère d'arrêt, tel qu'un nombre maximal d'itérations ou la stabilisation de la fitness. Un bon équilibre entre exploitation et exploration est crucial pour garantir la convergence vers une solution optimale [40].

Domaine d'application

- Optimisation des chaînes d'approvisionnement.
- Localisation des centres logistiques (p-médian, p-center).
- Optimisation des portefeuilles financiers.
- Séquencement des tâches en production et logistique.

2.3.2 Optimisation par essaim de particules (PSO)

Processus

L'optimisation par essaim de particules (PSO) est une métaheuristique inspirée du comportement collectif des essaims d'oiseaux et des bancs de poissons, introduite par Kennedy et Eberhart (1995) [45]. Dans ce processus, une population de particules représentant des solutions potentielles évolue dans l'espace de recherche en ajustant dynamiquement leurs positions en fonction de leurs expériences individuelles et collectives. Chaque particule est caractérisée par une position, une vitesse et une mémoire de sa meilleure position trouvée (pbest), ainsi que de la meilleure position globale identifiée par l'ensemble de l'essaim (gbest) [46]. À chaque itération, la vitesse et la position des particules sont mises à jour en fonction d'un équilibre entre exploration et exploitation, permettant une convergence progressive vers une solution optimale. L'algorithme s'arrête lorsqu'un critère prédéfini est atteint, tel qu'un nombre maximal d'itérations ou une stabilisation des solutions [47]. Grâce à sa simplicité et son efficacité, PSO est largement utilisé pour résoudre des problèmes d'optimisation continue et combinatoire dans divers domaines, y compris l'ordonnancement et la logistique [48].

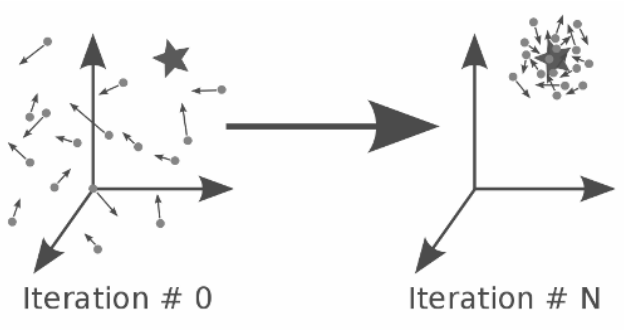

FIG. 2.5 : Processus de l'algorithme PSO.[49]

Mécanismes de l'algorithme PSO

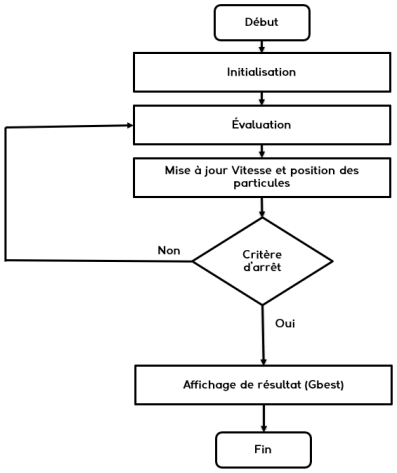

FIG. 2.6 : Diagramme des étapes de l'algorithme PSO.[50]

L'algorithme PSO repose sur une population de N particules évoluant dans un espace de recherche dimensionnel D. Chaque particule i possède :

- Une position $Xi = (x_{i1},x_{i2},...x_{iD})$: représente une solution candidate.
- Une vitesse $V_i = (v_{i1},v_{i2},...v_{iD})$: contrôle l'évolution de la position.
- Un historique de la meilleure solution personnelle atteinte $pBest_i$
- Un historique de la meilleure solution globale $gBest$ trouvée par l'ensemble des particules.

Initialisation

L'algorithme commence par initialiser aléatoirement la position et la vitesse de chaque particule dans l'espace de recherche. Les paramètres clés sont :

- Le nombre de particules N
- Les limites de l'espace de recherche (X_{min},X_{max})
- Les limites de vitesse (V_{min},V_{max})
- Les coefficients d'accélération c_1,c_2
- Le facteur d'inertie w

Chaque particule est évaluée à l'aide d'une fonction objectif spécifique. Dans le cas du problème Flowshop, cette fonction est le Makespan C_{max} .

Mise à jour des vitesses et des positions

Le déplacement d'une particule est guidé par la meilleure solution qu'elle a trouvée ($pBest$) et la meilleure solution trouvée par l'essaim ($gBest$). À chaque itération, la vitesse et la position sont mises à jour selon les équations suivantes :

$$V_i^{t+1} = w \cdot V_i^t + c_1 \cdot r_1 \cdot (pBest_i - X_i^t) + c_2 \cdot r_2 \cdot (gBest - X_i^t)$$

$$X_i^{t+1} = X_i^t + V_i^{t+1}$$

où :

w : facteur d'inertie qui contrôle l'influence de la vitesse précédente.

c_1,c_2 : coefficients d'accélération qui pondèrent respectivement l'influence de la mémoire individuelle et de l'expérience collective.

r_1,r_2 : variables aléatoires uniformes [0,1] introduisant une composante stochastique.

Interprétation des termes :

- $w.V_i^t$: Maintient un équilibre entre exploration et exploitation.
- $c_1.r_1.(pBest_i - Xi^t)$: Tendance de la particule à retourner à sa meilleure position.
- $c_2.r_2.(gBest - Xi^t)$: Attirance vers la meilleure position trouvée par l'essaim.

Gestion des paramètres

Le facteur d'inertie w Le paramètre w est crucial pour l'efficacité du PSO :

- Si w est élevé (≈0.9), l'algorithme favorise l'exploration de nouvelles solutions.
- Si w est faible (≈0.4), il privilégie l'exploitation autour des solutions prometteuses.

Les coefficients d'accélération c_1 et c_2 :

- c_1 détermine l'importance de l'apprentissage individuel.
- c_2 régule l'influence de l'expérience collective.

Typiquement, c_1 = c_2 = 2 pour garantir un équilibre.

Application du PSO au Flowshop

Dans un problème Flowshop, chaque particule représente une séquence ordonnée des tâches à exécuter sur plusieurs machines. L'objectif est de minimiser le Makespan.

Représentation des solutions Une solution peut être représentée par un vecteur d'entiers où chaque valeur correspond à un job à exécuter, par exemple :

$$X_i\text{=[3,1,2,5,4]}$$

où 3 signifie que le job 3 est exécuté en premier, suivi de 1, etc.

Fonction d'évaluation La fonction objectif est le Makespan $C_{\max} = \max\{C_{ij}\}$

où C_{ij} est le temps d'achèvement de la tâche j sur la machine i.

Adaptation des mouvements

Dans le cas du Flowshop, la mise à jour des positions doit être adaptée car les permutations ne suivent pas une logique vectorielle. On utilise :

- PSO Discret (DPSO) basé sur unopérateur de permutation.
- Un codage par permutations où les positions sont ajustées via des croisements inspirés des algorithmes génétiques.

Domaine d'application

- Optimisation des réseaux de neurones.
- l'apprentissage automatique.
- Optimisation des systèmes énergétiques et des réseaux électriques.
- Planification des horaires et allocation des ressources.
- Conception de structures et optimisation des paramètres en ingénierie.

2.3.3 Recherche Tabou (TS)

La recherche taboue (Tabu Search, TS) est une métaheuristique d'optimisation combinatoire introduite par Fred Glover en 1986 [51]. Elle repose sur une recherche locale améliorée qui intègre une mémoire adaptative pour éviter les cycles et améliorer l'exploration de l'espace de recherche. Contrairement aux méthodes heuristiques classiques qui risquent de stagner dans des optima locaux, la TS permet d'élargir la zone d'exploration en interdisant temporairement certains mouvements grâce à une liste taboue [52].

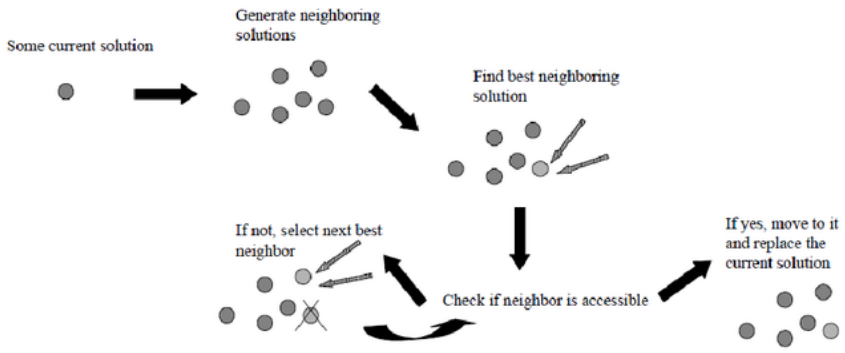

FIG. 2.7 : Les étapes de base d'une recherche tabou.[53]

Principe général

L'algorithme TS suit une approche itérative qui améliore une solution initiale en explorant son voisinage et en évitant les retours inutiles. L'élément clé est la liste taboue, qui stocke les solutions ou les mouvements récents et empêche leur réutilisation immédiate. Cette approche permet d'éviter la convergence prématurée vers des solutions sous-optimales et favorise la diversification de la recherche [54].

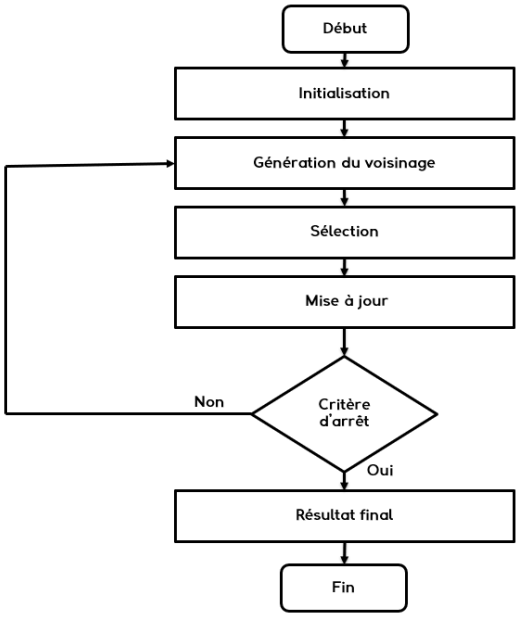

Fig. 2.8 : Diagramme de l'algorithme de Recherche Tabou.[55]

Processus et Étapes Clés

Initialisation

- Génération d'une solution initiale par une heuristique ou de manière aléatoire.
- Définition des paramètres, notamment la taille de la liste taboue et les critères d'arrêt.

Recherche locale améliorée

- Exploration du voisinage de la solution actuelle pour identifier une solution voisine améliorante.
- Sélection de la meilleure solution voisine, même si elle détériore temporairement la solution globale, afin d'éviter les minima locauxv [52].

Gestion de la mémoire taboue

- Ajout des mouvements récemment explorés à la liste taboue.

- Suppression des entrées les plus anciennes pour éviter un stockage excessif d'interdictions.

Critère d'aspiration

- Acceptation d'une solution taboue si elle offre une amélioration significative par rapport à la meilleure solution connue.

Critère d'arrêt

- Nombre maximal d'itérations atteint.
- Absence d'amélioration sur un certain nombre d'itérations consécutives.

Domaine d'application

- Problèmes de tournées de véhicules (VRP).
- Problèmes d'ordonnancement (flow shop, job shop).
- Planification des tâches et des ressources.
- Optimisation des réseaux (énergie, télécoms, transport).

2.3.4 Recuit Simulé (SA)

Le Recuit Simulé (Simulated Annealing - SA) est une métaheuristique stochastique inspirée d'un processus physique utilisé en métallurgie, appelé recuit. Ce processus consiste à chauffer un matériau à haute température, puis à le refroidir lentement afin de minimiser ses défauts et d'atteindre un état stable de faible énergie [56] .

En optimisation combinatoire, cette approche est transposée pour explorer un espace de solutions et éviter les optima locaux en acceptant temporairement des solutions moins bonnes selon une probabilité décroissante.[57] Cette propriété rend le SA particulièrement efficace pour résoudre des problèmes NP-difficiles, notamment les problèmes d'ordonnancement comme le Flowshop Scheduling Problem (FSP)[58].

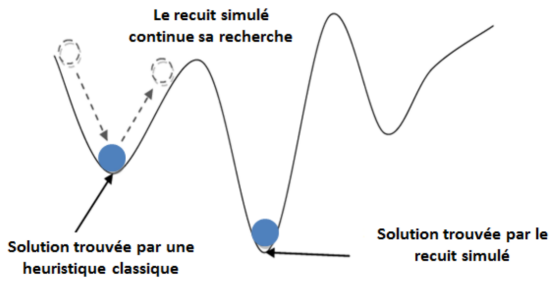

FIG. 2.9 : Performance du recuit simulé face à une heuristique classique.[59]

Principe Fondamental et Modèle Mathématique

L'algorithme du Recuit Simulé repose sur l'analogie entre la minimisation d'une fonction objectif et la minimisation de l'énergie d'un système thermodynamique. Il suit un mécanisme d'évolution basé sur la règle de **Metropolis** [60], qui définit la probabilité d'accepter une solution sous-optimale comme suit :

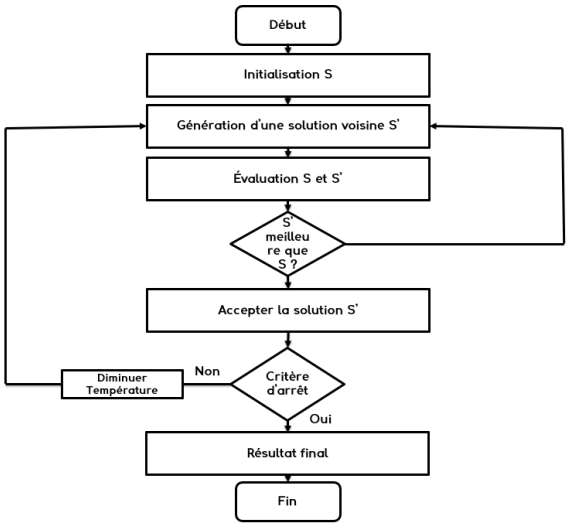

FIG. 2.10 : Étapes de l'algorithme de recuit simulé.[59]

$$P = e^{\frac{-(C(S')-C(S))}{T}} \tag{2.1}$$

où :

- $C(S)$ est la valeur de la fonction objectif de la solution courante S.
- $C(S')$ est la valeur de la fonction objectif de la solution candidate S'.
- T est la température actuelle, qui diminue progressivement.

Cette probabilité permet d'accepter des solutions moins bonnes avec une probabilité plus élevée lorsque la température est élevée, favorisant ainsi l'exploration de l'espace des solutions [61].

Algorithme du Recuit Simulé

L'algorithme SA suit une approche itérative où la température diminue progressivement selon une **fonction de refroidissement**. Son déroulement peut être formalisé en plusieurs étapes **suman2006survey** :

Étape 1 : Initialisation

- Générer une solution initiale S (ex. : une séquence de tâches pour un problème Flowshop).
- Fixer une température initiale élevée T_0.
- Déterminer un critère d'arrêt (nombre maximal d'itérations ou température minimale T_{min}).

Étape 2 : Génération d'une Solution Voisine

- Une nouvelle solution S' est générée par un mouvement dans l'espace des solutions (ex. : échange de deux tâches dans un ordonnancement).

Étape 3 : Évaluation et Acceptation de la Solution

- Calculer la différence de coût $\Delta C = C(S') - C(S)$.
- Si S' est meilleure ($\Delta C < 0$), elle est acceptée.
- Sinon, elle est acceptée avec une probabilité $P = e^{-\Delta C/T}$, favorisant l'exploration.

Étape 4 : Mise à Jour de la Température

- Réduire la température selon une loi de refroidissement exponentielle :

$$T_{k+1} = \alpha T_k \tag{2.2}$$

  où α est un paramètre de refroidissement $(0 < \alpha < 1)$.

Étape 5 : Critère d'Arrêt

- L'algorithme s'arrête lorsque la température atteint un seuil prédéfini ($T < T_{min}$) ou après un nombre d'itérations fixé.

Application du Recuit Simulé au Problème Flowshop

L'algorithme SA est largement utilisé pour résoudre le **Flowshop Scheduling Problem (FSP)**, où l'objectif est de minimiser le **Makespan** (C_{max}), c'est-à-dire le temps total d'achèvement des tâches [15].

Modélisation du problème

- **Représentation de la solution :** Un ordonnancement de tâches est représenté par une permutation de n tâches sur m machines.
- **Génération d'une solution initiale :** Utilisation d'une heuristique comme NEH [31] pour produire une première séquence efficace.
- **Définition du voisinage :** Modifier légèrement la séquence (ex. : échange ou inversion de deux tâches).
- **Refroidissement et convergence :** Un refroidissement progressif est appliqué pour améliorer la stabilité des solutions.

Domaine d'application

- Problèmes de routage et de logistique.
- Optimisation des trajectoires (ex. drones, robots).
- Répartition des charges dans les systèmes informatiques.
- Optimisation des réseaux électriques et de la production d'énergie.

2.4 Comparaison entre les métaheuristiques utilisées

Chaque algorithme présente des avantages et des limites, notamment en termes de qualité de solution, de temps de convergence et de robustesse, ce qui justifie une étude comparative approfondie.

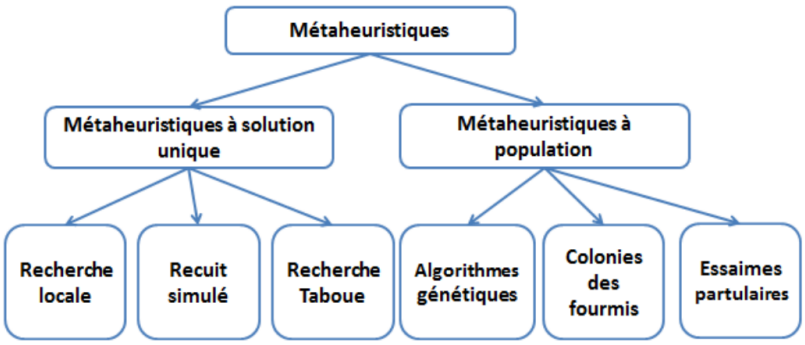

FIG. 2.11 : Analyse comparative des métaheuristiques.[59]

2.4.1 Tableau synthétique des caractéristiques théoriques

Ce tableau met en évidence les forces et limites de chaque métaheuristique. GA et PSO sont rapides mais peuvent converger prématurément, tandis que SA et TS sont plus lents mais évitent mieux les minima locaux. Le choix de l'algorithme dépend donc du problème à résoudre et des contraintes de calcul. Une approche hybride peut être une solution efficace pour améliorer les performances [62][63][64].

Tab. 2.2 : Tableau comparatif des métaheuristiques

Critères	**GA (Algorithmes Génétiques)**	**PSO (Optimisation par Essaim de Particules)**	**TS (Recherche Tabou)**	**SA (Recuit Simulé)**
Inspiration	Évolution biologique et sélection naturelle	Comportement des essaims d'oiseaux ou de poissons	Mécanismes de mémoire humaine	Processus de refroidissement thermodynamique
Représentation des solutions	Chromosomes codés en binaire ou réel	Particules représentant des solutions potentielles	Solutions avec une liste tabou pour éviter les cycles	États du système avec une fonction d'énergie
Mécanisme de mise à jour	Sélection, croisement et mutation des chromosomes	Mise à jour des vitesses et positions des particules	Exploration des voisins avec évitement des mouvements tabous	Acceptation probabiliste des solutions en fonction de la température
Critère de convergence	Stabilisation de la population vers une solution optimale	Convergence des particules vers les meilleures positions	Atteinte d'une solution optimale après exploration	Atteinte d'un état d'énergie minimale après refroidissement
Domaines d'application typiques	Optimisation combinatoire, problèmes de planification	Optimisation continue, problèmes de contrôle	Problèmes d'ordonnancement, conception de circuits	Problèmes NP-difficiles, optimisation globale

2.4.2 Points forts et limites de chaque algorithme

Dans cette section, nous présentons une comparaison détaillée des quatre métaheuristiques étudiées en fonction de six critères clés. Ces critères permettent d'évaluer leurs performances et leur pertinence pour la résolution du problème d'ordonnancement Flowshop.

Le tableau ci-dessous met en évidence les avantages et les limites de chaque algorithme. Ces informations sont essentielles pour orienter le choix de la métaheuristique la plus appropriée en fonction des contraintes du problème traité.

Pour évaluer les points forts et les limites des métaheuristiques, six critères techniques pertinents peuvent être pris en considération :

- Qualité des solutions – Capacité à trouver des solutions proches de l'optimal global.
- Vitesse de convergence – Rapidité avec laquelle l'algorithme atteint une bonne solution.
- Évasion des minima locaux – Aptitude à éviter le piégeage dans des solutions sous-optimales.
- Sensibilité aux paramètres – Importance du réglage des hyperparamètres pour la performance.
- Facilité d'implémentation – Complexité du développement et de l'utilisation.
- Scalabilité – Capacité à s'adapter à des problèmes de grande taille tout en maintenant une performance acceptable.

Tab. 2.3 : Points forts et limites des métaheuristiques

Critères	**GA**	**PSO**	**TS**	**SA**
Qualité des solutions	Bonne pour les problèmes combinatoires	Bonne pour les problèmes continus	Dépend du réglage de la liste taboue	Bonne mais dépend du taux de refroidissement
Vitesse de convergence	Modérée	Rapide mais risque de stagnation	Peut être lente pour de grands problèmes	Lente mais systématique
Évasion des minima locaux	Moyenne à élevée (mutation favorise l'exploration)	Moyenne, risque de convergence prématurée	Moyenne, dépend du mécanisme de diversification	Très bonne grâce à l'acceptation probabiliste
Sensibilité aux paramètres	Élevée (croisement, mutation, taille de la population)	Très élevée (poids d'inertie, coefficients d'accélération)	Moyenne (taille de la liste taboue critique)	Moyenne à élevée (taux de refroidissement, voisinage)
Facilité d'implémentation	Moyenne (besoin de représenter les individus et les opérateurs génétiques)	Facile (mise à jour simple des particules)	Moyenne (gestion de la liste taboue requise)	Facile (principe du refroidissement simple)
Scalabilité	Bonne pour problèmes de taille moyenne	Excellente pour optimisation continue	Bonne pour petits et moyens problèmes	Bonne mais peut devenir coûteuse

2.5 Hybridation des métaheuristiques

Afin de pallier les limites des métaheuristiques utilisées individuellement, notamment en termes de convergence prématurée ou de stagnation dans des optima locaux, il est pertinent d'explorer l'hybridation de deux méthodes complémentaires. Une stratégie efficace consiste à combiner une métaheuristique de recherche globale (telle que l'algorithme génétique ou la PSO), qui explore l'espace des solutions de manière large, avec une métaheuristique de recherche locale (comme la recherche tabou ou le recuit simulé), qui affine les solutions prometteuses pour atteindre des optima de meilleure qualité.

Cette combinaison permet de tirer parti de la capacité d'exploration initiale de la première méthode, tout en exploitant la capacité d'exploitation locale de la seconde. Ce principe s'inscrit dans la tendance actuelle des "algorithmes hybrides", largement adoptée dans la résolution de problèmes NP-difficiles tels que l'ordonnancement Flowshop. Par exemple, l'hybridation d'un algorithme génétique avec une recherche tabou intégrée en phase de post-traitement permet d'améliorer sensiblement la qualité des solutions finales [65].

L'intégration judicieuse de ces deux types de recherches doit être conçue de manière à assurer une complémentarité et à éviter la redondance ou le surcoût computationnel inutile. Dans ce travail, cette perspective sera discutée dans le cadre des perspectives futures, en tant que prolongement naturel à l'étude comparative initiale.

2.6 Conclusion

Ce chapitre a présenté en détail quatre métaheuristiques largement utilisées dans l'optimisation des problèmes d'ordonnancement en Flowshop : l'Algorithme Génétique (GA), l'Optimisation par Essaims Particulaires (PSO), la Recherche Tabou (TS) et le Recuit Simulé (SA).

Chaque algorithme repose sur des principes différents et offre des performances variées selon le contexte d'application. GA, basé sur l'évolution naturelle, s'est révélé efficace pour explorer de larges espaces de solutions, mais son efficacité dépend fortement du choix des paramètres[66] . PSO, inspiré du comportement des essaims, facilite la convergence rapide vers de bonnes solutions, bien qu'il soit sujet à la stagnation prématurée[67].

D'un autre côté, TS et SA se distinguent par leur approche locale et leur capacité à échapper aux minima locaux. TS utilise une mémoire adaptative pour éviter les cycles et intensifier la recherche dans des zones prometteuses [68], tandis que SA exploite un mécanisme stochastique inspiré du refroidissement des métaux pour explorer des solutions sous-optimales avec une probabilité contrôlée [56].

La comparaison entre ces méthodes a mis en évidence leurs forces et leurs limitations, ouvrant la voie à une analyse expérimentale approfondie dans le chapitre suivant. Les performances réelles de ces métaheuristiques seront évaluées sur des instances concrètes du problème Flowshop, permettant ainsi d'identifier les stratégies les plus adaptées à différentes configurations.

CHAPITRE 3

OUTILS ET ENVIRONNEMENT DE PROGRAMMATION POUR L'ÉXÉCUTION DES ALGORITHMES

3.1 Introduction

Dans ce chapitre, nous abordons la phase pratique de notre étude, qui consiste à modéliser, éxecuter et comparer les différentes métaheuristiques pour résoudre le problème d'ordonnancement dans un atelier de type flowshop, en visant la minimisation du makespan C_{max}. Cette démarche repose sur la mise en œuvre de plusieurs algorithmes bien établis dans la littérature : l'algorithme génétique, la recherche tabou, l'optimisation par essaim de particules et le recuit simulé.

Ces méthodes d'optimisation approximatives, souvent qualifiées de métaheuristiques, se distinguent par leur capacité à explorer efficacement de vastes espaces de recherche et à fournir des solutions de bonne qualité en un temps de calcul raisonnable, même pour des problèmes NP-difficiles tels que le flowshop. [30][14].

Chaque algorithme repose sur des mécanismes de recherche spécifiques — inspirés de phénomènes naturels, biologiques ou physiques — et suit une structure modulaire composée de plusieurs étapes : génération de solutions initiales, évaluation par une fonction objective, opérateurs d'exploration (mutation, voisinage, échange, etc.), et des critères d'arrêt pour la convergence de la solution.

Dans ce chapitre, nous décrirons dans un premier temps l'environnement de programmation dans lequel les algorithmes ont été développés et exécutés. Par la suite, chaque métaheuristique sera présentée avec ses principes fondamentaux, structures générales, ainsi que les paramètres techniques retenus pour la simulation. Enfin, une analyse comparative des performances sera menée à travers des indicateurs tels que le makespan, le temps de calcul et la stabilité des solutions, dans le but de déterminer la stratégie la plus éfficace pour le problème étudié.

3.2 Environnement de programmation

L'environnement de programmation joue un rôle fondamental dans le développement et l'évaluation des algorithmes d'optimisation. Un cadre de programmation efficace doit offrir des outils adaptés à la modélisation des problèmes, à la gestion des structures de données et à l'exécution des simulations. Il est essentiel de disposer d'un environnement permettant une bonne gestion des ressources de calcul, notamment en ce qui concerne l'allocation de la mémoire et la vitesse d'exécution des algorithmes. De plus, l'utilisation de bibliothèques et de plateformes dédiées à l'optimisation et à l'intelligence artificielle facilite la mise en œuvre des métaheuristiques en garantissant une manipulation efficace des données et des solutions. Enfin, un environnement bien configuré favorise la reproductibilité des expériences et l'analyse comparative des performances des différentes approches[38].

3.2.1 Python

Le choix du langage de programmation représente une étape stratégique dans le développement d'outils d'optimisation performants, notamment lorsqu'il s'agit de résoudre des problèmes complexes comme le flowshop. Dans ce contexte, Python s'est imposé comme une solution à la fois robuste, accessible et efficace. Son architecture simple, présence d'outils de programmation d'objets orientés, permet de structurer des algorithmes complexes tout en assurant une lisibilité du code élevée, ce qui est particulièrement appréciable dans un cadre académique et expérimental [69].

L'un des atouts majeurs de Python réside dans la richesse de son écosystème de bibliothèques. Des modules comme NumPy et SciPy facilitent la gestion des opérations mathématiques avancées, tandis que Matplotlib et Seaborn permettent de générer des visualisations claires pour l'analyse comparative des résultats [70]. En outre, plusieurs bibliothèques spécialisées dans les métaheuristiques, telles que DEAP, PyGAD, SimAnneal, ou encore ACO-Pants et optuna, offrent des cadres prêts à l'emploi pour éxecuter, personnaliser et tester les algorithmes d'optimisation [71].

Python s'adapte également aux exigences des projets de recherche, grâce à son intégration fluide avec d'autres outils, sa compatibilité multiplateforme et sa large communauté scientifique, qui en fait un choix de prédilection dans les domaines de l'intelligence artificielle, de l'apprentissage automatique et de la recherche opérationnelle [72]. Enfin, sa capacité à gérer des structures de données complexes, à manipuler des fichiers externes, et à exécuter des simulations intensives le rend tout à fait adapté à l'étude comparative de métaheuristiques dans un contexte de planification industrielle.

FIG. 3.1 : Logo Python.[73]

3.2.2 Visuel Code Studio

Dans le cadre de ce travail de recherche, Visual Studio Code (VS Code) a été choisi comme l'environnement principal de développement pour la mise en œuvre des différentes métaheuristiques utilisées dans cette étude comparative. Cet éditeur, réputé pour sa légèreté et sa rapidité, est particulièrement apprécié dans la communauté des développeurs pour sa flexibilité et sa puissance dans le traitement de multiples langages, dont Python [74]. Son architecture modulaire

permet de l'adapter facilement à divers besoins grâce à une richesse d'extensions, qui facilite l'intégration de bibliothèques spécifiques et améliore la productivité des utilisateurs [75]. En particulier, l'extension Python développée par Microsoft offre une prise en charge avancée du langage, permettant de travailler avec des environnements virtuels, de déboguer des scripts Python, d'exécuter des tests unitaires, ainsi que de bénéficier de l'autocomplétion de code via IntelliSense [76]. De plus, l'extension Jupyter pour VS Code permet d'ouvrir et d'exécuter des notebooks Jupyter, facilitant ainsi la visualisation interactive des résultats, ce qui est essentiel pour le développement, le test, et l'évaluation des performances des algorithmes de recherche heuristique comme les algorithmes génétiques, la recherche taboue, le recuit simulé et la PSO (Optimisation par essaim particulaire) [77].

L'utilisation du VS Code dans ce projet a non seulement optimisé la gestion du code et du processus de développement mais a également simplifié la reproductibilité des expériences, un aspect essentiel dans toute étude scientifique. La possibilité d'intégrer directement des outils de contrôle de version (via l'extension Git) et de gérer des environnements virtuels Python garantit une organisation et un suivi rigoureux des expérimentations [78]. Ces fonctionnalités ont été cruciales pour le développement de simulations fiables et la gestion des différentes variantes des métaheuristiques dans l'étude comparative du makespan dans le cadre d'un atelier flowshop.

FIG. 3.2 : Visuel Studio Code.[79]

3.2.3 Configuration matérielle de simulation

L'ensemble des simulations, implémentations et résultats présentés dans cette étude ont été réalisés sur un ordinateur personnel de configuration intermédiaire, spécifiquement le modèle DESKTOP-CLOUDI5, équipé d'un processeur Intel Core i5-1035G1 (1.00 GHz, jusqu'à 1.20 GHz), de 8 Go de mémoire RAM, et fonctionnant sous un système Windows 64 bits.

Il est important de préciser que les performances observées, notamment les temps d'exécution des algorithmes, sont directement influencées par cette configuration matérielle. Par conséquent, toute reproduction des expériences sur une machine disposant de ressources matérielles différentes pourrait engendrer des variations significatives des résultats, en particulier en termes de durée de calcul.Il est également important de signaler que la mémoire interne de la machine affecte directement la capacité de modéliser des problèmes de flowshop avec un nombre de machines élevé.

3.2.4 Configuration des instances

Dans cette étude, toutes les simulations ont été effectuées sur des instances comportant cinq machines (5), un choix motivé par sa représentativité comme valeur moyenne dans la littérature dédiée aux ateliers de type flowshop. Ce paramètre permet de maintenir une structure suffisamment réaliste tout en facilitant l'analyse comparative des résultats. Il est important de souligner que, dans ce type de problèmes, ce n'est pas le nombre de machines mais bien le nombre de

jobs qui influence significativement la complexité du problème et les performances des métaheuristiques, notamment en termes de temps de calcul et de qualité des solutions. Le choix de cinq machines permet donc de se focaliser sur l'impact des algorithmes eux-mêmes sans introduire de variabilité excessive liée à la configuration du système.

3.3 Implémentation des algorithmes

Afin de valider expérimentalement les performances des différentes métaheuristiques sélectionnées pour l'étude, une phase d'implémentation a été menée dans un environnement de programmation adapté. Chaque algorithme a été codé en Python, en s'appuyant sur les principes théoriques décrits dans la littérature. Le choix de Python repose principalement sur sa simplicité syntaxique, sa richesse en bibliothèques spécialisées pour l'optimisation, et sa large adoption dans la communauté scientifique.

L'implémentation a été conçue pour permettre la reproductibilité et la comparabilité des résultats. Pour cela, une structure modulaire a été adoptée : chaque métaheuristique possède un fichier dédié avec ses paramètres spécifiques, sa méthode de génération de solutions, et sa propre stratégie d'exploration du voisinage ou d'évolution de la population. La fonction d'évaluation commune, qui calcule le makespan, a été intégrée comme point central permettant d'unifier l'analyse comparative.

Les pseudocodes proposés dans la littérature scientifique ont été traduits en scripts Python, tout en respectant les logiques algorithmique originales [68], [80], [81]. Des ajustements ont été nécessaires pour adapter les représentations aux spécificités du problème de flowshop permutationnel. L'objectif est de garantir une convergence raisonnable des algorithmes tout en respectant les contraintes de calcul.

3.3.1 Structure des codes Python pour chaque métaheuristique

La conception des algorithmes a été réalisée de manière modulaire, afin de garantir une meilleure lisibilité, une maintenance facilitée et une extensibilité en cas de besoin futur. Chaque métaheuristique étudiée (algorithme génétique, recuit simulé, recherche taboue et optimisation par essaim de particules) a été développée dans un module Python distinct, intégrant ses composants fondamentaux et ses paramètres propres.

Les blocs de code ont été organisés en trois couches principales :

- Initialisation : génération aléatoire ou heuristique de la population (ou des solutions) initiale(s), en tenant compte des contraintes du problème de flowshop.
- Boucle principale d'optimisation : cœur de l'algorithme où s'opèrent la sélection, la mutation, la recherche de voisinage ou la mise à jour des positions, selon le paradigme de chaque métaheuristique.
- Évaluation : calcul systématique du makespan à chaque itération pour mesurer la qualité des solutions générées, en utilisant une fonction d'évaluation commune à tous les algorithmes.

L'approche suivie respecte rigoureusement les logiques issues des publications de référence [67], [82]-[84]. Par ailleurs, des paramètres adaptatifs ont été intégrés pour certains algorithmes (tel que la température pour le SA), afin d'améliorer leur convergence vers des solutions optimales.

Chaque fichier Python a été documenté avec des commentaires explicatifs, facilitant ainsi sa compréhension et sa réutilisation pour des extensions futures ou des hybridations algorithmiques.

3.3.2 Présentation des paramètres algorithmiques des métaheuristiques étudiées

Afin d'évaluer rigoureusement les performances des différentes métaheuristiques étudiées, nous avons adopté une démarche expérimentale consistant à faire varier un paramètre clé propre à chaque algorithme. Cette approche permet non seulement de tester la sensibilité de l'algorithme à ce paramètre, mais aussi d'identifier la configuration la plus performante selon la nature du problème traité.

L'algorithme génétique (GA)

Dans le cas de l'algorithme génétique (GA), nous avons porté une attention particulière à la taille de la population. Étant donné son influence significative sur la diversité des solutions et sur la convergence de l'algorithme, ce paramètre a été modulé en fonction de la taille des instances (nombre de jobs et de machines). Pour chaque matrice de test, plusieurs tailles de population ont été expérimentées afin de déterminer celle qui permettait d'obtenir les meilleurs résultats en termes de makespan. Le choix final du paramètre fixé pour l'évaluation comparative s'est donc appuyé sur les performances empiriques observées.

Tab. 3.1 : Paramètres de l'algorithme génétique

Paramètre	Valeur
Taux de croisement (Pc)	0.9
Taux de mutation (Pm)	0.1
Méthode de sélection	Sélection par tournoi
Type de croisement	Croisement en un point
Type de mutation	Swap
Nombre de générations	200

Optimisation par essaim de particules (PSO)

Dans le cadre de l'optimisation par essaim de particules (PSO), le paramètre central étudié est le nombre de particules constituant l'essaim. Ce choix se justifie par l'impact direct de ce paramètre sur la capacité d'exploration de l'espace de recherche et la convergence vers des solutions de qualité. Une population trop réduite peut conduire à une exploration insuffisante, tandis qu'un nombre excessif de particules peut alourdir le temps de calcul sans amélioration notable des performances.

Ainsi, pour chaque instance du problème flowshop, nous avons fait varier le nombre de particules en fonction de la taille de la matrice (nombre de tâches et de machines), dans le but d'observer l'effet de cette variation sur le makespan obtenu. L'objectif était d'identifier une configuration équilibrée entre efficacité de calcul et qualité des solutions.

TAB. 3.2 : Paramètres de l'algorithme PSO

Paramètre	Valeur
Poids d'inertie (w)	0.5
Coefficient cognitif (c1)	1.5
Coefficient social (c2)	1.5
Nombre maximal d'itérations	1000
Limitation de la vitesse (Vmax)	10

Recherche Tabou (TS)

Pour l'algorithme Tabu Search (TS), nous avons varié le paramètre de tenure en fonction de la taille des jobs. De même, pour les autres algorithmes, nous avons modifié la taille de la population pour chaque matrice de données, dans le but d'identifier la configuration optimale qui maximise la performance. Ces ajustements ont permis d'obtenir les meilleurs résultats en terme de qualité de solution et de temps de calcul.

TAB. 3.3 : Paramètres de l'algorithme Tabu Search

Paramètre	Valeur
Taille de la liste tabou	7
Nombre d'itérations	1000
Critère d'aspiration	Active
Structure de voisinage	Echange de deux jobs
Critère d'arrêt	Itérations $\geq$ 1000

Recuit Simulé (SA)

Pour l'algorithme Simulated Annealing (SA), le paramètre de température initiale a été varié en fonction de la taille des jobs. Cette variation a été réalisée pour chaque matrice de données, dans le but d'identifier la température optimale permettant d'obtenir les meilleurs résultats en termes de qualité de solution et de rapidité de convergence.

TAB. 3.4 : Paramètres de l'algorithme Simulated Annealing

Paramètre	Valeur
Température initiale (T_0)	1000
Type de refroidissement	Exponentiel
Taux de refroidissement (α)	0,95
Température finale (Tf)	0,001
Itérations par température	100

3.3.3 Pseudocodes des métaheuristiques étudiées

Pour mieux illustrer le fonctionnement des différentes métaheuristiques adoptées dans cette étude, nous avons choisi de présenter leur logique sous la forme de pseudocodes. Cette démarche permet de décrire de manière claire, précise et indépendante de tout langage de programmation, les étapes fondamentales de chaque algorithme.

L'algorithme génétique (GA)

Algorithm 1 L'algorithme génétique pour la minimisation du makespan dans un flowshop

Data : Matrice des temps de traitement pour chaque tâche et machine

Result : Meilleure séquence de jobs et son makespan associé

Définir les paramètres de l'algorithme :
Taux de mutation, taille de la population, nombre de générations

Définir la fonction de calcul du makespan :
- Calculer les temps de fin sur chaque machine
- Retourner le makespan (temps de fin du dernier job sur la dernière machine)

Générer la population initiale :
- Générer des séquences aléatoires de jobs

for *chaque génération* **do**

  Évaluer la population :
  - Calculer le makespan de chaque individu
  - Identifier le meilleur individu

  Sélection par tournoi :
  - Sélectionner les meilleurs individus pour reproduction

  Croisement (Order Crossover) :
  - Créer de nouveaux enfants à partir des parents

  Mutation :
  - Avec une probabilité donnée, échanger deux jobs

  Mise à jour de la population :
  - Remplacer l'ancienne génération par la nouvelle

Exécuter l'algorithme sur plusieurs essais :
for *chaque essai* **do**

  - Enregistrer le meilleur makespan obtenu

Calcul des statistiques :
- Minimum, maximum, moyenne des makespans
- Temps total d'exécution

Afficher les résultats.

Optimisation par essaim de particules (PSO)

Algorithm 2 PSO pour la minimisation du makespan dans un flowshop

Data : Matrice des temps de traitement [machines x jobs]

Result : Meilleure séquence de jobs et le makespan associé

Définir les paramètres du PSO :
- Nombre de particules (`num_particles`)
- Nombre d'itérations (`num_iterations`)
- Poids d'inertie (`inertia_weight`)
- Poids cognitif (`cognitive_weight`)
- Poids social (`social_weight`)

Initialiser les variables :
- Nombre de jobs (`num_jobs`)
- Nombre de machines (`num_machines`)

Définir la fonction de calcul du makespan (Cmax) :
- Initialiser `machine_times` pour suivre le temps de chaque machine
- Pour chaque job, calculer le temps de fin sur chaque machine
- Retourner le temps de fin de la dernière machine (Cmax)

Initialiser le PSO :
- Générer des `particles` aléatoires (permutations de jobs)
- Initialiser les `velocities` à zéro
- Copier les positions initiales comme `personal_best_positions`
- Évaluer et stocker `personal_best_scores`
- Définir `global_best_position` et `global_best_score`

for *chaque itération* $i = 1$ **to** *num_iterations* **do**
  for *chaque particule* $p = 1$ **to** *num_particles* **do**
    Générer des valeurs aléatoires $r1$ et $r2$ pour chaque job Copier la position actuelle de la particule **for** *chaque job* **do**
      Calculer un facteur de déplacement basé sur $r1$, $r2$ **if** *le facteur est inférieur à un seuil* **then**
        Échanger deux jobs dans la séquence
    Calculer le nouveau `Cmax` **if** *Cmax est meilleur que* *personal_best_score* **then**
      Mettre à jour `personal_best_position` et `personal_best_score` **if** *Cmax est meilleur que* *global_best_score* **then**
        Mettre à jour `global_best_position` et `global_best_score`
  Enregistrer le meilleur `Cmax` de cette itération dans `iteration_cmax_list`

Retourner :
- Meilleure séquence de jobs (`global_best_position`)
- Makespan associé (`global_best_score`)

Afficher les résultats :
- Ordonnancement optimal
- Cmax final, Cmax max, Cmax moyen
- Temps total d'exécution

Recherche Tabou (TS)

Algorithm 3 Tabu Search pour minimiser le makespan dans un flowshop

Data :

- Matrice des temps de traitement `processing_times`
- Nombre de jobs `n_jobs`
- Tenure `tenure` (durée d'interdiction d'un mouvement)
- Nombre maximal d'itérations `max_iter`

Result : Meilleure solution trouvée et son makespan

Initialiser l'heure de début de l'algorithme
Définir la fonction `Calculer_Makespan(sequence)` **begin**
    Créer une matrice `completion` de taille $(n\_machines, n\_jobs)$ **for** *chaque job dans la séquence* **do**
        for *chaque machine* **do**
            if *premier job et première machine* **then**
                Définir le temps de complétion
            else
                Calculer le temps de complétion selon la contrainte flow-shop
    Retourner le temps de complétion du dernier job sur la dernière machine
Initialiser une solution aléatoire `current_solution` Initialiser `best_solution` $\leftarrow$ `current_solution` Calculer `best_makespan` à partir de `best_solution` Initialiser la liste Tabu `tabu_list` comme vide
for $iter = 1$ **to** *`max_iter`* **do**
    Initialiser `neighborhood` comme vide
    for *chaque paire* (i, j) *de jobs* **do**
        Créer un voisin en échangeant les jobs i et j **if** *l'échange* (i, j) *n'est pas dans* *`tabu_list`* **then**
            Ajouter le voisin à `neighborhood`
    Trier les voisins par leur `makespan` croissant
    Sélectionner le meilleur voisin
    Mettre à jour `current_solution` avec le meilleur voisin Calculer le `makespan` du voisin sélectionné
    if *ce* *`makespan`* *est inférieur à* *`best_makespan`* **then**
        Mettre à jour `best_solution` et `best_makespan`
    Ajouter le mouvement (i, j) à `tabu_list` **if** *taille de* *`tabu_list`* $>$ *`tenure`* **then**
        Retirer le plus ancien mouvement
Retourner `best_solution` et `best_makespan`
Exécuter l'algorithme 10 fois indépendamment Initialiser `results` comme liste vide
for *chaque exécution* **do**
    Exécuter Tabu Search Ajouter le `makespan` obtenu à `results`
Afficher :

- Le makespan minimum (`Min_Cmax`), Le makespan maximum (`Max_Cmax`)
- La moyenne des makespans (`Moy_Cmax`), Le temps total d'exécution

Recuit Simulé (SA)

Algorithm 4 Recuit Simulé pour minimiser le makespan dans un flowshop

Data : Nombre de machines : `num_machines`, Nombre de jobs : `num_jobs`,
Matrice des temps : `processing_times`, Température initiale : `initial_temp`,
Taux de refroidissement : `cooling_rate`, Temp. min : `min_temp`,
Itérations par temp. : `iterations_per_temp`
Result : `best_solution`, `best_cost`

Fonction makespan : construire matrice de complétion et retourner dernière case.
Solution initiale : liste aléatoire de jobs.
Voisine : échanger 2 jobs dans une copie de la solution.

Algorithme principal :
Initialiser `current_solution` aléatoirement
Calculer `current_cost`
Initialiser `best_solution`, `best_cost`
Fixer `T` à `initial_temp`
while *T > min_temp* **do**
  for $i = 1$ **to** *iterations_per_temp* **do**
    Générer `new_solution` par swap
    Calculer `new_cost`, `delta` ← `new_cost` - `current_cost`
    if *delta < 0* ***ou*** *rand()* $< \exp(-delta/T)$ **then**
      Accepter `new_solution` comme `current_solution`
      MAJ `current_cost`
      if *current_cost < best_cost* **then**
        MAJ `best_solution` et `best_cost`
      end
    end
  end
  `T *= cooling_rate`
end
Répéter 10 fois :
Initialiser `global_best_makespan` élevé
for *chaque itération* **do**
  Exécuter l'algorithme
  MAJ `global_best_makespan` si besoin
end
Afficher résultats : `Cmax_min`, `Cmax_max`, `Cmax_moyen`, temps total

3.3.4 Pseudocodes des Métaheuristiques Hybridés

Algorithme Génétique et Recuit Simulé (GA & SA)

Algorithm 5 Algorithme Hybride (GA + SA) pour minimiser le makespan dans un flowshop

Data : Matrice des temps de traitement (`PROCESSING_TIMES`)

Result : Meilleure séquence de jobs et du makespan associé

1 **Définir les paramètres de l'algorithme génétique :**
- Taille de la population (`POP_SIZE`)
- Nombre de générations (`GENERATIONS`)
- Taux de mutation (`MUTATION_RATE`)

2 **Définir les paramètres du recuit simulé :**
- Température initiale (`initial_temp`)
- Taux de refroidissement (`cooling_rate`)
- Température minimale d'arrêt (`stopping_temp`)
- Nombre maximal d'itérations (`max_iter`)

3 **Initialiser les variables :**
- `results` ← liste vide
- `cmax_list` ← liste vide
- `nb_jobs` ← nombre total de jobs

4 **for** *exécution* $e = 1$ **to** 10 **do**

5 Afficher le numéro d'exécution

6 **Phase GA :**
- Générer une population initiale aléatoire - Évaluer le makespan de chaque séquence - Sélectionner les parents par tournoi - Effectuer les croisements et appliquer la mutation - Identifier la meilleure solution GA (`best_sequence_ga`)

7 **Phase SA :**
- Appliquer le recuit simulé à `best_sequence_ga` - Obtenir `best_sequence_hybrid`, `best_makespan_hybrid`

8 Ajouter les résultats à `results` et `cmax_list` Afficher les makespans GA et Hybrid

9 **Fin de l'exécution :**
- Calculer les statistiques finales :
Min_Cmax, Max_Cmax, Moyenne_Cmax - Afficher les résultats finaux et le temps total d'exécution

10 **Retourner :**
- Meilleure séquence optimisée par GA + SA
- Makespan final et performances globales

Optimisation par essaim de particules et Recherche Tabou (PSO & TS)

Algorithm 6 Hybride PSO + Tabu Search pour la Minimisation du Makespan dans un Flow Shop

Data : Matrice des temps de traitement `processing_times` [machines × jobs]

Result : Meilleure séquence de jobs et le makespan associé

Paramètres : `num_particles` ← 1000000; `num_iterations` ← 10; `num_replicates` ← 10; `tabu_tenure` ← 10; `tabu_iterations` ← 50; Déterminer `num_jobs`, `num_machines`

Fonction `calculate_makespan(schedule)`

Initialiser `machine_times` à 0 pour chaque machine

for *chaque job dans* `schedule` **do**

  for $m = 0$ **to** *num_machines-1* **do**

    if $m = 0$ **then**

      `start_time` ← `machine_times`[m]

    else

      `start_time` ← max(`machine_times`[m], `machine_times`[m-1])

    `machine_times`[m] ← `start_time` + `processing_times`[m][job]

return `machine_times[num_machines - 1]`

Fonction `tabu_search(initial_solution)`

`best_solution` ← `initial_solution`; `best_makespan` ← `calculate_makespan(best_solution)`; `tabu_list` ← vide

for $t = 1$ **to** *tabu_iterations* **do**

  Générer voisinage par permutations (i, j) non tabou

  Évaluer chaque voisin → makespan

  Choisir le meilleur voisin non tabou

  Mettre à jour la solution courante & `best_solution` si meilleure

  Ajouter (i, j) à `tabu_list` ; tronquer si dépassement de `tabu_tenure`

return `best_solution`, `best_makespan`

Fonction `pso()`

Générer `particles` aléatoires (permutations de jobs)

Initialiser `personal_best_positions`, `scores`, `global_best`

for $i = 1$ **to** *num_iterations* **do**

  for *chaque particule* p **do**

    Créer `new_position` en permutant des jobs avec $p = 0.5$

    Évaluer le makespan ; mettre à jour `personal_best` et `global_best` si amélioration

return `global_best_position`, `global_best_score`

Programme Principal

Initialiser `cmax_results`, `execution_times`

for $r = 1$ **to** *num_replicates* **do**

  Afficher "Réplication r"; démarrer chronomètre

  `pso()` → `pso_solution`, `pso_makespan`

  `tabu_search(pso_solution)` → `ts_solution`, `ts_makespan`

  Enregistrer le temps, stocker `ts_makespan` et durée

  Afficher les résultats de la réplication

Résultats Globaux

`Cmax min` = min(`cmax_results`); `Cmax max` = max(`cmax_results`);

`Cmax moyen` = moyenne(`cmax_results`); Temps total = somme(`execution_times`)

3.3.5 Fonction d'évaluation - Makespan

Dans les problèmes d'ordonnancement de type flowshop, la fonction d'évaluation la plus couramment utilisée est le makespan (C_{max}), représentant le temps total nécessaire pour terminer toutes les tâches dans le système. Cette mesure, largement adoptée dans la littérature [12], permet d'évaluer l'efficacité d'une séquence de production en identifiant le moment où la dernière tâche est complétée sur la dernière machine.

Le makespan se calcule en tenant compte des contraintes de précédence et de disponibilité des machines. Il constitue un indicateur clé dans les environnements industriels où l'objectif est de réduire les délais de production et améliorer l'utilisation des ressources [1].

3.3.6 Fonction d'évaluation - Temps d'éxecution

Le temps d'exécution désigne la durée nécessaire pour qu'un algorithme fournisse une solution à une instance donnée du problème. Cette métrique est particulièrement significative dans les problèmes d'ordonnancement de grande taille, où les méthodes exactes échouent souvent à produire des résultats dans un délai acceptable [12].

Dans le cadre des métaheuristiques, le temps d'exécution dépend de divers facteurs : paramètres de la machine (processeur et mémoire), structure de l'algorithme, paramètres, nombre d'itérations, complexité de la solution évaluée. Ainsi, l'analyse comparative ne peut se limiter au seul makespan, mais doit également intégrer le coût temporel du processus de recherche de solutions, comme le recommandent Baker et Trietsch (2009) [1].

3.4 Détails techniques pour chaque métaheuristique

L'exécution efficace des métaheuristiques pour la résolution du problème flowshop repose sur un ensemble de considérations techniques fondamentales. Tout d'abord, la représentation des solutions est généralement assurée par des permutations des tâches, une forme bien adaptée à la nature séquentielle de l'ordonnancement sur plusieurs machines, ce qui facilite l'application des opérateurs de recherche spécifiques à chaque méthode [39]. La performance de chaque solution est évaluée à l'aide d'une fonction objective qui calcule le makespan, c'est-à-dire le temps total requis pour exécuter l'ensemble des tâches selon la séquence proposée. Ce calcul prend en compte les contraintes d'enchaînement entre les machines, et constitue un indicateur clé dans la comparaison des performances [14]. Par ailleurs, l'efficacité globale d'un algorithme dépend aussi de la stratégie d'initialisation des solutions : celles-ci peuvent être générées aléatoirement pour favoriser la diversité, ou construites à l'aide d'heuristiques simples pour guider la recherche dès les premières itérations. Le choix des paramètres algorithmiques – tels que la taille de la population, le taux de mutation, ou encore la température initiale – affecte fortement la capacité de convergence et la stabilité des résultats [85]. Enfin, le processus de recherche est encadré par des critères d'arrêt tels qu'un nombre d'itérations maximal ou une limite de temps, et l'évaluation comparative repose sur plusieurs indicateurs de performance, notamment le makespan, le temps de calcul et la robustesse face aux variations d'instances [14], [30].

3.4.1 L'algorithme génétique (GA)

Les principaux paramètres de l'algorithme génétique, configurés en fonction des caractéristiques du problème de flowshop, sont définis comme suit :

Taille de la population

La taille de la population représente le nombre d'individus (solutions) maintenus à chaque génération de l'algorithme. Dans le contexte du flowshop, chaque individu correspond à une permutation des tâches. Une population trop réduite peut conduire à une convergence prématurée vers un optimum local, en limitant la diversité génétique. À l'inverse, une population trop grande augmente considérablement le temps de calcul sans forcément garantir une amélioration substantielle des résultats. Les recommandations issues de la littérature suggèrent généralement une taille variant entre 20 et 100 individus pour les problèmes d'ordonnancement de taille moyenne [14].

TAB. 3.5 : Variation de la taille de la population par rapport au nombre de jobs dans un problème flowshop

Nombre de jobs (n)	Taille recommandée de la population
5 à 10	20 – 30
11 à 20	30 – 50
21 à 50	50 – 80
51 à 100	80 – 120
> 100	120 – 200

Taux de croisement (crossover rate)

Le croisement est un opérateur fondamental dans les algorithmes génétiques, permettant de générer de nouveaux individus à partir de deux parents. Le taux de croisement détermine la proportion d'individus soumis à cette opération à chaque génération. Dans les problèmes de permutation comme le flowshop, des opérateurs spécifiques comme le Order Crossover (OX) ou le Partially Mapped Crossover (PMX) sont utilisés afin de garantir la validité des solutions générées. Un taux compris entre 0.7 et 0.9 est généralement recommandé pour maintenir une bonne exploration de l'espace de recherche tout en préservant la convergence [30].

Tab. 3.6 : Méthodes de crossover et leurs domaines d'application en ordonnancement

Méthode de crossover	Problèmes d'ordonnancement recommandés
PMX	Fortement recommandé pour les problèmes de type Flowshop et Job Shop, car il préserve les positions et la validité des permutations.
OX	Adapté aux problèmes de permutation comme le Flowshop, surtout pour préserver les relations d'ordre partiel entre tâches.
CX	Utilisé dans les problèmes où l'intégrité des cycles est importante, mais moins courant pour le Flowshop. Plus pertinent pour les tournées ou les cycles de production.
PBX	Convient aux problèmes de type Flowshop avec des contraintes fortes sur certaines positions de tâches.
ERX	Recommandé principalement pour le problème du voyageur de commerce (TSP) ou des tournées avec dépendance de voisinage, moins utilisé pour le Flowshop.

Taux de mutation

La mutation introduit des perturbations aléatoires dans les individus afin de préserver la diversité de la population et d'éviter les stagnations dans des optima locaux. Pour les problèmes de permutation, des mutations simples comme l'échange de deux positions ou l'inversion d'un sous-ensemble sont utilisées. Le taux de mutation est souvent fixé entre 0.01 et 0.1, selon la taille du problème. Une mutation trop faible peut réduire la capacité de l'algorithme à sortir des optima locaux, tandis qu'un taux excessif transforme le processus en une recherche aléatoire [85].

Tab. 3.7 : Taux de mutation recommandé selon le nombre de jobs dans un problème Flowshop

Nombre de jobs (n)	Taux de mutation recommandé
5 à 10	0.05 – 0.1
11 à 20	0.03 – 0.08
21 à 50	0.02 – 0.06
51 à 100	0.01 – 0.05
> 100	0.005 – 0.02

Méthode de sélection

La sélection détermine quels individus sont choisis pour se reproduire. Elle vise à favoriser les meilleures solutions tout en maintenant une certaine diversité. Les méthodes les plus courantes incluent la roulette wheel selection, qui est proportionnelle à la qualité des individus, et la tournament selection, qui compare un sous-ensemble d'individus choisis aléatoirement. Cette dernière est souvent préférée dans les problèmes combinatoires pour son efficacité et sa robustesse. Un mauvais choix de stratégie de sélection peut favoriser un appauvrissement génétique et une convergence prématurée.

TAB. 3.8 : Méthodes de sélection et domaines d'application recommandés

Méthode de sélection	Problèmes recommandés
Roulette Wheel Selection	Problèmes simples à petite échelle, fitness très variée
Tournament Selection	Flowshop, Job Shop — robuste et maintient la diversité
Rank-Based Selection	Flowshop avec disparité de fitness, évite domination prématurée
Elitist Selection	Tous types — utile pour préserver les meilleures solutions
Stochastic Universal Sampling	Problèmes continus ou hybrides, moins courant en Flowshop

Fonction de fitness

Dans le cas d'un problème de minimisation du makespan, la fonction de fitness doit être conçue de manière à ce que les meilleures solutions (i.e. celles avec un makespan plus faible) aient une valeur de fitness plus élevée. Une stratégie classique consiste à utiliser l'inverse du makespan, ou une fonction linéairement décroissante. Il est également possible de normaliser les valeurs de fitness pour éviter que les différences soient trop faibles et donc difficiles à distinguer pour les opérateurs de sélection.

Critère d'arrêt

Le critère d'arrêt définit la condition de terminaison de l'algorithme. Il peut s'agir d'un nombre maximal de générations, d'un temps de calcul limité ou encore d'une stagnation prolongée des résultats (c'est-à-dire aucune amélioration du makespan au cours d'un certain nombre d'itérations). Le choix de ce critère doit prendre en compte les ressources disponibles et le compromis entre qualité de solution et temps de calcul.

3.4.2 Optimisation par essaim de particules (PSO)

Les principaux paramètres de PSO, adaptés spécifiquement au problème de flowshop, sont définis comme suit :

Population de particules (Swarm Size)

La taille de la population représente le nombre total de particules dans l'essaim, chaque particule représentant une solution candidate. Une taille de population optimale est cruciale pour l'efficacité de l'algorithme. Selon les travaux de Kennedy et Eberhart (1995) [45], une taille de population entre 20 et 50 est souvent utilisée dans les applications de planification de type flowshop, en fonction de la complexité du problème.

Vitesse de la particule (Velocity)

Chaque particule ajuste sa position en fonction de sa vitesse, qui est déterminée par deux facteurs : sa propre meilleure position et la meilleure position globale. La mise à jour de la vitesse est généralement calculée comme suit :

$$v_i(t+1) = w \cdot v_i(t) + c_1 \cdot \text{rand}_1 \cdot (p^i_{best} - x_i(t)) + c_2 \cdot \text{rand}_2 \cdot (g_{best} - x_i(t)) \tag{3.1}$$

où $v_i(t)$ est la vitesse de la particule i au temps t, $pBest_i$ est la meilleure position de la particule, et $gBest$ est la meilleure position globale.

Les coefficients w,$c_1$et c_2 sont respectivement le facteur d'inertie et les poids associés à l'attraction vers la meilleure position personnelle et globale, souvent ajustés expérimentalement pour maximiser la performance de l'algorithme [46].

Position de la particule (Position)

La position de chaque particule dans l'espace de recherche représente une solution possible au problème de planification du flowshop. Les positions sont généralement des permutations des tâches ou des machines, en fonction de la représentation choisie. Les solutions sont mises à jour selon la formule suivante :

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{3.2}$$

Facteur d'inertie (Inertia Weight w)

Ce paramètre contrôle l'importance de la vitesse précédente de la particule dans le calcul de la nouvelle vitesse. Un facteur d'inertie élevé favorise l'exploration de nouvelles régions de l'espace de recherche, tandis qu'un faible facteur d'inertie favorise l'exploitation des solutions existantes [86].

Coefficients d'accélération (c_1 et c_2)

Ces coefficients représentent l'influence des meilleures positions personnelles et globales. Leur valeur influence directement la convergence de l'algorithme. Selon Eberhart et Shi (2000) [87], un bon compromis entre exploration et exploitation est obtenu en attribuant des valeurs égales à ces coefficients, généralement autour de 1.5 à 2.

Mécanisme de convergence

La PSO doit être capable de converger vers une solution optimale ou près de l'optimum global. Cela nécessite un équilibre entre exploration (découvrir de nouvelles solutions) et exploitation (améliorer les solutions existantes). Des techniques comme l'adaptation dynamique du facteur d'inertie et des stratégies de diversification sont souvent utilisées pour améliorer la performance de la PSO dans des problèmes complexes comme le flowshop [88].

Paramètres de PSO selon nombre des jobs

TAB. 3.9 : Meilleures valeurs des paramètres de PSO pour minimiser le makespan (Cmax) dans un atelier flowshop en fonction de la taille des jobs

Paramètre	10 jobs	20 jobs	50 jobs	100 jobs	200 jobs
Taille de la population	20	30	50	100	150
Inertie (w)	0.9	0.7	0.8	0.6	0.5
Coefficients d'accélération (c_1, c_2)	(1.5, 1.5)	(1.5, 1.5)	(1.6, 1.6)	(1.7, 1.7)	(1.8, 1.8)
Vitesse maximale	0.2	0.3	0.4	0.5	0.6
Cmax Min. (temps)	150	320	600	1200	2500

3.4.3 Recherche Tabou (TS)

Les paramètres fondamentaux de la recherche tabou, adaptés au contexte du flowshop, sont définis comme suit :

Structure de voisinage

Le voisinage d'une solution est généralement défini par des permutations simples, comme l'échange de deux tâches (swap) ou l'insertion d'une tâche à une autre position. Cette structure influence directement l'exploration de l'espace de recherche.

TAB. 3.10 : Structures de voisinage utilisées dans la recherche tabou pour le problème de flowshop

Structure de voisinage	Description	Avantages
Swap (échange simple)	Échange de deux tâches dans la séquence	Facile à exécuter, exploration rapide du voisinage, efficace pour des problèmes de petite taille
Insertion	Déplacement d'une tâche d'une position à une autre dans la séquence	Permet une exploration plus fine, améliore la qualité locale des solutions
Inversion partielle	Inversion d'une sous-séquence de tâches	Génère des mouvements plus profonds, favorise la diversification
Voisinage restreint aléatoire	Sélection aléatoire d'un sous-ensemble de mouvements (swap ou insertion)	Réduction du temps de calcul, adapté aux problèmes de grande taille
Voisinage adaptatif	Alternance entre plusieurs structures de voisinage selon l'état de la recherche	Combine intensification et diversification, s'adapte à la dynamique du processus de recherche

Liste tabou

Il s'agit d'une mémoire à court terme contenant les mouvements récents ou les attributs de ces mouvements pour empêcher leur répétition immédiate. La taille de cette liste (souvent appelée tenure) est un paramètre clé. Une taille trop petite peut entraîner des cycles, tandis qu'une taille trop grande peut restreindre l'exploration.

Critère d'aspiration

Ce mécanisme permet de surmonter les interdictions de la liste tabou si une solution interditée offre une amélioration significative du critère objectif (par exemple, un makespan plus faible que le meilleur trouvé jusqu'à présent).

Nombre d'itérations ou condition d'arrêt

La méthode peut s'interrompre après un nombre fixe d'itérations, un temps de calcul maximal ou lorsqu'aucune amélioration n'est observée pendant un certain nombre d'itérations consécutives [89], [90].

TAB. 3.11 : Nombre d'itérations recommandé selon la taille du problème flowshop

Taille du problème (n)	**Nombre d'itérations recommandé**	**Justification**
$n \leq 20$	500 à 1 000	Taille réduite, convergence rapide, exploration complète possible
$20 < n \leq 50$	1 000 à 2 000	Taille modérée, compromis entre intensification et diversification
$50 < n \leq 100$	2 000 à 5 000	Nécessite plus de cycles pour échapper aux optima locaux
$n > 100$	5 000 à 10 000 (ou adaptatif)	Problème de grande taille, importance de stratégies de diversification, ajustement dynamique conseillé

3.4.4 Recuit Simulé (SA)

Dans le contexte du flowshop, cette méthode est souvent exploitée pour sa capacité à échapper aux optima locaux grâce à l'acceptation contrôlée de solutions de moindre qualité au début de la recherche.

Les paramètres essentiels du recuit simulé sont les suivants :

Température initiale (T_0)

Ce paramètre joue un rôle crucial dans la probabilité d'acceptation des solutions moins bonnes. Une température initiale trop élevée peut entraîner une exploration excessive, tandis qu'une température trop basse limite la capacité d'évasion des optima locaux. Plusieurs études suggèrent d'utiliser une température initiale basée sur l'écart-type des valeurs de l'objectif dans une solution initiale aléatoire [84].

Fonction de refroidissement (cooling schedule)

Elle contrôle la diminution progressive de la température au fil des itérations. La fonction géométrique $T_{K+1} = T_K$ avec $\in (0.8, 0.99)$, est l'une des plus couramment employées en raison de sa simplicité et de son efficacité [82].

Nombre d'itérations par température (Markov chain length) :

Ce paramètre détermine combien de solutions sont explorées à une température donnée avant de la diminuer. Il peut être fixé comme un multiple de la taille du problème, par exemple $n \times 10$, où n est le nombre de tâches [91].

Critère d'arrêt

Le recuit simulé peut s'arrêter soit lorsqu'un nombre maximal d'itérations est atteint, soit lorsque la température devient inférieure à un seuil minimal T_{min} , ou encore lorsqu'aucune amélioration n'est observée pendant plusieurs étapes consécutives [38].

Fonction d'acceptation

L'acceptation d'une solution moins bonne est basée sur une probabilité donnée par la fonction de Metropolis :

$$P = \exp\left(-\frac{\Delta E}{T}\right) \tag{3.3}$$

où est l'augmentation de la valeur de la fonction objective. Cette propriété permet de diversifier la recherche, surtout en début d'exécution [92].

Stratégie de voisinage

Une solution voisine peut être générée par des permutations simples (par exemple, échange de deux tâches). Le choix de cette stratégie impacte directement la capacité de l'algorithme à explorer efficacement l'espace de recherche [82].

Variation des paramètres selon le nombre de jobs

Paramètres	10 jobs	25 jobs	50 jobs	75 jobs	100 jobs	150 jobs	200 jobs
Température initiale (T_0)	150	200	300	400	500	600	700
Facteur de refroidissement (α)	0,95	0,90	0,85	0,80	0,75	0,70	0,65
Itérations par température	100	150	200	300	500	750	1000
Température minimale (T_{min})	10	20	30	40	50	60	70
Critère d'arrêt (itérations sans amélioration)	100	150	200	250	300	400	500

TAB. 3.12 : Variation des paramètres du recuit simulé selon la taille des jobs

3.5 Conclusion

Ce chapitre a permis de poser les fondations techniques nécessaires à l'exécution des différentes métaheuristiques sélectionnées pour la résolution du problème d'ordonnancement dans un atelier de type flowshop. Le choix du langage Python a été motivé par sa simplicité d'utilisation, sa large communauté et la disponibilité de nombreuses bibliothèques adaptées à l'optimisation et au calcul scientifique. L'environnement de développement Visual Studio Code a été privilégié pour sa flexibilité, sa légèreté, et les nombreuses extensions facilitant le débogage et la visualisation du code.

Nous avons ensuite présenté la structure des programmes conçus pour chaque métaheuristique, en détaillant les fonctions d'évaluation (makespan et temps d'exécution), les paramètres spécifiques à chaque algorithme, ainsi que les pseudocodes correspondants. Une attention particulière a été accordée à l'exécution de deux approches hybridées : l'algorithme génétique combiné au recuit simulé (GA-SA), et l'optimisation par essaim de particules combinée à la recherche tabou (PSO-TS). Ces hybridations ont pour objectif de tirer parti des avantages complémentaires de chaque méthode, en associant la puissance exploratoire de l'un à la capacité d'exploitation locale de l'autre.

Ainsi, ce chapitre offre une vue d'ensemble complète des outils mobilisés et de la logique d'implémentation adoptée. Il constitue une base solide pour les expérimentations et l'analyse comparative des performances présentées dans le chapitre suivant.

CHAPITRE 4

RÉSULTATS DES SIMULATIONS, INTERPRÉTATION ET ÉTUDE COMPARATIVE

4.1 Introduction

Après avoir présenté le cadre théorique des principales métaheuristiques dans les chapitres précédents, ce chapitre se consacre à leur mise en œuvre pratique dans le contexte du problème d'ordonnancement de type flowshop, visant la minimisation du makespan C_{max}. Cette phase expérimentale constitue une étape cruciale permettant de confronter les modèles développés à des instances concrètes et d'évaluer leurs performances selon des critères bien définis.

Dans cette optique, plusieurs algorithmes d'optimisation tels que l'algorithme génétique, le recuit simulé, la recherche tabou et l'optimisation par essaim de particules, sont exécutés à l'aide du langage Python. Chaque approche est appliquée à un même ensemble d'instances tests issues de la littérature, permettant une comparaison équitable et rigoureuse.

Ce chapitre expose en détail le protocole expérimental adopté et les résultats obtenus en termes de qualité de solution et de temps de calcul. L'objectif est de mettre en lumière les forces et les limites de chaque métaheuristique, en préparation à l'analyse comparative approfondie qui sera développée par la suite dans ce même chapitre.

4.2 Résultats expérimentaux et interprétation

Afin d'évaluer les performances des différentes métaheuristiques exécutées (GA, PSO, SA, et TS), nous avons procédé à une série de simulations sur des instances de taille variable du problème flowshop. L'objectif est d'observer le comportement de chaque algorithme selon la complexité du problème, mesurée principalement par le nombre de tâches (jobs) à ordonnancer.

Pour ce faire, nous avons généré plusieurs instances tests, réparties comme suit :

- Une instance de 10 jobs, de petite taille, permettant de comparer les solutions obtenues avec celles d'une méthode exacte (de type branch-and-bound ou énumération complète), servant ainsi de référence.
- Des instances plus grandes, comportant 25, 50, 75, 100, 150 et 200 jobs, représentant des cas où les méthodes exactes deviennent inapplicables en raison du temps de calcul sur la machine devenant trés grand et impraticable pour trouver la valeur exacte du C_{max} (problème NP-difficile).

Chaque instance a été simulée avec les cinq algorithmes, en répétant l'expérience plusieurs fois (généralement 10 à 30 exécutions) afin d'observer la robustesse (écart-type) et la stabilité des solutions proposées. Pour chaque simulation, les indicateurs suivants ont été collectés :

- Le makespan minimal obtenu,
- La moyenne des Makespan obtenus,
- Le temps de calcul,
- Le Makespan maximal obtenu.

L'objectif est de fournir une analyse comparative non seulement en termes de qualité de la solution, mais aussi en prenant en compte l'efficacité algorithmique (temps de réponse) et la stabilité.

4.2.1 Visualisation des performances

4.2.2 Instance à 10 jobs

Cette instance de petite taille permet de comparer les performances des métaheuristiques à une solution exacte, obtenue par une méthode déterministe. La matrice utilisée pour créer les examples pour l'instance à 10 jobs est présenté dans la section A.1 de l'Annexe A.

L'algorithme génétique (GA)

Pop size	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
10	618	645	624.7	0.904
20	618	642	626.2	1.693
30	618	629	623.2	2.449
40	618	627	620.4	3.419
50	618	629	623.6	4.105

TAB. 4.1 : Performances de l'algorithme génétique (GA) pour 10 jobs dans un problème flowshop.

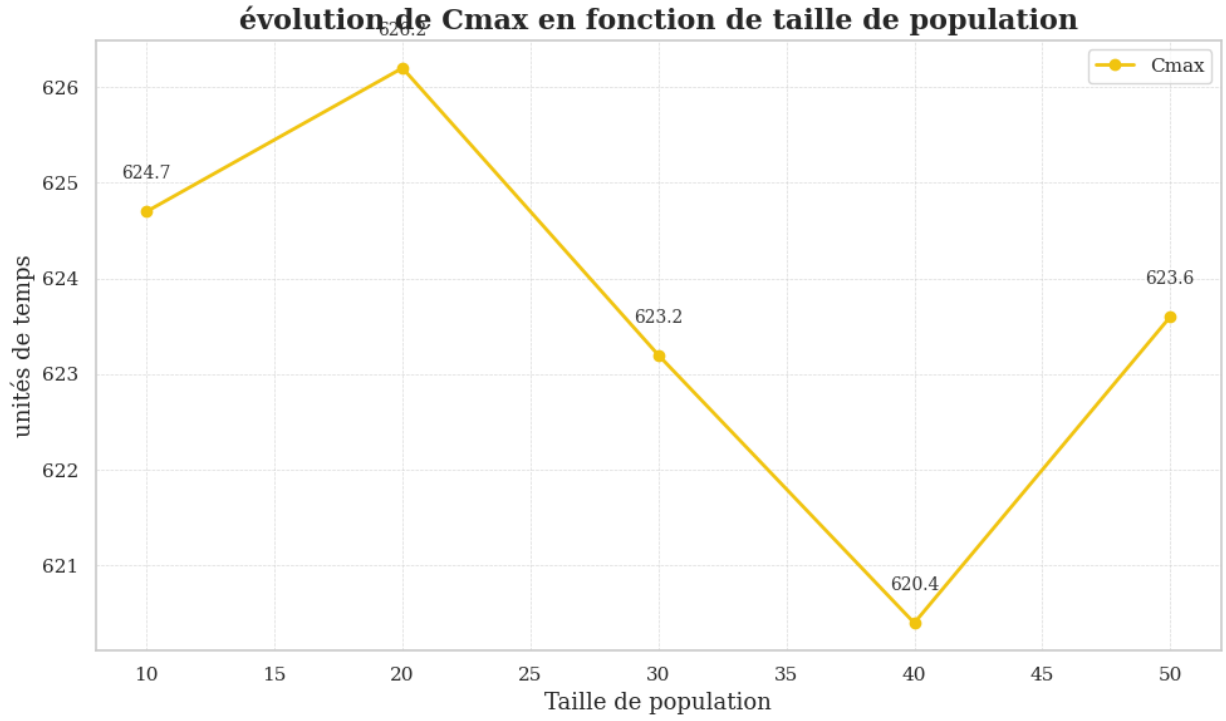

FIG. 4.1 : Évolution du makespan moyen (C_{max}) pour différentes tailles de population avec l'algorithme génétique (GA), pour 10 jobs.

Interprétation des résultats

Les résultats montrent que l'augmentation de la taille de la population n'améliore pas significativement la qualité des solutions (min C_{max} stable à 618), mais réduit légèrement C_{max} moyen pour une population de 40. Le temps d'exécution augmente linéairement avec la taille de la population, indiquant un compromis entre qualité et efficacité.

Optimisation par essaim de particules (PSO)

Num particles	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
1000	636	657	640.2	2.128
2000	642	647	645.6	4.383
3000	629	639	632.0	6.525
4000	624	645	630.9	6.338
5000	634	634	634.0	4.948

TAB. 4.2 : Performances de l'optimisation par Essaim de Particules (PSO) pour 10 jobs dans un problème flowshop.

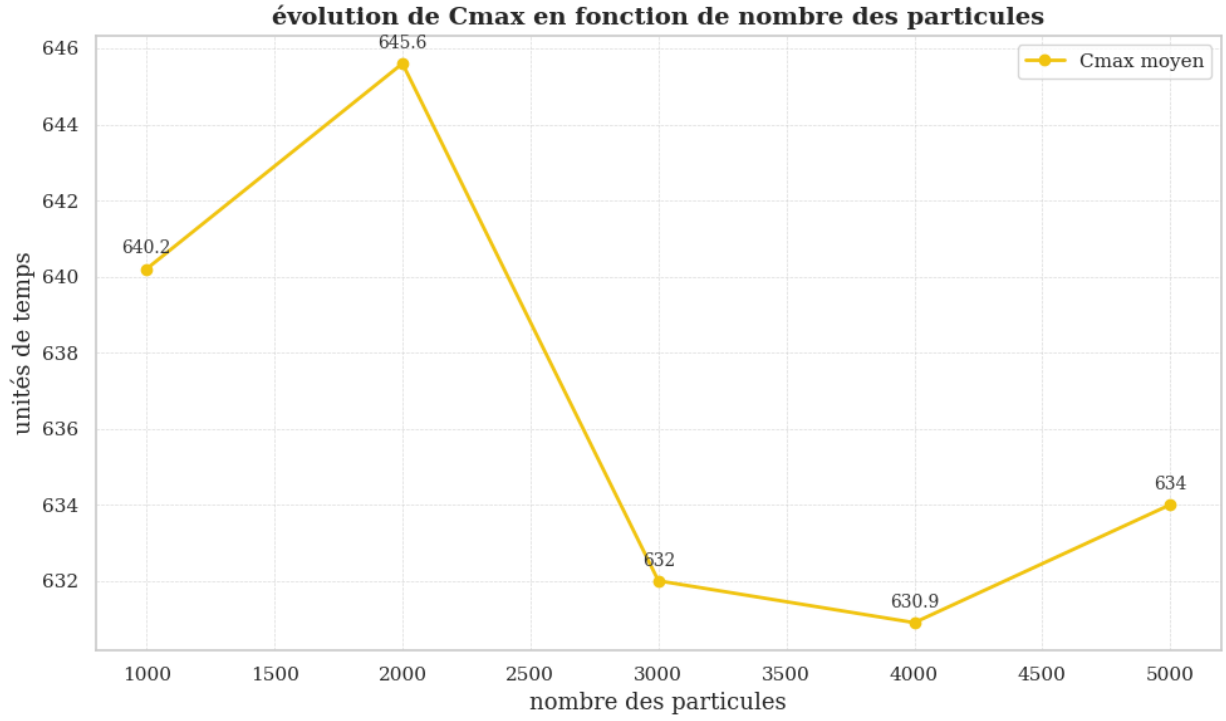

FIG. 4.2 : Évolution du makespan moyen (C_{max}) pour différentes tailles de nuées dans l'algorithme d'optimisation par essaim de particules (PSO), pour 10 jobs.

Interprétation des résultats

Le PSO présente une variabilité des résultats (min C_{max} entre 624 et 642) et un temps d'exécution modéré. Les performances ne s'améliorent pas nécessairement avec l'augmentation du nombre de particules, suggérant un besoin d'optimisation des paramètres.

Recherche tabou (TS)

Tenure	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
10	618	624	619.3	4.931
20	618	629	620.3	3.235
30	618	629	622.8	2.179
40	618	637	622.5	1.181

TAB. 4.3 : Performances de la Recherche Tabou (TS) pour 10 jobs dans un problème flowshop.

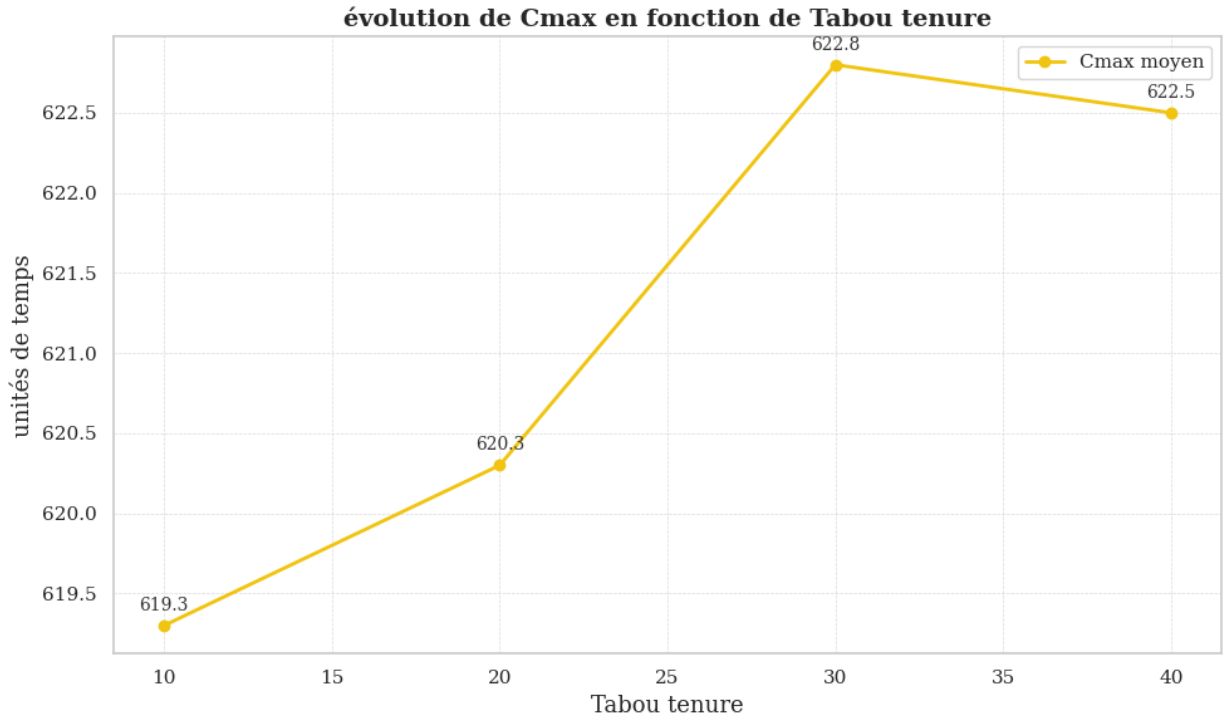

FIG. 4.3 : Évolution du makespan moyen (C_{max}) en fonction de la taille de la mémoire (tenure) dans la méthode Tabu Search (TS), pour 10 jobs.

Interprétation des résultats

La méthode TS obtient des résultats compétitifs (min C_{max} à 618) avec des temps d'exécution courts. La tenure influence peu la qualité des solutions, mais une tenure de 10 offre le meilleur équilibre entre qualité et rapidité.

Recuit simulé (SA)

Initial temp	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
10	618	618	618	12.974
20	618	618	618	12.492
30	618	618	618	14.926
40	618	618	618	19.737
50	618	618	618	21.283

TAB. 4.4 : Performances du Recuit Simulé (SA) pour 10 jobs dans un problème flowshop.

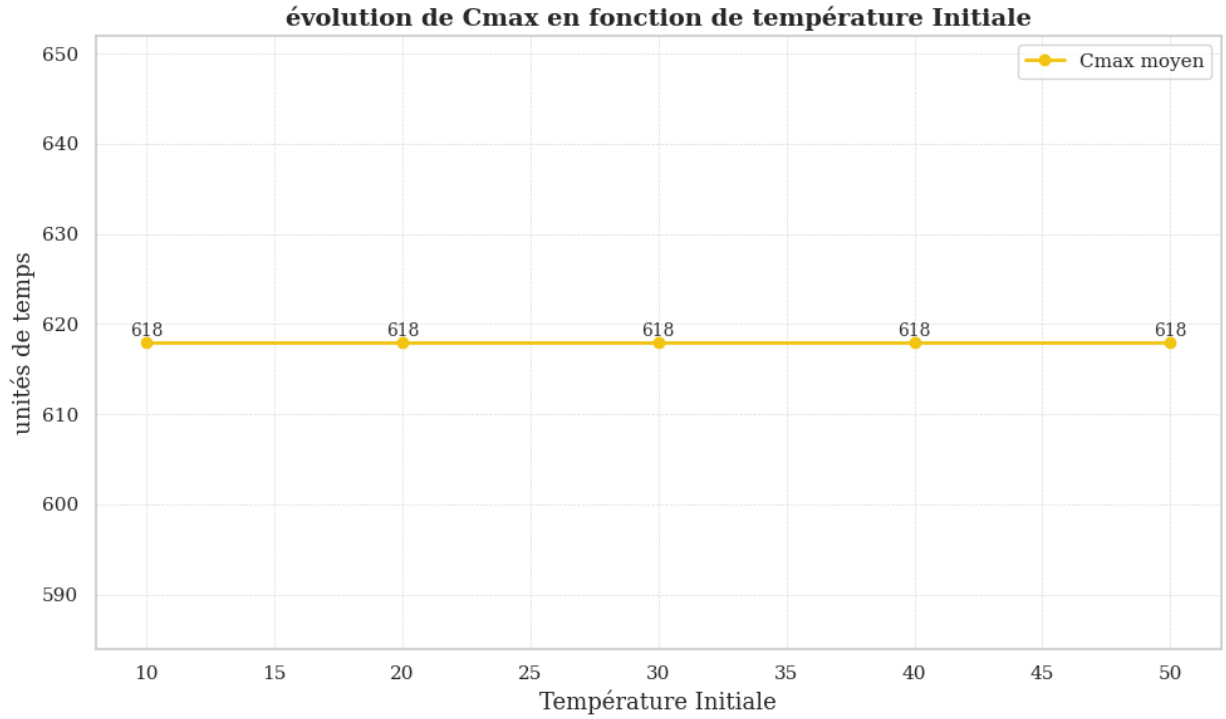

FIG. 4.4 : Évolution du makespan moyen (C_{max}) pour différentes températures initiales avec l'algorithme de recuit simulé (SA), pour 10 jobs.

Interprétation des résultats

Le SA atteint systématiquement le meilleur makespan (618) pour toutes les températures initiales, démontrant une robustesse exceptionnelle. Cependant, son temps d'exécution est nettement plus élevé que les autres algorithmes, ce qui peut être un inconvénient pour des applications en temps réel.

Hybridation GA & SA

Initial temp	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
10	618	620	618.2	6.352
20	618	618	618.0	6.872
30	618	618	618.0	7.185
40	618	618	618.0	7.293
50	618	618	618.0	7.215

TAB. 4.5 : Performances de l'hybridation GA–SA pour 10 jobs dans un problème flowshop.

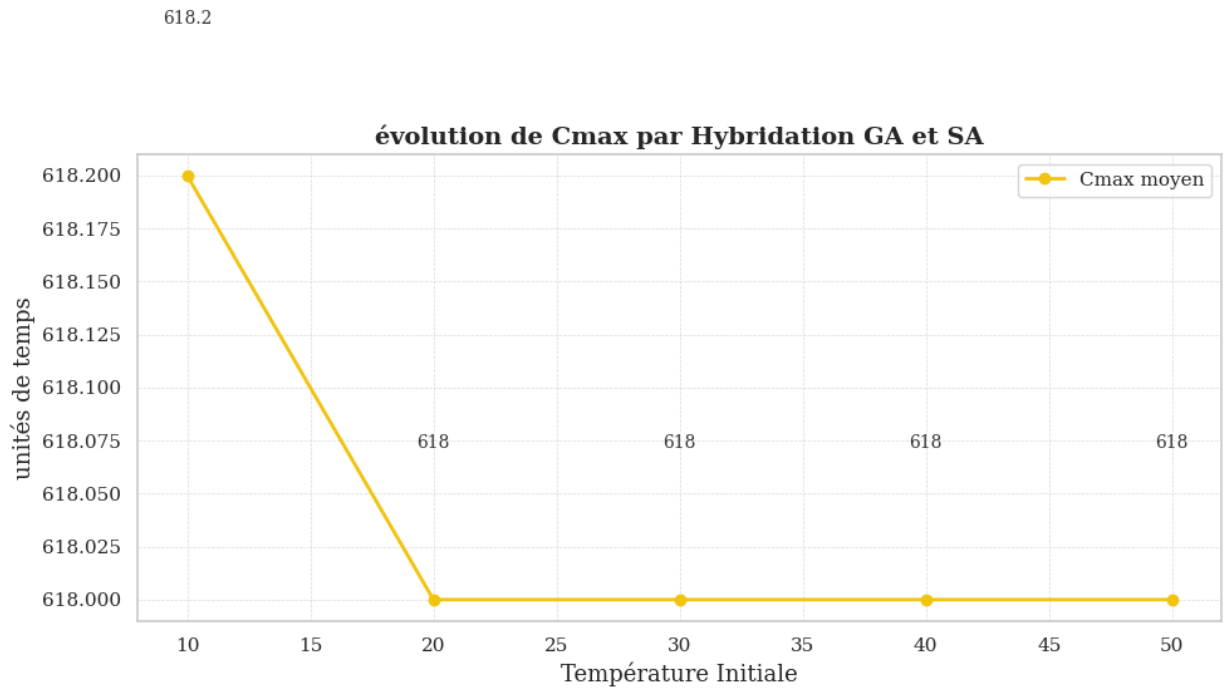

FIG. 4.5 : Évolution du makespan moyen (C_{max}) pour différentes tailles de population dans l'hybridation GA & SA, pour 10 jobs.

Interprétation des résultats

L'hybridation GA-SA maintient les performances optimales du SA (min C_{max} = 618) tout en réduisant considérablement le temps d'exécution par rapport au SA seul (6.352s vs 12.974s pour une température initiale de 10). Cependant, l'avg C_{max} reste très proche de l'optimum (618.2 à 620.8), confirmant la robustesse de cette combinaison. Cette approche semble bénéficier de l'exploitation locale du SA tout en exploitant la diversité générée par le GA.

Hybridation PSO & TS

Tenure	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
10	618	629	620.8	10.254
20	618	624	620.1	9.445
30	618	637	623.0	9.757
40	618	629	621.4	9.570
50	618	633	620.3	9.690

TAB. 4.6 : Performances de l'hybridation PSO–TS pour 10 jobs du problème flowshop.

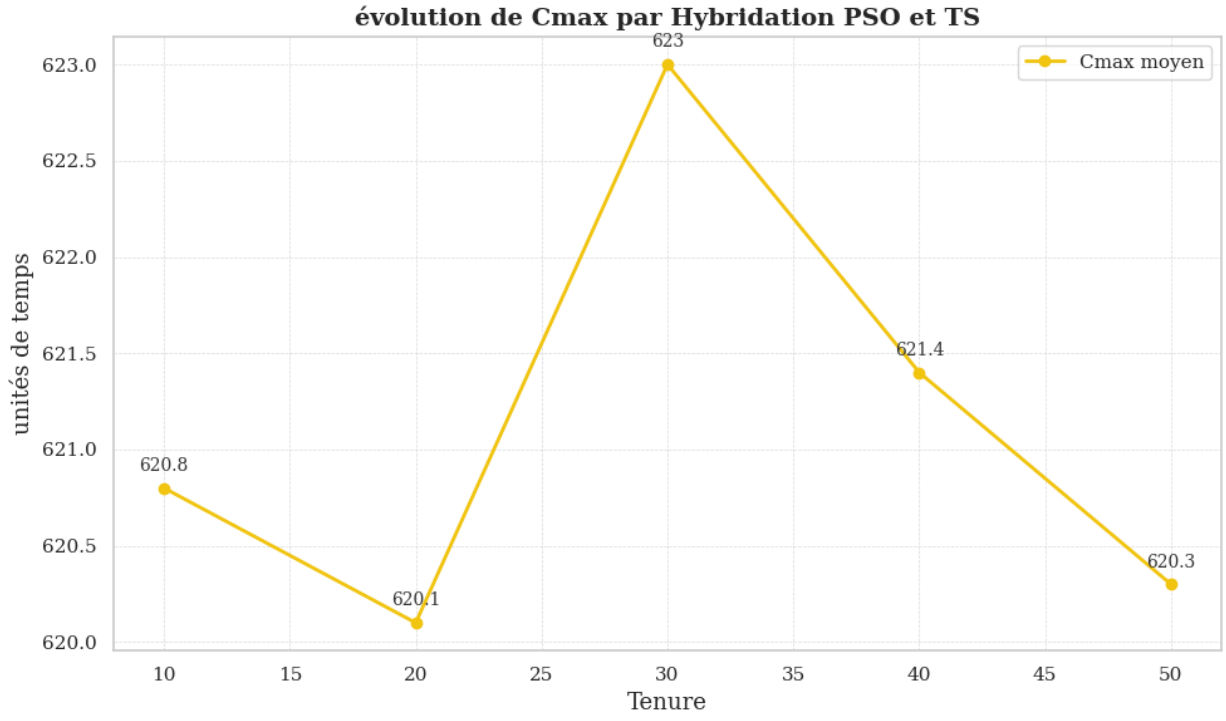

FIG. 4.6 : Évolution du makespan moyen (C_{max}) en fonction du paramètre de mémoire (tenure) dans l'hybridation PSO-TS, pour 10 jobs.

Interprétation des résultats

Cette combinaison conserve les bonnes performances de la TS (min C_{max} = 618) mais avec des temps d'exécution plus élevés (9.445s à 10.254s) que la TS seule (1.181s à 4.931s). L'avg C_{max} (620.1 à 623) reste compétitif, mais l'hybridation n'apporte pas de gain significatif en qualité par rapport à la TS seule. Cela suggère que l'ajout du PSO n'améliore pas suffisamment l'exploration pour justifier l'augmentation du temps de calcul.

4.2.3 Instance à 25 jobs

Cette instance de taille moyenne permet d'évaluer la performance des métaheuristiques sur un problème plus complexe, pour lequel l'obtention d'une solution exacte devient plus coûteuse en temps de calcul. L'analyse se focalise donc sur l'efficacité relative des différentes approches heuristiques. La matrice utilisée pour créer les examples pour l'instance à 25 jobs est présenté dans la section A.2 de l'Annexe A.

L'algorithme génétique (GA)

Pop size	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
25	1513	1548	1532.7	3.458
50	1513	1526	1516.3	6.763
75	1513	1522	1514.1	10.684
100	1513	1522	1515.0	14.266
125	1513	1514	1513.1	20.423

TAB. 4.7 : Performances de l'algorithme génétique (GA) pour 25 jobs dans un problème flowshop.

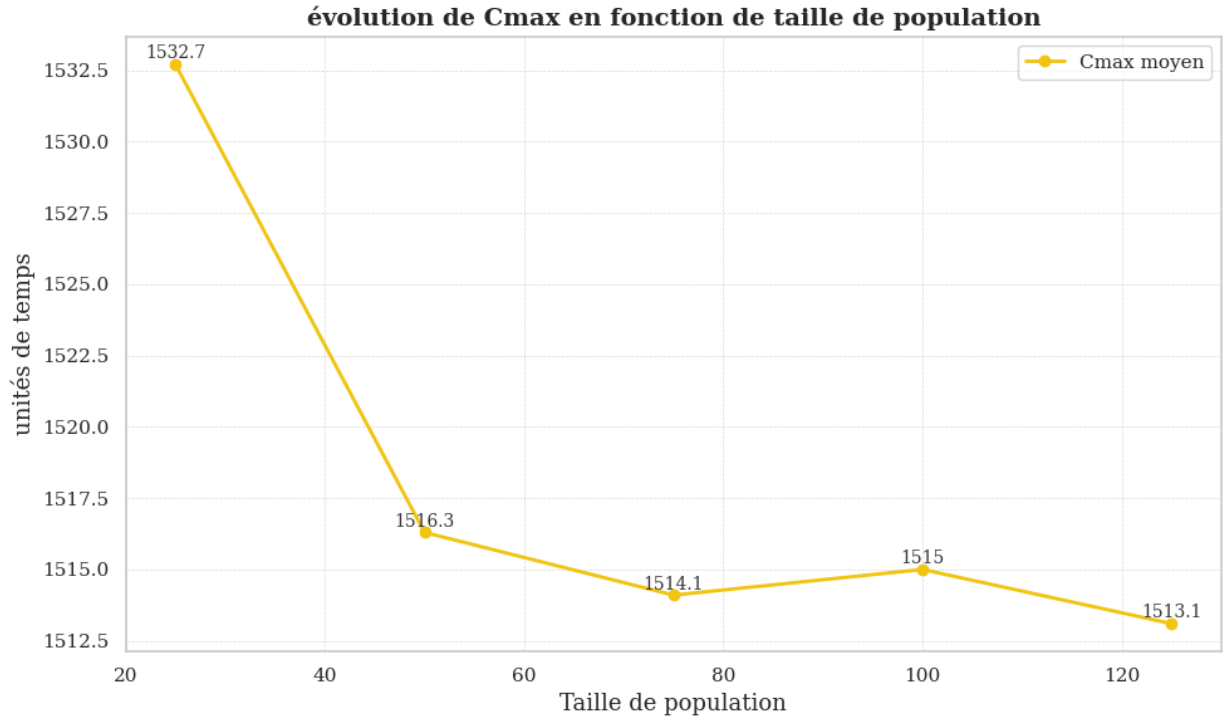

FIG. 4.7 : Évolution du makespan moyen (C_{max}) pour différentes tailles de population avec l'algorithme génétique (GA), pour 25 jobs.

Interprétation des résultats

Le GA atteint systématiquement le meilleur makespan (1513) dès une population de 25, avec une légère amélioration de l'avg C_{max} (1532.7 à 1513.1) lorsque la population augmente. Le temps d'exécution croît linéairement, mais la qualité des solutions se stabilise rapidement, suggérant qu'une population modérée (50-75) offre un bon compromi

Optimisation par essaim de particules (PSO)

Num particles	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
25000	1581	1601	1583.9	36.294
50000	1578	1578	1578.0	67.041
75000	1563	1567	1564.4	97.583
100000	1570	1579	1573.3	139.731
125000	1559	1584	1563.3	155.492

TAB. 4.8 : Performances de l'optimisation par Essaim de Particules (PSO) pour 25 jobs dans un problème flowshop.

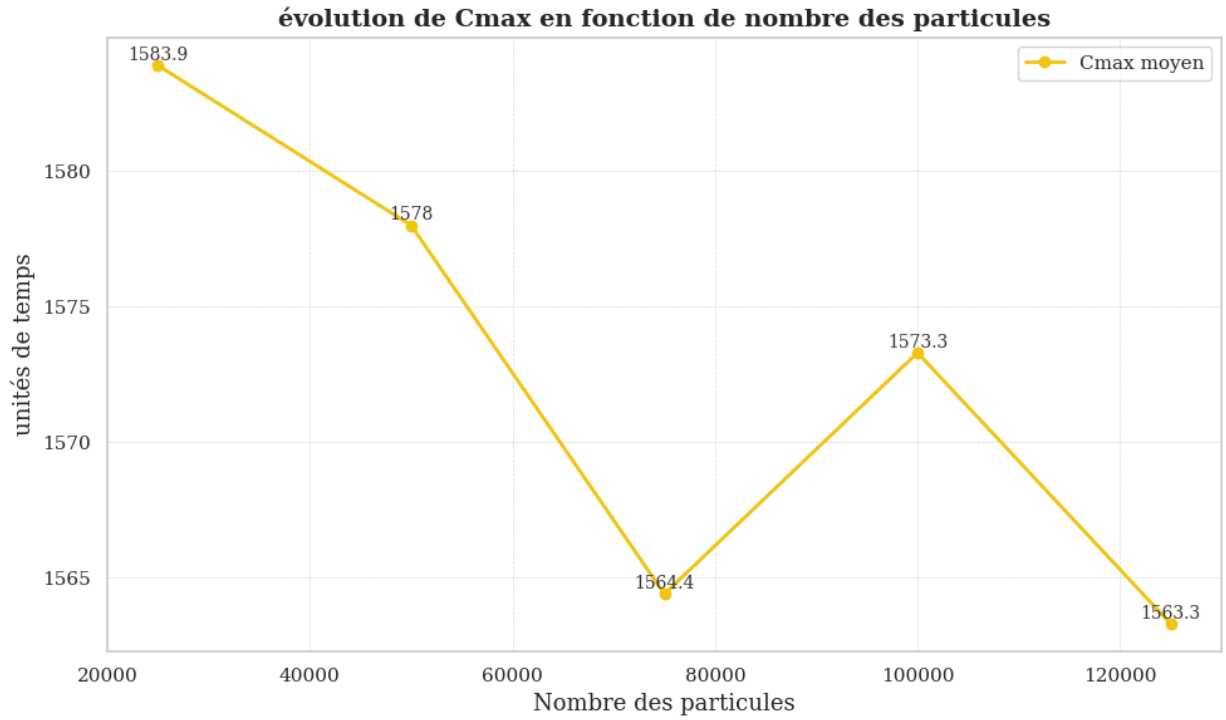

FIG. 4.8 : Évolution du makespan moyen (C_{max}) pour différentes tailles de nuées dans l'algorithme PSO, pour 25 jobs.

Interprétation des résultats

Le PSO est moins performant que les autres algorithmes (min C_{max} entre 1559 et 1581), avec des temps de calcul prohibitifs (36 à 155s). L'augmentation du nombre de particules n'améliore pas significativement les résultats, ce qui souligne ses limites pour ce type de problème.

Recherche tabou (TS)

Tenure	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
25	1513	1537	1517.7	36.956
50	1513	1522	1515.7	36.828
75	1513	1537	1520.5	38.777
100	1513	1537	1520.3	35.054
125	1513	1537	1516.5	32.220

TAB. 4.9 : Performances de la Recherche Tabou (TS) pour 25 jobs dans un problème flowshop.

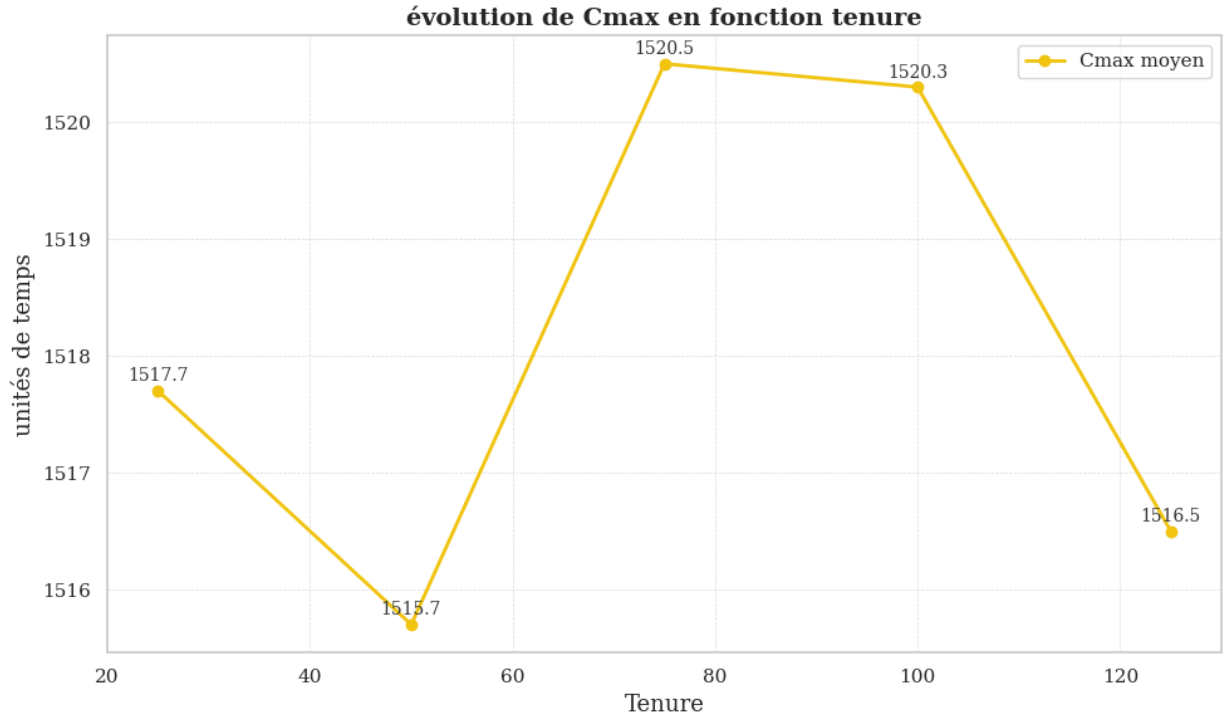

FIG. 4.9 : Évolution du makespan moyen (Cmax) en fonction de la taille de la mémoire (tenure) dans la méthode Tabu Search (TS), pour 25 jobs.

Interprétation des résultats

La TS atteint le meilleur makespan (1513) mais avec une variabilité accrue (max C_{max} jusqu'à 1537). Son temps d'exécution (35s) est raisonnable, bien que supérieur à celui du GA. La tenure a peu d'impact sur la qualité, mais une valeur de 50 semble optimale.

Recuit simulé (SA)

Initial temp	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
25	1513	1514	1513.1	25.997
50	1513	1513	1513.0	30.007
75	1513	1513	1513.0	30.174
100	1513	1513	1513.0	31.708
125	1513	1513	1513.0	32.815

TAB. 4.10 : Performances du recuit simulé (SA) pour 25 jobs dans un problème flowshop.

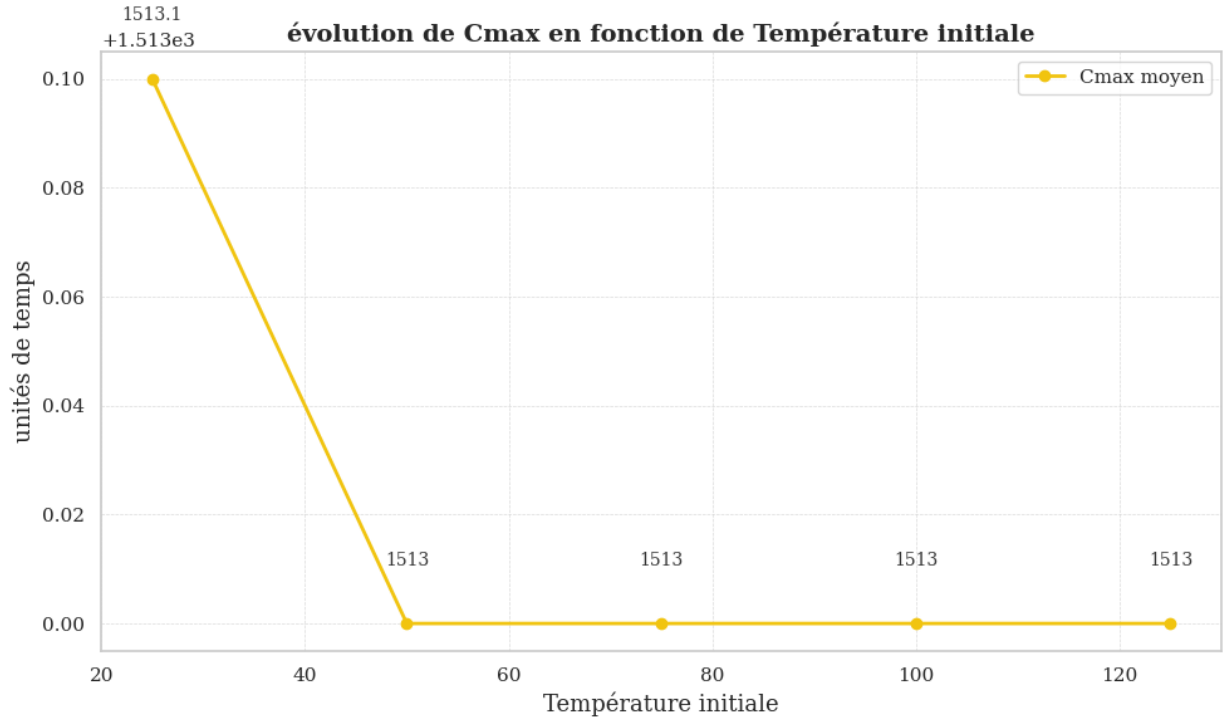

FIG. 4.10 : Évolution du makespan moyen (C_{max}) pour différentes températures initiales avec l'algorithme de recuit simulé (SA), pour 25 jobs.

Interprétation des résultats

Le SA obtient des résultats parfaits (min C_{max} = 1513 pour toutes les températures) mais avec des temps d'exécution élevés (25 à 33s). Contrairement au cas des 10 jobs, l'hybridation GA-SA réduit significativement le temps (16s vs 25s) tout en conservant l'optimalité, ce qui renforce son intérêt pour les instances plus grandes.

Hybridation GA & SA

Initial temp	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
25	1513	1513	1513	16.132
50	1513	1513	1513	15.226
75	1513	1513	1513	16.368
100	1513	1513	1513	16.587
125	1513	1513	1513	16.689

TAB. 4.11 : Performances de l'hybridation GA–SA pour 25 jobs dans un problème flowshop.

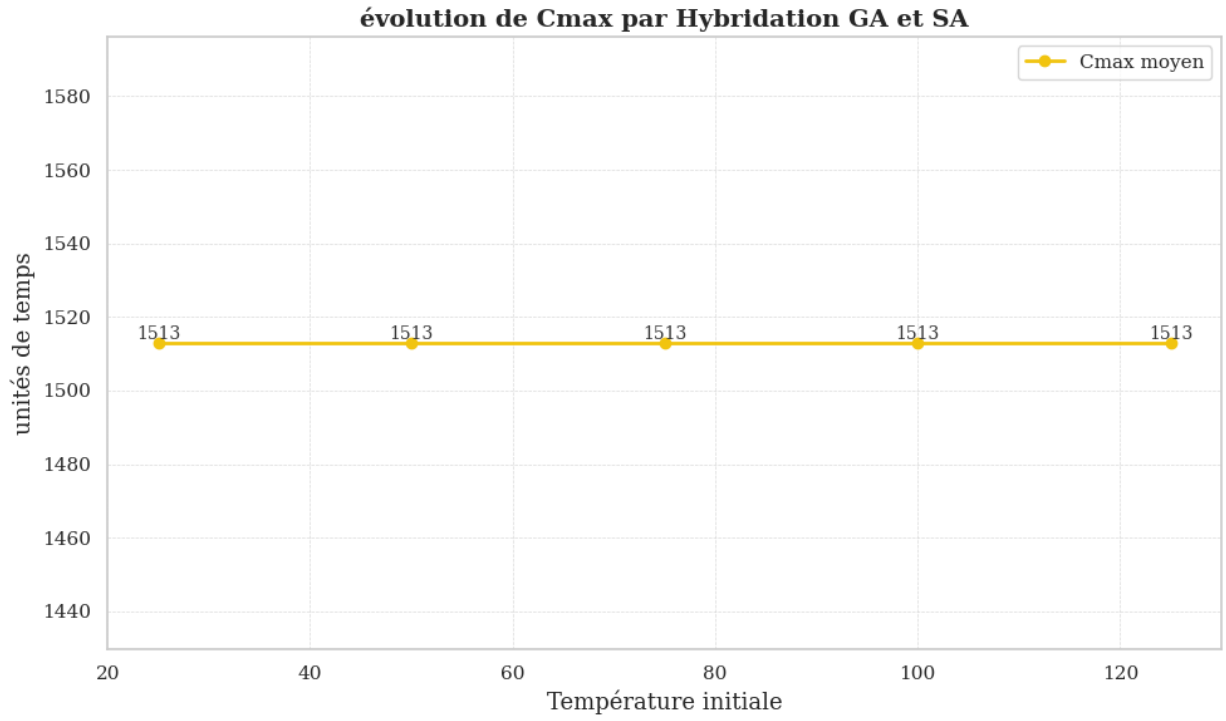

FIG. 4.11 : Évolution du makespan moyen (C_{max}) pour différentes tailles de population dans l'hybridation GA & SA, pour 25 jobs.

Interprétation des résultats

Cette combinaison est très efficace : elle conserve la perfection du SA (min C_{max} = 1513) tout en divisant par deux le temps d'exécution (16s vs 25s pour le SA seul). L'hybridation exploite la diversité du GA et la précision du SA, ce qui en fait une approche idéale pour cette taille d'instance.

Hybridation PSO & TS

Tenure	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
10	1513	1532	1517.3	786.506
20	1513	1525	1516.0	762.391
30	1513	1522	1518.0	783.117
40	1513	1522	1517.6	767.369
50	1513	1522	1515.1	775.158

TAB. 4.12 : Performances de l'hybridation PSO–TS pour 25 jobs dans un problème flowshop.

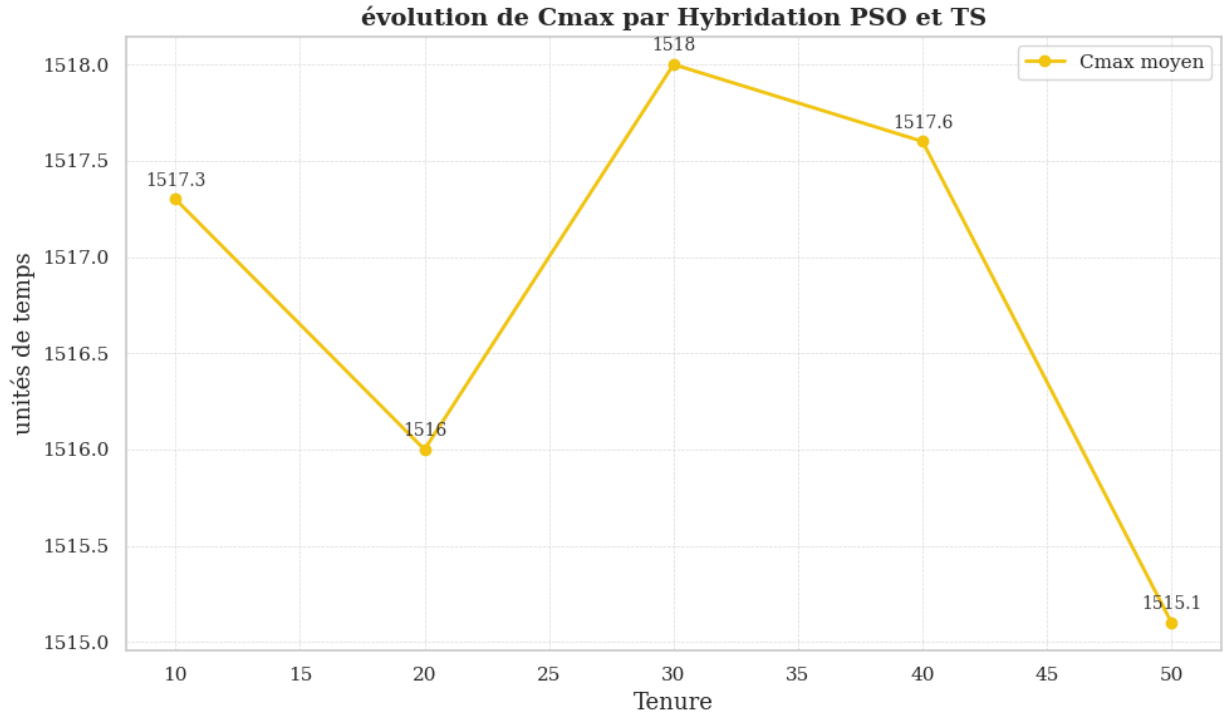

FIG. 4.12 : Évolution du makespan moyen (C_{max}) en fonction du paramètre de mémoire (tenure) dans l'hybridation PSO & TS, pour 25 jobs.

Interprétation des résultats

Peu convaincante : malgré un min C_{max} optimal (1513), les temps d'exécution (762–786s) sont extrêmement élevés, et l'avg C_{max} (1515–1518) n'est pas meilleure que celle de la TS seule. L'ajout du PSO alourdit inutilement le processus sans gain notable.

4.2.4 Instance à 75 jobs

Avec cette instance de grande taille, la résolution exacte devient difficilement envisageable. L'objectif ici est de tester la robustesse et la capacité des métaheuristiques à fournir des solutions satisfaisantes dans un délai raisonnable, face à un problème plus réaliste en termes de taille. La matrice utilisée pour créer les examples pour l'instance à 75 jobs est présenté dans la section A.4 de l'Annexe A.

L'algorithme génétique (GA)

Pop size	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
75	4203	4253	4209.5	27.750
150	4203	4262	4209.8	55.596
225	4203	4211	4203.8	107.481
300	4203	4203	4203.0	124.025
375	4203	4203	4203.0	180.340

TAB. 4.13 : Performances de l'algorithme génétique (GA) pour 75 jobs dans un problème flowshop.

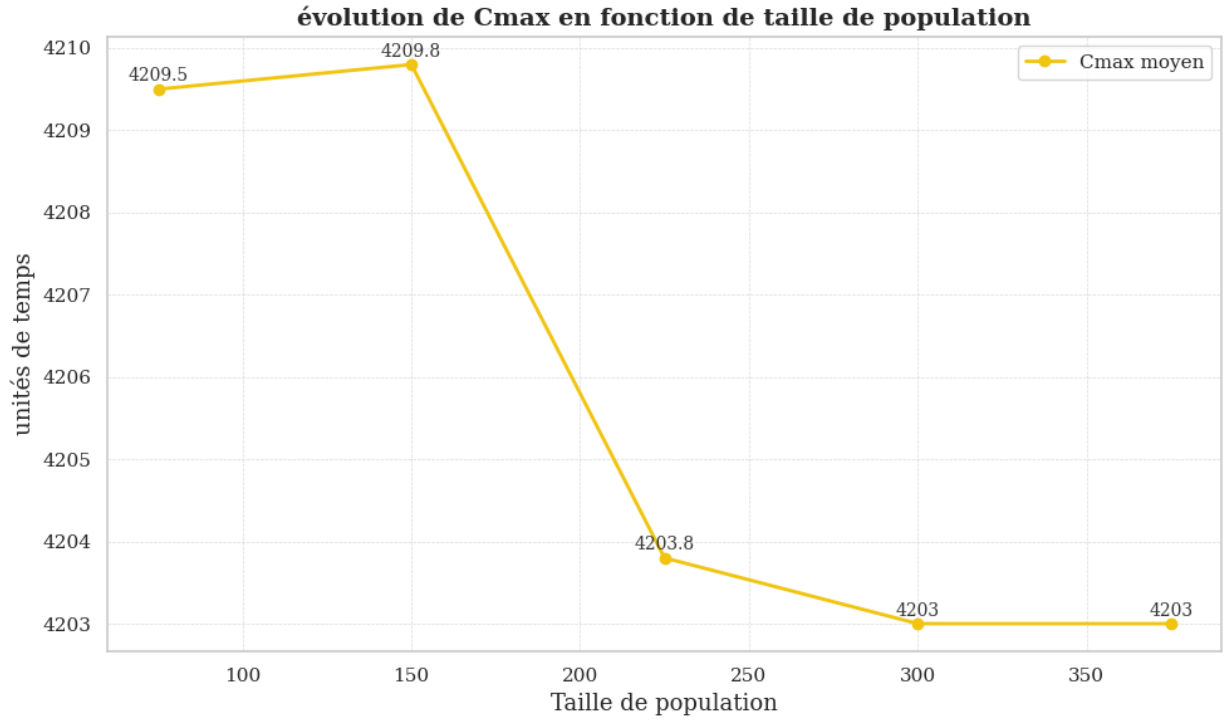

FIG. 4.13 : Évolution du makespan moyen (C_{max}) pour différentes tailles de population avec l'algorithme génétique (GA), pour 75 jobs.

Interprétation des résultats

Le GA atteint le makespan optimal (4203) à partir d'une population de 300, avec un temps d'exécution raisonnable (124s). L'augmentation de la population améliore la stabilité (avg C_{max} = 4203 pour 300+ individus), mais le temps de calcul augmente significativement (180s pour 375). Une taille de population de 300 semble être le meilleur compromis.

Optimisation par essaim de particules (PSO)

Num particles	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
75000	4271	4303	4281.0	243.360
150000	4280	4285	4281.0	492.875
225000	4241	4279	4267.6	735.776
300000	4241	4279	4267.6	735.776
375000	4261	4261	4261.0	1240.559

TAB. 4.14 : Performances de l'optimisation par essaim de particules (PSO) pour 75 jobs dans un problème flowshop.

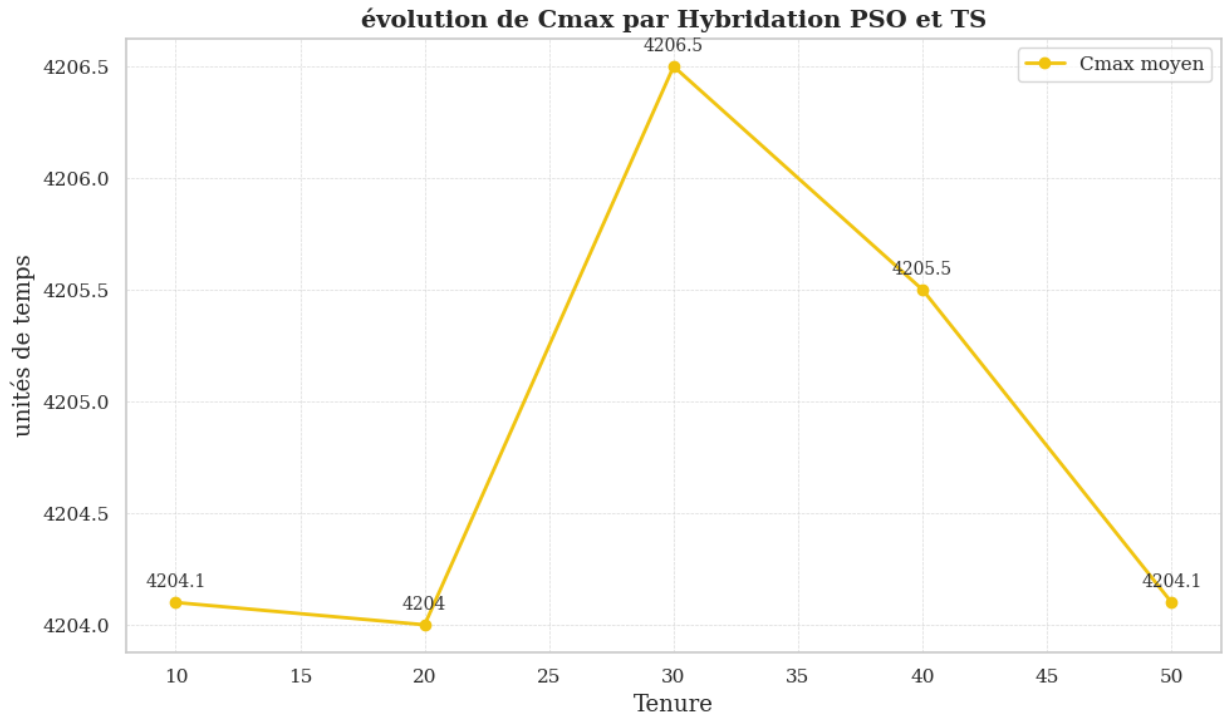

FIG. 4.14 : Évolution du makespan moyen (C_{max}) pour différentes tailles de nuées dans l'algorithme d'optimisation par essaim de particules (PSO), pour 75 jobs.

Interprétation des résultats

Le PSO présente des performances médiocres (min C_{max} entre 4241 et 4281) avec des temps d'exécution très élevés (243s à 1240s). L'augmentation de la taille de l'essaim n'améliore pas significativement les résultats, confirmant

les limites du PSO pour les problèmes de grande taille.

Recherche tabou (TS)

Tenure	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
10	4203	4208	4205.6	1895.834
20	4203	4213	4206.2	1948.344
30	4203	4208	4204.5	1725.698
40	4203	4208	4204.5	1514.832
50	4203	4213	4206.5	1581.715

TAB. 4.15 : Performances de la Recherche Tabou (TS) pour 75 jobs dans un problème flowshop.

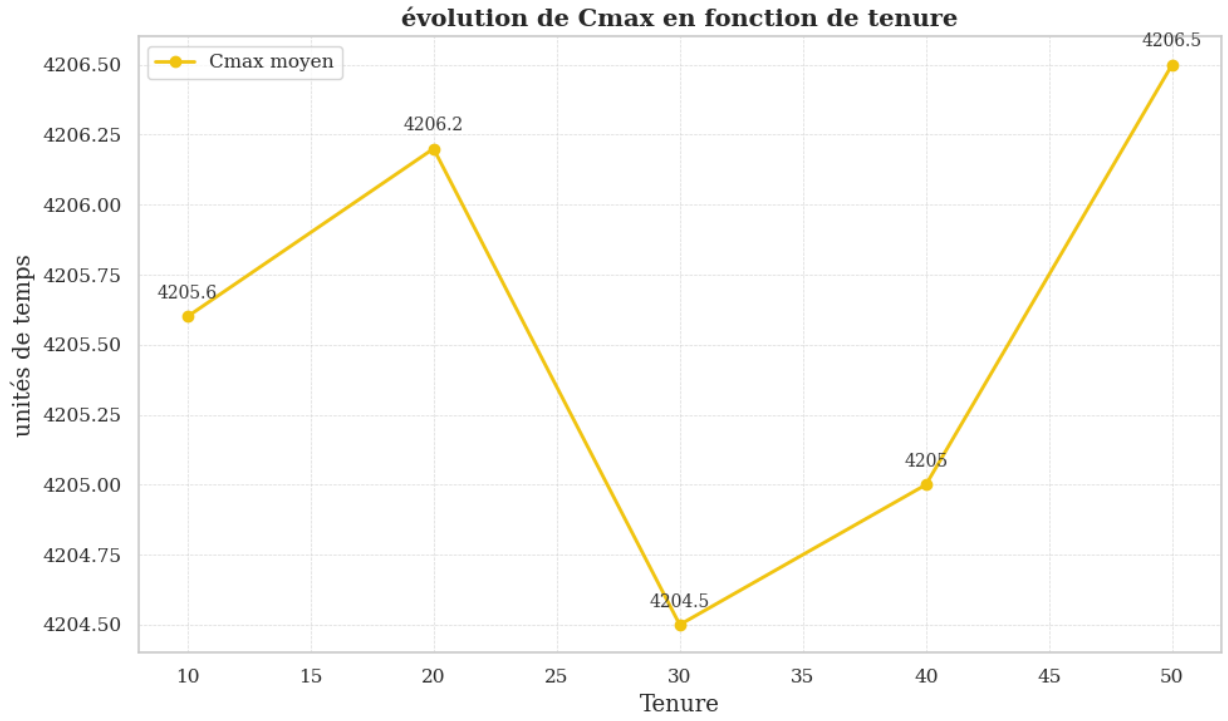

FIG. 4.15 : Évolution du makespan moyen (C_{max}) en fonction de la taille de la mémoire (tenure) dans la méthode Tabu Search (TS), pour 75 jobs

Interprétation des résultats

La TS obtient le makespan optimal (4203) mais avec des temps d'exécution extrêmement longs (1514s à 1948s). La variabilité des résultats (max C_{max} jusqu'à 4213) et les temps prohibitifs en font une option peu attractive pour cette instance.

Recuit simulé (SA)

Initial temp	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
75	4203	4203	4203	67.563
150	4203	4203	4203	73.132
225	4203	4203	4203	80.893
300	4203	4203	4203	83.001
375	4203	4203	4203	82.750

TAB. 4.16 : Performances du recuit simulé (SA) pour 75 jobs dans un problème flowshop.

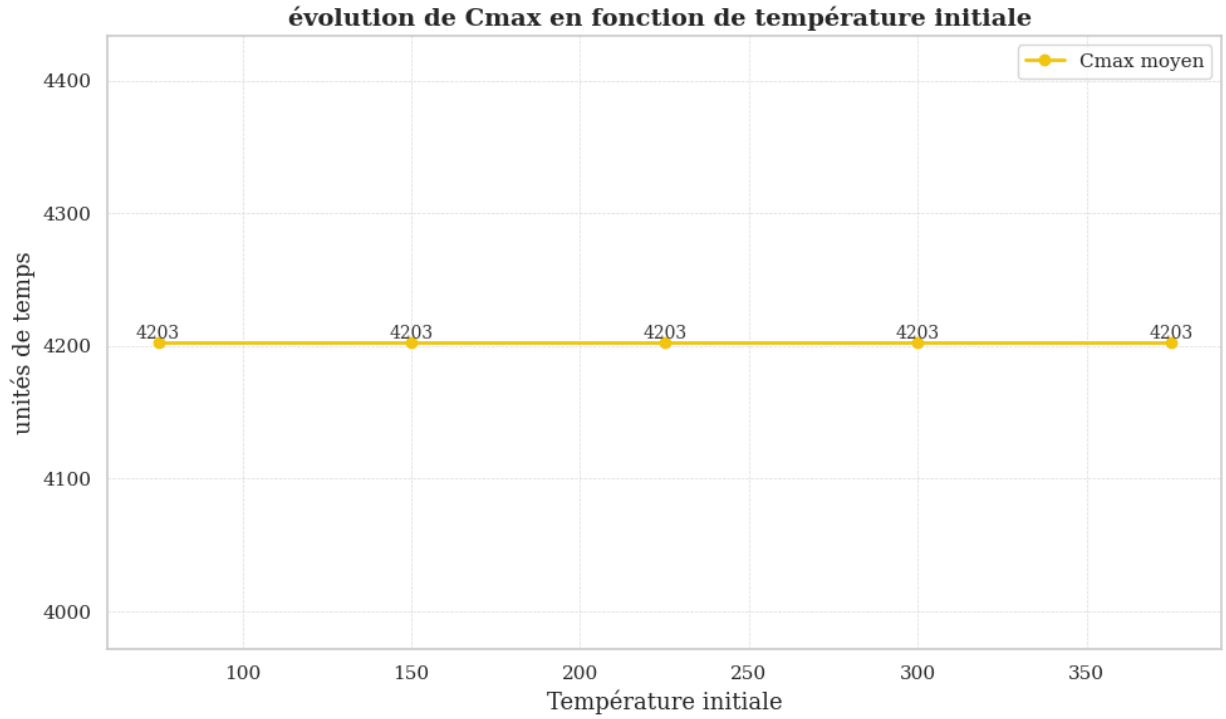

FIG. 4.16 : Évolution du makespan moyen (C_{max}) pour différentes températures initiales avec l'algorithme de recuit simulé (SA), pour 75 jobs.

Interprétation des résultats

Le SA se distingue par des résultats parfaits (min C_{max} = 4203) et des temps d'exécution modérés (67s à 83s). Sa robustesse et son efficacité en font l'un des meilleurs choix pour cette taille de problème.

Hybridation GA & SA

Initial temp	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
75	4203	4203	4203	49.983
150	4203	4203	4203	54.936
225	4203	4203	4203	55.698
300	4203	4203	4203	56.263
375	4203	4203	4203	56.891

TAB. 4.17 : Performances de l'hybridation GA–SA pour 75 jobs dans un problème flowshop.

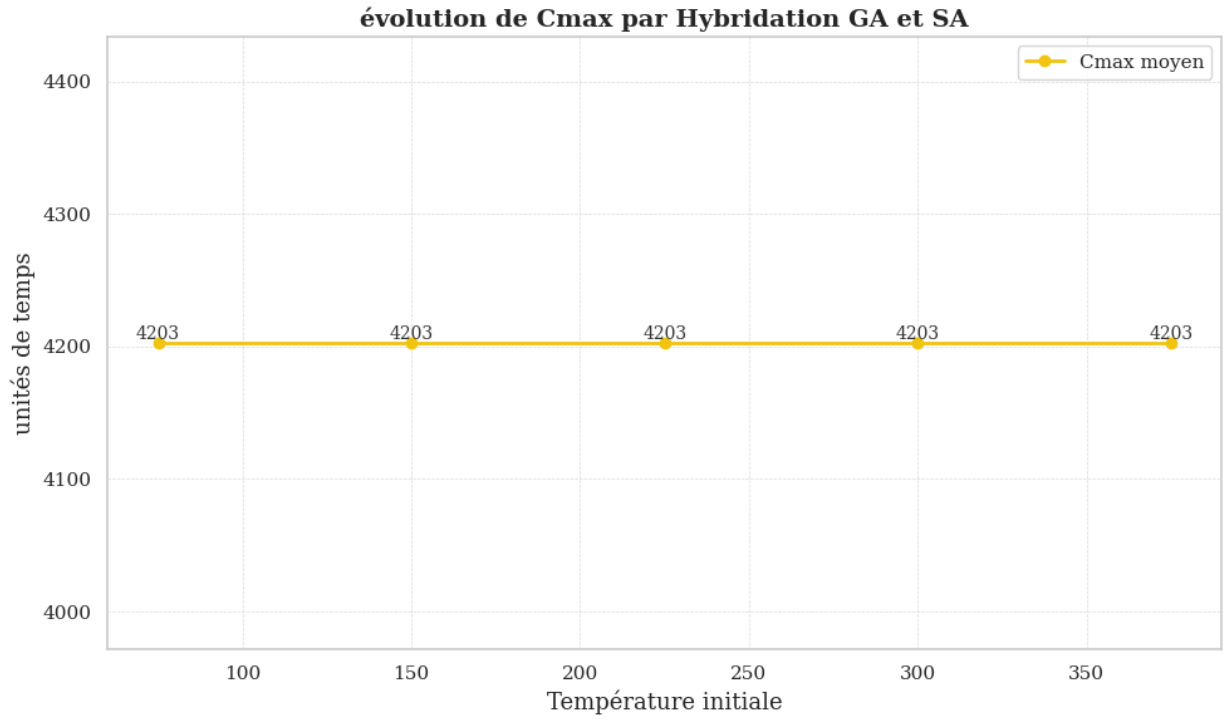

FIG. 4.17 : Évolution du makespan moyen (C_{max}) pour différentes tailles de population dans l'hybridation GA & SA, pour 75 jobs.

Interprétation des résultats

Cette combinaison conserve l'optimalité du SA (min C_{max} = 4203) tout en réduisant le temps d'exécution (49s à 57s vs 67s à 83s pour le SA seul). L'efficacité de cette hybridation est démontrée, combinant rapidité et précision.

Hybridation PSO & TS

Tenure	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
10	4203	4208	4204.1	12562.381
20	4203	4208	4204.1	9561.336
30	4203	4208	4206.5	9579.123
40	4203	4208	4205.5	9826.209
50	4203	4208	4204.1	9855.652

TAB. 4.18 : Performances de l'hybridation PSO–TS pour 75 jobs dans un problème flowshop.

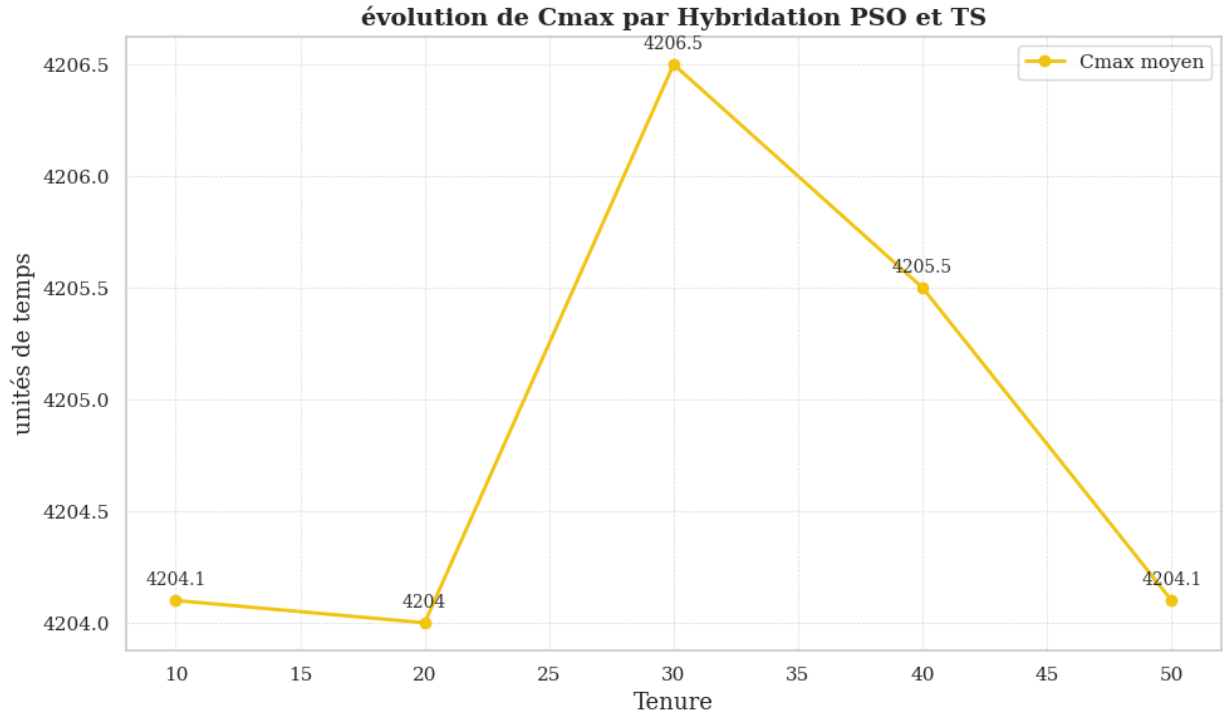

FIG. 4.18 : Évolution du makespan moyen (C_{max}) en fonction du paramètre de mémoire (tenure) dans l'hybridation PSO & TS, pour 75 jobs.

Interprétation des résultats

Les résultats sont décevants : bien que le min C_{max} soit optimal (4203), les temps d'exécution sont prohibitifs (9561s à 12562s) et l'avg C_{max} (4204 à 4206.5) n'est pas meilleure que celle de la TS seule. Cette hybridation n'apporte aucun avantage pour cette instance.

4.2.5 Instance à 200 jobs

Cette instance de très grande taille simule un cas extrême, destiné à pousser les algorithmes à leurs limites. Elle permet d'observer leur comportement en termes de convergence, de stabilité et de qualité de solutions sur des problèmes industriels de grande envergure. La matrice utilisée pour créer les examples pour l'instance à 200 jobs est présenté dans la section A.7 de l'Annexe A.

L'algorithme génétique (GA)

Pop size	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
200	10706	10741	10716.7	258.732
400	10706	10707	10706.1	535.961
600	10706	10722	10707.7	784.826
800	10706	10707	10706.1	1026.729
1000	10706	10707	10706.1	1299.040

TAB. 4.19 : Performances de l'algorithme génétique (GA) pour 200 jobs dans un problème flowshop.

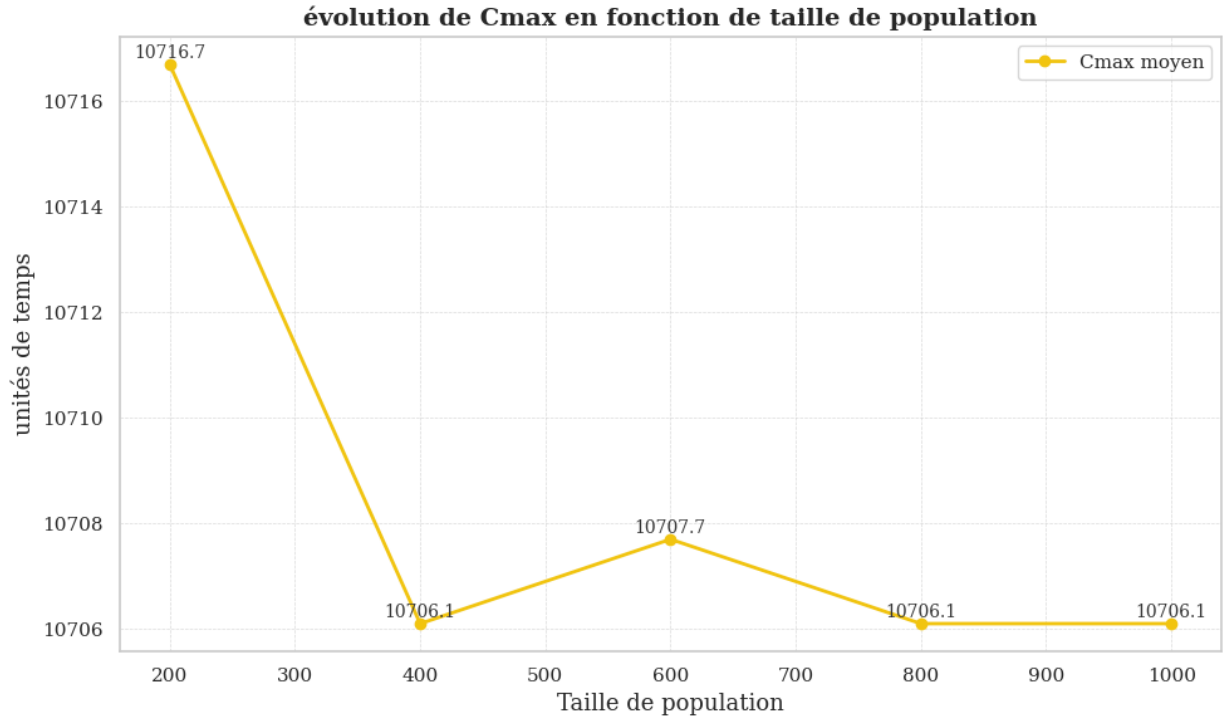

FIG. 4.19 : Évolution du makespan moyen (C_{max}) pour différentes tailles de population avec l'algorithme génétique (GA), pour 200 jobs.

Interprétation des résultats

L'algorithme génétique atteint un makespan minimum de 10706 pour toutes tailles de population (200-1000), avec un makespan moyen de 10706.1 à 10716.7 et une bonne stabilité (écart maximal de 35 unités). Le temps d'exécution varie de 258.732s à 1299.04s, augmentant avec la taille de la population. GA est efficace pour des populations petites, mais des tailles plus grandes n'améliorent pas les résultats.

Optimisation par essaim de particules (PSO)

Num particle	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
200000	10809	10876	10822.4	1770.416
400000	10832	10842	10834.0	3262.169
600000	10816	10816	10816.0	5353.516
800000	10820	10830	10820.0	6930.236
1000000	10854	10876	10859.6	1694.616

TAB. 4.20 : Performances de l'optimisation par essaim de particules (PSO) pour 200 jobs dans un problème flowshop.

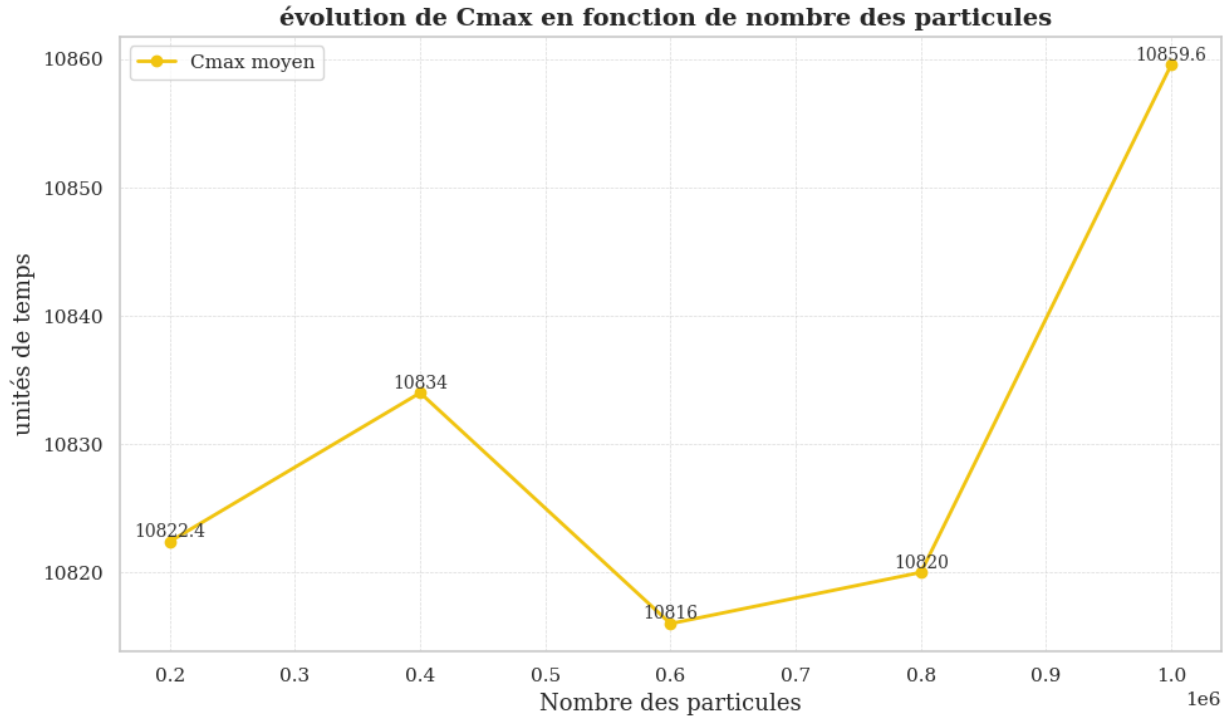

FIG. 4.20 : Évolution du makespan moyen (C_{max}) pour différentes tailles de nuées dans l'algorithme d'optimisation par essaim de particules (PSO), pour 200 jobs.

Interprétation des résultats

L'Optimisation par essaim de particules (PSO) obtient un makespan minimum de 10809 (200000 particules), un makespan moyen de 10816 à 10859.6, et une stabilité modérée (écart jusqu'à 67 unités). Le temps d'exécution, de 1694.616s à 6930.236s, est élevé. La méthode PSO est moins performante, avec des solutions sous-optimales et un coût de calcul important.

Recherche tabou (TS)

Tenure	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
10	10706	107040	10712.4	19082.416
20	10706	10808	10716.2	19083.260
30	10706	10706	10706.0	19319.526
40	10706	10706	10706.0	19142.006
50	10706	107021	10707.5	19020.451

TAB. 4.21 : Performances de la recherche tabou (TS) pour 200 jobs dans un problème flowshop.

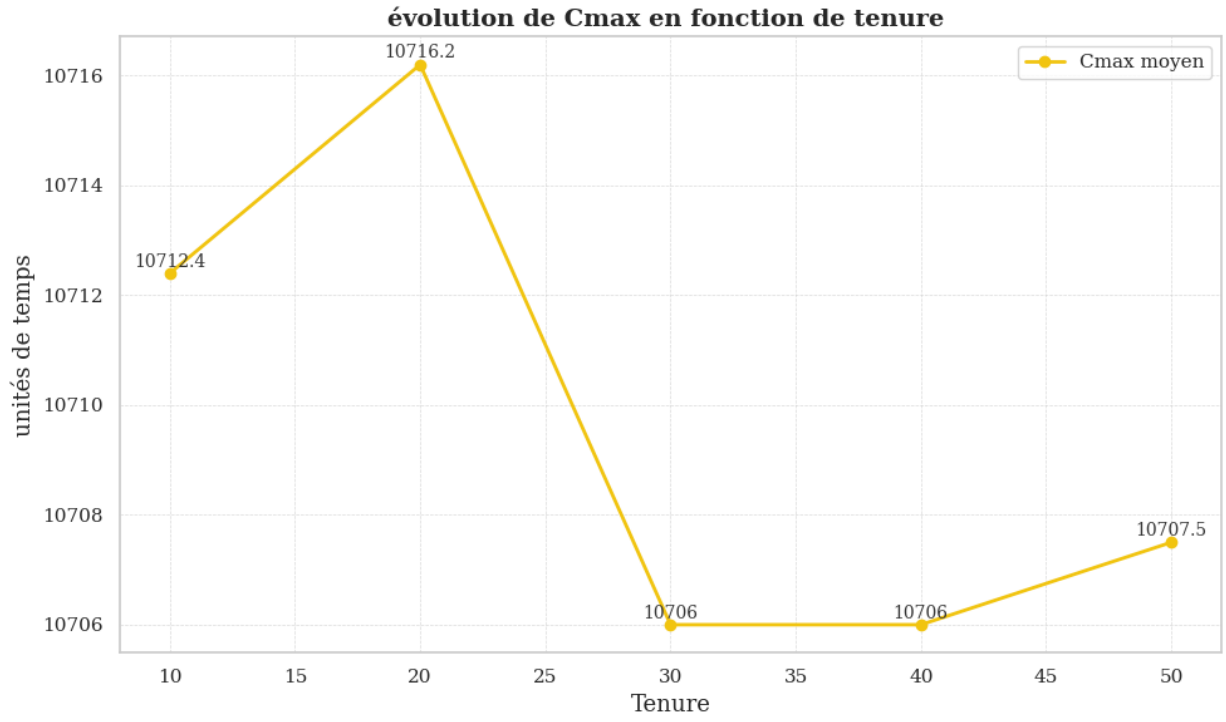

FIG. 4.21 : Évolution du makespan moyen (C_{max}) en fonction de la taille de la mémoire (tenure) dans la méthode Tabu Search (TS), pour 200 jobs.

Interprétation des résultats

La Recherche Tabou atteint un makespan minimum de 10706 pour toutes tenures (10-50), optimal à tenure 30 et 40 (stabilité parfaite). Des anomalies (max C_{max} = 107040 pour tenure 10) suggèrent des erreurs. Le temps d'exécution (19020s-19319s) est prohibitif, limitant l'applicabilité de TS malgré ses solutions optimales.

Recuit simulé (SA)

TAB. 4.22 : Performances du recuit simulé (SA) pour 200 jobs dans un problème flowshop.

Initial temp	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
200	10706	10706	10706.0	242.847
400	10706	10707	10706.1	247.001
600	10706	10706	10706.0	271.329
800	10706	10706	10706.0	268.502
1000	10706	10707	10706.2	276.563

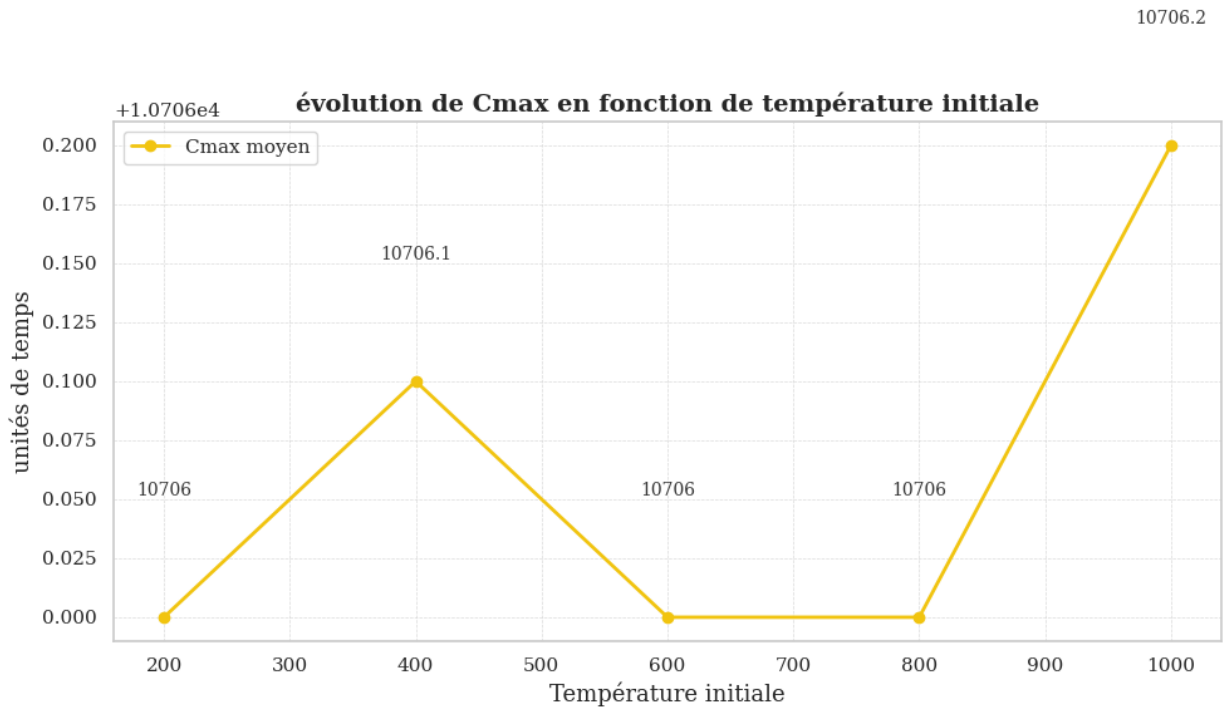

FIG. 4.22 : Évolution du makespan moyen (C_{max}) pour différentes températures initiales avec l'algorithme de recuit simulé (SA), pour 200 jobs.

Interprétation des résultats

Le Recuit Simulé atteint un makespan minimum de 10706 pour toutes températures (200-1000), avec un makespan moyen de 10706 à 10706.2 et une stabilité excellente (écart maximal d'une unité). Le temps d'exécution, de 242.847s

à 276.563s, est faible. SA est rapide et efficace, idéale pour ce problème.

Hybridation GA & SA

Initial temp	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
200	10706	10706	10706	219.523
400	10706	10706	10706	227.825
600	10706	10706	10706	229.045
800	10706	10706	10706	226.332
1000	10706	10706	10706	229.650

TAB. 4.23 : Performances de l'hybridation GA–SA pour 200 jobs dans un problème flowshop.

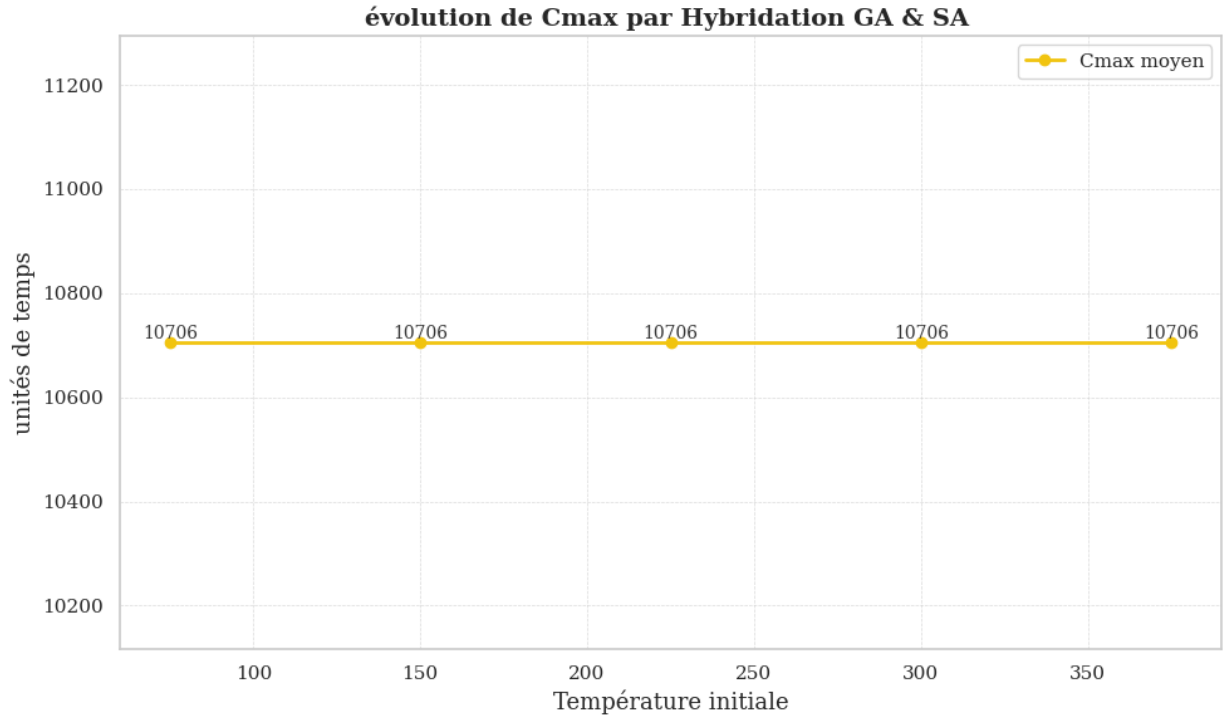

FIG. 4.23 : Évolution du makespan moyen (C_{max}) pour différentes tailles de population dans l'hybridation GA & SA, pour 200 jobs.

Interprétation des résultats

L'hybridation GA-SA atteint un makespan constant de 10706 (min, max, moyen) pour toutes températures (200-1000), avec une stabilité parfaite. Le temps d'exécution, de 219.523s à 229.65s, est le plus bas. GA-SA est la méthode la plus performante, combinant rapidité et fiabilité.

Hybridation PSO & TS

Tenure	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
10	10706	10707	10706.1	9694.834
20	10706	10706	10706.0	9699.494
30	10706	10740	10709.4	9701.220
40	10706	10733	10708.7	9698.065
50	10706	10706	10706.0	9735.429

TAB. 4.24 : Performances de l'hybridation PSO–TS pour 200 jobs dans un problème flowshop.

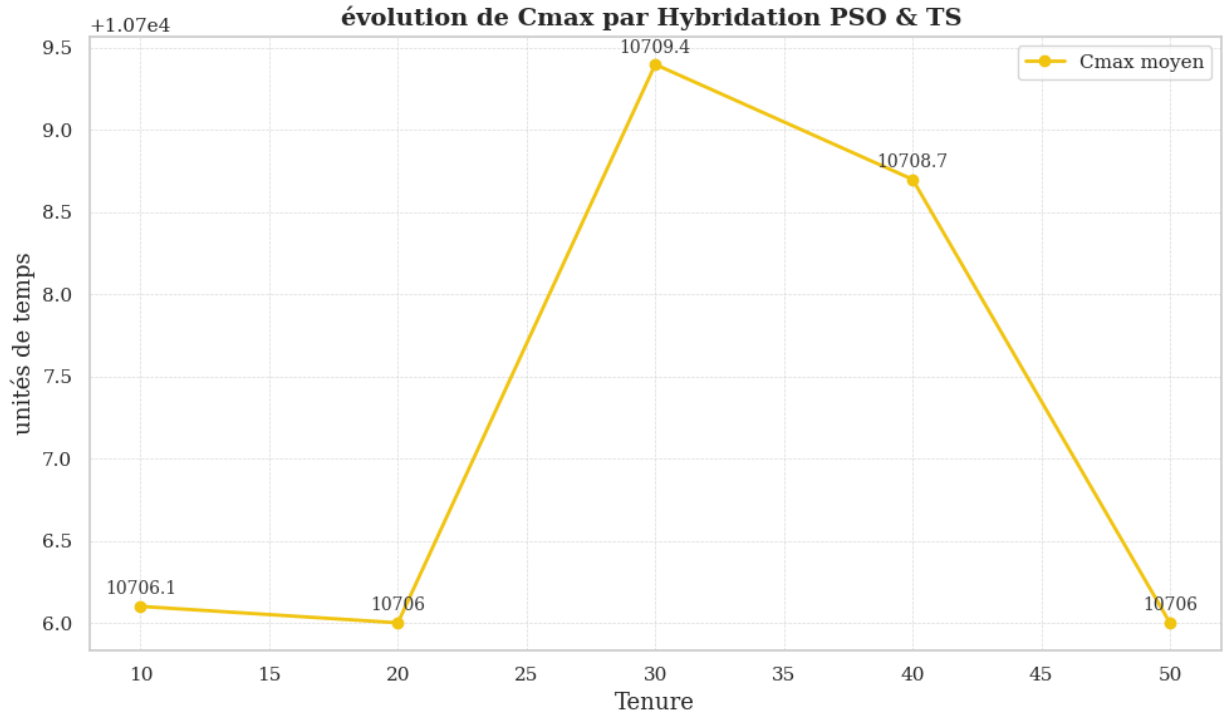

FIG. 4.24 : Évolution du makespan moyen (C_{max}) en fonction du paramètre de mémoire (tenure) dans l'hybridation PSO & TS, pour 200 jobs.

Interprétation des résultats

L'hybridation PSO-TS atteint un makespan minimum de 10706 pour toutes tenures (10-50), avec un makespan moyen de 10706 à 10709.4 et une bonne stabilité (écart maximal de 34 unités). Le temps d'exécution (9694s-9735s) reste élevé. PSO-TS améliore PSO, mais est moins compétitive que GA-SA ou SA.

4.3 Analyse comparative

Dans cette section, nous procédons à une analyse comparative approfondie des performances des différentes métaheuristiques exécutées dans le cadre de cette étude, à savoir : l'algorithme génétique (GA), le recuit simulé (SA), la recherche tabou (TS), l'optimisation par essaim de particules (PSO), l'hybridation GA & SA et l'hybrydation PSO & TS. Cette comparaison s'appuie principalement sur deux indicateurs clés : le makespan moyen (C_{max}) obtenu pour chaque instance testée et le temps de calcul requis par chaque algorithme.

Les résultats ont été organisés selon trois catégories d'instances, représentant respectivement les cas de petite, moyenne et grande taille (en termes de nombre de jobs et de machines). Pour chaque catégorie, des histogrammes ont été générés afin de visualiser de manière claire et synthétique les performances relatives des différentes métaheuristiques. Ces graphiques permettent non seulement de comparer les valeurs de C_{max}, mais également d'évaluer l'efficacité computationnelle des méthodes testées.

C_{max} moyen

Le C_{max} moyen, calculé sur dix itérations, permet de comparer la qualité des solutions produites par chaque métaheuristique. Les résultats mettent en évidence que certaines approches parviennent à générer des ordonnancements plus efficaces, notamment pour les instances de taille moyenne et grande. Cet indicateur est crucial pour juger de la performance globale des algorithmes en termes d'optimisation.

Temps de calcul moyen

Le temps de calcul moyen, obtenu sur dix itérations pour chaque métaheuristique, permet d'évaluer l'efficacité en termes de rapidité d'exécution. Les résultats montrent que certains algorithmes restent performants même pour les grandes instances, tandis que d'autres voient leur temps de traitement augmenter significativement. Ce critère s'avère déterminant dans les contextes où la réactivité est primordiale.

Instance à 10 jobs

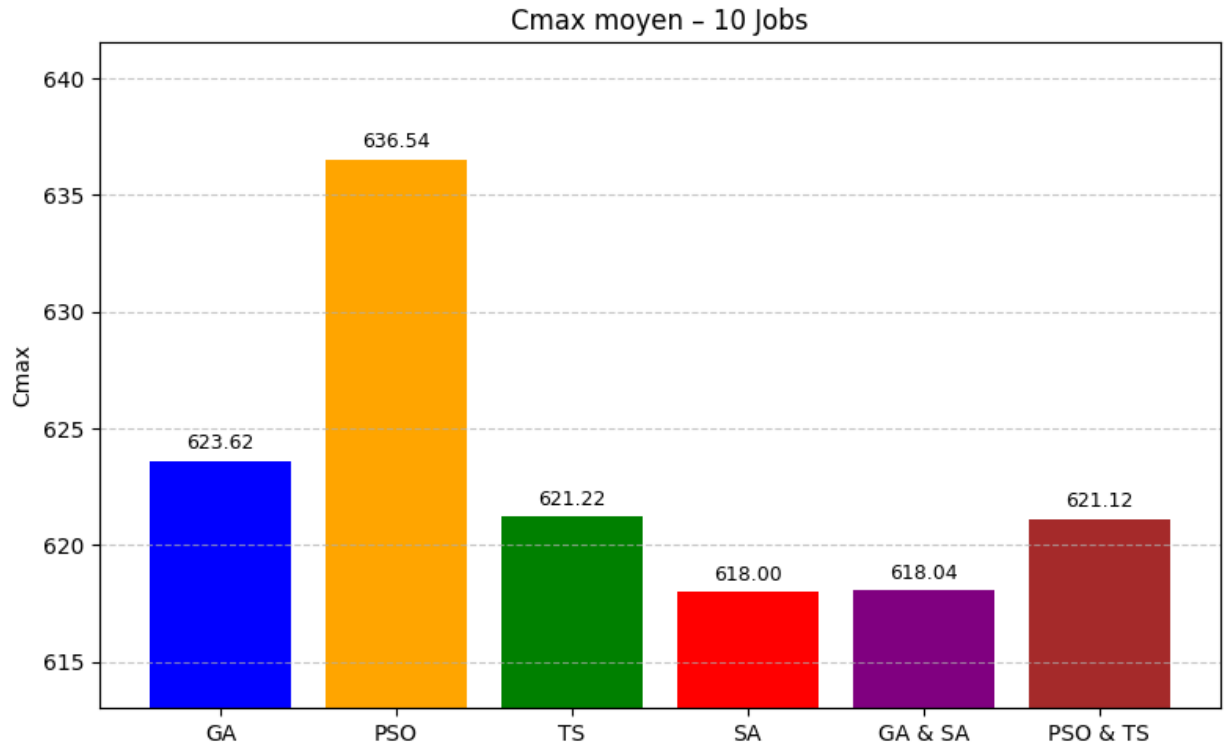

FIG. 4.25 : Histogramme comparatif des makespans moyens obtenus par chaque métaheuristique sur 10 jobs.

Analyse comparative

L'algorithme du recuit simulé (SA) obtient la meilleure performance avec un C_{max} moyen de 618, suivi de près par l'hybride GA/SA (618.04) et la recherche tabou (TS) (621.22). Ces résultats démontrent l'efficacité des approches combinant recherche locale et mécanismes d'évitement de minima locaux. Les algorithmes génétique (GA) (623.62) et l'optimisation par essaim de particules (PSO) (636.54) présentent des performances légèrement inférieures, ce qui pourrait s'expliquer par leur nature populationnelle nécessitant plus d'itérations pour converger sur des problèmes de cette taille. L'hybride PSO/TS (621.12) montre des résultats comparables au TS seul, confirmant l'intérêt potentiel des approches hybrides.

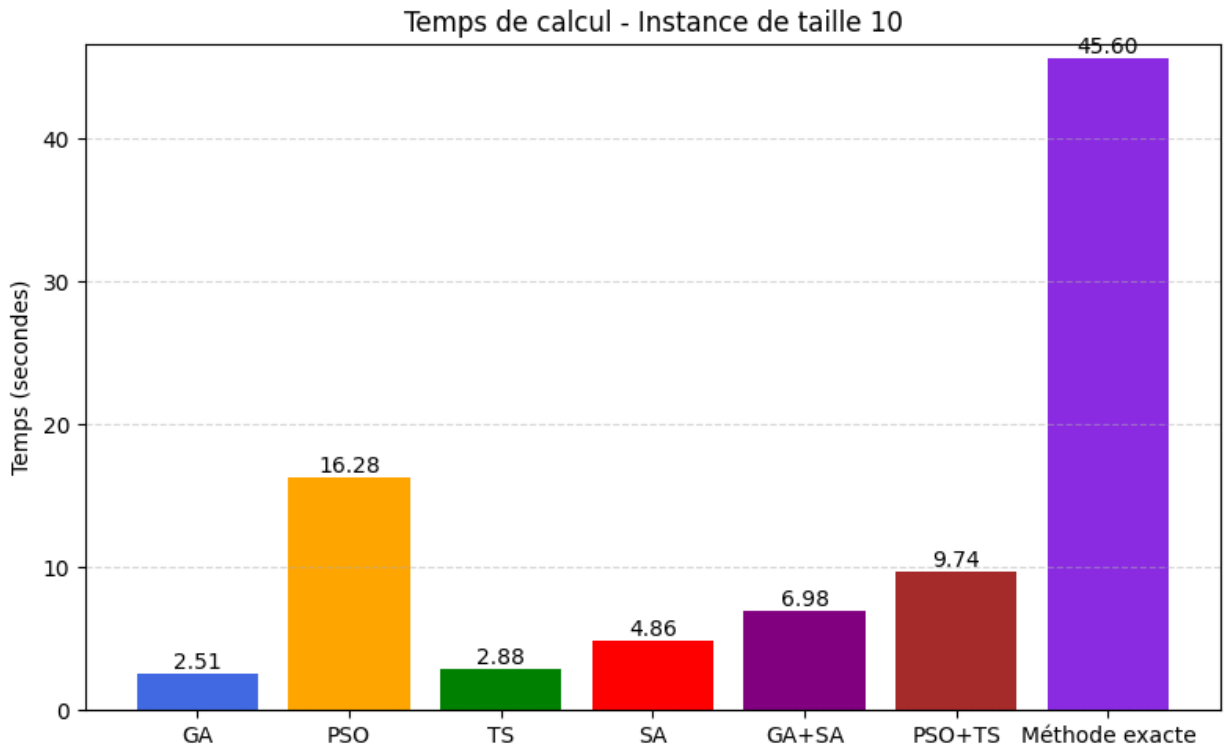

FIG. 4.26 : Histogramme comparatif des temps de calcul moyens obtenus pour chaque métaheuristique sur un ensemble de 10 jobs

Analyse comparative

L'algorithme génétique (GA) se révèle le plus rapide (2.51s), suivi de la recherche tabou (TS) (2.88s), faisant de ces deux approches les plus efficaces pour des applications nécessitant des temps de réponse courts. Le recuit simulé (SA) présente un temps intermédiaire (4.86s), tandis que l'hybride GA/SA (6.98s) montre qu'une combinaison d'algorithmes peut maintenir des temps raisonnables. En revanche, l'optimisation par essaim de particules (PSO) (16.28s) et surtout l'hybride PSO/TS (45.60s) s'avèrent significativement plus lents, ce qui limite leur utilisation dans des contextes où le temps de calcul est critique.

Instance à 25 jobs

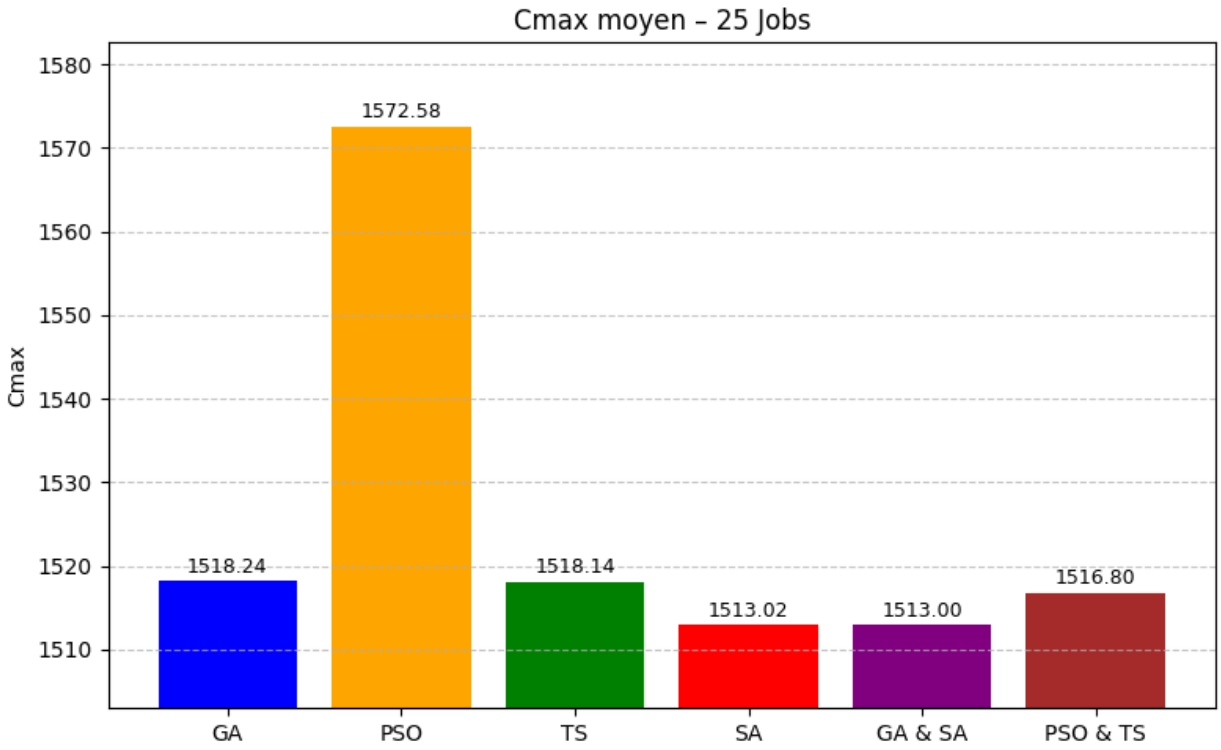

FIG. 4.27 : Histogramme comparatif des makespans moyens obtenus par chaque métaheuristique sur 25 jobs.

Analyse comparative

L'examen des résultats pour l'instance plus conséquente de 25 jobs révèle des tendances similaires mais amplifiées par rapport à l'instance précédente. L'algorithme du recuit simulé (SA) confirme sa supériorité avec un C_{max} moyen de 1513.02, suivi de très près par l'hybride GA/SA (1513.00), démontrant ainsi la robustesse de ces approches pour des problèmes de plus grande taille. La recherche tabou (TS) (1518.14)) et l'algorithme génétique (GA) (1518.24) présentent des performances quasi-équivalentes, tandis que l'hybride PSO/TS (1516.80) se positionne légèrement mieux que l'optimisation par essaim de particules (PSO) seul (1572.58). Ces résultats confirment que les méthodes combinant recherche locale et mécanismes d'optimisation globale tendent à offrir les meilleures solutions pour des problèmes de cette envergure.

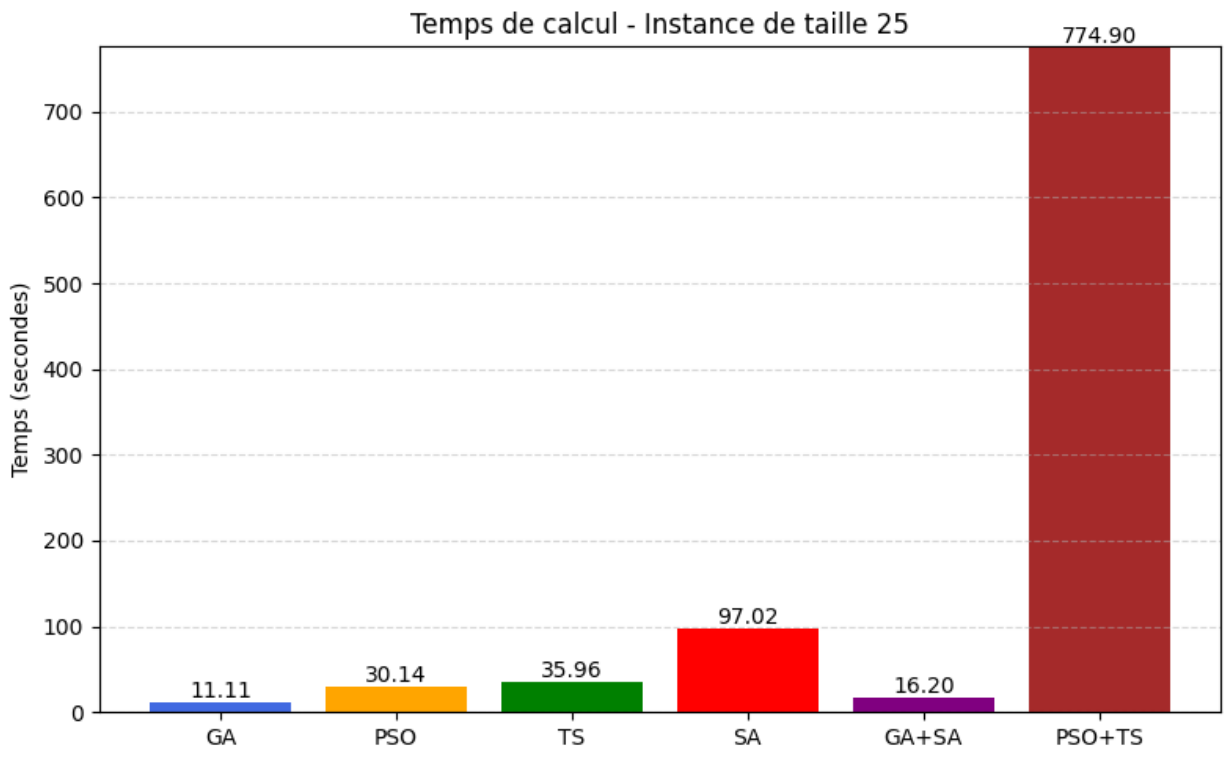

FIG. 4.28 : Histogramme comparatif des temps de calcul moyens obtenus pour chaque métaheuristique sur un ensemble de 25 jobs.

Analyse comparative

L'analyse des temps d'exécution met en lumière l'impact croissant de la taille du problème sur les performances de calcul. L'algorithme génétique (GA) conserve son avantage en termes de rapidité (11.11s), bien que son temps de calcul ait significativement augmenté par rapport à l'instance plus petite. La recherche tabou (TS) (35.96s) et l'optimisation par essaim de particules (PSO) (30.14s) présentent des temps d'exécution raisonnables, tandis que le recuit simulé (SA) (97.02s) voit son temps de calcul multiplié, reflétant la complexité accrue du problème. Les approches hybrides montrent des comportements divergents : GA/SA (16.20s) maintient un temps de calcul compétitif, alors que PSO/TS (774.9s) devient prohibitif, suggérant que certaines combinaisons algorithmiques peuvent entraîner des surcoûts computationnels insoutenables pour des problèmes de cette taille.

Instance à 75 jobs

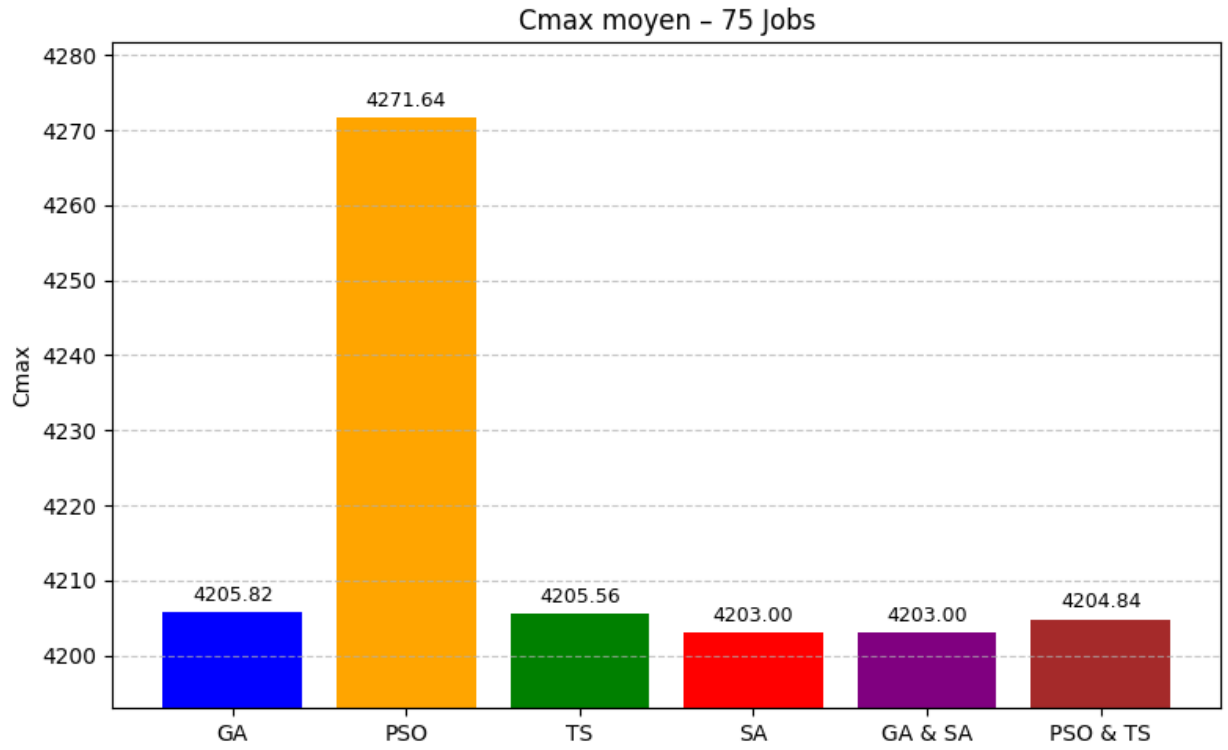

FIG. 4.29 : Histogramme comparatif des makespans moyens obtenus par chaque métaheuristique sur 75 jobs.

Analyse comparative

Pour cette instance de grande taille (75 jobs), les résultats confirment certaines tendances observées précédemment tout en révélant des comportements nouveaux. L'algorithme du recuit simulé (SA) et son hybride GA/SA obtiennent conjointement les meilleures performances avec un C_{max} moyen identique de 4203.00, démontrant une remarquable stabilité face à l'augmentation de la complexité du problème. Les approches de recherche tabou (TS) (4205.56) et l'algorithme génétique (GA) (4205.82) présentent des résultats très proches, avec un écart négligeable, tandis que l'hybride PSO/TS (4204.84) se positionne légèrement mieux que ces méthodes individuelles. L'optimisation par essaim de particules (PSO) seul (4271.64) montre à nouveau les performances les moins convaincantes, confirmant sa moindre adaptabilité aux problèmes de grande taille dans cette configuration.

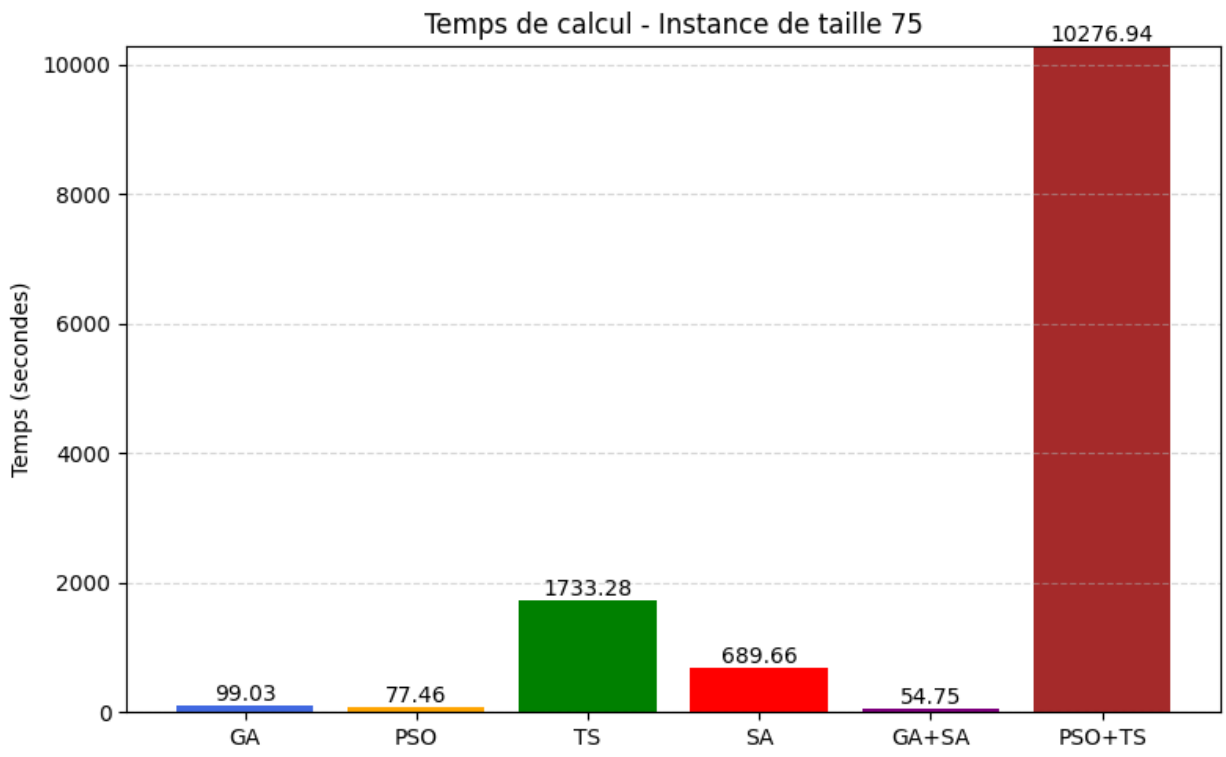

FIG. 4.30 : Histogramme comparatif des temps de calcul moyens obtenus pour chaque métaheuristique sur un ensemble de 75 jobs.

Analyse comparative

L'analyse des temps d'exécution révèle des écarts spectaculaires entre les différentes approches. De manière surprenante, PSO seul (77.46s) et GA (99.03s) présentent les temps les plus raisonnables, inversant la tendance observée sur les instances plus petites. L'hybride GA/SA (45.75s) se distingue particulièrement par son efficacité de calcul inattendue. À l'opposé, TS seul (28.88 min) et surtout SA seul (11.49 min) voient leurs temps de calcul s'envoler, tandis que l'hybride PSO/TS (171.28 min, soit près de 3 heures) devient totalement impraticable, révélant les limites de certaines combinaisons algorithmiques face à des problèmes de très grande taille.

Instance à 200 jobs

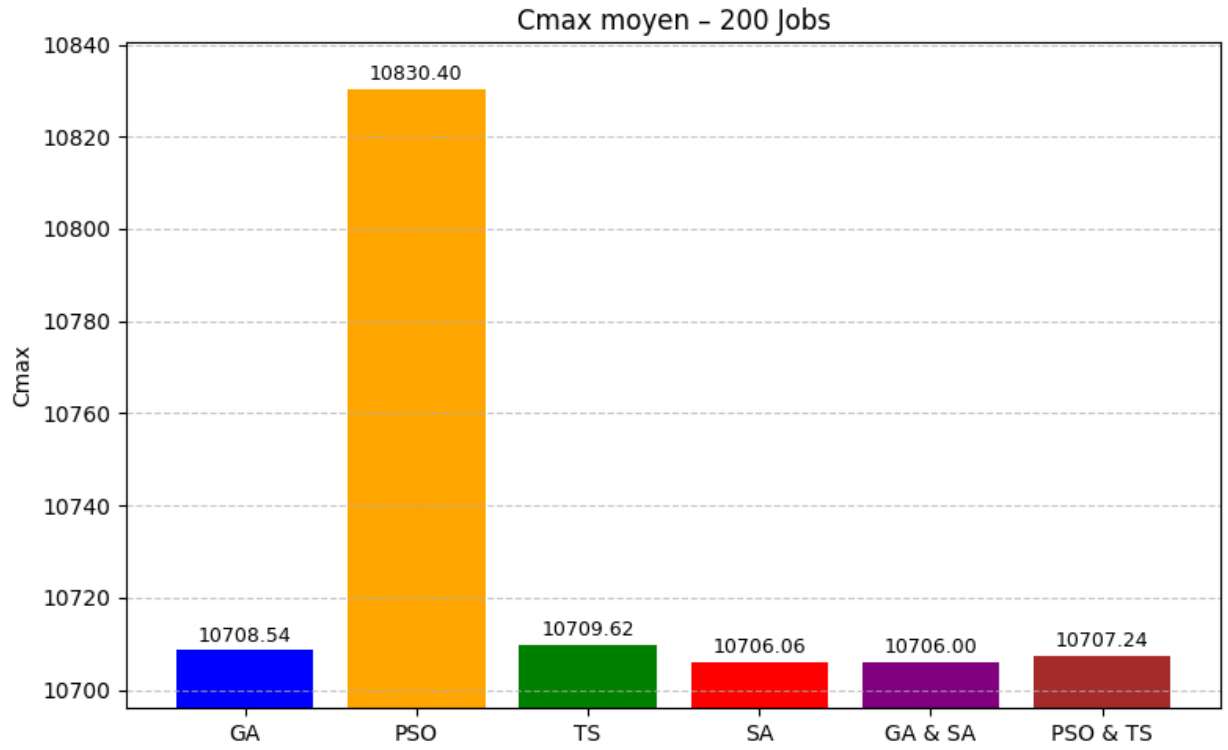

FIG. 4.31 : Histogramme comparatif des makespans moyens obtenus par chaque métaheuristique sur 200 jobs.

Analyse comparative

L'analyse de cette instance critique de 200 jobs révèle des différences significatives entre les approches. L'algorithme du recuit simulé (SA) et son hybride GA/SA confirment leur domination avec des C_{max} moyens respectifs de 10706.06 et 10706.00, établissant un nouveau record de précision pour cette échelle de problème. L'hybride PSO/TS (10707.24) se positionne comme une alternative intéressante, devançant légèrement les méthodes individuelles de recherche tabou (TS) (10709.62) et l'algorithme génétique (GA) (10708.54). L'optimisation par essaim de particules (PSO) seul (10830.40) continue à montrer les performances les moins convaincantes, avec un écart notable par rapport aux autres méthodes.

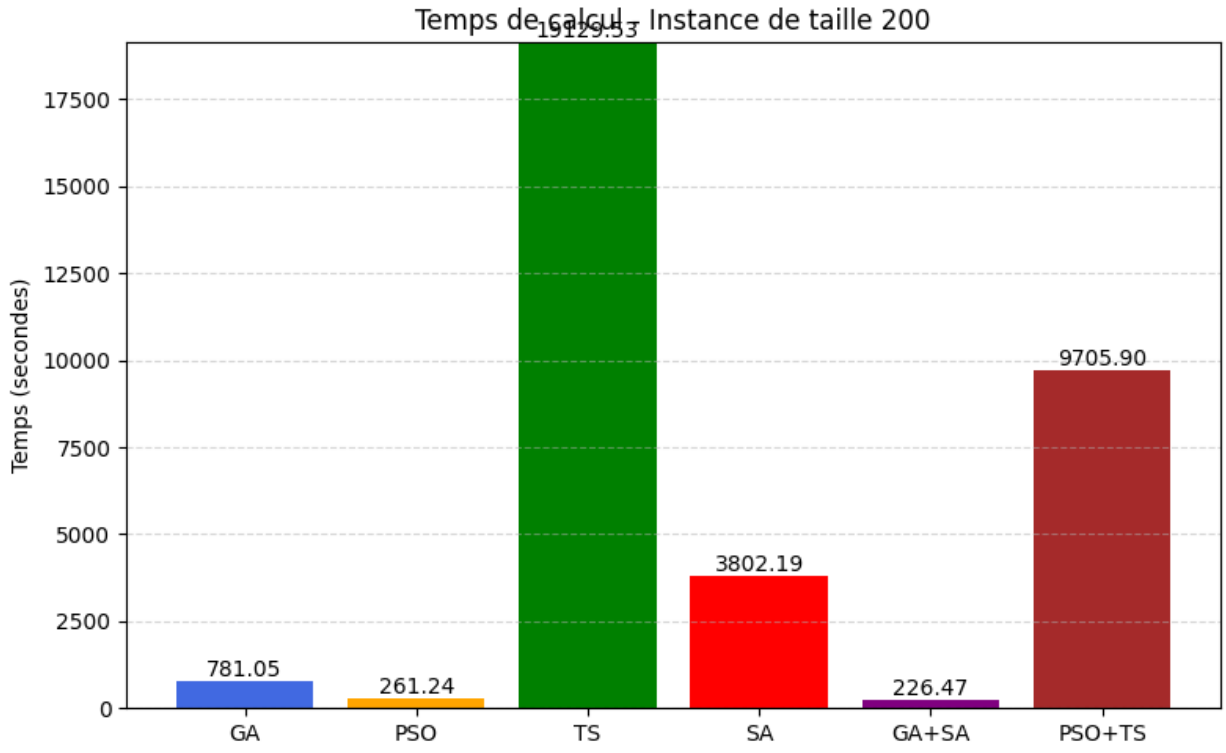

FIG. 4.32 : Histogramme comparatif des temps de calcul moyens obtenus pour chaque métaheuristique sur un ensemble de 200 jobs.

Analyse comparative

L'examen des temps d'exécution dévoile des tendances paradoxales. PSO seul (4.35 min) et l'hybride GA/SA (3.77 min) surprennent par leur efficacité de calcul, offrant les temps les plus compétitifs malgré la complexité accrue. GA seul (13.02 min) maintient des performances temporelles acceptables, tandis que SA seul (63.37 min) voit son temps de calcul devenir problématique. Les approches TS seul (318.82 min, soit plus de 5 heures) et PSO/TS (161.77 min, environ 2h40) deviennent clairement impraticables pour des applications nécessitant des temps de réponse rapides.

4.4 Discussion et prise de décision

4.4.1 Instance à 10 jobs

La comparaison des résultats montre que le choix de l'algorithme doit tenir compte d'un compromis entre qualité de solution et temps de calcul. Les approches comme SA et TS offrent le meilleur équilibre pour cette taille de problème, tandis que les méthodes hybrides présentent des performances variables. L'excellente performance de GA/SA en termes de C_{max}, couplée à un temps de calcul raisonnable, suggère que les stratégies hybrides méritent une exploration plus approfondie. À l'inverse, le mauvais rapport performance-temps de PSO/TS indique que certaines combinaisons peuvent s'avérer contre-productives. Ces observations soulignent l'importance d'adapter le choix de l'algorithme aux contraintes spécifiques du problème à résoudre.

4.4.2 Instance à 25 jobs

L'hybridation GA-SA est l'algorithme privilégié pour cette instance en raison de son makespan optimal et de son temps de calcul raisonnable. La synergie entre la recherche globale de GA et le raffinement local de SA lui permet d'échapper aux optima locaux et de converger efficacement vers des solutions de haute qualité. Cet équilibre rend GA-SA particulièrement adapté aux problèmes de petite à moyenne taille où la qualité des solutions et la rapidité de calcul sont essentielles.

4.4.3 Instance à 75 jobs

Cette analyse met en lumière plusieurs enseignements majeurs. Premièrement, l'excellence persistante du SA et GA/SA en termes de qualité de solution, même pour des instances importantes, confirme leur robustesse. Deuxièmement, la contre-performance de PSO seul en termes de C_{max} contraste avec son temps de calcul compétitif, posant la question de son utilité pratique. Troisièmement, l'effondrement des performances temporelles de TS et PSO/TS suggère que certaines méthodes voient leur complexité augmenter de manière non-linéaire avec la taille du problème. Enfin, la remarquable performance de GA/SA, à la fois en qualité de solution et en temps de calcul, en fait un candidat sérieux pour des applications industrielles nécessitant un bon compromis performance-temps. Ces résultats ouvrent des perspectives intéressantes pour l'optimisation des paramètres des hybrides et leur adaptation à des problèmes de très grande échelle.

4.4.4 Instance à 200 jobs

L'analyse des performances sur l'instance de 200 jobs révèle la supériorité de l'hybride GA/SA, combinant la qualité de solution (C_{max} optimal) et rapidité (226.47s). À l'inverse, TS et PSO/TS s'avèrent trop lents pour un usage pratique, tandis que PSO, bien que rapide, donne des résultats médiocres. Le SA seul offre une bonne qualité de solution mais avec un temps de calcul élevé. Ces résultats soulignent l'importance du choix de la méthode selon les contraintes opérationnelles, GA/SA émergeant comme le meilleur compromis pour les problèmes complexes. Ils ouvrent aussi des pistes pour optimiser les approches hybrides sur des problèmes de plus grande échelle.

4.4.5 Synthèse

L'analyse des quatre instances (10, 25, 75 et 200 tâches) révèle des forces distinctes des algorithmes. Pour l'instance à 10 tâches, SA est optimal en raison de son obtention constante du makespan minimum, bien que GA-SA offre une alternative plus rapide avec des résultats quasi-optimaux. Pour les instances à 25, 75 et 200 tâches, l'hybridation GA-SA surpasse systématiquement les autres, obtenant les makespans les plus petits avec des temps de calcul nettement inférieurs. Son succès repose sur la combinaison de la recherche globale de GA avec l'optimisation locale de SA, garantissant des solutions de haute qualité pour différentes tailles de problème. Ces résultats soulignent l'efficacité des métaheuristiques hybrides dans l'ordonnancement flowshop et suggèrent que des ajustements supplémentaires des paramètres et des stratégies hybrides pourraient améliorer les performances pour divers scénarios d'ordonnancement.

Instance	Algorithme recommandé	Makespan moyen	Temps de calcul moyen (s)
10 tâches	Recuit Simulé (SA)	618	16.282
25 tâches	Hybridation GA-SA	1513	16.200
75 tâches	Hybridation GA-SA	4203	54.754
200 tâches	Hybridation GA-SA	10706	226.475

Tab. 4.25 : Algorithmes recommandés pour les instances de 10, 25, 75 et 200 tâches.

4.5 Conclusion

Ce chapitre a été consacré à la présentation, à l'analyse et à la comparaison des résultats issus de l'application de différentes métaheuristiques au problème d'ordonnancement flowshop, avec pour objectif la minimisation du makespan. Les simulations ont porté sur plusieurs instances variées, permettant d'évaluer les algorithmes selon trois critères fondamentaux : la qualité des solutions, le temps de calcul, et la robustesse.

Suite à une première phase d'expérimentation, les analyses ont alors été concentrées sur quatre approches : GA, PSO, TS, et SA. Parmi celles-ci, le Recuit Simulé (SA) s'est distingué par sa capacité remarquable à fournir des solutions de haute qualité, avec un bon compromis entre exploration et exploitation de l'espace de recherche.

Cependant, l'hybridation du SA avec l'algorithme génétique (GA) a permis d'atteindre des performances encore supérieures. Cette combinaison a montré une amélioration notable des résultats pour l'ensemble des instances testées, tant en termes de réduction du makespan qu'en stabilité des solutions. Cette synergie entre les mécanismes de diversification du GA et le raffinement local du SA a mis en évidence l'intérêt d'approches hybrides pour traiter des problèmes complexes tels que le flowshop.

En résumé, cette étude comparative a permis d'identifier l'hybridation GA–SA comme l'approche la plus prometteuse parmi les algorithmes testés. Elle ouvre des perspectives intéressantes pour la conception de métaheuristiques robustes et adaptatives, capables de s'ajuster efficacement à différentes configurations du problème d'ordonnancement.

CONCLUSION GENÉRALE

Dans un contexte industriel en constante évolution, marqué par la recherche continue de performance, de flexibilité et de compétitivité, l'optimisation des systèmes de production revêt une importance stratégique majeure. Le problème d'ordonnancement dans un atelier de type flowshop, notamment lorsqu'il s'agit de minimiser le makespan, constitue un défi classique mais toujours d'actualité, en raison de sa complexité combinatoire et de sa nature NP-difficile.

Ce mémoire a proposé une étude comparative approfondie de plusieurs métaheuristiques, à savoir l'algorithme génétique (GA), le recuit simulé (SA), l'optimisation par essaim de particules (PSO) et la recherche tabou (TS), ainsi que deux approches hybridées (GA-SA et PSO-TS), afin d'évaluer leur efficacité dans la résolution du problème flowshop. Après une revue théorique rigoureuse des principes de chaque algorithme, une implémentation en Python a permis de tester et d'analyser leurs performances sur des instances issues de la littérature scientifique.

Les résultats obtenus ont mis en évidence la pertinence des approches hybrides, en particulier la combinaison GA–SA, qui a démontré une capacité remarquable à produire des solutions de haute qualité tout en assurant une bonne stabilité et un temps de calcul raisonnable. Ce succès réside dans la complémentarité des stratégies d'exploration globale et d'intensification locale, permettant à l'algorithme de naviguer efficacement dans l'espace de solutions.

Par ailleurs, l'étude a souligné l'influence déterminante des paramètres de chaque métaheuristique sur la qualité des résultats. Cela ouvre la voie à des pistes de recherche prometteuses autour de l'optimisation dynamique et adaptative des paramètres, ainsi que de l'intégration de mécanismes d'auto-apprentissage ou de méta-optimisation.

En définitive, cette contribution s'inscrit dans une démarche de valorisation des techniques métaheuristiques dans le domaine de l'ordonnancement avancé. Elle démontre non seulement leur efficacité opérationnelle mais aussi leur potentiel évolutif dans des environnements industriels de plus en plus complexes. À l'avenir, l'enrichissement de ces méthodes par des approches hybrides intelligentes et adaptatives, couplées à des données en temps réel, pourrait ouvrir de nouvelles perspectives pour l'optimisation robuste et agile des systèmes de production.

BIBLIOGRAPHIE

[1] K. R. Baker et D. Trietsch, *Principles of Sequencing and Scheduling*. Hoboken, NJ : John Wiley Sons, 2009, isbn : 978-0-470-39240-4. doi : 10.1002/9780470451793.

[2] J. Y.-T. Leung, éd., *Handbook of Scheduling : Algorithms, Models, and Performance Analysis*. Boca Raton, FL : Chapman Hall/CRC, 2004, isbn : 978-1-58488-397-5. doi : 10.1201/9780203489802.

[3] M. L. Pinedo, *Scheduling : Theory, Algorithms, and Systems*, 4th. New York, NY : Springer US, 2012, isbn : 978-1-4614-1986-0. doi : 10.1007/978-1-4614-2361-4.

[4] G. Zobolas, C. D. Tarantilis et G. Ioannou, "Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm", *Computers & Operations Research*, t. 36, nº 4, p. 1249-1267, 2009.

[5] Younes2000, *Représentation schématique d'un flowshop*, Consulté le 11 mai 2025, 2012. adresse : https://fr.m.wikipedia.org/wiki/Fichier:Flow_Shop_Ordonnancement.JPEG.

[6] ResearchGate. "Permutation flow shop scheduling problem consisting of 4 jobs and 4 machines, Scientific Figure on ResearchGate". Consulté le : 11 Mai 2025. (2020), adresse : https://www.researchgate.net/figure/Permutation-flow-shop-scheduling-problem-consisting-of-4-jobs-and-4-machines_fig1_343519322.

[7] ResearchGate. "A two-stage hybrid flow shop, Scientific Figure on ResearchGate". Consulté le : 11 Mai 2025. (2017), adresse : https://www.researchgate.net/figure/A-two-stage-hybrid-flow-shop_fig3_318400360.

[8] R. L. Milidiú, A. A. Pessoa et E. S. Laber, "The complexity of makespan minimization for pipeline transportation", *Theoretical computer science*, t. 306, nº 1-3, p. 339-351, 2003.

[9] G. J. Woeginger, "Exact algorithms for NP-hard problems : A survey", in *Combinatorial Optimization—Eureka, You Shrink ! Papers Dedicated to Jack Edmonds 5th International Workshop Aussois, France, March 5–9, 2001 Revised Papers*, Springer, 2003, p. 185-207.

[10] ResearchGate. "Diagram of intersection among classes P, NP, NP-complete and NP-hard problems, Scientific Figure on ResearchGate". Consulté le : 11 Mai 2025. (2020), adresse : https://www.researchgate.net/figure/Diagram-of-intersection-among-classes-P-NP-NP-complete-and-NP-hard-problems_fig13_336890186.

[11] M. R. Garey et D. S. Johnson, *Computers and Intractability : A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[12] M. L. Pinedo, *Scheduling : Theory, Algorithms, and Systems*, 5th. Springer, 2016.

[13] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt et J. Weglarz, "Scheduling Computer and Manufacturing Processes", *Springer*, 1996.

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

[15] R. Ruiz et T. Stützle, "A Simple and Effective Iterated Greedy Algorithm for the Permutation Flowshop Scheduling Problem", *European Journal of Operational Research*, t. 177, nº 3, p. 2033-2049, 2007.

[16] F. Xhafa et A. Abraham, *Metaheuristics for scheduling in industrial and manufacturing applications*. Springer, 2008, t. 128.

[17] H. Fakhravar, "Combining heuristics and exact algorithms : A review", *arXiv preprint arXiv :2202.02799*, 2022.

[18] O. Y. M. Al-Rawi et T. Mukherjee, "Application of Linear Programming in Optimizing Labour Scheduling", *Journal of Mathematical Finance*, t. 9, nº 3, p. 272-285, août 2019. doi : 10.4236/jmf.2019.93017.

[19] G. B. Dantzig, "Linear programming", *Operations research*, t. 50, nº 1, p. 42-47, 2002.

[20] W. Song, X. Chen, Q. Li et Z. Cao, "Flexible job-shop scheduling via graph neural network and deep reinforcement learning", *IEEE Transactions on Industrial Informatics*, t. 19, nº 2, p. 1600-1610, 2022.

[21] ResearchGate. "Graphe disjonctif de l'ordonnancement de groupes, Scientific Figure on ResearchGate". Consulté le : 11 Mai 2025. (2018), adresse : https://www.researchgate.net/figure/Graphe-disjonctif-de-lordonnancement-de-groupes_fig29_323219280.

[22] Y. B. Canbolat et E. Gundogar, "Fuzzy priority rule for job shop scheduling", *Journal of intelligent manufacturing*, t. 15, p. 527-533, 2004.

[23] R. L. Graham, "Bounds for Certain Multiprocessing Anomalies", *Bell System Technical Journal*, t. 47, nº 5, p. 1563-1581, 1969.

[24] K. R. Baker et D. Trietsch, *Principles of Sequencing and Scheduling*. John Wiley Sons, 2008.

[25] J. R. Jackson, "Scheduling a Production Line to Minimize Maximum Tardiness", *Management Science*, t. 3, nº 4, p. 409-421, 1955.

[26] W. L. M. Richard W. Conway et L. W. Miller, "Theory of Scheduling", 1967.

[27] ResearchGate. "Graphe de Branche and Bound, Scientific Figure on geeksforgeeks". Consulté le : 11 Mai 2025. (2023), adresse : https://www.geeksforgeeks.org/introduction-to-branch-and-bound-data-structures-and-algorithms-tutorial/.

[28] M. Dorigo et T. Stützle, *Ant Colony Optimization*. MIT Press, 2004.

[29] H. H. Hoos et T. Stützle, *Stochastic Local Search : Foundations and Applications*. Elsevier, 2005.

[30] C. Blum et A. Roli, "Metaheuristics in Combinatorial Optimization : Overview and Conceptual Comparison", *ACM Computing Surveys*, t. 35, nº 3, p. 268-308, 2003.

[31] M. Nawaz, E. E. Enscore et I. Ham, "A Heuristic Algorithm for the m-Machine, n-Job Flow-Shop Sequencing Problem", *OMEGA*, t. 11, nº 1, p. 91-95, 1983.

[32] É. Taillard, "Some Efficient Heuristic Methods for the Flow Shop Sequencing Problem", *European Journal of Operational Research*, t. 47, nº 1, p. 65-74, 1990.

[33] P. J. M. van Laarhoven et E. H. L. Aarts, "Simulated Annealing : Theory and Applications", *Kluwer Academic Publishers*, 1992.

[34] T. Liao, X. Wang et W. Wang, "An Adaptive Simulated Annealing for Permutation Flowshop Scheduling Problem", *Computers Operations Research*, t. 39, p. 2056-2064, 2012.

[35] R. Zhang et Z. Sun, "Particle Swarm Optimization Algorithm for Flowshop Scheduling Problem", *Applied Mathematics and Computation*, t. 185, nº 2, p. 911-918, 2007.

[36] C. Rajendran et H. Ziegler, "Ant Colony Optimization for Flowshop Scheduling Problems", *European Journal of Operational Research*, t. 155, nº 3, p. 426-438, 2004.

[37] G. Zobolas, C. D. Tarantilis et G. Ioannou, "Hybrid Metaheuristics for the Flowshop Scheduling Problem", *Journal of Heuristics*, t. 15, nº 5, p. 495-518, 2009.

[38] E.-G. Talbi, *Metaheuristics : from design to implementation*. John Wiley & Sons, 2009.

[39] C. Blum et A. Roli, "Metaheuristics in combinatorial optimization : Overview and conceptual comparison", *ACM Computing Surveys (CSUR)*, t. 35, nº 3, p. 268-308, 2003.

[40] S. N. Sivanandam et S. N. Deepa, *Introduction to Genetic Algorithms*. Springer, 2008. doi : 10.1007/978-3-540-73190-0.

[41] D. Whitley, "Genetic Algorithms : A Review and a New Taxonomy", *Evolutionary Computation*, t. 30, nº 1, p. 1-35, 2022. doi : 10.1162/evco_a_00323.

[42] A. E. Eiben et J. E. Smith, *Introduction to Evolutionary Computing*, 2nd. Springer, 2015. doi : 10.1007/978-3-662-44874-8.

[43] "Optimisation Multi-Objectifs des paramètres d'usinage par Algorithmes Génétiques - Scientific Figure on ResearchGate". Consulté le 13 Mai 2025. (2025), adresse : https://www.researchgate.net/figure/Principe-de-fonction-dun-algorithme-genetique_fig5_279940927.

[44] "INTRODUCTION TO GENETIC ALGORITHMS". Consulté le 13 Mai 2025. (2022), adresse : https://arisaftech.co.jp/introduction-to-genetic-algorithms/.

[45] J. Kennedy et R. Eberhart, "Particle Swarm Optimization", *Proceedings of IEEE International Conference on Neural Networks*, t. 4, p. 1942-1948, 1995. doi : 10.1109/ICNN.1995.488968.

[46] Y. Shi et R. Eberhart, "A Modified Particle Swarm Optimizer", *Proceedings of IEEE International Conference on Evolutionary Computation*, p. 69-73, 1998. doi : 10.1109/ICEC.1998.699146.

[47] M. Clerc, *Particle Swarm Optimization*. ISTE, 2006. doi : 10.1002/9780470612163.

[48] M. R. Bonyadi et Z. Michalewicz, "Particle Swarm Optimization for Single Objective Continuous Space Problems : A Review", *Evolutionary Computation*, t. 25, nº 1, p. 1-54, 2017. doi : 10.1162/EVCO_a_00180.

[49] "PARTICLE SWARM OPTIMIZATION (PSO)". Consulté le 13 Mai 2025. (2024), adresse : https://esa.github.io/pagmo2/docs/cpp/algorithms/pso.html.

[50] "Strategic Planning for Minimizing CO_2 Emissions Using LP Model Based on Forecasted Energy Demand by PSO Algorithm and ANN - Scientific Figure on ResearchGate". Consulté le 13 Mai 2025. (2025), adresse : https://www.researchgate.net/figure/Particle-swarm-optimization-flowchart_fig1_252629792.

[51] F. Glover, "Future paths for integer programming and links to artificial intelligence", *Computers & operations research*, t. 13, nº 5, p. 533-549, 1986.

[52] F. Glover et M. Laguna, *Tabu Search*. Springer US, 1997.

[53] ResearchGate. "The basic steps of a tabu search, Using a Tabu Search Approach to Align DNA Sequences". Consulté le : 18 Mai 2025. (2025), adresse : https://www.researchgate.net/figure/The-basic-steps-of-a-tabu-search_fig1_269336602.

[54] F. Glover, "Tabu search—Uncharted domains", *Annals of Operations Research*, t. 149, nº 1, p. 89-98, 2007.

[55] "Intra-Platoon Vehicle Sequence Optimization for Eco-Cooperative Adaptive Cruise Control - Scientific Figure on ResearchGate". Consulté le 13 Mai 2025. (2025), adresse : https://www.researchgate.net/figure/Flowchart-of-tabu-search-algorithm_fig1_320508257.

[56] S. Kirkpatrick, C. D. Gelatt et M. P. Vecchi, "Optimization by Simulated Annealing", *Science*, t. 220, nº 4598, p. 671-680, 1983.

[57] "A New Hybrid Approach in Selection of Optimum Establishment Location of the Biogas Energy Production Plant - Scientific Figure on ResearchGate". Consulté le 13 Mai 2025. (2025), adresse : https://www.researchgate.net/figure/Simulated-annealing-algorithm-flowchart-25_fig2_351459987.

[58] E. H. L. Aarts et J. H. M. Korst, *Simulated Annealing and Boltzmann Machines : A Stochastic Approach to Combinatorial Optimization and Neural Computing*. Chichester, UK : John Wiley Sons, 1989, isbn : 978-0471921467.

[59] "OPTIMISATION COMBINATOIRE". Consulté le 13 Mai 2025. (2025), adresse : https://www.mcours.net/cours/pdf/hassbg/hassbgli902.pdf.

[60] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller et E. Teller, "Equation of State Calculations by Fast Computing Machines", *The Journal of Chemical Physics*, t. 21, nº 6, p. 1087-1092, 1953. doi : 10.1063/1.1699114.

[61] V. Černý, "Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm", *Journal of Optimization Theory and Applications*, t. 45, p. 41-51, 1985.

[62] Author(s), "Metaheuristic Algorithms : A Comprehensive Review", *Journal Name*, t. Volume, nº Number, Pages, Year. doi : 10.1016/B978-0-12-813314-9.00010-4. adresse : https://www.sciencedirect.com/science/article/pii/B9780128133149000104.

[63] Author(s), "Metaheuristics : A Comprehensive Overview and Classification Along with Bibliometric Study", *Artificial Intelligence Review*, t. Volume, nº Number, Pages, Year. doi : 10.1007/s10462-020-09952-0. adresse : https://link.springer.com/article/10.1007/s10462-020-09952-0.

[64] Author(s), "Comparison of Robustness of Metaheuristic Algorithms for Steel Frame Design Optimization", *Engineering Structures*, t. Volume, nº Number, Pages, Year. doi : 10.1016/j.engstruct.2015.05.042. adresse : https://www.sciencedirect.com/science/article/pii/S0141029615005118.

[65] E.-G. Talbi, *Metaheuristics : From Design to Implementation*. John Wiley & Sons, 2002.

[66] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[67] J. Kennedy et R. Eberhart, "Particle Swarm Optimization", *Proceedings of IEEE International Conference on Neural Networks*, p. 1942-1948, 1995.

[68] F. Glover, "Tabu Search—Part I", *ORSA Journal on Computing*, t. 1, nº 3, p. 190-206, 1989.

[69] M. Lutz, *Learning Python*, 5th. O'Reilly Media, 2013.

[70] T. E. Oliphant, *Guide to NumPy*. Trelgol Publishing, 2007.

[71] A. Milani et F. Rossi, "Comparison of Metaheuristic Algorithms Using Python Libraries for Solving Optimization Problems", *Journal of Artificial Intelligence Research*, t. 70, p. 233-258, 2021.

[72] G. Van Rossum et F. L. Drake, *The Python Language Reference*, https://docs.python.org/3/reference/, 2020.

[73] P. S. Foundation. "Python Logo". consulté : 4 Mai 2025, Python Software Foundation. (2024), adresse : https://www.python.org/static/community_logos/python-logo.png.

[74] Microsoft, *Visual Studio Code - Documentation*, Consulté en mars 2025, 2023. adresse : https://code.visualstudio.com/docs.

[75] J. Han et Y. Li, “Why Visual Studio Code is Emerging as the Most Popular IDE”, *Software Development Review*, t. 8, nº 2, p. 34-38, 2021.

[76] Microsoft, *Python in Visual Studio Code*, Extension officielle, consultée en mars 2025, 2023. adresse : `https://marketplace.visualstudio.com/items?itemName=ms-python.python`.

[77] J. D. Team, *Jupyter Extension for Visual Studio Code*, Consulté en mars 2025, 2023. adresse : `https://marketplace.visualstudio.com/items?itemName=ms-toolsai.jupyter`.

[78] Microsoft, *Git in Visual Studio Code*, Consulté en mars 2025, 2023. adresse : `https://code.visualstudio.com/docs/editor/versioncontrol`.

[79] Microsoft Corporation. “Visual Studio Code Branding Resources”. Consulté le : 4 Mai 2025, Microsoft. (2025), adresse : `https://code.visualstudio.com/brand`.

[80] X. Li, L. Gao et X. Shao, “A comparative study of metaheuristic algorithms for permutation flow shop scheduling problem”, *International Journal of Production Research*, t. 48, nº 2, p. 525-541, 2010.

[81] M. Dorigo et L. M. Gambardella, “Ant colony system : a cooperative learning approach to the traveling salesman problem”, *IEEE Transactions on Evolutionary Computation*, t. 1, nº 1, p. 53-66, 1997.

[82] C. R. Reeves, *Modern heuristic techniques for combinatorial problems*. Wiley, 1995.

[83] M. Dorigo et T. Stützle, *Ant Colony Optimization*. MIT Press, 2004.

[84] I. H. Osman et G. Laporte, “Meta-heuristics : a bibliography”, *Annals of Operations Research*, t. 63, p. 513-623, 1996.

[85] C. R. Reeves, “Modern heuristic techniques for combinatorial problems”, *Blackwell Scientific Publications*, 1993.

[86] Z.-H. Zhan, H. Li et X. He, “A comprehensive review on particle swarm optimization : Variants and applications”, *Journal of Artificial Evolution and Applications*, t. 2013, p. 1-15, 2013.

[87] R. Eberhart et Y. Shi, “Comparing inertia weights and constriction factors in particle swarm optimization”, *Proceedings of the IEEE Swarm Intelligence Symposium*, p. 84-88, 2000.

[88] J. Liang, H. Qin et S. Zhao, “Particle swarm optimization with an aging leader and challenger for global optimization”, *IEEE Transactions on Evolutionary Computation*, t. 10, nº 3, p. 282-292, 2006.

[89] E. Nowicki et C. Smutnicki, “A fast taboo search algorithm for the permutation flow-shop problem”, *European Journal of Operational Research*, t. 91, nº 1, p. 160-175, 1996.

[90] I. H. Osman, “Metastrategy simulated annealing and tabu search algorithms for the job shop scheduling problem”, *Annals of Operations Research*, t. 41, nº 4, p. 421-451, 1993.

[91] Y. Liu et C. R. Reeves, “A hybrid metaheuristic for permutation flow-shop scheduling”, *Computers & Operations Research*, t. 34, nº 10, p. 3209-3227, 2007.

[92] P. J. M. V. Laarhoven et E. H. L. Aarts, *Simulated Annealing : Theory and Applications*. Springer, 1987.

ANNEXE A

ANNEXE A - MATRICES DES JOBS UTILISÉS DANS LES SIMULATIONS

Cette annexe regroupe les matrices générées pour lancer les tests présentés dans le chapitre 4 pour les divers algorithmes métaheuristiques. Etant donné que le nombre de machine a été fixé à 5, c'est uniquement le nombre de jobs qui a été varié pour créer les matrices différentes.

A.1 Instance à 10 jobs

$$\begin{bmatrix} 55 & 56 & 32 & 20 & 44 & 37 & 18 & 80 & 19 & 84 \\ 42 & 35 & 29 & 85 & 61 & 74 & 26 & 40 & 6 & 47 \\ 20 & 85 & 76 & 5 & 1 & 42 & 63 & 82 & 72 & 43 \\ 2 & 1 & 60 & 31 & 83 & 74 & 79 & 11 & 63 & 31 \\ 98 & 31 & 31 & 50 & 73 & 61 & 33 & 7 & 20 & 28 \end{bmatrix}$$

A.2 Instance à 25 jobs

$$\begin{bmatrix} 81 & 90 & 74 & 93 & 82 & 40 & 29 & 62 & 15 & 89 & 47 & 71 & 31 & 42 & 33 & 91 & 79 & 80 & 24 & 52 & 46 & 11 & 46 & 27 & 53 \\ 8 & 82 & 39 & 78 & 33 & 57 & 56 & 33 & 48 & 93 & 40 & 93 & 40 & 42 & 66 & 68 & 13 & 27 & 84 & 79 & 33 & 86 & 94 & 55 & 98 \\ 11 & 64 & 61 & 40 & 59 & 92 & 100 & 45 & 78 & 99 & 67 & 64 & 34 & 55 & 2 & 18 & 100 & 57 & 10 & 44 & 21 & 66 & 47 & 36 & 82 \\ 70 & 29 & 32 & 40 & 62 & 44 & 5 & 52 & 46 & 31 & 68 & 85 & 61 & 50 & 43 & 16 & 22 & 100 & 50 & 47 & 89 & 27 & 14 & 8 & 59 \\ 11 & 21 & 68 & 22 & 41 & 99 & 73 & 98 & 86 & 68 & 12 & 8 & 36 & 54 & 21 & 20 & 99 & 2 & 70 & 53 & 33 & 74 & 96 & 9 & 22 \end{bmatrix}$$

A.3 Instance à 50 jobs

Vue la taille importante de la matrice par son grand nombre de colonnes, la représentation habituelle en forme matricielle n'est pas possible pour les instances dépassant les 25 jobs. Au lieu, cette matrice est affichée en forme de listes ou les éléments de chaque ligne de la matrice sont écrits entre deux crochets [] et sont séparés par des virgules comme présentés ci-aprés.

[9, 61, 32, 39, 94, 63, 76, 25, 63, 46, 46, 34, 17, 99, 47, 42, 50, 9, 24, 7, 56, 96, 30, 21, 5, 47, 18, 50, 83, 34, 66, 54, 61, 54, 8, 78, 70, 42, 32, 78, 7, 55, 69, 95, 48, 18, 71, 10, 71, 99], [57, 85, 98, 36, 4, 20, 48, 36, 66, 43, 68, 76, 56, 20, 95, 39, 30, 94, 64, 55, 49, 57, 12, 57, 84, 47, 61, 52, 32, 94, 92, 93, 70, 19, 18, 59, 94, 18, 78, 25, 43, 29, 4, 37, 54, 73, 61, 61, 51, 53], [70, 98, 47, 89, 62, 23, 35, 95, 17, 35, 86, 87, 91, 26, 71, 84, 68, 91, 49, 9, 36, 96, 92, 25, 31, 54, 2, 75, 64, 67, 2, 57, 36, 27, 82, 2, 74, 60, 100, 17, 5, 34, 76, 37, 98, 22, 78, 10, 77, 31], [41, 45, 74, 52, 31, 2, 23, 51, 75, 45, 49, 53, 76, 88, 81, 37, 33, 19, 79, 85, 35, 2, 29, 96, 15, 61, 50, 19, 60, 57, 25, 98, 29, 10, 58, 69, 60, 26, 66, 28, 46, 92, 74, 42, 25, 90, 58, 47, 37, 74], [52, 77, 52, 25, 6, 3, 90, 10, 1, 92, 20, 19, 88, 36, 17, 38, 54, 70, 54, 46, 85, 11, 41, 70, 99, 76, 20, 72, 78, 14, 56, 60, 78, 89, 32, 26, 14, 72, 33, 22, 85, 57, 49, 73, 84, 36, 58, 35, 54, 78]]

A.4 Instance à 75 jobs

[15, 29, 43, 29, 66, 42, 1, 66, 32, 73, 8, 52, 51, 62, 21, 88, 32, 52, 91, 41, 72, 55, 62, 98, 4, 37, 59, 87, 13, 98, 68, 65, 24, 22, 59, 7, 100, 3, 52, 81, 96, 84, 86, 41, 1, 70, 27, 5, 71, 39, 5, 88, 34, 81, 23, 10, 32, 93, 29, 72, 23, 89, 10, 56, 36, 4, 30, 52, 24, 45, 72, 17, 64, 22, 89], [57, 43, 18, 85, 7, 28, 49, 20, 56, 10, 33, 59, 30, 17, 35, 99, 97, 76, 81, 74, 45, 73, 82, 99, 25, 60, 65, 79, 49, 9, 70, 83, 7, 50, 42, 82, 11, 6, 1, 93, 25, 6, 23, 79, 72, 51, 59, 60, 39, 24, 87, 47, 12, 74, 14, 31, 67, 82, 75, 36, 38, 47, 79, 81, 51, 57, 58, 49, 48, 4, 40, 100, 31, 91, 14], [37, 57, 34, 56, 59, 6, 9, 86, 15, 59, 44, 97, 9, 84, 77, 55, 69, 83, 5, 83, 41, 33, 56, 37, 79, 55, 28, 87, 94, 38, 3, 74, 74, 80, 32, 7, 45, 4, 64, 85, 26, 58, 58, 45, 72, 37, 16, 74, 35, 13, 93, 65, 20, 55, 68, 31, 21, 99, 73, 82, 94, 8, 31, 48, 86, 8, 93, 58, 13, 33, 43, 16, 34, 5, 77], [31, 78, 59, 67, 56, 35, 24, 47, 11, 39, 83, 83, 95, 30, 89, 95, 84, 25, 49, 27, 65, 96, 60, 13, 50, 57, 12, 82, 47, 83, 94, 69, 12, 31, 85, 82, 72, 67, 75, 98, 18, 65, 67, 71, 54, 31, 84, 74, 51, 52, 66, 78, 41, 95, 52, 65, 39, 43, 33, 77, 69, 36, 92, 84, 55, 38, 29, 24, 25, 9, 13, 70, 22, 23, 87], [51, 64, 25, 6, 71, 69, 36, 48, 22, 22, 37, 37, 84, 87, 38, 82, 6, 44, 72, 90, 65, 84, 99, 73, 59, 96, 64, 12, 19, 67, 94, 40, 23, 6, 86, 10, 21, 43, 40, 23, 52, 53, 13, 50, 44, 59, 7, 6, 59, 78, 60, 36, 1, 11, 45, 100, 89, 95, 71, 51, 31, 57, 34, 25, 81, 77, 32, 2, 64, 86, 38, 82, 48, 51, 76]

A.5 Instance à 100 jobs

[63, 80, 45, 25, 18, 98, 75, 63, 24, 56, 66, 19, 93, 70, 78, 14, 53, 92, 77, 11, 13, 17, 46, 39, 92, 99, 33, 48, 42, 29, 7, 65, 41, 67, 65, 48, 2, 48, 59, 28, 53, 42, 65, 34, 83, 99, 3, 92, 39, 96, 61, 27, 6, 58, 45, 66, 60, 13, 24, 67, 15, 68, 100, 38, 96, 48, 59, 97, 51, 35, 71, 37, 88, 85, 24, 42, 51, 26, 53, 60, 46, 10, 99, 18, 30, 77, 32, 99, 52, 97, 3, 27, 61, 93, 47, 9, 53, 33, 43, 72], [22, 24, 90, 89, 76, 90, 38, 96, 3, 40, 55, 22, 36, 21, 64, 86, 30, 56, 52, 39, 94, 24, 57, 60, 75, 100, 82, 13, 46, 75, 93, 40, 68, 93, 33, 57, 76, 90, 46, 36, 97, 98, 6, 97, 13, 50, 69, 68, 25, 60, 92, 57, 92, 2, 54, 49, 86, 42, 53, 31, 84, 75, 83, 18, 61, 20, 100, 73, 68, 48, 11, 9, 83, 95, 90, 53, 96, 53, 9, 15, 40, 76, 2, 48, 6, 25, 87, 99, 25, 31, 28, 18, 84, 15, 59, 46, 97, 16, 48, 24], [80, 77, 46, 49, 38, 58, 61, 63, 83, 68, 16, 25, 25, 61, 36, 59, 13, 18, 41, 48, 67, 7, 53, 18, 81, 92, 86, 32, 90, 7, 90, 33, 30, 84, 9, 74, 25, 83, 7, 64, 86, 56, 42, 69, 52, 16, 72, 98, 16, 62, 77, 42, 14, 51, 15, 92, 51, 36, 53, 31, 70, 13, 56, 96, 18, 38, 61, 4, 1, 78, 90, 73, 44, 59, 98, 7, 8, 77, 7, 98, 33, 96, 70, 51, 57, 93, 7, 7, 32, 66, 11, 27, 8, 86, 61, 67, 21, 42, 76, 31], [16, 11, 55, 2, 76, 54, 12, 94, 20, 73, 11, 10, 41, 56, 19, 92, 87, 75, 62, 74, 75, 16, 86, 88, 80, 76, 25, 46, 67, 55, 78, 63, 65, 89, 2, 25, 91, 28, 19, 29, 49, 85, 40, 62, 37, 32, 52, 71, 74, 3, 83, 55, 57, 70, 37, 17, 86, 50, 7, 50, 71, 84, 77, 67, 26, 22, 8, 61, 45, 88, 53, 38, 7, 65, 79, 57, 47, 24, 83, 29, 70, 44, 90, 12, 44, 25, 91, 46, 20, 50, 36, 33, 19, 39, 23, 83, 22, 40, 83, 100], [91, 28, 45, 84, 66, 71, 54, 15, 90, 16, 20, 30, 60, 68, 18, 84, 69, 18, 2, 92, 7, 59, 84, 46, 51, 46, 99, 16, 38, 99, 54, 62, 15, 82, 47, 63, 47, 64, 60, 100, 73, 46, 70, 95, 63, 65, 81, 28, 35, 68, 86, 72, 95, 16, 47, 49, 92, 68, 22, 53, 32, 74, 89, 30, 78, 58, 73, 53, 74, 51, 25, 98, 19, 7, 11, 43, 40, 51, 88, 11, 52, 16, 63, 88, 32, 69, 11, 6, 53, 47, 85, 51, 77, 4, 49, 66, 34, 23, 79, 2]

A.6 Instance à 150 jobs

[52, 88, 66, 6, 31, 37, 64, 6, 74, 17, 100, 5, 77, 1, 5, 69, 28, 98, 77, 23, 10, 72, 50, 20, 91, 89, 87, 32, 59, 50, 100, 4, 82, 66, 2, 94, 40, 20, 95, 93, 6, 39, 92, 84, 17, 55, 9, 10, 37, 4, 57, 52, 27, 2, 97, 28, 8, 84, 74, 31, 56, 62, 3, 70, 82, 43, 57, 34, 75, 53, 75, 74, 100, 89, 32, 66, 96, 64, 63, 58, 98, 44, 66, 75, 34, 63, 31, 57, 35, 96, 18, 64, 41, 54, 34, 55, 82, 45, 87, 87, 60, 38, 6, 16, 27, 4, 30, 77, 50, 100, 2, 46, 43, 8, 76, 89, 36, 68, 70, 72, 9, 77, 54, 60, 94, 35, 17, 75, 64, 77, 32, 70, 89, 7, 79, 15, 32, 48, 19, 42, 4, 60, 92, 83, 67, 59, 99, 44, 62, 57], [79, 87, 5, 88, 73, 27, 65, 40, 4, 76, 8, 13, 16, 74, 52, 3, 49, 71, 28, 83, 33, 96, 65, 30, 57, 63, 4, 97, 36, 42, 70, 28, 83, 16, 59, 46, 10, 34, 13, 33, 60, 29, 26, 53, 10, 5, 1, 29, 52, 70, 35, 59, 67, 42, 77, 11, 33, 20, 14, 30, 35, 8, 87, 35, 15, 13, 32, 28, 89, 24, 38, 50, 1, 89, 53, 40, 71, 49, 16, 44, 20, 79, 57, 93, 9, 16, 46, 16, 77, 31, 98, 74, 65, 43, 88, 84, 80, 25, 60, 52, 12, 56, 11, 92, 47, 98, 28, 9, 59, 21, 37, 93, 9, 3, 41, 67, 67, 58, 36, 24, 60, 2, 59, 54, 79, 88, 94, 91, 7, 72, 43, 2, 23, 66, 25, 88, 86, 15, 87, 60, 95, 4, 26, 28, 12, 97, 19, 78, 30, 70], [82, 25, 25, 53, 71, 29, 93, 45, 49, 42, 8, 16, 62, 43, 65, 58, 94, 88, 85, 35, 88, 75, 37, 31, 49, 34, 10, 95, 62, 37, 9, 59, 80, 99, 13, 31, 21, 84, 80, 80, 20, 18, 52, 18, 72, 12, 39, 44, 27, 39, 32, 15, 9, 4, 12, 17, 4, 31, 61, 63, 44, 82, 14, 13, 71, 59, 9, 26, 78, 27, 73, 38, 29, 74, 70, 97, 19, 73, 28, 70, 17, 21, 48, 91, 47, 7, 65, 95, 54, 24, 48, 53, 86, 10, 29, 90, 87, 91, 81, 33, 78, 33, 14, 4, 18, 99, 63, 4, 80, 42, 70, 37, 87, 92, 32, 42, 58, 54, 37, 4, 61, 73, 82, 56, 78, 58, 96, 27, 77, 60, 58, 61, 100, 89, 5, 71, 69, 11, 93, 42, 71, 46, 46, 95, 89, 63, 15, 25, 81, 55], [79, 38, 79, 83, 36, 66, 88, 78, 92, 98, 64, 11, 16, 92, 37, 28, 60, 22, 11, 69, 34, 35, 27, 72, 18, 73, 55, 98, 14, 6, 80, 4, 54, 46, 31, 43, 20, 8, 33, 66, 84, 98, 84, 63, 15, 96, 13, 96, 60, 33, 24, 52, 78, 14, 97, 19, 93, 90, 40, 7, 71, 88, 22, 27, 16, 58, 65, 47, 22, 63, 19, 50, 68, 4, 4, 89, 69, 92, 58, 4, 7, 100, 98, 91, 45, 1, 99, 28, 13, 77, 18, 29, 10, 93, 60, 53, 96, 9, 37, 49, 34, 29, 8, 19, 15, 80, 72, 75, 6, 31, 52, 73, 59, 4, 27, 27, 3, 47, 22, 36, 59, 76, 55, 79, 85, 61, 83, 93, 32, 66, 52, 49, 32, 77, 20, 62, 35, 66, 70, 80, 12, 28, 72, 72, 79, 58, 89, 41, 92, 76], [91, 65, 37, 35, 44, 53, 42, 65, 46, 91, 45, 85, 71, 1, 19, 15, 91, 94, 71, 15, 14, 23, 27, 35, 93, 31, 98, 55, 86, 79, 63, 34, 22, 53, 88, 52, 88, 60, 47, 18, 98, 72, 69, 35, 48, 31, 99, 56, 59, 35, 53, 42, 17, 46, 73, 92, 62, 85, 82, 14, 50, 73, 2, 79, 65, 3, 22, 23, 81, 77, 32, 94, 47, 91, 17, 22, 7, 87, 35, 11, 58, 99, 88, 36, 25, 38, 47, 86, 52, 81, 92, 72, 50, 90, 60, 33, 42, 71, 72, 83, 26, 1, 35, 69, 39, 35, 42, 76, 28, 64, 88, 2, 78, 87, 38, 25, 66, 5, 84, 7, 75, 88, 96, 54, 14, 44, 92, 41, 4, 66, 6, 44, 78, 73, 49, 4, 38, 69, 84, 73, 65, 29, 7, 99, 62, 40, 77, 21, 4, 25]

A.7 Instance à 200 jobs

[81, 96, 78, 49, 78, 40, 25, 10, 79, 35, 21, 80, 80, 29, 61, 56, 75, 94, 66, 86, 75, 79, 15, 23, 2, 74, 62, 81, 56, 78, 96, 37, 87, 18, 73, 60, 82, 32, 30, 1, 77, 45, 62, 34, 90, 4, 59, 87, 38, 43, 6, 57, 91, 42, 35, 56, 45, 5, 33, 32, 67, 4, 70, 67, 10, 6, 72, 94, 29, 50, 99, 18, 79, 76, 72, 22, 36, 21, 46, 11, 100, 88, 100, 55, 72, 36, 55, 40, 57, 94, 7, 78, 3, 23, 84, 91, 89, 86, 83, 90, 88, 62, 43, 67, 24, 52, 34, 55, 1, 35, 68, 82, 24, 98, 82, 35, 32, 66, 91, 35, 1, 87, 28, 86, 89, 63, 100, 93, 55, 74, 40, 97, 44, 21, 58, 12, 39, 37, 11, 81, 2, 18, 11, 80, 1, 90, 86, 98, 79, 12, 69, 74, 11, 16, 75, 44, 27, 73, 14, 55, 23, 95, 87, 1, 39, 66, 98, 18, 48, 61, 12, 51, 54, 63, 39, 45, 52, 67, 16, 32, 41, 90, 12, 46, 4, 31, 32, 35, 18, 1, 9, 16, 74, 99, 20, 7, 38, 66, 57, 79], [60, 52, 95, 100, 2, 91, 1, 5, 89, 93, 67, 64, 81, 42, 18, 21, 46, 4, 24, 79, 6, 58, 7, 99, 80, 64, 92, 51, 32, 65, 68, 42, 75, 95, 55, 7, 42, 98, 41, 28, 68, 48, 29, 73, 87, 50, 66, 61, 97, 72, 28, 57, 2, 6, 78, 100, 44, 78, 60, 38, 2, 50, 9, 42, 4, 12, 98, 60, 68, 25, 50, 70, 71, 37, 30, 38, 58, 89, 60, 46, 59, 57, 67, 10, 36, 31, 62, 64, 57, 57, 72, 77, 74, 98, 90, 51, 85, 93, 54, 82, 71, 96, 41, 22, 26, 55, 83, 86, 23, 27, 55, 86, 90, 43, 74, 47, 93, 32, 91, 69, 2, 63, 41, 68, 79, 80, 15, 3, 47, 34, 100, 33, 80, 70, 41, 42, 88, 8, 45, 64, 27, 91, 100, 52, 1, 43, 91, 85, 98, 30, 61, 69, 68, 16, 75, 76, 66, 3, 5, 51, 18, 82, 52, 24, 67, 6, 90, 6, 10, 71, 52, 64, 38, 19, 43, 72, 18, 15, 46, 25, 50, 1, 4, 98, 87, 54, 91, 96, 53, 31, 39, 11, 17, 71, 66, 54, 95, 4, 78, 74], [72, 93, 26, 32, 92, 96, 37, 7, 37, 34, 47, 15, 2, 65, 22, 74, 62, 21, 35, 5, 88, 88, 80, 34, 20, 76, 7, 50, 70, 3, 4, 93, 20, 26, 41, 68, 49, 73, 49, 67, 57, 76, 57, 23, 50, 81, 93, 44, 56, 65, 11, 12, 12, 18, 34, 81, 93, 44, 18, 91, 31, 71, 69, 84, 51, 15, 62, 48, 48, 12, 7, 32, 65, 35, 77, 5, 94, 4, 73, 93, 12, 81, 95, 98, 62, 97, 54, 98, 43, 35, 55, 89, 86, 88, 65, 40, 65, 16, 22, 75, 88, 18, 76, 41, 23, 19, 4, 19, 24, 62, 95, 60, 4, 98, 25, 33, 53, 30, 20, 25, 80, 10, 19, 33, 88, 60, 9, 2, 2, 44, 33, 83, 19, 67, 55, 86, 8, 100, 84, 76, 72, 61, 71, 77, 65, 8, 55, 72, 25, 39, 45, 49, 100, 15, 13, 26, 14, 91, 67, 61, 58, 98, 54, 20, 14, 87, 95, 1, 84, 89, 6, 9, 95, 35, 69, 36, 96, 93, 68, 30, 77, 59, 99, 58, 4, 89, 58, 3, 5, 59, 68, 36, 71, 23, 50, 93, 56, 69, 59, 45], [70, 64, 7, 61, 57, 44, 43, 14, 60, 15, 75, 40, 89, 62, 90, 99, 72, 93, 6, 13, 3, 60, 1, 28, 80, 77, 8, 73, 85, 59, 44, 58, 56, 95, 84, 85, 36, 71, 87, 43, 74, 50, 38, 100, 95, 35, 15, 96, 78, 60, 13, 89, 21, 11, 73, 71, 14, 28, 71, 60, 14, 42, 71, 14, 35, 61, 22, 33, 89, 53, 80, 32, 88, 63, 14, 99, 98, 75, 73, 81, 13, 30, 83, 77, 10, 73, 100, 35, 8, 97, 57, 39, 11, 26, 53, 43, 63, 67, 45, 60, 27, 59, 71, 40, 7, 82, 35, 63, 31, 82, 81, 25, 6, 88, 48, 55, 20, 11, 29, 81, 49, 69, 68, 59, 57, 10, 15, 38, 69, 56, 96, 93, 65, 34, 90, 39, 28, 92, 47, 51, 24, 46, 90, 80, 25, 5, 54, 45, 4, 26, 24, 85, 67, 44, 51, 54, 16, 62, 7, 48, 15, 31, 69, 27, 100, 4, 41, 6, 2, 32, 13, 34, 96, 29, 3, 31, 26, 86, 97, 84, 34, 86, 69, 7, 33, 13, 11, 74, 42, 4, 37, 83, 82, 92, 46, 98, 67, 71, 92, 7], [37, 43, 57, 27, 66, 96, 99, 1, 100, 59, 33, 73, 68, 9, 18, 98, 58, 48, 60, 93, 12, 72, 94, 48, 95, 1, 1, 77, 74, 57, 49, 84, 18, 70, 63, 41, 4, 20, 8, 29, 76, 52, 58, 56, 82, 94, 7, 76, 55, 8, 86, 17, 75, 7, 54, 54, 89, 84, 12, 76, 96, 43, 44, 64, 63, 64, 96, 54, 83, 14, 46, 68, 28, 30, 20, 4, 33, 60, 85, 37, 73, 23, 29, 6, 52, 46, 100, 40, 23, 52, 6, 82, 99, 4, 2, 42, 15, 81, 3, 3, 44, 76, 27, 31, 92, 9, 11, 19, 22, 74, 44, 40, 19, 100, 7, 74, 36, 1, 1, 33, 47, 11, 57, 2, 37, 95, 19, 74, 31, 67, 85, 98, 88, 45, 44, 92, 7, 86, 24, 58, 6, 22, 45, 28, 43, 75, 34, 38, 80, 15, 80, 87, 39, 30, 57, 92, 45, 81, 79, 52, 60, 65, 45, 34, 60, 15, 61, 13, 89, 32, 44, 97, 52, 42, 12, 61, 17, 90, 99, 2, 54, 84, 47, 43, 53, 51, 97, 79, 80, 67, 45, 3, 35, 2, 25, 75, 14, 60, 96, 31]

ANNEXE B

ANNEXE B - VISUALISATION DES PERFORMANCES

Visualisation des performances

B.1 Instance à 50 jobs

B.1.1 L'algorithme génétique (GA)

Pop size	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
50	2784	2807	2790.1	12.204
100	2783	2802	2787.6	24.503
150	2776	2792	2784.0	36.974
200	2776	2788	2781.9	50.403
250	2776	2784	2780.6	61.883

TAB. B.1 : : Performances de l'algorithme génétique (GA) pour 50 jobs dans un problème flowshop.

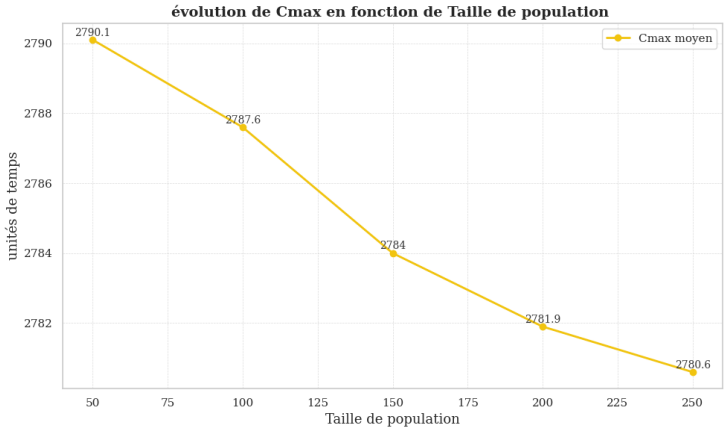

FIG. B.1 : Évolution du makespan moyen (C_{max}) pour différentes tailles de population avec l'algorithme génétique (GA), pour 50 jobs.

B.1.2 Optimisation par essaim de particules (PSO)

Num particles	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
50000	2859	2911	2899.2	114.392
100000	2859	2880	2861.1	218.658
150000	2854	2869	2861.5	324.082
200000	2869	2899	2881.5	437.240
250000	2887	2899	2889.6	538.532

TAB. B.2 : Performances de l'optimisation par essaim de particules (PSO) pour 50 jobs dans un problème flowshop.

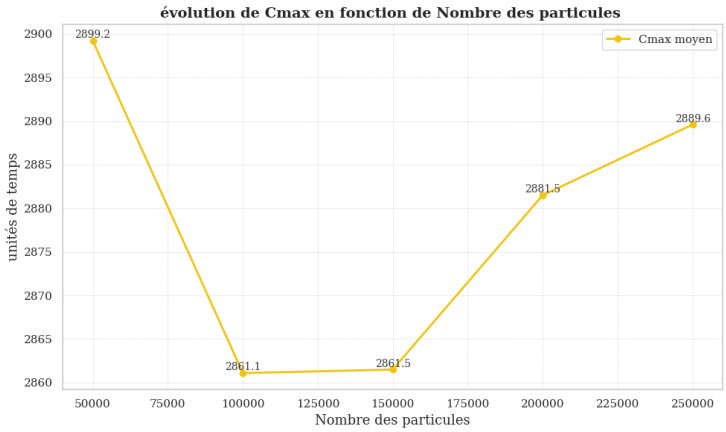

FIG. B.2 : Évolution du makespan moyen (C_{max}) pour différentes tailles de nuées dans l'algorithme d'optimisation par essaim de particules (PSO), pour 50 jobs.

B.1.3 Recherche tabou (TS)

Tenure	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
10	2783	2813	2790.4	370.636
20	2778	2794	2787.0	405.162
30	2776	2813	2789.5	491.454
40	2778	2793	2786.9	327.089
50	2776	2813	2787.7	422.014

TAB. B.3 : Performances de la recherche tabou (TS) pour 50 jobs dans un problème flowshop.

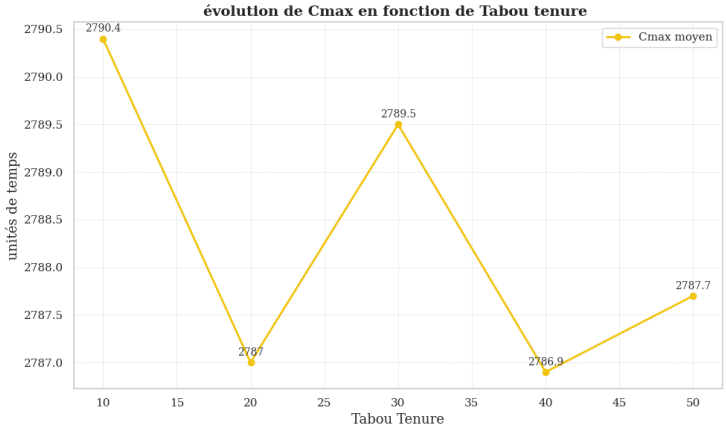

FIG. B.3 : Évolution du makespan moyen (C_{max}) en fonction de la taille de la mémoire (tenure) dans la méthode tabu search (TS), pour 50 jobs.

B.1.4 Recuit simulé (SA)

Initial temp	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
50	2759	2779	2776.2	42.404
100	2759	2779	2776.2	46.433
150	2776	2779	2777.5	49.262
200	2776	2779	2777.3	53.268
250	2776	2779	2777.9	53.201

TAB. B.4 : Performances du recuit simulé (SA) pour 50 jobs dans un problème flowshop.

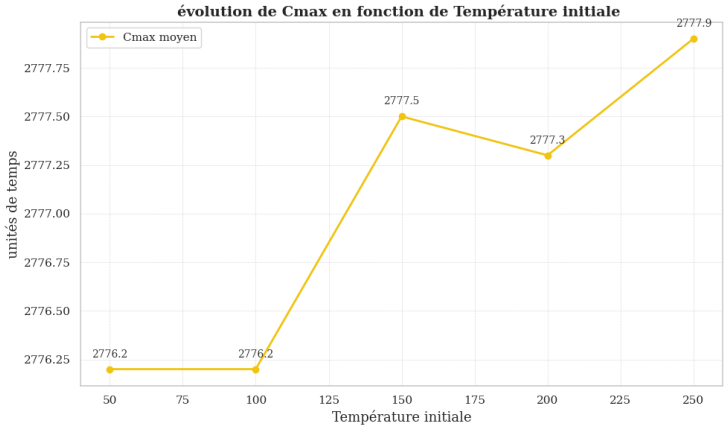

FIG. B.4 : Évolution du makespan moyen (C_{max}) pour différentes températures initiales avec l'algorithme de recuit simulé (SA), pour 50 jobs.

B.2 Instance à 100 jobs

B.2.1 L'algorithme génétique (GA)

Pop size	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
100	5495	5511	5503.2	66.330
200	5495	5518	5500.3	133.792
300	5495	5508	5498.7	207.703
400	5495	5507	5496.5	206.394
500	5495	5507	5496.7	206.889

TAB. B.5 : : Performances de l'algorithme génétique (GA) pour 100 jobs dans un problème flowshop.

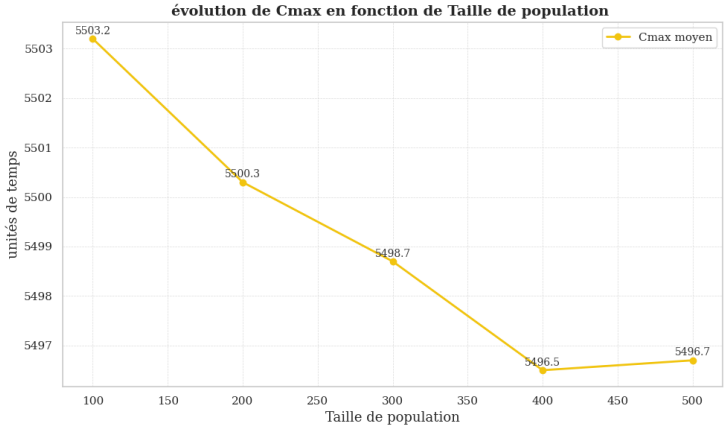

FIG. B.5 : Évolution du makespan moyen (C_{max}) pour différentes tailles de population avec l'algorithme génétique (GA), pour 100 jobs.

B.2.2 Optimisation par essaim de particules (PSO)

Num particles	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
100000	5624	5632	5625.6	528.370
200000	5581	5581	5581.0	1032.208
300000	5612	5612	5612.0	1549.992
400000	5581	5620	5598.9	2686.969
500000	5601	5620	5604.4	2439.813

TAB. B.6 : Performances de l'Optimisation par Essaim de Particules (PSO) pour 100 jobs dans un problème flowshop.

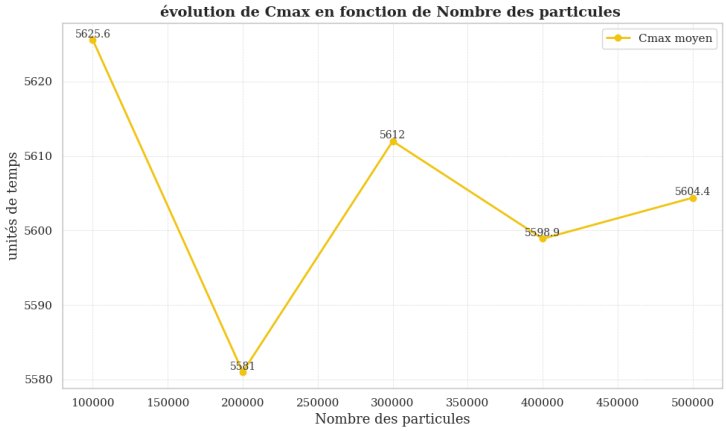

FIG. B.6 : Évolution du makespan moyen (C_{max}) pour différentes tailles de nuées dans l'algorithme d'optimisation par essaim de particules (PSO), pour 100 jobs.

B.2.3 Recherche tabou (TS)

Tenure	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
10	5495	5518	5506.0	4747.797
20	5495	5579	5514.0	6347.914
30	5495	5507	5502.0	5151.847
40	5495	5495	5504.5	4822.505
50	5495	5508	5504.7	5920.873

TAB. B.7 : Performances de la Recherche Tabou (TS) pour 100 jobs dans un problème flowshop.

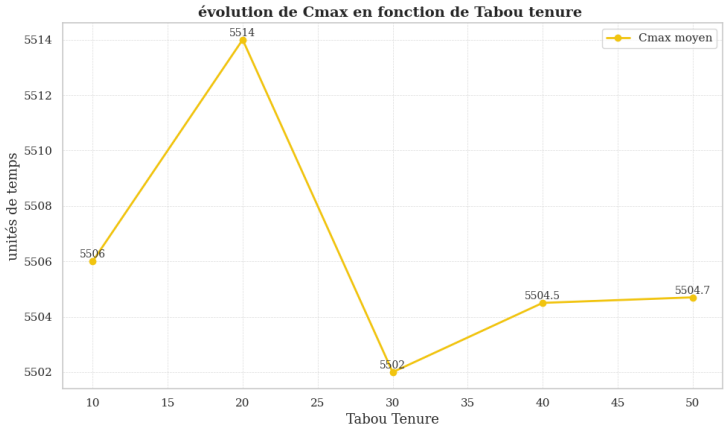

FIG. B.7 : Évolution du makespan moyen (C_{max}) en fonction de la taille de la mémoire (tenure) dans la méthode Tabu Search (TS), pour 100 jobs.

B.2.4 Recuit simulé (SA)

Initial temp	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
100	5495	5496	5495.1	94.964
200	5495	5496	5495.2	100.544
300	5495	5507	5497.6	108.247
400	5495	5495	5507.0	109.539
500	5495	5495	5507.0	114.674

TAB. B.8 : Performances du recuit simulé (SA) pour 100 jobs dans un problème flowshop.

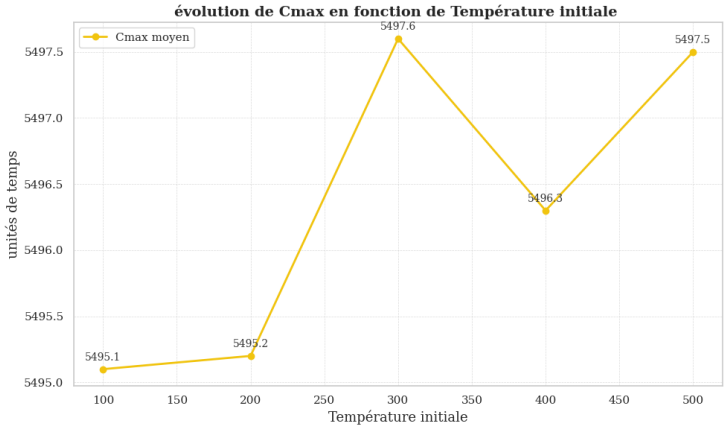

FIG. B.8 : Évolution du makespan moyen (C_{max}) pour différentes températures initiales avec l'algorithme de recuit simulé (SA), pour 100 jobs.

B.3 Instance à 150 jobs

B.3.1 L'algorithme génétique (GA)

Pop size	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
150	7964	8002	7977.8	131.179
300	7964	7981	7970.5	268.867
450	7964	7981	7972.8	399.203
600	7964	7981	7968.1	534.208
750	7964	7971	7964.7	672.401

TAB. B.9 : : Performances de l'algorithme génétique (GA) pour 150 jobs dans un problème flowshop.

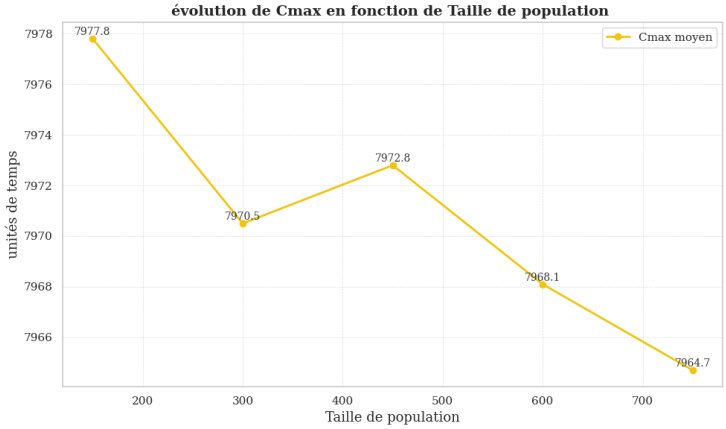

FIG. B.9 : Évolution du makespan moyen (C_{max}) pour différentes tailles de population avec l'algorithme génétique (GA), pour 150 jobs

B.3.2 Optimisation par essaim de particules (PSO)

Num particles	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
150000	8177	8192	8185.3	956.48
300000	8158	8158	8158.0	1913.25
450000	8141	8172	8154.4	3218.32
600000	8155	8168	8157.5	4429.12
750000	8128	8177	8141.0	6003.84

TAB. B.10 : Performances de l'Optimisation par Essaim de Particules (PSO) pour 150 jobs dans un problème flowshop.

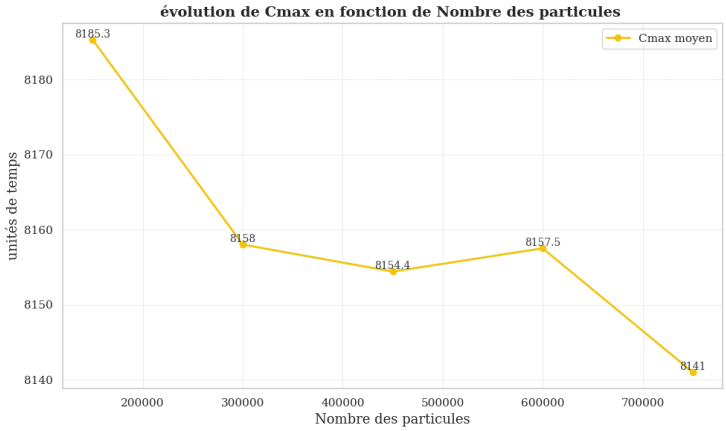

FIG. B.10 : Évolution du makespan moyen (C_{max}) pour différentes tailles de nuées dans l'algorithme d'optimisation par essaim de particules (PSO), pour 150 jobs.

B.3.3 Recherche tabou (TS)

Tenure	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
10	7958	8002	7964.3	8979.064
20	7958	7981	7962.7	8144.174
30	7958	7964	7959.2	8326.996
40	7958	8002	7966.5	8099.609
50	7958	8017	7966.3	8244.729

TAB. B.11 : Performances de la recherche tabou (TS) pour 150 jobs dans un problème flowshop.

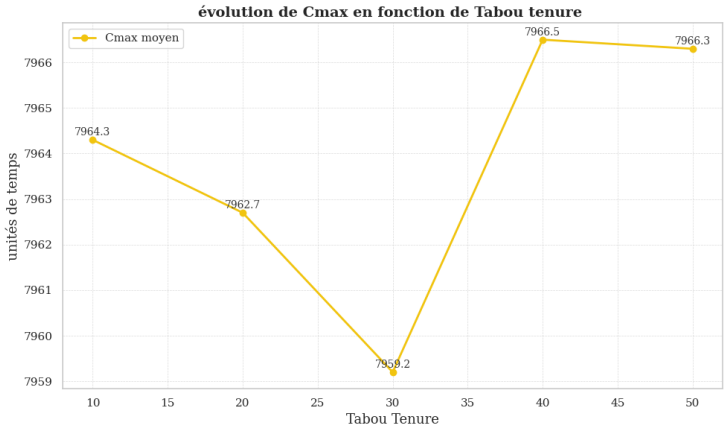

FIG. B.11 : Évolution du makespan moyen (C_{max}) en fonction de la taille de la mémoire (tenure) dans la méthode Tabu Search (TS), pour 150 jobs.

B.3.4 Recuit simulé (SA)

Initial temp	min C_{max}	max C_{max}	avg C_{max}	Runtime (s)
150	7964	7964	7964.0	188.042
300	7958	7970	7963.4	185.183
450	7958	7971	7964.1	194.148
600	7958	7964	7961.4	173.511
750	7958	7970	7964.0	176.825

TAB. B.12 : Performances du recuit simulé (SA) pour 150 jobs dans un problème flowshop.

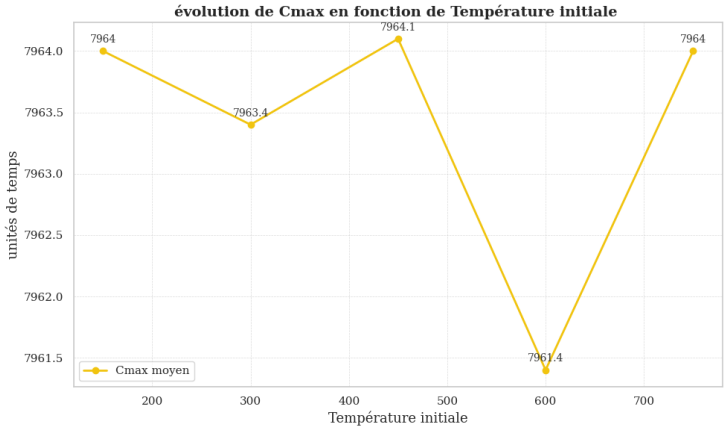

FIG. B.12 : Évolution du makespan moyen (C_{max}) pour différentes températures initiales avec l'algorithme de recuit simulé (SA), pour 150 jobs

Résumé : Ce mémoire porte sur l'optimisation de l'ordonnancement dans un atelier flowshop, un problème NP-difficile. Il compare les performances de quatre métaheuristiques (GA, SA, PSO, TS) ainsi que deux approches hybrides (GA-SA et PSO-TS) pour la minimisation du makespan (C_{max}). Les algorithmes ont été codés en Python et testés sur des instances de référence. L'analyse repose sur la qualité des solutions, le temps de calcul et la robustesse. Les résultats montrent la supériorité de l'approche hybride GA-SA, qui allie exploration globale et intensification locale. Cette étude confirme l'intérêt des méthodes hybrides pour résoudre efficacement des problèmes complexes d'ordonnancement industriel.

Mots clés : Flowshop, Ordonnancement, Métaheuristiques, Makespan, Algorithme Génétique (GA), Recuit Simulé (SA), Optimisation par Essaim de Particules (PSO), Recherche Tabou (TS), Approches hybrides, Python, Robustesse, Temps de calcul, NP-difficile.

ملخص : يهدف هذا البحث إلى دراسة ومقارنة أداء مجموعة من الخوارزميات الميتاابتكارية لحل مشكلة جدولة المهام في ورشات فلوشوب، والتي تُعد من المشاكل المعقدة. تم تطبيق أربع خوارزميات رئيسية: الخوارزمية الجينية، التبريد المحاكى، تحسين سرب الجسيمات، والبحث المحظور، إضافة إلى مقاربتين هجنتين. اعتمد التقييم على جودة الحلول، زمن التنفيذ، والاستقرار. تمت البرمجة باستخدام لغة بايثون واختبار النتائج على بيانات معيارية. أظهرت النتائج فعالية المقاربة الهجينة GA-SA في تحسين الأداء. تُبرز الدراسة دور التهجين بين الاستكشاف والتكثيف في حل مشاكل الجدولة. وتوصي بتطوير مقاربات أكثر تكيفًا للبيئات الصناعية المعقدة.

الكلمات المفتاحية : فلوشوب، جدولة الإنتاج، الخوارزميات الميتاابتكارية، زمن الإنجاز الأقصى، الخوارزمية الجينية، التبريد المحاكى، تحسين سرب الجسيمات، البحث المحظور، المقاربات الهجينة، بايثون.

Abstaract : Scheduling in flow shop environments represents a well-known NP-hard problem with significant practical relevance in industrial production systems. This thesis presents a comprehensive comparative study of four classical metaheuristics: Genetic Algorithm (GA), Simulated Annealing (SA), Particle Swarm Optimization (PSO), and Tabu Search (TS), alongside two hybrid approaches combining GA with SA and PSO with TS. The algorithms were developed and implemented using the Python programming language and evaluated on standard benchmark instances. The performance evaluation is based on solution quality, computational efficiency, and robustness. The experimental results indicate that the hybrid GA-SA approach outperforms other methods by effectively integrating global exploration and local intensification strategies to minimize the makespan (C_{max}). These findings substantiate the efficacy of hybrid metaheuristics in addressing complex scheduling problems in industrial contexts and suggest promising avenues for the development of more adaptive and efficient optimization techniques.

Key words : Flowshop, Scheduling, Metaheuristics, Makespan, Genetic Algorithm (GA), Simulated Annealing (SA), Particle Swarm Optimization (PSO), Tabu Search (TS), Hybrid Approaches, Python, Robustness, Computational Time, NP-hard.