République algérienne démocratique et populaire
Ministère de l'enseignement supérieur et de la recherche scientifique
Université 20 août 1955—Skikda
Département d'informatique

Ordonnancement en ateliers à flux continu par approches métaheuristiques

Mémoire de fin d'études
pour l'obtention du diplôme de master en informatique
Spécialité : systèmes informatiques

Préparé par :
M. Yamine Kahoul
Mlle Oujdane Boustar

Membres du jury :

M. Abdelhak Mensoul	Maître-assistant 'A'	Président
M. Rabah Mazouzi	Maître-assistant	Examinateur
M. Mohammed Redjimi	Professeur	Encadrant

Année universitaire 2024/2025

ملخص

يتناول هذا البحث المشكلة المعقدة المتمثلة في الجدولة في الورشات ذات التدفق المستمر، وهي قضية استراتيجية رئيسية لتحسين أداء الأنظمة الصناعية الحديثة. الهدف هو تحسين كفاءة خطوط الإنتاج من خلال خفض قيم معايير مقياسية مثل الوقت الإجمالي التصنيع. بعد استعراض الطرق الحالية، وخاصة الخوارزميات التقديرية والخوارزميات فوق التقديرية، يقترح البحث منهجية تعتمد على التحسين بسرب الجسيمات، مصممة خصيصًا للتعامل مع القيود التوليفية للمشكلة. تم اختبار المنهجية المقترحة على مجموعات بيانات معيارية وتمت مقارنتها مع الطرق الرائدة في المجال، حيث أنها أظهرت متانة وكفاءة تنافسية. تؤكد النتائج جدوى هذا النهج وتفتح آفاقاً لتحسينات مستقبلية، لا سيما من خلال دمج جوانب منهجية متقدمة أخرى وتطبيقات عملية في البيئة الصناعية.

الكلمات المفتاحية—الجدولة في الورشات ذات التدفق المستمر، الخوارزميات التقديرية، الخوارزميات فوق التقديرية، التحسين بسرب الجسيمات، إدارة الإنتاج، الصناعة التحويلية.

Abstract

This thesis addresses the complex problem of flow shop scheduling, a major strategic challenge for optimizing modern industrial systems. The objective is to enhance the performance of production lines by minimizing criteria such as makespan. After reviewing existing methods, mainly heuristics and metaheuristics, the study proposes an approach based on particle swarm optimization, adapted to the combinatorial constraints of the problem. The proposed approach was tested on standardized datasets and compared with state-of-the-art methods, demonstrating competitive robustness and efficiency. The results confirm the relevance of this approach and open up prospects for future improvements, particularly through the integration of other advanced methodological aspects and practical applications in industrial settings.

Keywords—flow shop scheduling, heuristics, metaheuristics, particle swarm optimization, production management, manufacturing industry.

Résumé

Ce mémoire traite du problème complexe de l'ordonnancement en ateliers à flux continu, un enjeu stratégique majeur pour l'optimisation des systèmes industriels modernes. L'objectif est d'améliorer la performance des lignes de production en minimisant des critères tels que le temps de fabrication total. Après une revue des méthodes existantes, principalement les heuristiques et métaheuristiques, l'étude propose une approche basée sur l'optimisation par essaims de particules, adaptée aux contraintes combinatoires du problème. L'approche proposée a été testée sur des jeux de données standardisés et comparée aux méthodes de l'état de l'art, démontrant une robustesse et une efficacité compétitives. Les résultats confirment la pertinence de cette approche et ouvrent des perspectives pour des améliorations futures, notamment par la prise en compte d'autres aspects méthodologiques avancés et des applications pratiques en milieu industriel.

Mots clés—ordonnancement en ateliers à flux continu, heuristiques, métaheuristiques, optimisation par essaims de particules, gestion de production, industrie manufacturière.

Remerciements

Nous remercions tout particulièrement M. Mohammed Redjimi pour avoir dirigé ce travail, ainsi que pour son accompagnement, ses conseils précieux et sa disponibilité tout au long de cette recherche. Nous adressons également nos sincères remerciements à M^{me} Amina Remichi et à M. Rabah Mazouzi pour nous avoir fait l'honneur d'accepter d'évaluer ce mémoire. Nos remerciements vont aussi à nos familles respectives et à nos amis pour leur soutien moral constant. Enfin, nous tenons à exprimer notre profonde gratitude à toutes les personnes qui ont contribué, de près ou de loin, à la réalisation de ce travail.

Yamine Kahoul Oujdane Boustar

Table des matières

Table des figures vii

Liste des tableaux ix

Liste des algorithmes x

1 Introduction 1
- 1.1 Contexte et motivations . . . 1
- 1.2 Plan du mémoire . . . 2

2 Ordonnancement en ateliers à flux continu 3
- 2.1 Ordonnancement de tâches . . . 3
- 2.2 Ordonnancement en ateliers à flux continu . . . 5
- 2.3 Complexité, contraintes et critères d'évaluation . . . 6
- 2.4 Génération et visualisation des solutions . . . 8

3 Métaheuristiques 15
- 3.1 Optimisation . . . 15
- 3.2 Métaheuristiques . . . 20
- 3.3 Catégorisation et principales approches . . . 23
- 3.4 Enjeux et perspectives . . . 25

4 Approche proposée 29
- 4.1 Hypothèses de travail . . . 29
- 4.2 Optimisation par essaims de particules . . . 30
- 4.3 Prise en compte de l'aspect combinatoire . . . 35
- 4.4 Gestion de la stagnation . . . 40

5 Résultats et discussion 43
- 5.1 Jeu de données utilisé . . . 43
- 5.2 Tests effectués . . . 45
- 5.3 Analyse et discussion . . . 48

6 Conclusion **64**
6.1 Principaux résultats 64
6.2 Améliorations et perspectives 65

Références **68**

Table des figures

1 Ouvriers sur la première chaîne de montage des automobiles Ford de 1913. . . . 4
2 Diagramme de Gantt associé au planning $(3, 1, 2)$. 14

3 Encodage des solutions dans le cadre de l'optimisation (Talbi, 2009). 17
4 Problème d'optimisation avec plusieurs optimums locaux (Siarry, 2016). 18
5 Méthodes exactes et approximatives d'optimisation (Talbi, 2009). 19
6 Méthodes exactes d'optimisation (Talbi, 2009). 20
7 Généalogie des métaheuristiques (Talbi, 2009). 21

8 Une nuée d'étourneaux se comportant comme un essaim. 31
9 Mécanismes guidant le mouvement des particules (Gendreau and Potvin, 2019). 33
10 Exemples de voisinages utilisés dans le modèle l_i^{best} (Gendreau and Potvin, 2019). 34

11 Traitement de l'instance 20_5_1 en utilisant l'interface graphique. 47
12 Corrélations entre les paramètres et les résultats. 49
13 Influence de la taille de l'essaim sur la meilleure adaptation globale. 50
14 Influence de la taille de l'essaim sur le temps d'exécution. 51
15 Influence de la taille des voisinages sur la meilleure adaptation globale. 51
16 Influence de la taille des voisinages sur le temps d'exécution. 52
17 Influence de l'inertie sur la meilleure adaptation globale. 53
18 Influence de l'inertie sur le temps d'exécution. 53
19 Influence du coefficient cognitif sur la meilleure adaptation globale. 54
20 Influence du coefficient cognitif sur le temps d'exécution. 54
21 Influence du coefficient social sur la meilleure adaptation globale. 55
22 Influence du coefficient social sur le temps d'exécution. 55
23 Influence du taux de diversification sur la meilleure adaptation globale. 56
24 Influence du taux de diversification sur le temps d'exécution. 57
25 Influence du nombre maximal d'itérations autorisées sur la meilleure adaptation globale. 57
26 Influence du nombre maximal d'itérations autorisées sur le temps d'exécution. . 58
27 Évolution de la longueur moyenne des vélocités des particules de l'essaim. . . . 61

28 Évolution de la meilleure adaptation globale de l'essaim. 61
29 Diagramme de Gantt associé au planning trouvé pour l'instance 20_5_1. 62

Liste des tableaux

1 Exemple de temps de traitement en minutes T pour 3 tâches et 5 machines. . . 5
2 Explosion combinatoire du nombre de plannings possibles. 6
3 Makespan en minutes des 6 plannings possibles. 8
4 Informations temporelles associées au planning optimal $(3, 1, 2)$. 13
5 Espaces de solutions continus et à base de permutations. 35
6 Instance 20_5_1 du jeu de données de Taillard (1993). 44
7 Valeurs possibles pour les paramètres. 45
8 Résultats obtenus en utilisant les paramètres extraits des tests. 60

Liste des algorithmes

1 Calcul du makespan. 7
2 Règle de Johnson ($m = 2$). 9
3 Heuristique de Palmer (SIR). 10
4 Heuristique de Campbell-Dudek-Smith (CDS) ($m \geq 2$). 11
5 Heuristique de Gupta. 11
6 Heuristique de Dannenbring (RAH). 12
7 Heuristique de Nawaz-Enscore-Ham (NEH). 12
8 Heuristique de Suliman. 13

9 Structure générale d'une métaheuristique. 22

10 Optimisation par essaims de particules. 32
11 Calcul de la vélocité entre deux permutations. 37
12 Fusion probabiliste des vélocités. 39
13 Changement d'une position selon une vélocité. 40

Chapitre 1

Introduction

L'ordonnancement de la production constitue un enjeu stratégique majeur dans les systèmes industriels modernes. Face à la complexité croissante des processus de fabrication et à la demande accrue de flexibilité et d'efficacité, l'optimisation des plannings de production est devenue une priorité incontournable. Ce chapitre introduit le contexte général de cette problématique, présente les motivations de cette étude, et offre un aperçu de la structure du mémoire, permettant ainsi au lecteur de suivre les différentes étapes de l'analyse et des développements proposés.

1.1 Contexte et motivations

Un atelier à flux continu est caractérisé par une série de machines disposées en séquence, où chaque tâche doit être traitée sur chaque machine selon le même ordre (Baker and Trietsch, 2019 ; Blazewicz et al., 2019 ; Emmons and Vairaktarakis, 2013 ; Pinedo, 2022). Ce problème d'ordonnancement est dit NP-difficile, ce qui implique qu'il n'existe pas d'algorithme de résolution efficace pour les instances de grande taille. En pratique, cela se traduit par la nécessité de recourir à des méthodes d'optimisation approchées pour trouver des solutions satisfaisantes en un temps de calcul raisonnable. Les conséquences d'un ordonnancement inefficace peuvent être désastreuses pour une entreprise, entrainant des retards de production, une augmentation des couts, une baisse de la qualité et une insatisfaction des clients. Par conséquent, le développement de méthodes d'ordonnancement efficaces est un domaine de recherche crucial en génie industriel et en gestion des opérations.

Ce mémoire s'inscrit dans ce contexte et vise à étudier le problème d'ordonnancement dans les ateliers à flux continu, en mettant l'accent sur le développement et l'évaluation de méthodes d'optimisation adéquates. Plus précisément, les objectifs de cette recherche sont les suivants :

1. **Analyser les approches existantes :** effectuer une revue de la littérature consacrée à l'ordonnancement des ateliers à flux continu, en examinant les différentes méthodes de résolution proposées, leurs avantages et leurs limites. Cette analyse permettra de situer la

présente recherche par rapport aux travaux antérieurs et d'identifier les points qui justifient de nouvelles investigations.

2. **Proposer une nouvelle approche :** concevoir et implémenter une méthode d'optimisation, basée sur une métaheuristique, pour résoudre le problème d'ordonnancement des ateliers à flux continu. Cette démarche revêt un double intérêt, à la fois théorique et pratique. D'un point de vue théorique, elle permettra d'appliquer et de mettre en œuvre les notions de recherche et de méthodologie étudiées dans le cadre de notre formation académique, en explorant les potentialités des métaheuristiques pour résoudre un problème d'optimisation complexe (Gendreau and Potvin, 2019 ; Martí et al., 2018 ; Siarry, 2016 ; Talbi, 2009). D'un point de vue pratique, l'approche proposée devra être capable de trouver des solutions de bonne qualité tout en garantissant un temps de calcul raisonnable.

3. **Évaluer les performances :** comparer les performances de la méthode proposée avec celles d'autres méthodes de l'état de l'art, en utilisant des jeux de données de référence issus de la littérature (Taillard, 1993). Cette évaluation permettra de mesurer l'efficacité et la robustesse de l'approche proposée, et de déterminer dans quelles conditions elle est la plus performante. Cette recherche vise également à ouvrir de nouvelles perspectives pour les travaux futurs dans le domaine de l'ordonnancement des ateliers à flux continu.

1.2 Plan du mémoire

L'organisation de ce document suit une progression méthodique, où chaque chapitre traite une problématique précise, permettant ainsi une compréhension graduelle et cohérente des enjeux et des résultats exposés :

- Le **chapitre 2** présente les concepts de base de l'ordonnancement et les spécificités des ateliers à flux continu, en abordant leur formalisation, leurs contraintes et leurs critères d'évaluation.
- Le **chapitre 3** est consacré à l'étude des métaheuristiques, en présentant les principes généraux, les principales approches et les enjeux de ce domaine de l'optimisation.
- Le **chapitre 4** décrit la méthode d'optimisation proposée pour résoudre le problème d'ordonnancement des ateliers à flux continu.
- Le **chapitre 5** présente les résultats expérimentaux obtenus, en comparant les performances de la méthode proposée avec celles d'autres méthodes de l'état de l'art.
- Enfin, le **chapitre 6** conclut ce mémoire en résumant les principales contributions de la recherche et en proposant des perspectives pour les travaux futurs.

Chapitre 2

Ordonnancement en ateliers à flux continu

L'ordonnancement en ateliers à flux continu vise à organiser l'enchainement des tâches pour améliorer la performance du système de production, ce qui est essentiel pour optimiser l'utilisation des ressources et respecter les contraintes opérationnelles. Ce chapitre commence par une présentation des principes de l'ordonnancement et des différentes structures d'ateliers avant de se concentrer sur le cas spécifique des ateliers à flux continu, en abordant leur formalisation, leurs contraintes et leurs critères d'évaluation. La complexité du problème étant exponentielle, diverses approches sont étudiées, notamment les approches heuristiques de Johnson, de Palmer, de Campbell-Dudek-Smith et de Nawaz-Enscore-Ham. Enfin, une attention particulière est accordée à la visualisation et à l'analyse des solutions, à travers des outils comme les diagrammes de Gantt, afin d'éclairer les choix stratégiques en matière de planification industrielle.

2.1 Ordonnancement de tâches

L'ordonnancement (Scheduling) est une discipline clé de la *recherche opérationnelle* et de l'*optimisation combinatoire* qui vise à organiser de façon optimale l'exécution de tâches sur des ressources limitées afin d'atteindre un objectif spécifique. Il joue un rôle fondamental dans la gestion des *ateliers*, des systèmes de production et dans le domaine informatique, où l'optimisation de l'affectation des ressources peut influencer considérablement les performances globales (Figure 1). L'importance de l'ordonnancement réside dans sa capacité à réduire les couts, améliorer l'efficacité et maximiser l'utilisation des équipements. Dans l'industrie manufacturière, un ordonnancement efficient permet d'éviter les goulots d'étranglement et d'améliorer la livraison des produits dans les délais impartis. En informatique, il permet d'optimiser l'exécution des processus sur des processeurs multicœurs. L'ordonnancement s'applique donc à un large éventail de secteurs et représente un défi majeur en raison de la complexité croissante des systèmes à gérer (Baker and Trietsch, 2019 ; Blazewicz et al., 2019 ; Emmons and Vairaktarakis, 2013 ; Pinedo,

FIG. 1: Ouvriers sur la première chaine de montage des automobiles Ford de 1913[1].

2022).

Les notions fondamentales de l'ordonnancement incluent plusieurs concepts clés, comme les *tâches (jobs)*, les *machines*, les *plannings* et les *contraintes de traitement*. Chaque tâche doit être exécutée selon un ordre précis, parfois avec des dépendances entre elles. Quant aux machines, elles peuvent être unitaires, avec une seule machine par tâche, ou intégrées à des systèmes plus complexes où plusieurs machines opèrent en parallèle ou en série. L'ordonnancement peut également inclure des contraintes, comme la disponibilité des ressources, les délais d'exécution ou les couts associés. Il est essentiel de prendre en compte ces paramètres pour définir des algorithmes efficaces capables de produire des solutions optimales ou quasi optimales.

Plus particulièrement, les problèmes d'ordonnancement en atelier se classent en plusieurs catégories en fonction de la structure de l'atelier et des règles de séquencement des tâches. Il est possible de distinguer principalement trois types :

- **Atelier à tâches (Job Shop) :** chaque tâche suit un itinéraire unique à travers les machines, mais ces itinéraires peuvent différer d'une tâche à l'autre, rendant la planification complexe.

[1]Image dans le domaine public aux États-Unis, disponible sur Wikimedia Commons.

TAB. 1: Exemple de temps de traitement en minutes T pour 3 tâches et 5 machines.

	Machine 1	**Machine 2**	**Machine 3**	**Machine 4**	**Machine 5**
Tâche 1	5	8	3	6	4
Tâche 2	6	7	4	5	3
Tâche 3	4	9	5	7	6

- **Atelier à flux continu (Flow Shop) :** ce cas impose un ordre fixe de passage sur les machines pour toutes les tâches simplifiant certains aspects, mais posant d'autres défis d'optimisation.
- **Atelier ouvert (Open Shop) :** l'ordre de passage des tâches sur les machines n'est pas prédéfini, ce qui introduit une flexibilité maximale, mais une difficulté accrue pour trouver des solutions optimales.

2.2 Ordonnancement en ateliers à flux continu

L'ordonnancement en ateliers à flux continu (Flow Shop Scheduling) consiste à organiser le traitement de n tâches qui doivent suivre le même parcours à travers m machines, chacune effectuant une opération spécifique. Pour chaque tâche i (avec $i = 1, 2, \ldots, n$), un temps d'exécution t_{ij} est alloué sur la machine j (avec $j = 1, 2, \ldots, m$). L'objectif principal est généralement de minimiser le temps global d'exécution tout en optimisant la séquence de passage des n tâches sur les m machines. Mathématiquement, cela revient à trouver la *permutation* optimale des tâches, c'est-à-dire l'ordre dans lequel elles seront traitées, afin de minimiser le temps d'occupation des ressources tout en respectant les contraintes du système de production. Algorithmiquement, cela revient typiquement à prendre en entrée la matrice des temps d'exécution $T = (t_{ij})$ et à produire en sortie un planning $P = (p_i)$ reflétant la séquence de passage des n tâches sur les m machines[2].

Pour illustrer ce concept de manière concrète, considérons un atelier de production dans lequel 3 tâches doivent être traitées sur 5 machines disposées en série. Par exemple, une entreprise de fabrication mécanique peut organiser la production de 3 types de pièces, chaque pièce devant passer par 5 opérations successives : préparation, assemblage, inspection, emballage et étiquetage. Le tableau 1 donne les temps de traitement (exprimés en minutes) de chaque tâche sur les différentes machines[3].

En optimisant l'ordre de passage des tâches, l'atelier peut réduire les temps d'attente entre les opérations et éviter les goulots d'étranglement. Du point de vue mathématique, le problème se formalise à l'aide de modèles combinatoires ou de programmes linéaires en nombres entiers. Ces formulations traduisent précisément les contraintes liées à l'ordre fixe des machines et aux durées

[2]Les notations utilisées ici restent valables tout au long du document.

[3]Les temps de traitement présentés dans ce tableau servent uniquement à illustrer le concept étudié.

Tab. 2: Explosion combinatoire du nombre de plannings possibles.

Nombre de tâches (n)	**Nombre de plannings possibles** ($n!$)
1	1
2	2
3	6
4	24
5	120
⋮	⋮
10	3628800
⋮	⋮
15	1307674368000
⋮	⋮
20	2432902008176640000
⋮	⋮

variables de traitement pour chaque tâche. Toutefois, la complexité du problème croît de manière exponentielle avec le nombre de tâches et de machines, rendant l'utilisation de méthodes exactes impraticable pour des applications industrielles à grande échelle (Tableau 2). Pour l'exemple précédemment cité avec 3 tâches, on dénombre seulement $3! = 6$ plannings possibles pour représenter les différents ordres de passage des tâches sur les 5 machines : $(1, 2, 3)$, $(1, 3, 2)$, $(2, 1, 3)$, $(2, 3, 1)$, $(3, 1, 2)$ et $(3, 2, 1)$. En revanche, dans un cas plus réaliste avec 20 tâches, ce nombre dépasse deux-trillions, rendant toute approche exhaustive inenvisageable. Il devient alors essentiel de recourir à des méthodes d'optimisation approchées pour résoudre efficacement le problème.

2.3 Complexité, contraintes et critères d'évaluation

Le problème d'ordonnancement en ateliers à flux continu est un exemple typique des défis posés par l'optimisation combinatoire. Il est classé comme *NP-difficile*, ce qui signifie qu'aucun algorithme polynomial n'est connu pour garantir une solution optimale pour toutes les instances. Cette complexité provient non seulement de l'explosion combinatoire du nombre de plannings possibles, mais aussi du nombre de machines impliquées, qui influence directement la difficulté du problème. Plus le nombre de machines est élevé, plus le calcul des critères d'évaluation de la qualité de chaque planning devient complexe, rendant l'exploration exhaustive des solutions encore plus difficile.

En parallèle, plusieurs contraintes viennent complexifier l'ordonnancement. Certaines sont inhérentes à la structure du système de production, tandis que d'autres reflètent des exigences opérationnelles et stratégiques :

- **Contraintes de séquencement :** chaque tâche doit suivre un ordre précis sur les ma-

chines, conformément au processus de fabrication.

- **Variabilité des temps de traitement :** les durées d'exécution peuvent dépendre de la machine utilisée, nécessitant une gestion adaptative de l'ordonnancement.
- **Échéances strictes :** certains produits doivent être livrés à des dates précises, imposant des contraintes fortes sur l'ordonnancement.
- **Contraintes de ressources :** les disponibilités en main-d'œuvre, en matières premières ou en équipements influencent la faisabilité d'un ordonnancement.
- **Considérations environnementales :** la réduction de l'empreinte carbone et de la consommation énergétique devient un critère déterminant dans les stratégies d'ordonnancement, notamment pour répondre aux exigences règlementaires et aux objectifs de développement durable.

Algorithme 1 : Calcul du makespan.

Entrées : Planning P, matrice des temps d'exécution T.
Sorties : Makespan de l'exécution de P en prenant en compte T.

```
1 O ← [0, ... 0] ;
2 pour chaque i de P faire
3 |   O[1] ← O[1] + T[i][1] ;
4 |   pour j de 2 à m faire
5 |   |   O[j] ← max(O[j], O[j − 1]) + T[i][j] ;
6 retourner O[m] ;
```

Plusieurs critères sont couramment employés pour évaluer la qualité d'un planning. La mesure la plus classique est le *makespan*, défini comme la durée totale nécessaire pour accomplir l'ensemble des tâches. Il représente le temps écoulé entre le début du traitement de la première tâche et la fin du traitement de la dernière tâche sur la dernière machine. Un makespan plus court indique généralement une meilleure utilisation des ressources et une meilleure efficacité de l'ordonnancement. Optimiser le makespan consiste donc à *minimiser* sa valeur. L'algorithme 1 permet de calculer le makespan en prenant en compte un planning et une matrice des temps d'exécution comme entrées. Il utilise un tableau de longueur m pour suivre l'avancement du traitement des tâches sur chaque machine et met à jour les temps en tenant compte des dépendances entre machines. En prenant l'exemple précédent avec 3 tâches et 5 machines, représenté par la matrice des temps d'exécution T, le makespan des 6 plannings possibles est donné dans le tableau 3. Les résultats montrent que l'exécution des tâches dans l'ordre $(3, 1, 2)$ dure exactement 40 minutes, ce qui correspond au meilleur cas possible (valeurs encadrées).

En plus du makespan, d'autres indicateurs permettent de capturer différents aspects de la performance d'un planning. À cet effet, il est possible de citer :

Tab. 3: Makespan en minutes des 6 plannings possibles.

Planning	**Makespan**
$(1, 2, 3)$	47
$(1, 3, 2)$	43
$(2, 1, 3)$	48
$(2, 3, 1)$	44
$\boxed{(3, 1, 2)}$	$\boxed{40}$
$(3, 2, 1)$	41

- **Temps d'inactivité (Idle time) :** il correspond aux périodes durant lesquelles certaines machines restent inemployées, ce qui peut affecter l'efficacité globale.
- **Retard global (Tardiness) :** il quantifie l'écart entre les dates d'achèvement réelles des tâches et leurs échéances prévues.
- **Équilibrage de charge (Load Balancing) :** visant à répartir de manière homogène les tâches entre les machines afin d'éviter les surcharges ou les sous-utilisations.

2.4 Génération et visualisation des solutions

Comme l'exemple précédent le montre, la *recherche exhaustive*, bien que théoriquement capable de trouver une solution optimale en explorant toutes les combinaisons possibles, devient rapidement impraticable pour les problèmes de grande taille en raison de son explosion combinatoire. Pour remédier à cela, des *méthodes exactes* comme la programmation dynamique et les algorithmes de séparation et évaluation (Branch and Bound) sont utilisées. Ces méthodes garantissent une solution optimale, mais leur complexité exponentielle limite leur application à des instances de taille modérée, ce qui correspond aux cas rencontrés en pratique.

Dans ces situations, il devient nécessaire d'adopter des approches plus pragmatiques. C'est pourquoi des *méthodes approximatives*, telles que les *heuristiques*, constituent une alternative efficace. Une heuristique est une méthode de résolution de problèmes qui utilise des règles empiriques ou des stratégies simplifiées pour trouver rapidement une solution satisfaisante, sans garantir qu'elle soit optimale.

Dans le contexte de l'ordonnancement en ateliers à flux continu, une heuristique permet d'établir un ordre de passage des tâches sur les machines en fonction de critères spécifiques, comme les temps de traitement, les délais ou la charge des ressources. Contrairement aux méthodes exactes, les heuristiques privilégient l'efficacité de calcul et la facilité d'application, ce qui les rend particulièrement adaptées aux problèmes complexes où une solution optimale est trop couteuse à obtenir.

Plusieurs heuristiques ont été développées pour améliorer l'ordonnancement en ateliers à flux

Algorithme 2 : Règle de Johnson ($m = 2$).

Entrées : Matrice des temps de traitement T.
Sorties : Planning P et makespan associé.

$G_1 \leftarrow [i \mid 1 \leq i \leq n \wedge T[i][1] \leq T[i][2]]$;
$G_2 \leftarrow [i \mid 1 \leq i \leq n \wedge T[i][1] > T[i][2]]$;
Trier G_1 selon $T[\cdot][1]$ en ordre croissant ;
Trier G_2 selon $T[\cdot][2]$ en ordre décroissant ;
$P \leftarrow G_1 \oplus G_2$[4] ;
retourner $P, \text{makespan}(P)$;

continu, chacune adoptant une approche spécifique pour optimiser la séquence des tâches et réduire le makespan. Ces heuristiques varient en complexité et en efficacité selon la structure du problème, la taille de l'atelier et les contraintes spécifiques du système de production. Parmi les plus connues, il est possible de citer :

- **Règle de Johnson (1954) :** cette heuristique conçue exclusivement pour des configurations à deux machines consiste à trier n tâches en fonction de leurs temps d'exécution sur les deux machines cibles. Si le temps d'exécution d'une tâche sur la première machine est plus court que sur la seconde, la tâche est placée en premier dans la séquence ; sinon, elle est placée en dernier (Algorithme 2).

- **Heuristique de Palmer (1965) :** connue aussi sous le nom de SIR (Slope Index Rule), elle classe les tâches en fonction de l'évolution de leurs temps de traitement d'une machine à l'autre. Elle attribue un score à chaque tâche en tenant compte de la tendance générale : les tâches dont les temps de traitement diminuent progressivement sont favorisées au début de la séquence, tandis que celles dont les temps augmentent sont placées en fin d'ordonnancement. Cette approche vise à répartir la charge de travail de manière plus équilibrée et à limiter les temps d'attente entre les machines (Algorithme 3).

- **Heuristique de Campbell et al. (1970) :** connue aussi sous le nom CDS (Campbell-Dudek-Smith), elle généralise la règle de Johnson en décomposant un problème à plusieurs machines en une série de sous-problèmes équivalents à un ordonnancement sur deux machines. En appliquant la règle de Johnson à ces sous-problèmes, elle génère plusieurs séquences possibles, parmi lesquelles elle sélectionne celle offrant le meilleur makespan. Cette approche permet d'explorer différentes configurations tout en conservant l'efficacité de Johnson pour les problèmes simplifiés (Algorithme 4).

- **Heuristique de Gupta (1971) :** elle établit un ordre de priorité des tâches en fonction d'un critère combinant les temps d'exécution sur les différentes machines. Chaque tâche se voit attribuer un score calculé comme une somme pondérée de ses temps de traitement,

[4] Le symbole $\oplus$ désigne l'opérateur de concaténation de séquences.

où le poids de chaque machine reflète son importance relative dans la séquence d'ordonnancement. Les tâches sont ensuite classées selon ce score afin de minimiser le makespan et d'améliorer l'équilibrage du flux de production (Algorithme 5).

- **Heuristique de Dannenbring (1977) :** connue aussi sous le nom RAH (Rapid Access Heuristic), elle vise à fournir une solution efficace en un temps réduit en évitant des calculs complexes. Chaque tâche reçoit un score basé sur une combinaison pondérée de ses temps de traitement, où les premières machines du processus ont un poids plus élevé afin de privilégier les tâches influençant davantage le flux de production (Algorithme 6).
- **Heuristique de Nawaz et al. (1983) :** connue aussi sous le nom NEH (Nawaz-Enscore-Ham), elle repose sur l'idée que les tâches ayant un temps de traitement total plus élevé doivent être priorisées, car elles influencent davantage la durée d'exécution. L'heuristique commence par trier les tâches par ordre décroissant de leur temps total, puis les insère une à une à la position minimisant le makespan, en testant plusieurs insertions possibles (Algorithme 7).
- **Heuristique de Suliman (2000) :** elle repose sur une approche en deux phases pour améliorer l'ordonnancement. Un planning initial est d'abord construit à l'aide d'heuristiques comme Palmer ou CDS. Ensuite, il est affiné par un échange directionnel de paires, où les tâches ne peuvent être déplacées que selon une règle prédéfinie, limitant ainsi le nombre de permutations évaluées (Algorithme 8).

Algorithme 3 : Heuristique de Palmer (SIR).

Entrées : Matrice des temps de traitement T.
Sorties : Planning P et makespan associé.

1 $P \leftarrow [i \mid 1 \leq i \leq n]$;
2 Trier P selon $\sum_j (m - 2j + 1) \cdot T[\cdot][j]$ en ordre décroissant ;
3 **retourner** $P, \text{makespan}(P)$;

Des études comparatives montrent que la performance des heuristiques dépend fortement de la taille et de la structure du problème. Par exemple, l'heuristique NEH tend à fournir la meilleure qualité de solution pour des problèmes de grande envergure, bien qu'elle implique un cout de calcul plus élevé. En revanche, l'heuristique CDS propose un compromis intéressant en offrant des solutions de qualité acceptable avec des temps de calcul plus courts, ce qui la rend particulièrement adaptée aux problèmes de taille moyenne. Les heuristiques telles que SIR et RAH se distinguent par leur capacité à générer rapidement des solutions, un atout dans les contextes où la réactivité est cruciale, même si cela se fait parfois au détriment d'une performance optimale. Enfin, les approches développées par Gupta et Suliman intègrent des raffinements spécifiques visant à optimiser l'équilibrage du flux de production dans des environnements complexes. Cependant, sur des problèmes de grande envergure, ces heuristiques peuvent fournir des solutions

de qualité inférieure par rapport à d'autres plus intensives en calcul, illustrant ainsi le compromis entre temps de calcul et performance (Baker and Trietsch, 2019 ; Blazewicz et al., 2019 ; Emmons and Vairaktarakis, 2013 ; Pinedo, 2022).

Algorithme 4 : Heuristique de Campbell-Dudek-Smith (CDS) ($m \geq 2$).

Entrées : Matrice des temps de traitement T.
Sorties : Planning P^* et makespan associé M^*.

$P^* \leftarrow [\,]$;
$M^* \leftarrow \infty$;
pour j **de** *1* **à** $m-1$ **faire**
  $S_1 \leftarrow [\sum_{k=1}^{j} T[i][k] \mid 1 \leq i \leq n]$;
  $S_2 \leftarrow [\sum_{k=j+1}^{m} T[i][k] \mid 1 \leq i \leq n]$;
  $G_1 \leftarrow [i \mid 1 \leq i \leq n \wedge S_1[i] \leq S_2[i]]$;
  $G_2 \leftarrow [i \mid 1 \leq i \leq n \wedge S_1[i] > S_2[i]]$;
  Trier G_1 selon $S_1[\cdot]$ en ordre croissant ;
  Trier G_2 selon $S_2[\cdot]$ en ordre décroissant ;
  $P \leftarrow G_1 \oplus G_2$;
  $M \leftarrow \text{makespan}(P)$;
  si $M < M^*$ **alors**
    $P^* \leftarrow P$;
    $M^* \leftarrow M$;
retourner P^*, M^* ;

Algorithme 5 : Heuristique de Gupta.

Entrées : Matrice des temps de traitement T.
Sorties : Planning P et makespan associé.

$P \leftarrow [i \mid 1 \leq i \leq n]$;
Trier P selon $\sum_j (m-j+1) \cdot T[\cdot][j]$ en ordre décroissant ;
retourner $P, \text{makespan}(P)$;

Les concepts abordés précédemment s'adressent principalement aux applications industrielles, où la *prise de décision* est un enjeu stratégique. Dans ce contexte, la visualisation des résultats se révèle être un outil indispensable pour analyser, interpréter et communiquer les données issues des processus d'ordonnancement, tout en orientant efficacement les choix en matière de planification et d'allocation des ressources. La présentation des informations sous forme de tableaux offre une structure claire et synthétique, particulièrement utile pour réaliser des comparaisons rapides entre différentes solutions. Ainsi, en se référant à l'exemple de l'atelier comportant 3 tâches et 5 machines, le tableau 4 expose les informations temporelles (temps de début et de fin, et durée d'exécution) relatives au déroulement du planning $(3, 1, 2)$, identifié comme étant le meilleur parmi les six configurations possibles.

Au-delà du format tabulaire, la *visualisation graphique* joue un rôle essentiel dans l'analyse et l'optimisation des processus de production. Elle permet de mieux comprendre la dynamique

Algorithme 6 : Heuristique de Dannenbring (RAH).

Entrées : Matrice des temps de traitement T.
Sorties : Planning P^* et makespan associé M^*.

$P^* \leftarrow [\,]$;
$M^* \leftarrow \infty$;
pour j **de** *1* **à** $m-1$ **faire**
  $S_1 \leftarrow [\sum_{k=1}^{j} \frac{m-k}{m} \cdot T[i][k] \mid 1 \leq i \leq n]$;
  $S_2 \leftarrow [\sum_{k=j+1}^{m} \frac{k}{m} \cdot T[i][k] \mid 1 \leq i \leq n]$;
  $G_1 \leftarrow [i \mid 1 \leq i \leq n \wedge S_1[i] \leq S_2[i]]$;
  $G_2 \leftarrow [i \mid 1 \leq i \leq n \wedge S_1[i] > S_2[i]]$;
  Trier G_1 selon $S_1[\cdot]$ en ordre croissant ;
  Trier G_2 selon $S_2[\cdot]$ en ordre décroissant ;
  $P \leftarrow G_1 \oplus G_2$;
  $M \leftarrow \text{makespan}(P)$;
  si $M < M^*$ **alors**
    $P^* \leftarrow P$;
    $M^* \leftarrow M$;
retourner P^*, M^* ;

Algorithme 7 : Heuristique de Nawaz-Enscore-Ham (NEH).

Entrées : Matrice des temps de traitement T.
Sorties : Planning P^* et makespan associé M^*.

$I \leftarrow [i \mid 1 \leq i \leq n]$;
Trier I selon $\sum_j T[\cdot][j]$ en ordre décroissant ;
$P \leftarrow [\,]$;
pour chaque i **de** I **faire**
  $P^* \leftarrow [\,]$;
  $M^* \leftarrow \infty$;
  pour j **de** *1* **à** longueur$(P)+1$ **faire**
    $P' \leftarrow P$;
    Insérer i à la position j dans P' ;
    $M' \leftarrow \text{makespan}(P')$;
    si $M' < M^*$ **alors**
      $P^* \leftarrow P'$;
      $M^* \leftarrow M'$;
  $P \leftarrow P^*$;
retourner P^*, M^* ;

Algorithme 8 : Heuristique de Suliman.

Entrées : Matrice des temps de traitement T.
Sorties : Planning P^* et makespan associé M^*.

$P^* \leftarrow [i \mid 1 \leq i \leq n]$;
Trier P^* selon $\sum_j T[\cdot][j]$ en ordre décroissant ;
$M^* \leftarrow \text{makespan}(P^*)$;
pour i **de** *1* **à** $\text{longueur}(P^*) - 1$ **faire**
    $P \leftarrow P^*$;
    Échanger $P[i]$ et $P[i+1]$;
    $M \leftarrow \text{makespan}(P)$;
    si $M < M^*$ **alors**
        $P^* \leftarrow P$;
        $M^* \leftarrow M$;
retourner P^*, M^* ;

Tab. 4: Informations temporelles associées au planning optimal $(3, 1, 2)$.

	Machine 1	**Machine 2**	**Machine 3**	**Machine 4**	**Machine 5**
Tâche 1	$(4, 9, 5)$	$(13, 21, 8)$	$(21, 24, 3)$	$(25, 31, 6)$	$(31, 35, 4)$
Tâche 2	$(9, 15, 6)$	$(21, 28, 7)$	$(28, 32, 4)$	$(32, 37, 5)$	$(37, 40, 3)$
Tâche 3	$(0, 4, 4)$	$(4, 13, 9)$	$(13, 18, 5)$	$(18, 25, 7)$	$(25, 31, 6)$

d'ordonnancement, d'identifier visuellement les goulots d'étranglement et de détecter d'éventuels déséquilibres dans l'utilisation des ressources. Différentes approches existent, allant des graphiques statiques aux représentations interactives et animées, chacune offrant des niveaux variés d'analyse et d'interprétation.

L'*interactivité* constitue un atout majeur dans l'exploration des données d'ordonnancement. Grâce aux outils modernes, il est possible de manipuler dynamiquement les plannings, de simuler différents scénarios et d'observer en temps réel l'impact des ajustements. Des visualisations avancées, telles que les *heatmaps* et *graphiques d'occupation*, offrent une représentation synthétique de la charge de travail sur les machines, facilitant ainsi l'identification des périodes critiques et l'optimisation des ressources.

Parmi les méthodes les plus répandues, le *diagramme de Gantt (1919)* reste un outil de référence pour représenter le déroulement du planning. Chaque tâche y est illustrée par une barre horizontale indiquant son début et sa fin sur un axe temporel, répartie sur les différentes machines. Cette représentation permet de visualiser l'enchainement des opérations, d'identifier les périodes d'inactivité ou de chevauchement et de repérer rapidement les machines surchargées ou sous-utilisées. Son utilisation facilite la lecture et la communication des plannings, aussi bien pour les ingénieurs que pour les décideurs. La figure 2 illustre le diagramme de Gantt correspondant à l'exemple précédent, mettant en évidence la répartition temporelle des tâches sur les différentes machines. On y observe les 5 machines (axe vertical) exécutant différentes

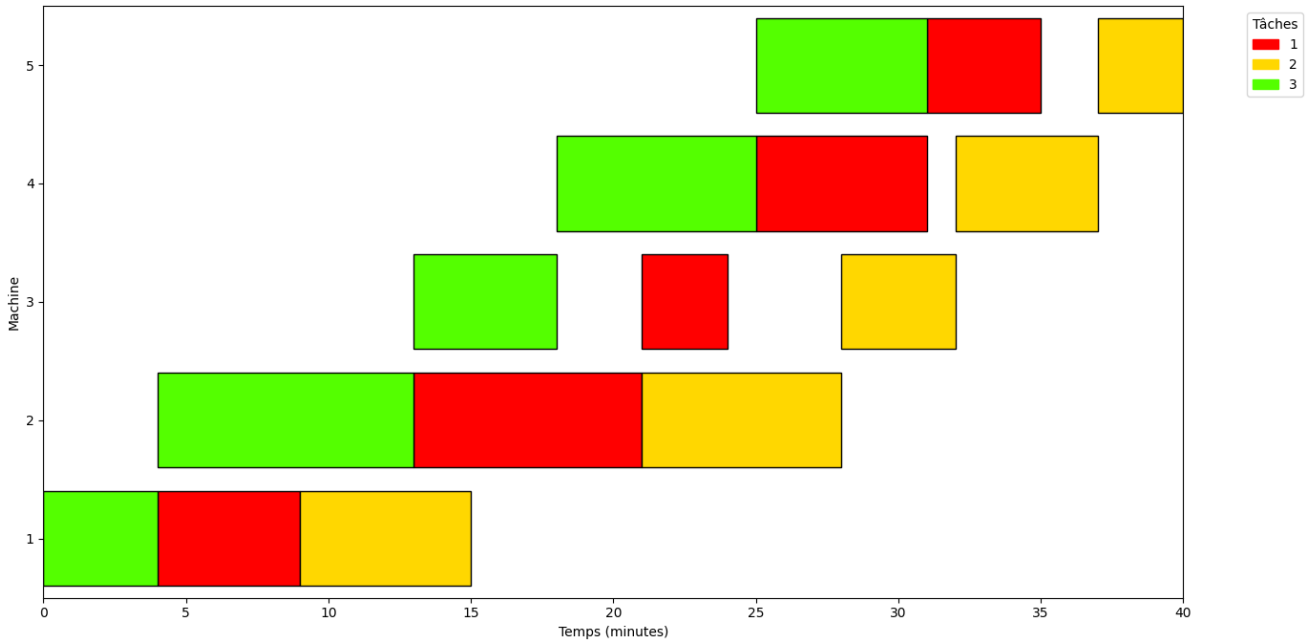

FIG. 2: Diagramme de Gantt associé au planning $(3, 1, 2)$.

tâches (axe horizontal en minutes). Chaque tâche est représentée par un rectangle coloré : rouge pour la tâche 1, jaune pour la tâche 2 et vert pour la tâche 3. Dans cette solution optimale, les tâches sont réparties de manière à minimiser le temps d'exécution total tout en équilibrant la charge entre les machines. Cette configuration met en évidence un ordonnancement bien optimisé, où les périodes d'inactivité sont réduites au minimum, garantissant ainsi une utilisation efficace des ressources disponibles et un temps d'achèvement global optimal.

Après avoir exploré les fondements de l'ordonnancement en ateliers à flux continu et les différentes approches heuristiques permettant d'optimiser les séquences de production, il est essentiel d'approfondir l'analyse des méthodes avancées. Le chapitre suivant se consacre donc aux approches métaheuristiques, qui offrent des solutions performantes pour résoudre des problèmes d'ordonnancement plus complexes et mieux adaptés aux contraintes industrielles réelles.

Chapitre 3

Métaheuristiques

Les métaheuristiques sont des algorithmes développés pour traiter des problèmes d'optimisation complexes, où l'utilisation de méthodes exactes devient inapplicable en raison de leur cout computationnel élevé. Elles reposent sur des stratégies dédiées permettant d'atteindre des solutions de qualité en un temps raisonnable. Ce chapitre commence par une présentation des fondements de l'optimisation et des défis associés, justifiant le recours aux métaheuristiques. Il expose ensuite les principes généraux de ces approches, en distinguant celles basées sur une seule trajectoire de recherche et celles exploitant des populations de solutions. Une classification des principales méthodes est ensuite proposée, incluant le recuit simulé, la recherche par tabous et les algorithmes génétiques. Enfin, le chapitre aborde les enjeux actuels et les perspectives d'amélioration, notamment l'optimisation automatique des paramètres, l'apprentissage automatique et l'hybridation avec d'autres approches.

3.1 Optimisation

L'*optimisation* est une discipline fondamentale en mathématiques appliquées et en informatique, jouant un rôle clé dans la résolution de problèmes complexes dans des domaines variés, tels que la logistique, la finance et l'ingénierie. Son objectif est de déterminer la meilleure solution parmi un ensemble de solutions admissibles en minimisant ou maximisant une *fonction objectif* sous *contraintes* (Gendreau and Potvin, 2019 ; Martí et al., 2018 ; Siarry, 2016 ; Talbi, 2009). Formellement, un problème d'optimisation s'écrit :

$$\text{Trouver } x^* \in S \mid f(x^*) \leq f(x), \forall x \in S,$$

où S désigne l'espace des solutions possibles (espace de recherche) et f la fonction objectif à optimiser[1] qui sert à évaluer lesdites solutions possibles. Une *instance de problème* I désigne une configuration spécifique d'un problème général, caractérisée par un ensemble précis de données

[1] Pour une maximisation, il suffit d'inverser l'inégalité.

d'entrée. Ces données définissent les éléments du problème et influencent directement l'espace de recherche ainsi que les solutions possibles. Ainsi, un même problème peut donner lieu à une multitude d'instances distinctes, chacune correspondant à une combinaison particulière de ces données. Par exemple, dans le cas de l'ordonnancement en ateliers à flux continu évoqué précédemment, le problème général consiste à organiser l'exécution des tâches sur des machines afin d'optimiser un critère donné. Toutefois, chaque instance varie en fonction du nombre et de la nature des tâches, du nombre de machines disponibles, ainsi que des contraintes spécifiques associées (délais, priorités, temps de traitement, etc.). Chaque combinaison de ces éléments génère une instance distincte, influençant directement l'espace de recherche S et les valeurs de la fonction objectif f, qui peut correspondre, par exemple, au makespan, au temps d'inactivité, au retard global ou encore à l'équilibrage de charge.

Les problèmes d'optimisation peuvent être classés selon plusieurs critères, en fonction de la nature des variables, du type de contraintes et du nombre d'objectifs à atteindre. Parmi les grandes catégories, on distingue :

- **Optimisation continue :** les variables de décision sont des nombres réels et l'objectif est souvent formulé sous forme d'une fonction mathématique différentiable. Un exemple typique est l'optimisation en programmation linéaire, largement utilisée en économie et en recherche opérationnelle pour la gestion de stocks, la planification de production ou la maximisation des profits.

- **Optimisation discrète et combinatoire :** dans ce cas, les solutions possibles sont en nombre fini ou dénombrable. Ce type d'optimisation est au cœur de nombreux problèmes industriels et scientifiques, notamment le problème du voyageur de commerce, le problème d'affectation, ou encore le problème du sac à dos. Ces problèmes sont généralement NP-difficiles et nécessitent des approches spécifiques pour être résolus efficacement.

- **Optimisation multiobjectif :** contrairement aux problèmes classiques qui reposent sur un seul critère, les problèmes multiobjectifs impliquent plusieurs fonctions objectif pouvant être *conflictuelles*. Par exemple, en agriculture durable, il est nécessaire d'augmenter le rendement des cultures pour répondre à la demande alimentaire mondiale tout en limitant l'utilisation de pesticides et d'engrais chimiques qui impactent l'environnement. De même, dans le développement urbain, il faut construire davantage de logements pour accompagner la croissance démographique tout en préservant les espaces verts et en réduisant la pollution urbaine.

L'étude des problèmes d'optimisation repose sur la *théorie de la complexité algorithmique*, qui classe ces problèmes en différentes catégories selon leur difficulté computationnelle. Les problèmes nécessitant une optimisation figurent souvent parmi les plus complexes, beaucoup appartenant à la classe NP-difficile, où la recherche d'une solution optimale s'avère bien plus difficile que sa simple vérification. L'efficacité d'un algorithme d'optimisation se mesure alors selon plusieurs

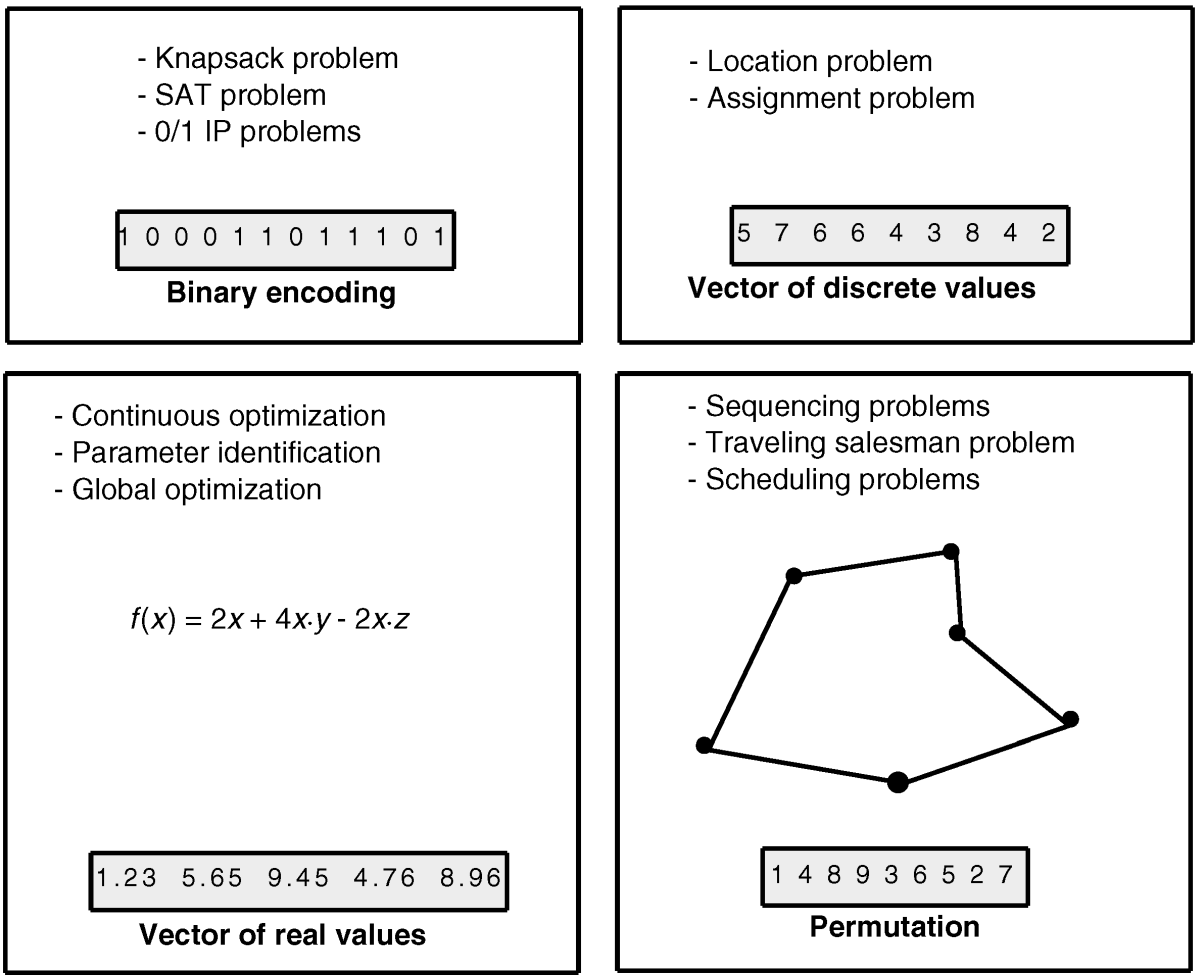

FIG. 3: Encodage des solutions dans le cadre de l'optimisation (Talbi, 2009).

critères : la qualité de la solution, qui doit être proche de l'optimum global, le temps de calcul, crucial pour les problèmes de grande taille, la robustesse, qui garantit des performances stables malgré les variations du problème, et la *scalabilité*, assurant que l'algorithme reste performant à grande échelle. Par ailleurs, les problèmes d'optimisation sont souvent soumis à des contraintes restreignant l'ensemble des solutions admissibles ; celles-ci peuvent être linéaires ou non linéaires, dures (incontournables) ou molles (violables avec pénalisation). La gestion de ces contraintes constitue un défi majeur, influençant directement le choix des méthodes de résolution.

L'encodage des solutions joue un rôle central dans l'efficacité des algorithmes d'optimisation, influençant directement leur capacité à explorer et exploiter l'espace de recherche (Figure 3). Il s'agit du processus par lequel une solution candidate est représentée sous une forme manipulable tout au long du processus d'optimisation. Le choix d'un encodage approprié dépend fortement de la nature du problème à résoudre. Pour les problèmes combinatoires, comme le voyageur de commerce ou l'ordonnancement, les solutions peuvent être encodées sous forme de permutations, de graphes ou de matrices d'affectation. En optimisation continue, les solutions sont

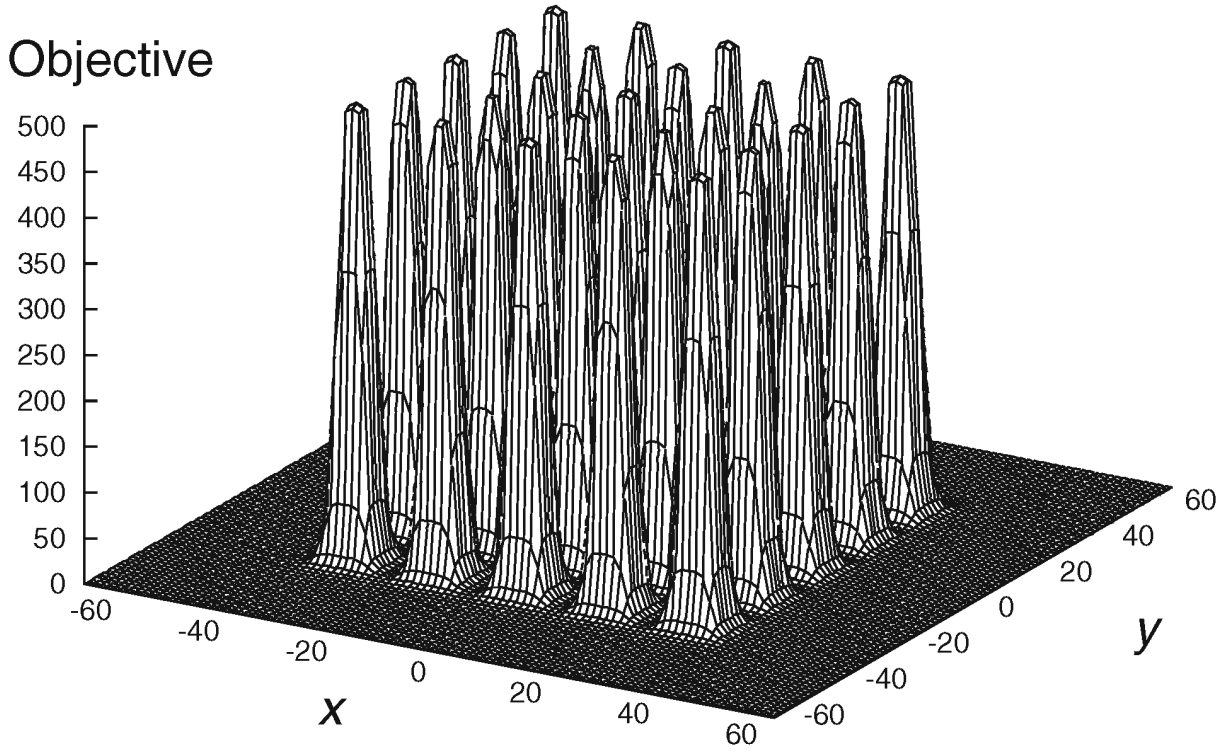

FIG. 4: Problème d'optimisation avec plusieurs optimums locaux (Siarry, 2016).

souvent représentées par des vecteurs de réels ou de nombres discrets. L'encodage doit permettre une exploration efficace de l'espace de recherche tout en garantissant que les solutions générées respectent les contraintes du problème. Un encodage mal adapté peut entraver la convergence de l'algorithme en introduisant des solutions non valides ou en rendant la navigation dans l'espace de recherche inefficace.

Un défi majeur en optimisation réside dans la présence d'optimums locaux. Un optimum local est une solution meilleure que toutes ses voisines immédiates, mais qui peut être éloignée de l'optimum global, la meilleure solution possible sur l'ensemble de l'espace de recherche. La figure 4 illustre un paysage d'optimisation parsemé de multiples pics et vallées, représentant des optimums locaux. Pour éviter que la recherche ne se bloque prématurément dans ces optimums locaux, il est essentiel d'adopter des stratégies favorisant une exploration plus diversifiée de l'espace des solutions. Cela peut passer par l'introduction de mécanismes permettant de s'éloigner temporairement d'un optimum local, d'explorer différentes régions de l'espace de recherche ou encore de maintenir un équilibre entre exploitation et exploration. Ces principes sont particulièrement cruciaux dans des contextes comme l'ordonnancement en ateliers à flux continu, où la structure du problème peut induire un grand nombre d'optimums locaux.

Généralement, les approches de résolution des problèmes d'optimisation se divisent en deux grandes familles : les *méthodes exactes* et les *méthodes approximatives*. Les méthodes exactes ga-

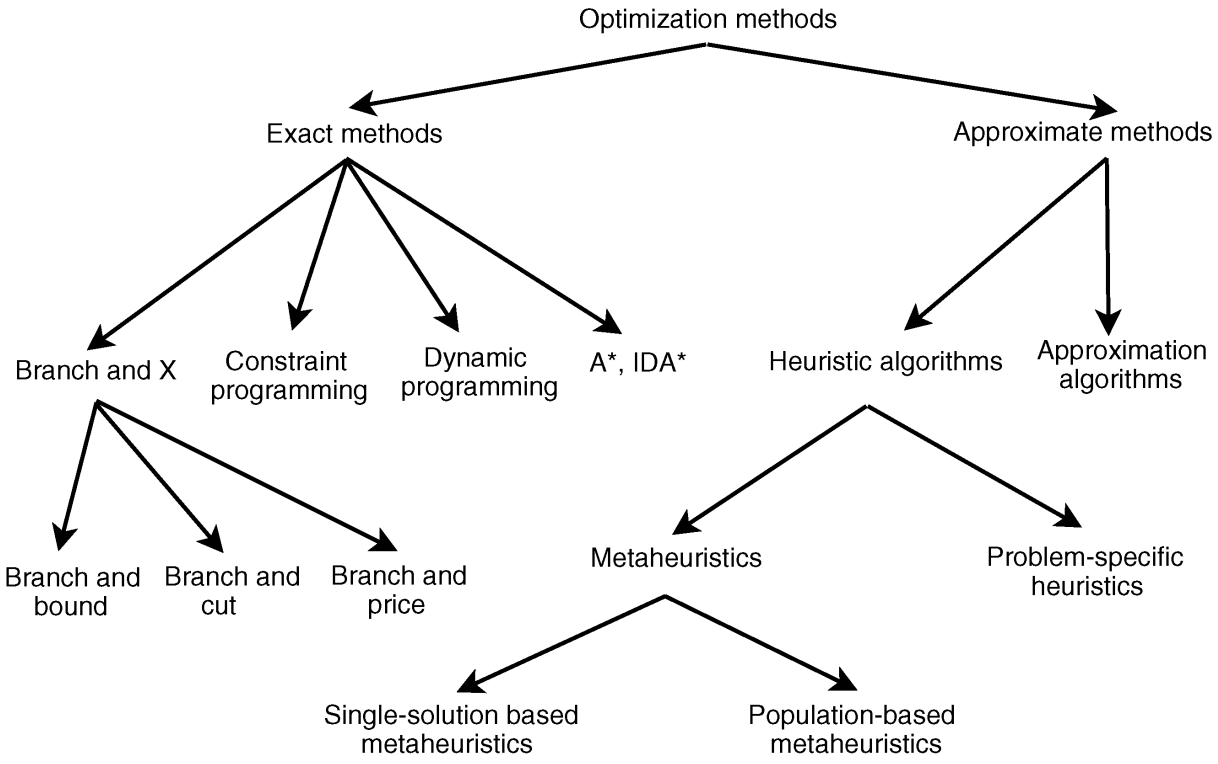

FIG. 5: Méthodes exactes et approximatives d'optimisation (Talbi, 2009).

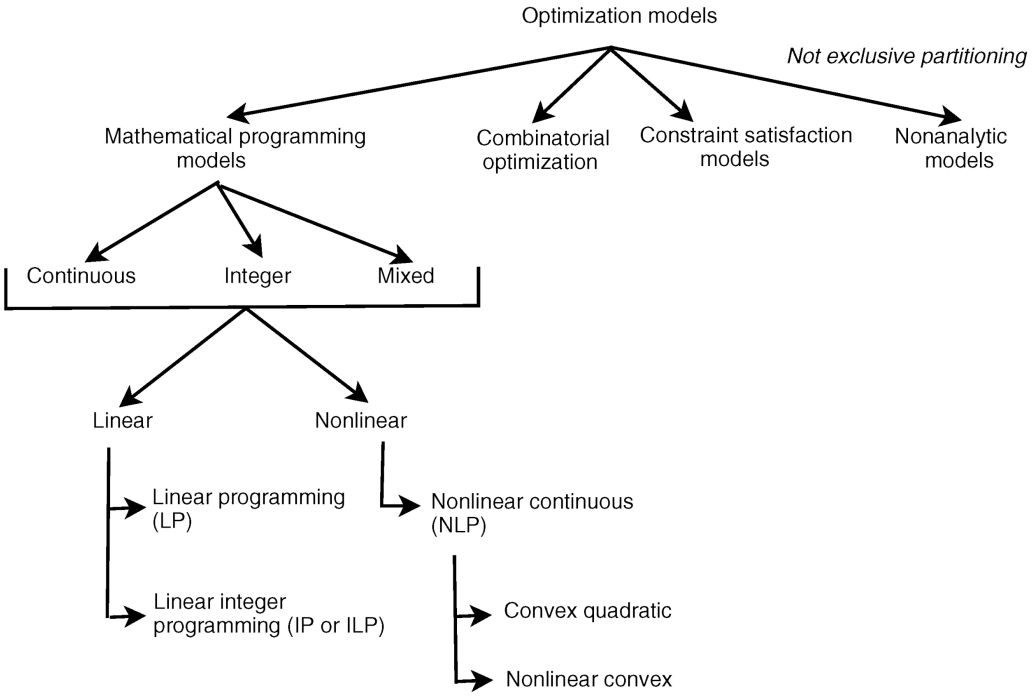

FIG. 6: Méthodes exactes d'optimisation (Talbi, 2009).

rantissent d'obtenir la solution optimale, mais leur cout computationnel peut rapidement devenir prohibitif pour les problèmes de grande taille ou de nature combinatoire (Figures 5–6). Parmi ces techniques, on retrouve la programmation linéaire et non linéaire, qui modélise la fonction objectif et les contraintes sous forme d'équations mathématiques ; les algorithmes de séparation et évaluation, qui explorent l'espace des solutions en éliminant intelligemment les branches non prometteuses ; et la programmation dynamique, qui résout certains problèmes en construisant une solution optimale à partir de sous-problèmes optimaux, comme dans le problème du sac à dos.

3.2 Métaheuristiques

Dans de nombreux problèmes d'optimisation, l'explosion combinatoire rend les méthodes exactes impraticables dans des contextes réels, imposant ainsi le recours à des approches approximatives. Parmi celles-ci, on distingue les heuristiques et les métaheuristiques, qui visent à fournir des solutions de qualité en un temps raisonnable. Les heuristiques, en particulier, ont été développées comme une alternative pragmatique aux méthodes exactes. Fondées sur des règles empiriques issues de l'intuition ou de l'expérience, elles permettent d'obtenir rapidement une solution approchée. Toutefois, la spécialisation des heuristiques limite leur capacité à s'adapter à des problèmes variés et leur tendance à se bloquer dans des optimums locaux compromet la qualité des solutions obtenues. Pour pallier ces limites, les métaheuristiques ont été développées

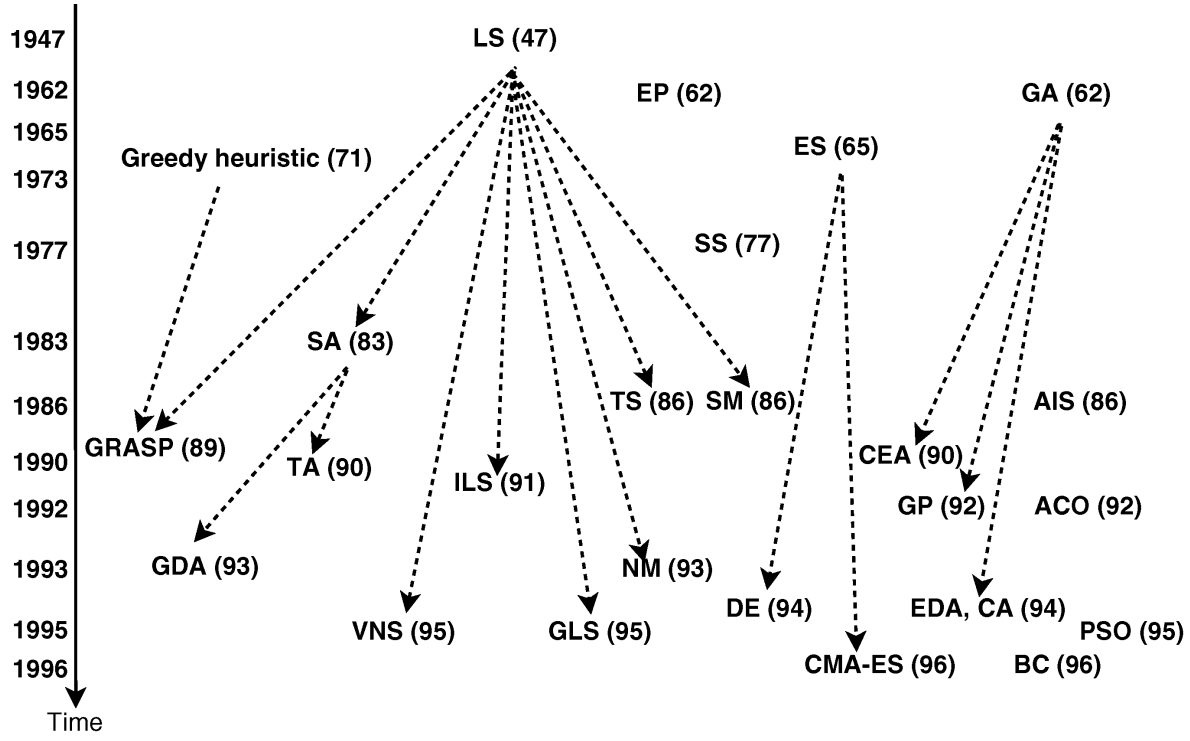

FIG. 7: Généalogie des métaheuristiques[2](Talbi, 2009).

afin d'offrir une approche plus générale et plus efficace de l'optimisation.

Ainsi, les métaheuristiques se distinguent des heuristiques par leur capacité à mieux explorer l'espace des solutions et à éviter le piégeage dans des optimums locaux. Bien que des approches adaptatives combinant plusieurs heuristiques aient existé bien avant leur formalisation, c'est Glover (1986) qui a introduit le terme *métaheuristique* pour désigner une stratégie d'optimisation de haut niveau (Figure 7). Ces méthodes, qui s'appuient sur des heuristiques comme briques de base, fournissent un cadre flexible applicable à divers problèmes d'optimisation. Contrairement aux heuristiques, souvent conçues pour un problème spécifique et privilégiant l'exploitation locale, les métaheuristiques équilibrent exploration et exploitation afin d'améliorer progressivement la qualité des solutions. Plus robustes face aux variations des problèmes et mieux adaptées aux contextes complexes, elles impliquent cependant un cout computationnel généralement plus élevé.

Toute métaheuristique suit une structure générale qui peut être appliquée à divers problèmes d'optimisation, qu'elle repose sur une solution unique ou une population de solutions (Gendreau and Potvin, 2019 ; Martí et al., 2018 ; Siarry, 2016 ; Talbi, 2009). Cette structure permet de guider la recherche d'une solution optimale tout en équilibrant l'exploration de l'espace des solutions et l'exploitation des solutions prometteuses (Algorithme 9). De manière générale, une métaheuristique appliquée sur une instance de problème (I) suit trois phases principales, qui

[2]Pour une explication plus détaillée, voir les pages 23 à 25 de l'ouvrage de référence de Talbi (2009).

Algorithme 9 : Structure générale d'une métaheuristique.

Entrées : Instance du problème I et paramètres θ.
Sorties : Meilleure solution trouvée x^* et son évaluation f^*.

```
Générer une solution x (ou une population de solutions P) de base ;
Évaluer x (ou P) ;
répéter
    Générer une nouvelle solution x' (ou une population de solutions P') ;
    Évaluer x' (ou P') ;
    Mettre à jour x (ou P) en utilisant x' (ou P') ;
    Mettre à jour x* et f* ;
jusqu'à critère d'arrêt satisfait ;
retourner x*, f* ;
```

définissent son déroulement et influencent son efficacité :

- **Phase d'initialisation :** elle commence par générer une ou plusieurs solutions initiales, qui constituent soit un point de départ unique (x), soit une population de solutions (P). La génération peut être aléatoire, heuristique ou basée sur des connaissances à priori du problème. Ces solutions sont ensuite évaluées en fonction d'une fonction objectif afin d'établir un premier repère de qualité. L'initialisation joue un rôle clé, car une bonne distribution initiale des solutions peut favoriser la convergence de la métaheuristique vers une solution optimale.

- **Phase d'exploration et de mise à jour :** cette phase constitue le cœur de la métaheuristique. À chaque itération, de nouvelles solutions (x' ou P') sont générées en appliquant des mécanismes spécifiques à la méthode utilisée. Chaque solution générée est évaluée, et une stratégie d'acceptation est appliquée pour déterminer si elle doit remplacer une solution existante. Cette phase est largement influencée par les *paramètres* θ qui régissent son comportement et son efficacité. Contrairement aux solutions x ou P, qui sont des candidats potentiels à l'optimisation du problème, les paramètres θ contrôlent le fonctionnement même de la métaheuristique : ils déterminent la manière dont les solutions sont générées, évaluées et sélectionnées. Leur nature et leurs valeurs dépendent de la logique de la métaheuristique considérée et influencent directement le compromis entre exploration et exploitation, la convergence vers une solution optimale et la capacité à éviter les optimums locaux.

- **Phase d'arrêt :** la métaheuristique s'arrête lorsque l'un des critères prédéfinis est atteint. Ces critères peuvent inclure un nombre maximal d'itérations, l'atteinte d'un seuil de performance, une stagnation prolongée des solutions ou encore une contrainte de temps. Une fois l'arrêt déclenché, la métaheuristique retourne la meilleure solution trouvée x^* ainsi que son évaluation f^*, qui représente l'approximation finale de l'optimum du problème considéré.

L'intérêt des métaheuristiques réside dans leur capacité à s'adapter à des problèmes variés grâce à cette structure générique. En ajustant les stratégies d'exploration, d'exploitation, ainsi qu'en *calibrant* leurs paramètres en fonction des spécificités de la méthode utilisée, elles peuvent être spécialisées pour différents contextes, qu'il s'agisse d'optimisation combinatoire, de programmation non linéaire ou encore de problèmes complexes à très grande échelle.

Ce contexte souligne un point fondamental : les métaheuristiques doivent trouver un équilibre entre diversification et intensification afin d'optimiser leur performance. La diversification (ou exploration) vise à explorer l'espace des solutions de manière large afin d'éviter le piégeage dans des optimums locaux et de découvrir de nouvelles régions prometteuses. Elle repose souvent sur des mécanismes aléatoires ou semi-dirigés permettant de générer une diversité de solutions initiales ou d'introduire des perturbations contrôlées au cours de la recherche. L'intensification (ou exploitation), quant à elle, se concentre sur l'amélioration des solutions existantes en exploitant les zones les plus prometteuses de l'espace de recherche. Elle s'appuie sur des stratégies locales d'optimisation, comme la descente de gradient ou la recherche locale, afin de raffiner progressivement les solutions et d'atteindre des performances optimales. Un bon compromis entre ces deux phases est essentiel : un excès de diversification peut entrainer une recherche inefficace et aléatoire, tandis qu'une intensification excessive risque de conduire à une convergence prématurée vers un optimum local sous-optimal. C'est pourquoi les métaheuristiques intègrent souvent des mécanismes adaptatifs ou hybrides pour ajuster dynamiquement l'équilibre entre ces deux composantes au cours du processus d'optimisation.

3.3 Catégorisation et principales approches

Les métaheuristiques reposent sur des mécanismes d'exploration et d'exploitation inspirés de divers principes, allant des processus physiques aux systèmes biologiques et sociaux. Ces algorithmes offrent une approche flexible pour résoudre des problèmes d'optimisation complexes en équilibrant l'exploration de nouvelles solutions et l'exploitation des régions prometteuses de l'espace de recherche. Leur classification peut se faire selon leur mode de recherche et leur adaptabilité aux différents types de problèmes. Certaines métaheuristiques fonctionnent en explorant une seule trajectoire de solution à la fois, tandis que d'autres manipulent simultanément un ensemble de solutions afin d'améliorer la couverture de l'espace de recherche. Ce choix d'approche influence directement la performance et l'efficacité de la métaheuristique en fonction du problème traité. De plus, certains de ces algorithmes intègrent des mécanismes d'adaptation et d'apprentissage, les rendant encore plus robustes face à des problèmes complexes et dynamiques.

Les algorithmes basés sur une trajectoire unique avancent progressivement en transformant une solution initiale pour en améliorer la qualité. Ils sont bien adaptés aux problèmes où l'on cherche à affiner une solution à partir d'un point de départ donné. Ces méthodes sont particulièrement efficaces lorsqu'il est essentiel d'exploiter localement une solution sans explorer

massivement l'ensemble de l'espace de recherche. Elles sont souvent privilégiées lorsque la structure du problème permet une convergence rapide vers un optimum de bonne qualité. Parmi ces méthodes, on trouve :

- **Recuit simulé (Simulated Annealing) :** inspirée du recuit métallurgique, cette métaheuristique imite le processus par lequel un matériau atteint un état stable lorsqu'il est refroidi lentement. Elle accepte temporairement des solutions de moindre qualité afin d'éviter le piégeage dans un optimum local et de favoriser une exploration plus large de l'espace de recherche. Son efficacité repose sur un paramétrage précis du taux de refroidissement, qui détermine la vitesse à laquelle la probabilité d'accepter des solutions sous-optimales diminue au fil des itérations (Kirkpatrick et al., 1983).

- **Recherche par tabous (Tabu Search) :** cette métaheuristique repose sur une mémoire adaptative qui empêche de revisiter fréquemment les mêmes solutions, évitant ainsi la stagnation et favorisant l'exploration de nouvelles régions de l'espace de recherche. Son utilisation d'une liste d'évitement permet de guider la recherche vers des solutions inexplorées tout en contrôlant le risque de revenir en arrière (Glover, 1986).

- **Procédure de recherche adaptative gloutonne et aléatoire (Greedy Randomized Adaptive Search Procedure) :** cette métaheuristique combine une phase de construction aléatoire contrôlée, où chaque solution est générée en sélectionnant de manière probabiliste des éléments parmi un ensemble restreint, avec une phase d'optimisation locale visant à affiner la solution obtenue. En intégrant à la fois exploration et exploitation, elle permet d'éviter un biais excessif vers des solutions sous-optimales tout en maintenant une diversité dans l'espace de recherche (Feo and Resende, 1989).

- **Recherche à voisinage variable (Variable Neighborhood Search) :** cette approche repose sur un changement dynamique du voisinage exploré, permettant ainsi d'améliorer la diversité des solutions testées et d'éviter la stagnation dans les optimums locaux. Elle exploite le principe selon lequel une solution de bonne qualité peut souvent être améliorée en examinant différents ensembles de solutions voisines (Mladenović and Hansen, 1997).

D'autres métaheuristiques exploitent une approche basée sur des populations de solutions qui évoluent simultanément. Ces algorithmes sont particulièrement utiles pour les problèmes comportant de nombreux optimums locaux, car ils permettent une exploration plus large et parallèle de l'espace de recherche. Leur capacité à traiter plusieurs solutions en parallèle les rend adaptés à des problèmes complexes où la diversité des solutions candidates est essentielle. Ces algorithmes sont souvent privilégiés pour les problèmes multiobjectifs et les problèmes à grande échelle. On retrouve notamment :

- **Recherche par dispersion (Scatter Search) :** cette métaheuristique génère de nouvelles solutions en combinant intelligemment des solutions existantes de haute qualité. Elle

se distingue par son approche déterministe, qui contraste avec les algorithmes génétiques plus stochastiques. Sa capacité à exploiter les caractéristiques communes des meilleures solutions en fait une méthode très efficace pour des problèmes complexes (Glover, 1977).

- **Algorithmes génétiques (Genetic Algorithms) :** inspirés des principes des théories de l'évolution et de la sélection naturelle ainsi que de certains mécanismes biologiques, ils reposent sur le principe de la survie des individus les plus adaptés à chaque génération. Grâce aux opérateurs de sélection, croisement et mutation, ces algorithmes permettent à une population de solutions d'évoluer progressivement vers un optimum. Ils sont largement utilisés dans des problèmes où l'espace de recherche est très vaste et où la diversification est essentielle (Holland, 1992).

- **Optimisation par essaims de particules (Particle Swarm Optimization) :** modélisée à partir du comportement collectif des essaims d'oiseaux ou de poissons, cette métaheuristique repose sur l'interaction entre agents qui partagent des informations et ajustent progressivement leurs trajectoires pour converger vers de meilleures solutions. Sa simplicité et son efficacité en font un choix populaire pour l'optimisation continue (Eberhart and Kennedy, 1995 ; Kennedy and Eberhart, 1995).

- **Optimisation par colonies de fourmis (Ant Colony Optimization) :** inspirée du comportement des colonies de fourmis, cette métaheuristique exploite un processus d'apprentissage collectif basé sur le dépôt de phéromones. Les solutions sont construites de manière incrémentale par un ensemble d'agents qui interagissent indirectement en ajustant les niveaux de phéromones selon la qualité des solutions explorées. Ce mécanisme favorise la convergence vers des solutions optimales tout en maintenant une exploration efficace de l'espace de recherche (Dorigo and Di Caro, 1999a,b ; Dorigo et al., 1999).

3.4 Enjeux et perspectives

Malgré le succès des métaheuristiques, ces approches restent confrontées à des défis fondamentaux qui nécessitent des recherches approfondies pour améliorer leur efficacité, leur robustesse et leur adaptabilité. Parmi ces défis, l'amélioration de la convergence, l'optimisation automatique des paramètres et l'hybridation avec d'autres approches constituent des axes d'exploration prometteurs (Gendreau and Potvin, 2019 ; Martí et al., 2018 ; Siarry, 2016 ; Talbi, 2009). Ces défis se déclinent en plusieurs axes majeurs qui structurent les recherches actuelles et futures sur les métaheuristiques :

- ***No-Free Lunch Theorem***[3] ***(NFLT)*** **:** ce théorème constitue un enjeu théorique fondamental dans la recherche sur les métaheuristiques. Formulé par Wolpert and Macready

[3] Littéralement : théorème d'absence de repas gratuits.

(1995, 1997), il établit qu'aucun algorithme de recherche ou d'optimisation ne peut surpasser tous les autres sur l'ensemble des problèmes possibles. En d'autres termes, toute amélioration obtenue sur une classe spécifique de problèmes se fait nécessairement au détriment d'autres classes. Cette observation met en évidence l'importance d'adapter les métaheuristiques aux caractéristiques des problèmes traités, plutôt que de chercher une *solution universelle*. Ainsi, l'enjeu ne se limite pas à concevoir des approches plus performantes, mais aussi à comprendre comment les configurer de manière optimale pour chaque cas particulier.

- **Équilibre entre exploration et exploitation :** un des autres défis les plus critiques dans cette perspective est l'équilibre entre exploration et exploitation (Gendreau and Potvin, 2019 ; Martí et al., 2018 ; Siarry, 2016 ; Talbi, 2009). Une recherche trop exploratoire risque d'être inefficace, tandis qu'un excès d'exploitation peut entrainer une stagnation dans des minimums locaux. Pour répondre à cette problématique, de nombreux algorithmes intègrent déjà des mécanismes permettant d'adapter dynamiquement cette balance. Par exemple, l'introduction de perturbations contrôlées, comme dans la recherche à voisinage variable, modifie dynamiquement la structure de la recherche pour échapper aux pièges de l'optimisation locale. De même, l'utilisation de mémoires adaptatives, comme en recherche par tabous, ou de critères d'acceptation probabilistes, comme dans le recuit simulé, contribue à une meilleure exploration de l'espace des solutions. Il faut noter toutefois que ces ajustements impliquent généralement un réglage manuel des paramètres, ce qui constitue une autre limitation importante.

- **Optimisation automatique des paramètres :** cette technique s'impose comme une nécessité pour renforcer l'autonomie des métaheuristiques. Traditionnellement, les paramètres en question sont déterminés empiriquement après de nombreux tests, un processus long et spécifique à chaque problème. Afin d'automatiser cette tâche, des approches issues de l'apprentissage automatique permettent aux algorithmes d'ajuster dynamiquement leurs propres paramètres en fonction des performances observées. Par exemple, le taux de mutation dans un algorithme génétique peut être modifié en temps réel afin de maintenir un niveau optimal de diversité au sein de la population.

- **Apprentissage automatique :** plus largement, l'apprentissage automatique joue un rôle croissant dans l'amélioration des métaheuristiques en leur offrant des capacités d'adaptation inédites. L'apprentissage profond et l'apprentissage par renforcement permettent notamment de modéliser et d'anticiper le comportement des algorithmes, optimisant ainsi leur exécution. Par exemple, des réseaux de neurones peuvent être entrainés à prédire les régions prometteuses de l'espace de recherche, guidant plus efficacement l'exploration. De même, ces techniques permettent d'ajuster dynamiquement les hyperparamètres, réduisant ainsi la nécessité d'une configuration manuelle et améliorant l'adaptabilité des métaheu-

ristiques à divers types de problèmes. Cette capacité d'autoadaptation améliore à la fois la flexibilité et l'efficacité des approches d'optimisation.

- **Hybridation :** l'hybridation des métaheuristiques avec d'autres approches constitue une voie prometteuse pour surmonter leurs limitations individuelles. L'idée est de combiner les forces de différentes méthodes afin d'obtenir des algorithmes plus performants et robustes. Une première stratégie consiste à incorporer des éléments d'une métaheuristique dans une autre. Par exemple, l'ajout d'une recherche locale inspirée de la recherche par tabous dans un algorithme génétique permet d'améliorer la qualité des solutions générées à chaque itération. Une autre approche consiste à exécuter plusieurs métaheuristiques en parallèle ou en séquence, en exploitant leurs complémentarités pour couvrir différentes facettes du problème.

- ***Matheuristiques* :** cette façon de faire, qui combine des métaheuristiques avec des méthodes exactes issues de la recherche opérationnelle, émerge comme une alternative particulièrement efficace. À cet effet, les matheuristiques exploitent la puissance des techniques mathématiques, telles que la programmation linéaire, la programmation par contraintes ou encore la relaxation lagrangienne, pour guider l'exploration heuristique de l'espace de recherche. Par exemple, une heuristique peut être utilisée pour générer rapidement des solutions approximatives qui sont ensuite affinées à l'aide d'une méthode exacte. Inversement, une optimisation exacte peut initialiser un algorithme métaheuristique avec des solutions de haute qualité, améliorant ainsi sa convergence. Cette synergie entre heuristique et rigueur mathématique permet d'exploiter le meilleur des deux mondes et de traiter efficacement des problèmes d'optimisation complexes.

- **Parallélisation :** l'essor du calcul parallèle a également transformé le domaine des métaheuristiques en leur offrant des capacités d'exploration accrues. Grâce aux processeurs multicœurs (Multi-core Processors), aux grappes de calcul (Computing Clusters) et aux unités de traitement graphique (Graphics Processing Units), il est désormais possible d'exécuter plusieurs instances d'un algorithme simultanément, accélérant ainsi la convergence. Par exemple, dans les algorithmes évolutionnaires, des sous-populations peuvent évoluer indépendamment avant d'échanger périodiquement des solutions, favorisant à la fois diversité et efficacité. D'autres stratégies exploitent la parallélisation des évaluations de la fonction objectif ou la décomposition du problème en sous-problèmes traités en parallèle. Cette capacité à tirer parti du calcul haute performance renforce la pertinence des métaheuristiques pour des problèmes de grande envergure.

Ce chapitre a présenté les métaheuristiques en détaillant leurs principes, leurs classifications et leurs défis. L'optimisation par essaims de particules, faisant partie de ces approches, est explorée dans le chapitre suivant à travers son application au problème de l'ordonnancement en ateliers à flux continu. Cette étude permettra d'évaluer son efficacité et son adaptation à ce contexte spécifique.

Chapitre 4

Approche proposée

Ce chapitre présente l'approche proposée pour résoudre les problèmes d'ordonnancement en ateliers à flux continu par une adaptation de l'optimisation par essaims de particules. Initialement, il détaille les hypothèses de travail, axées sur la reformulation de la position et de la vélocité pour les permutations et l'intégration d'heuristiques contre la stagnation. Ensuite, il rappelle les fondements de l'optimisation par essaims de particules. Le chapitre se penche alors sur son adaptation au contexte combinatoire en manipulant des permutations et des transpositions pour la position et la vélocité. Enfin, il expose la stratégie de gestion de la stagnation par injection adaptative d'heuristiques.

4.1 Hypothèses de travail

Cette approche explore l'idée que l'adaptation de l'algorithme d'optimisation par essaims de particules, à travers des stratégies spécifiques de gestion de la diversité et d'intégration d'heuristiques d'ordonnancement, pourrait améliorer son efficacité pour résoudre les problèmes d'ordonnancement en ateliers à flux continu. Plus précisément, l'approche proposée comprend principalement :

- **Reformulation des concepts de position et de vélocité pour l'ordonnancement :** nous suggérons que la reformulation des concepts traditionnels de position et de vélocité de l'optimisation par essaims de particules, qui sont basées sur des vecteurs numériques, pourrait être cruciale pour une application efficace aux problèmes d'ordonnancement. Dans ce contexte, où les solutions sont représentées par des permutations de tâches, nous suggérons que la définition de la position comme une *permutation* spécifique et de la vélocité comme une séquence de *transpositions* pourrait permettre une exploration plus naturelle et plus efficace de l'espace des solutions. Cette adaptation pourrait permettre à l'optimisation par essaims de particules de mieux naviguer dans l'espace discret des solutions d'ordonnancement, en maintenant la validité des solutions à chaque itération et en exploitant plus efficacement les caractéristiques du problème.

- **Gestion de la stagnation :** nous envisageons que l'*injection* d'heuristiques d'ordonnancement bien établies d'une façon contrôlée et adaptative, pourrait améliorer la capacité de l'optimisation par essaims de particules à converger vers des solutions de haute qualité. Contrairement aux approches qui emploient ces heuristiques uniquement pour l'initialisation des solutions, nous suggérons qu'une introduction dynamique de ces heuristiques en réponse à la stagnation de l'essaim (c'est-à-dire lorsque les particules cessent d'améliorer leurs solutions) pourrait fournir un mécanisme plus efficace pour équilibrer l'exploration et l'exploitation. Cette stratégie pourrait réinjecter de la diversité dans l'essaim à des moments critiques, en évitant une convergence prématurée vers des optimums locaux, tout en guidant la recherche vers des régions prometteuses de l'espace des solutions.

4.2 Optimisation par essaims de particules

L'optimisation par essaims de particules (Particle Swarm Optimization), connue aussi sous l'appellation PSO est une métaheuristique d'optimisation inspirée du comportement collectif observé dans la nature (Algorithme 10), notamment dans les nuées d'oiseaux et les bancs de poissons (Figure 8). Introduite par Eberhart and Kennedy (1995) ; Kennedy and Eberhart (1995), cette approche repose sur la coopération et la communication entre des entités individuelles appelées particules, qui explorent l'espace de recherche en suivant des règles simples d'adaptation (Figure 9).

Pour ce faire, l'optimisation par essaims de particules repose sur un ensemble de particules se déplaçant dans un espace de recherche défini (Gendreau and Potvin, 2019 ; Martí et al., 2018 ; Siarry, 2016 ; Talbi, 2009). Chaque particule représente une solution candidate au problème d'optimisation et est caractérisée par :

- **Position** x_i **:** un vecteur de dimension n, représentant une solution possible.
- **Vélocité** v_i **:** un vecteur indiquant l'ampleur et la direction du déplacement futur.
- **Mémoire :** pour sauvegarder la meilleure solution trouvée par la particule (p_i^{best}).
- **Communication :** établie avec les autres particules pour partager la meilleure solution trouvée.

À chaque itération, une particule ajuste sa vélocité et met à jour sa position en fonction de trois composantes :

- **Inertie :** la particule conserve une part de son mouvement précédent pour maintenir l'exploration.
- **Attraction cognitive :** la particule tend à revenir vers sa meilleure solution trouvée jusqu'à présent.

FIG. 8: Une nuée d'étourneaux se comportant comme un essaim[1].

- **Attraction sociale :** la particule est influencée par la meilleure solution trouvée par l'ensemble du groupe.

L'actualisation de la vélocité et de la position de chaque particule à l'instant $t+1$ suit les équations suivantes. Dans le modèle g^{best}, chaque particule est influencée par la meilleure solution trouvée par l'ensemble du groupe :

$$v_i^{(t+1)} = w \cdot v_i^{(t)} + c_1 \cdot r_1 \cdot (p_i^{best} - x_i^{(t)}) + c_2 \cdot r_2 \cdot (g^{best} - x_i^{(t)}) \tag{4.1}$$

$$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)} \tag{4.2}$$

où :

- w est le facteur d'inertie, qui contrôle l'équilibre entre exploration et exploitation.
- c_1 et c_2 sont des coefficients d'accélération régulant l'influence des composantes cognitive et sociale.
- r_1 et r_2 sont des *valeurs aléatoires* uniformément distribuées dans $[0, 1]$, introduisant un aspect stochastique dans la mise à jour.
- p_i^{best} est la meilleure position atteinte par la particule i.

[1]Image utilisée sous licence Creative Commons BY-SA 2.0, disponible sur Wikimedia Commons.

- g^{best} est la meilleure position atteinte par n'importe quelle particule du groupe.

Dans certaines applications, une alternative pour exploiter la composante sociale consiste à limiter l'influence de chaque particule à un voisinage restreint, où elle est guidée uniquement par la meilleure solution trouvée au sein de ce sous-groupe :

$$v_i^{(t+1)} = w \cdot v_i^{(t)} + c_1 \cdot r_1 \cdot (p_i^{best} - x_i^{(t)}) + c_2 \cdot r_2 \cdot (l_i^{best} - x_i^{(t)}) \tag{4.3}$$

où l_i^{best} est la meilleure solution trouvée parmi un sous-ensemble de particules appelées *voisins* de la particule i.

Algorithme 10 : Optimisation par essaims de particules.

Entrées : Instance du problème I et paramètres θ.
Sorties : Meilleure solution trouvée x^* et son évaluation f^*.

```
pour chaque particule faire
    Initialiser le voisinage (pour le modèle l_i^best) ;
    Initialiser la position x_i ;
    Initialiser la vélocité v_i ;
    Initialiser la meilleure position personnelle p_i ← x_i ;
répéter
    pour chaque particule faire
        Mettre à jour la vélocité selon l'équation (4.1) ou (4.3) (pour le modèle l_i^best) ;
        Mettre à jour la position selon l'équation (4.2) ;
        Évaluer x_i et mettre à jour p_i, x* et f* ;
jusqu'à critère d'arrêt satisfait ;
retourner x*, f* ;
```

Comme pour toutes les métaheuristiques, l'optimisation par essaims de particules est un processus itératif qui nécessite la définition d'un nombre maximal d'itérations. Ce paramètre, crucial pour assurer la convergence de l'algorithme vers une solution optimale, doit être déterminé le plus souvent d'une manière empirique avant son exécution. En plus de ce critère d'arrêt principal, d'autres conditions peuvent être considérées, telles que la convergence de la solution, l'atteinte d'un seuil de performance, la perte de diversité de l'essaim, ou encore le dépassement d'un temps de calcul maximal.

Dans la même optique, le choix des valeurs les plus appropriées des paramètres est un aspect crucial dans le bon déroulement de l'exécution de la métaheuristique. À cet effet, dans le modèle g^{best}, chaque particule est influencée par la meilleure solution trouvée par l'ensemble du groupe, ce qui favorise une convergence rapide, mais peut conduire à un piégeage prématuré dans un optimum local si la diversité des solutions diminue trop rapidement. À l'inverse, le modèle l_i^{best} introduit la notion de voisinage, où chaque particule ne communique qu'avec un sous-ensemble restreint de l'essaim, souvent défini selon une structure topologique fixe (anneau, grille, etc.). Une taille typique de voisinage est de 3 à 6 particules, notamment dans une structure en anneau,

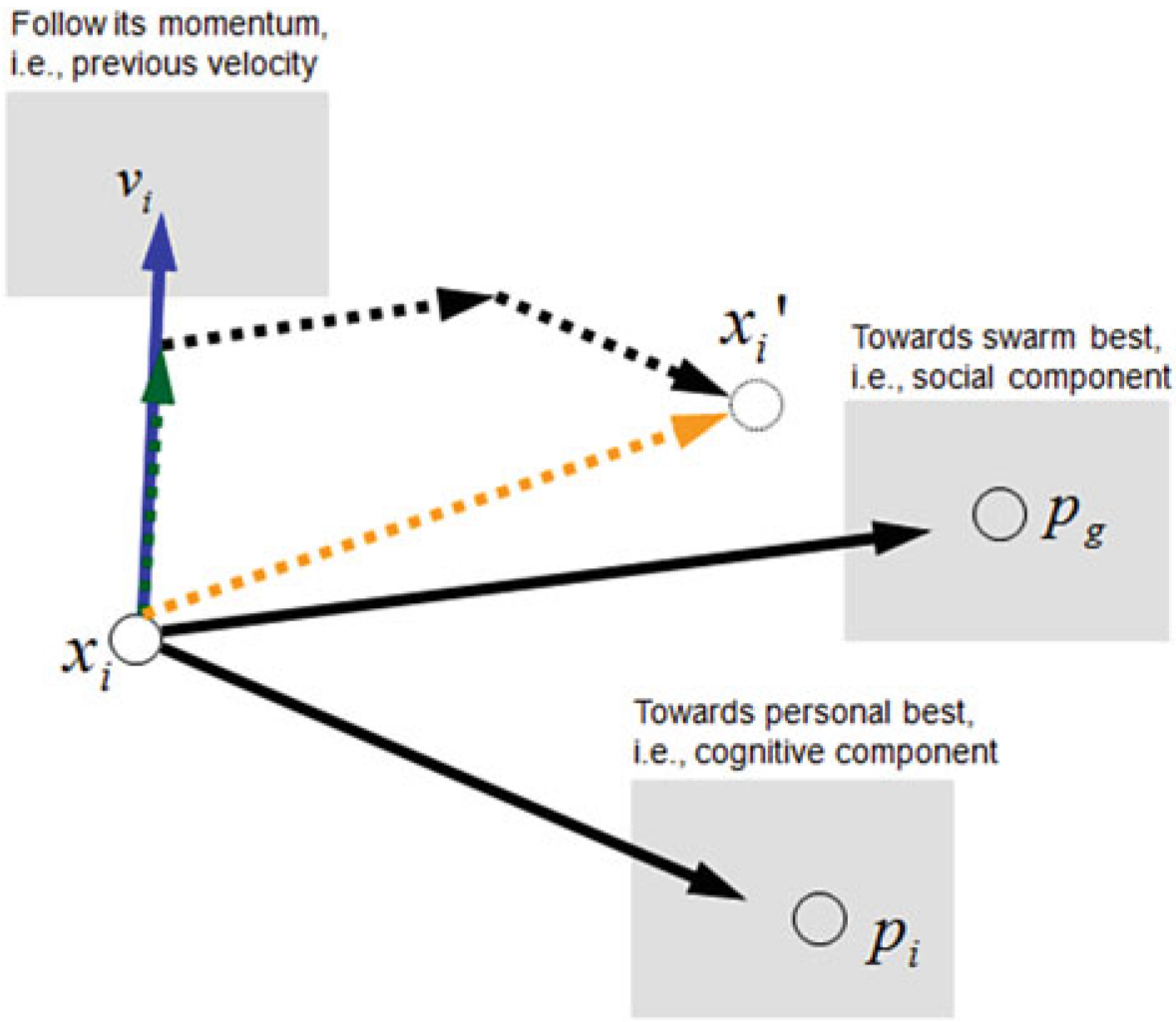

FIG. 9: Mécanismes guidant le mouvement des particules (Gendreau and Potvin, 2019).

ce qui permet de conserver une diversité plus élevée et d'éviter une convergence trop rapide vers une solution sous-optimale (Figure 10).

En plus du choix entre g^{best} et l_i^{best}, les performances de l'optimisation par essaims de particules sont fortement influencées par plusieurs paramètres clés qui régissent l'équilibre entre exploration et exploitation. Le facteur d'inertie w joue un rôle essentiel dans ce compromis : une valeur élevée ($w \in [0.8, 1.2]$) encourage l'exploration en maintenant la vélocité des particules, tandis qu'une valeur plus faible ($w \in [0.4, 0.6]$) favorise l'exploitation en les incitant à se concentrer autour des solutions déjà découvertes. Une stratégie couramment adoptée consiste à faire décroitre w progressivement au fil des itérations, par exemple en le réduisant linéairement de 0.9 à 0.4, afin de débuter par une exploration large avant d'affiner la recherche.

Les coefficients d'accélération c_1 et c_2 déterminent l'influence respective des expériences personnelles et collectives des particules. Un c_1 trop élevé rend les particules trop dépendantes de leurs positions précédentes, limitant leur capacité à s'adapter au contexte global, tandis qu'un

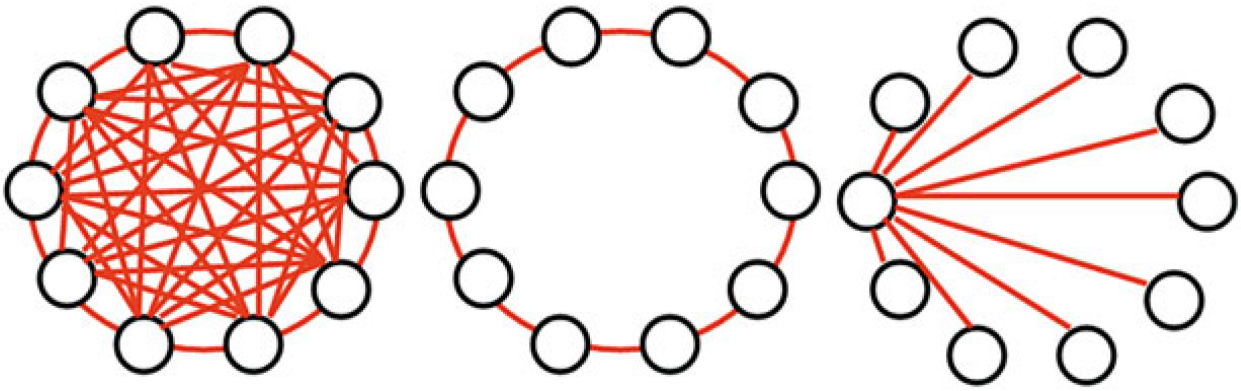

FIG. 10: Exemples de voisinages utilisés dans le modèle l_i^{best} (Gendreau and Potvin, 2019).

c_2 trop grand les pousse à suivre aveuglément la meilleure solution connue, au risque d'un piégeage dans un optimum local. Des études empiriques suggèrent que des valeurs $c_1, c_2 \in [1.5, 2.0]$ avec $c_1 + c_2 \approx 4$ assurent un bon compromis, limitant les oscillations tout en maintenant une exploration efficace.

De même que pour les paramètres cités précédemment, la taille de l'essaim influence directement la diversité des solutions explorées. Un essaim de petite taille (10 à 20 particules) permet une convergence plus rapide, mais augmente le risque de stagnation dans un optimum local. En revanche, une taille plus importante (50 à 100 particules) améliore la couverture de l'espace de recherche et favorise une plus grande robustesse face aux minimums locaux, au prix d'une complexité computationnelle accrue.

De nombreuses variantes de l'optimisation par essaims de particules ont été proposées afin d'améliorer sa robustesse, d'adapter son comportement à des contextes spécifiques, ou encore de mieux équilibrer exploration et exploitation. La version standardisée, connue sous le nom de SPSO (Standard PSO), a été définie afin d'harmoniser les paramètres et les comportements de l'algorithme de base, permettant ainsi une comparaison rigoureuse des performances (Zambrano-Bigiarini et al., 2013). Par ailleurs, des adaptations ont été développées pour traiter des espaces de recherche discrets, comme la BPSO (Binary PSO), introduite par Kennedy and Eberhart (1997), dans laquelle les positions des particules sont représentées par des vecteurs binaires, et la mise à jour des vélocités s'interprète sous forme de probabilités de changement d'état, souvent à l'aide d'une fonction sigmoïde. D'autres variantes explorent les problèmes multiobjectifs, les versions hybrides avec d'autres métaheuristiques, ou encore l'adaptativité, où les paramètres évoluent dynamiquement au cours de la recherche. Ces extensions témoignent de la flexibilité du cadre général de l'optimisation par essaims de particules et de sa capacité à s'adapter à une large variété de problèmes d'optimisation (Gendreau and Potvin, 2019 ; Martí et al., 2018 ; Siarry, 2016 ; Talbi, 2009).

Tab. 5: Espaces de solutions continus et à base de permutations.

Notion	Espaces continus	Espaces à base de permutations
Position	Vecteurs de $\mathbb{R}^n$	Permutations de S_n
Vélocité	Vecteurs de $\mathbb{R}^n$	Séquences de transpositions
Coefficients	Constantes dans $\mathbb{R}$	Constantes dans $\mathbb{R}$
Mise à jour	Formules (4.1), (4.2) et (4.3)	Application séquentielle des transpositions
Implémentation	Opérateurs vectoriels classiques	Opérateurs basés sur des heuristiques (non définis mathématiquement)

4.3 Prise en compte de l'aspect combinatoire

L'optimisation par essaims de particules est une technique puissante dans les espaces continus, mais son application aux problèmes combinatoires, tels que l'ordonnancement en ateliers à flux continu, nécessite des adaptations spécifiques. Cette section décrit les modifications apportées pour permettre la manipulation de solutions sous forme de permutations, l'exploitation des heuristiques classiques d'ordonnancement et les mécanismes d'optimisation adoptés pour améliorer la convergence et la diversité des solutions.

L'optimisation par essaims de particules classique, fonctionnant avec des vecteurs continus, ne peut pas directement gérer les permutations des tâches dans le problème d'ordonnancement. Pour résoudre cette limitation, nous avons remplacé la position d'une particule par une séquence ordonnée de tâches, à savoir une permutation, et la vélocité par une liste d'opérations transformatrices sur ces permutations (Tableau 5). La vélocité est ainsi modélisée comme une liste de transpositions, où chaque transposition définit une paire d'indices à échanger dans la permutation actuelle. Le calcul de la vélocité consiste à déterminer la différence entre deux permutations (position actuelle et position cible) et à exprimer cette différence correctement. Ensuite, la vélocité est appliquée à la position en effectuant successivement les transpositions en question pour générer la nouvelle position.

Une permutation est une bijection d'un ensemble fini sur lui-même, c'est-à-dire une fonction qui associe à chaque élément de cet ensemble un élément unique de l'ensemble, de manière réversible (Bóna, 2022, 2025). L'ensemble de toutes les permutations d'un ensemble à n éléments forme le groupe symétrique, noté S_n. Dans ce cadre, une permutation peut être représentée de plusieurs façons :

- **Notation fonctionnelle :** cela consiste à lister directement les images des éléments d'une permutation π selon un ordre de référence. Par exemple[2], si $\pi(a) = b$, $\pi(b) = d$, $\pi(c) = a$, $\pi(d) = c$, on peut écrire $\pi = (b, d, a, c)$, en supposant que l'ordre de référence est (a, b, c, d).

[2]Ces exemples utilisent des lettres pour les valeurs afin d'éviter toute confusion avec les indices.

- **Notation à deux lignes :** elle montre explicitement la correspondance entre chaque élément et son image :
$$\pi = \begin{pmatrix} a & b & c & d \\ b & d & a & c \end{pmatrix}$$
- **Notation à trois lignes :** elle indique les indices, les éléments de référence, puis leurs images. Ici, l'élément en position 1 (a) est envoyé vers b, celui en position 2 (b) vers d et ainsi de suite :
$$\pi = \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{pmatrix} a & b & c & d \\ b & d & a & c \end{pmatrix}$$

Enfin, une permutation peut être représentée par sa décomposition en cycles, une notation particulièrement utile en combinatoire. Un cycle est une suite ordonnée d'éléments $(x_1\ x_2\ \ldots\ x_k)$ tels que :
$$\pi(x_1) = x_2, \quad \pi(x_2) = x_3, \quad \ldots, \quad \pi(x_k) = x_1.$$
Il est possible de visualiser ce comportement comme une boucle fermée :
$$x_1 \to x_2 \to x_3 \to \cdots \to x_k \to x_1.$$
Par exemple, si $\pi(a) = b$, $\pi(b) = d$, $\pi(d) = c$, et $\pi(c) = a$, alors π forme un seul cycle :
$$a \to b \to d \to c \to a,$$
ce qui se note $(a\ b\ d\ c)$.

Une permutation peut aussi contenir plusieurs cycles disjoints. Par exemple, si $\pi(a) = b$, $\pi(b) = a$, $\pi(c) = d$, et $\pi(d) = c$, alors :
$$a \to b \to a \quad \text{et} \quad c \to d \to c,$$
et la permutation s'écrit $(a\ b)(c\ d)$, représentant deux cycles indépendants. Dans ce cas, un point fixe est un élément qui reste inchangé : $\pi(x) = x$. Il forme un cycle de longueur 1, noté (x). Ces cycles sont souvent omis dans la notation, sauf s'ils sont nécessaires pour l'analyse.

La transformation d'une permutation π en une autre permutation σ peut être réalisée en effectuant un certain nombre de transpositions. Une transposition modifie localement l'ordre des éléments dans la permutation. Il est possible de transformer π en σ en un nombre fini de transpositions, allant de 0 à $n-1$, où n est le nombre d'éléments dans la permutation. En effet, si π est déjà égal à σ, aucune transposition n'est nécessaire (cas 0). Dans le cas général, chaque transposition permet de rapprocher π de σ en corrigeant une position incorrecte. Ainsi, au plus $n-1$ transpositions suffisent pour réarranger les éléments de π afin qu'ils correspondent

exactement à ceux de σ, car chaque transposition peut corriger au moins un emplacement erroné sans en créer davantage. Cette propriété est fondamentale en théorie des permutations et en algèbre, notamment dans l'étude du groupe symétrique. Par exemple, soient $\pi = (c,\ a,\ d,\ b)$ et $\sigma = (a,\ b,\ c,\ d)$, avec $n = 4$. Pour transformer π en σ, il est possible d'effectuer les transpositions suivantes :

1. En appliquant (1 2) sur $(c,\ a,\ d,\ b)$ pour obtenir $(a,\ c,\ d,\ b)$
2. En appliquant (2 4) sur $(a,\ c,\ d,\ b)$ pour obtenir $(a,\ b,\ d,\ c)$
3. En appliquant (3 4) sur $(a,\ b,\ d,\ c)$ pour obtenir $(a,\ b,\ c,\ d)$

Ainsi, trois transpositions, soit $n - 1 = 3$, suffisent pour obtenir σ à partir de π. À cet égard, la vélocité obtenue se note par $[(1\ 2), (2\ 4), (3\ 4)]$.

Algorithme 11 : Calcul de la vélocité entre deux permutations.

Entrées : Une permutation source s et une permutation cible t.
Sorties : Une vélocité v sous forme de séquence de transpositions.

```
1  v ← [] ;
2  e ← s ;
3  m ← {e[i] ↦ i | 1 ≤ i ≤ ‖s‖} ;
4  pour i ← 1 à ‖s‖ faire
5      si e[i] ≠ t[i] alors
6          j ← m[t[i]] ;
7          Ajouter (i j) à v ;
8          Échanger e[i] et e[j] ;
9          m[e[i]] ← i ;
10         m[e[j]] ← j ;
11 retourner v
```

D'un point de vue pratique, cette séquence de transpositions peut être construite efficacement à l'aide d'un algorithme itératif (Algorithme 11). Celui-ci maintient une copie de la permutation source ainsi qu'un dictionnaire associant chaque valeur à sa position actuelle. En parcourant chaque position de gauche à droite, on identifie les désaccords entre la source et la cible. Lorsqu'un tel désaccord est détecté, l'élément attendu est localisé via le dictionnaire, puis une transposition est appliquée pour corriger la position. La structure du dictionnaire est ensuite mise à jour pour refléter la permutation modifiée. Ce processus garantit que chaque transposition rapproche la permutation courante de la cible, jusqu'à obtenir une correspondance complète. Cette propriété bien connue en combinatoire justifie l'utilisation des séquences de transpositions comme opérations élémentaires dans notre adaptation de l'optimisation par essaims de particules. Elles permettent de mesurer et d'orienter efficacement les déplacements dans l'espace des permutations tout en garantissant la validité des vélocités générées aléatoirement.

Comme dans la définition du modèle l_i^{best} selon les formules (4.2) et (4.3), notre approche détermine la vélocité d'une particule en combinant trois composantes fondamentales : l'inertie, la composante cognitive, et la composante sociale. Toutefois, dans le contexte des permutations, il n'existe pas de méthode mathématique canonique pour fusionner des séquences de transpositions, contrairement aux vecteurs dans $\mathbb{R}^n$ où l'addition pondérée est bien définie. Pour contourner cette difficulté, nous avons conçu un mécanisme de fusion probabiliste qui agit à deux niveaux complémentaires : la détermination de la longueur de la nouvelle vélocité, et la sélection des transpositions qui la composent. Concrètement, l'inertie, le coefficient cognitif, et le coefficient social sont multipliés par un tirage aléatoire uniforme dans $[0, 1]$, générant trois valeurs aléatoires w_1, w_2, w_3. Ces valeurs sont ensuite normalisées pour obtenir des probabilités (p_1, p_2, p_3) telles que $p_1 + p_2 + p_3 = 1$. Ces probabilités jouent un rôle double. D'une part, elles sont utilisées pour calculer la longueur cible de la vélocité résultante selon une moyenne pondérée[3], comme montré dans l'algorithme 12 :

$$l = \text{arrondi}(p_1 \cdot \|v_1\| + p_2 \cdot \|v_2\| + p_3 \cdot \|v_3\|).$$

D'autre part, elles servent à guider aléatoirement l'échantillonnage des transpositions parmi les trois séquences d'origine v_1, v_2 et v_3, représentant respectivement l'inertie, l'expérience individuelle et l'influence sociale. À chaque itération, une des trois sources est sélectionnée aléatoirement selon ces probabilités, et la prochaine transposition disponible est extraite si elle n'a pas encore été ajoutée à la nouvelle vélocité. Le processus se répète jusqu'à ce que la longueur cible soit atteinte ou dépassée. Ce mécanisme assure ainsi que la fusion soit à la fois diversifiée, grâce à l'aléa contrôlé, et non redondante, en évitant les transpositions dupliquées. Par ailleurs, il introduit une stochasticité maitrisée qui favorise l'exploration de l'espace de recherche sans compromettre la pertinence des solutions générées.

Bien que des approches telles que le vote majoritaire ou la méthode de Borda soient généralement peu couteuses et efficaces dans des contextes standards de fusion de classements, elles se révèlent inadaptées à notre cas d'usage intensif en optimisation par essaims de particules. Le vote majoritaire, appliqué à des séquences de transpositions, nécessite une analyse combinatoire pour identifier les opérations les plus fréquentes entre plusieurs solutions (Li et al., 2022 ; Raol, 2015), ce qui devient rapidement complexe à mesure que la taille des permutations et le nombre de particules augmentent. De son côté, la méthode de Borda attribue des scores aux éléments en fonction de leur position dans différentes séquences, puis les agrège pour produire un ordre final. Cependant, cette approche repose sur une logique de classement global difficilement applicable à des séquences de transpositions, où les dépendances locales entre opérations sont cruciales. De plus, son utilisation dans un cadre hautement itératif comme celui de l'optimisation par essaims peut induire un surcout non négligeable en raison du grand nombre de traitements requis à

[3]Le symbole $\|\cdot\|$ désigne ici l'opérateur qui donne la longueur d'une vélocité.

Algorithme 12 : Fusion probabiliste des vélocités.

Entrées : Trois vélocités v_1, v_2, v_3 ;
Probabilités associées p_1, p_2, p_3 telles que $p_1 + p_2 + p_3 = 1$.

Sorties : Vélocité fusionnée v.

```
1  v ← [] ;
2  l ← arrondi(p_1 · ‖v_1‖ + p_2 · ‖v_2‖ + p_3 · ‖v_3‖) ;
3  i ← 1 ;
4  j ← 1 ;
5  k ← 1 ;
6  tant que ‖v‖ < l faire
7      Tirer c ∈ {1, 2, 3} selon les probabilités respectives p_1, p_2, p_3 ;
8      si c = 1 alors
9          si i ≤ ‖v_1‖ alors
10             si v_1[i] n'est pas dans v alors
11                 Ajouter v_1[i] à v ;
12             i ← i + 1 ;
13     sinon si c = 2 alors
14         si j ≤ ‖v_2‖ alors
15             si v_2[j] n'est pas dans v alors
16                 Ajouter v_2[j] à v ;
17             j ← j + 1 ;
18     sinon
19         si k ≤ ‖v_3‖ alors
20             si v_3[k] n'est pas dans v alors
21                 Ajouter v_3[k] à v ;
22             k ← k + 1 ;
23 retourner v ;
```

chaque itération.

D'autres méthodes, comme la fusion par distance minimale (utilisant par exemple les distances de Kendall tau ou de Cayley), ou les approches fondées sur l'alignement ou la recherche de sous-séquences communes, posent également problème. Bien qu'elles puissent produire des résultats précis, elles reposent sur des techniques d'optimisation combinatoire ou de programmation dynamique, souvent couteuses en temps de calcul, et donc peu compatibles avec les contraintes de rapidité propres à notre cadre.

À l'inverse, notre méthode de fusion probabiliste se distingue par sa légèreté algorithmique et sa flexibilité. Elle ne dépend d'aucun cadre rigide de classement ni d'agrégation exhaustive, et repose simplement sur un tirage aléatoire contrôlé par des probabilités (p_1, p_2, p_3), permettant de générer des vélocités diversifiées tout en conservant l'information essentielle des composantes initiales. Ce mécanisme offre un bon compromis entre exploration et exploitation, et se révèle particulièrement bien adapté au caractère dynamique et fortement itératif de l'optimisation par essaims dans des espaces discrets, comme ceux des permutations. Elle offre ainsi une solution pragmatique, à la fois efficace, pertinente et facile à mettre en œuvre, ce qui en fait une stratégie robuste et extensible pour des applications à grande échelle.

Algorithme 13 : Changement d'une position selon une vélocité.

Entrées : Vélocité v et position p.
Sorties : Position p changée selon v.

```
pour chaque transposition (i j) de v faire
    Échanger p[i] et p[j] ;
retourner p ;
```

Une fois les notions de vélocité, ainsi que leur calcul et leur combinaison définis, il ne reste qu'à appliquer cette vélocité à une position donnée pour en déduire une nouvelle. Cette opération consiste à interpréter la vélocité comme une séquence ordonnée de transpositions, que l'on applique successivement à la permutation initiale. Chaque transposition modifie localement l'ordre des éléments, déplaçant ainsi progressivement la particule dans l'espace de recherche. L'algorithme 13 formalise ce processus.

4.4 Gestion de la stagnation

Dans notre approche, nous introduisons une utilisation adaptative d'heuristiques classiques pour améliorer la qualité des solutions, tout en préservant la diversité initiale de la population. Contrairement à certaines méthodes qui emploient ces heuristiques pour initialiser l'essaim, notre implémentation évite cette approche afin de prévenir une convergence prématurée. L'essaim est donc initialisé de manière aléatoire, favorisant ainsi une exploration plus large de l'espace de recherche dès le départ.

L'injection d'heuristiques est déclenchée uniquement en cas de stagnation locale, détectée lorsque la vélocité d'une particule devient nulle. En d'autres termes, la stagnation est identifiée lorsque la combinaison des vélocités d'une particule devient nulle. Dans ce cas, pour relancer la recherche, la position de la particule est réinitialisée en utilisant une solution issue d'une des heuristiques suivantes : Johnson, Palmer, Campbell-Dudek-Smith, Gupta, Dannenbring, Nawaz-Enscore-Ham ou Suliman. Si aucune heuristique n'est fournie, une nouvelle position aléatoire est générée. Simultanément, sa vélocité est réinitialisée aléatoirement, et son voisinage est redéfini. De plus, même si plusieurs particules utilisent la même heuristique comme point de départ, leurs trajectoires divergent grâce à la diversité introduite par la réinitialisation aléatoire de la vélocité et la redéfinition du voisinage.

Le paramètre nommée *taux de diversification*, introduit pour les besoins de notre approche, joue un rôle clé : il contrôle la probabilité avec laquelle une particule, lorsqu'elle est en stagnation, va effectivement appliquer une réinitialisation de sa position ainsi que de son voisinage. Concrètement, après avoir détecté une stagnation, la correction de position (via une heuristique ou par génération aléatoire) et la correction de voisinage sont chacune déclenchées de manière probabiliste selon ce paramètre. Un taux de diversification élevé favorise donc une exploration plus agressive de l'espace des solutions en multipliant les réinitialisations, tandis qu'un taux plus faible rend la recherche plus conservatrice, limitant ces diversifications pour se concentrer sur l'exploitation locale. Cela permet d'ajuster finement l'équilibre entre exploration et exploitation au sein de l'algorithme.

Afin de favoriser une convergence fine vers les solutions les plus prometteuses identifiées au cours du processus d'optimisation, notre approche inclut également une réduction progressive de la vélocité des particules au fil des itérations. Cette stratégie est mise en place par l'ajustement itératif d'un paramètre qui contrôle la vélocité maximale des particules :

$$l(t) = \text{arrondi}\left(L - 1 - (L-2) \cdot \left(\frac{t}{T}\right)^2\right),$$

où :

- L représente la taille des particules, ce qui correspond ici au nombre de machines de l'atelier.
- t est l'itération en cours.
- T est le nombre maximal d'itérations.

Cette façon de faire permet initialement une exploration étendue de l'espace de recherche grâce à des déplacements plus amples des particules. Cette formule, qui détermine la borne supérieure de la vélocité à chaque itération, se justifie par sa capacité à moduler la décroissance de la vélocité de manière non linéaire. Plus précisément, la descente n'est pas linéaire, mais concave,

ce qui permet de maintenir une vélocité relativement élevée pendant les itérations initiales. Cette caractéristique est cruciale dans le contexte de l'optimisation par essaims de particules, car elle favorise une exploration approfondie de l'espace de recherche et permet aux particules de découvrir des régions prometteuses. Ensuite, à mesure que l'algorithme progresse et que des régions potentiellement optimales sont découvertes, la diminution de la vélocité devient plus prononcée. Cette transition vers une décroissance plus rapide incite les particules à effectuer des mouvements plus localisés, favorisant ainsi un affinement de la recherche autour de ces zones prometteuses. Cette transition d'une exploration globale à une exploitation locale, orchestrée par la décroissance concave de la vélocité, est une stratégie couramment employée dans l'optimisation par essaims de particules. Elle permet d'améliorer la précision de la solution finale obtenue en concentrant les efforts de recherche dans les régions les plus prometteuses, tout en évitant une convergence prématurée vers un optimum local.

Ce chapitre a présenté l'approche proposée pour résoudre les problèmes d'ordonnancement en ateliers à flux continu via une adaptation de l'optimisation par essaims de particules, détaillant la reformulation de la position et de la vélocité pour les permutations et l'intégration dynamique d'heuristiques contre la stagnation. Les bases théoriques étant posées, le chapitre suivant évaluera l'efficacité, la robustesse et le potentiel de cette approche par des expérimentations rigoureuses.

Chapitre 5

Résultats et discussion

Dans ce chapitre, nous présentons les résultats expérimentaux obtenus à partir de l'application de notre approche d'optimisation par essaims de particules au problème d'ordonnancement en ateliers à flux continu. Après avoir décrit en détail le jeu de données de référence utilisé, notamment les instances proposées par Taillard, nous détaillons les protocoles de test mis en place, incluant la génération systématique de configurations de paramètres et l'automatisation des expériences. Nous analysons ensuite les résultats obtenus, en nous appuyant sur des outils statistiques et de visualisation afin de dégager les tendances générales, d'identifier les paramètres les plus influents, et de comparer les performances de notre méthode aux meilleures solutions connues. Ce chapitre a pour objectif d'évaluer rigoureusement l'efficacité, la robustesse et les limites de l'approche proposée, en fournissant des éléments concrets de discussion et de recommandation pour de futurs travaux.

5.1 Jeu de données utilisé

Le jeu de données de Taillard (1993) pour le problème d'ordonnancement, constitue une référence incontournable dans le domaine de l'optimisation combinatoire[1]. Les données en question comprennent des *instances* variées, caractérisées par des combinaisons de 20 à 500 tâches et de 5 à 20 machines, offrant ainsi un large éventail de complexité pour l'évaluation des algorithmes d'ordonnancement pour les problèmes d'ateliers à tâches, d'ateliers à flux continu ainsi que des ateliers ouverts. Chaque instance est définie par une matrice de temps de traitement, où chaque ligne représente une tâche et chaque colonne une machine, les valeurs indiquant le temps nécessaire pour traiter une tâche sur une machine donnée. Ces données sont générées de manière pseudoaléatoire selon des distributions contrôlées, assurant la reproductibilité et la diversité des cas de test. Le format des fichiers est structuré de manière à faciliter leur utilisation dans divers environnements de programmation, avec des outils disponibles pour leur génération et leur vérification. Le tableau 6 illustre l'une de ces instances.

[1] http ://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html.

Tab. 6: Instance 20_5_1 du jeu de données de Taillard (1993).

	Machine 1	**Machine 2**	**Machine 3**	**Machine 4**	**Machine 5**
Tâche 1	54	79	16	66	58
Tâche 2	83	3	89	58	56
Tâche 3	15	11	49	31	20
Tâche 4	71	99	15	68	85
Tâche 5	77	56	89	78	53
Tâche 6	36	70	45	91	35
Tâche 7	53	99	60	13	53
Tâche 8	38	60	23	59	41
Tâche 9	27	5	57	49	69
Tâche 10	87	56	64	85	13
Tâche 11	76	3	7	85	86
Tâche 12	91	61	1	9	72
Tâche 13	14	73	63	39	8
Tâche 14	29	75	41	41	49
Tâche 15	12	47	63	56	47
Tâche 16	77	14	47	40	87
Tâche 17	32	21	26	54	58
Tâche 18	87	86	75	77	18
Tâche 19	68	5	77	51	68
Tâche 20	94	77	40	31	28

Le jeu de données de Taillard est largement utilisé pour comparer les performances des algorithmes heuristiques et exacts, et est considéré comme un standard dans la littérature scientifique sur l'ordonnancement. Le protocole de comparaison des performances des algorithmes sur les instances fournies repose sur l'évaluation du makespan. Pour chaque instance, les meilleures solutions connues sont reportées, offrant des références précieuses pour mesurer l'efficacité des méthodes développées (Blazewicz et al., 2019 ; Emmons and Vairaktarakis, 2013). Ces résultats proviennent de contributions issues de la communauté scientifique, qui mobilisent des approches particulièrement élaborées, telles que les métaheuristiques hybrides, les recherches locales intensives, les matheuristiques combinant méthodes exactes et heuristiques, ainsi que des variantes sophistiquées de l'algorithme NEH. Ces techniques, bien que puissantes, nécessitent souvent des temps de calcul importants et une certaine maitrise méthodologique pour être mises en œuvre efficacement. Tout cela fait que le jeu de données de référence de Taillard constitue non seulement un standard incontournable pour la comparaison des algorithmes, mais représente aussi un ensemble de défis techniques qui illustrent les limites actuelles de l'optimisation en ordonnancement.

Dans le cadre de cette étude, nous nous concentrons sur les 30 instances du problème d'ordonnancement en ateliers à flux continu, correspondant aux combinaisons de 20 et 50 tâches, avec 5, 10 et 20 machines, et comprenant cinq exemples distincts par configuration. Cette diversité per-

TAB. 7: Valeurs possibles pour les paramètres.

Paramètre	**Valeurs possibles**
Taille de l'essaim	$\{20, 50, 100\}$
Taille des voisinages	$\{3, 4, 5, 6\}$
Inertie	$[0, 1]$
Coefficient cognitif	$[0, 1]$
Coefficient social	$[0, 1]$
Taux de diversification	$[0, 1]$
Nombre maximal d'itérations autorisées	$\{200, 500, 1000\}$
Positions heuristiques injectées	{Non, Oui}

met de couvrir plusieurs configurations, allant des petites tailles où les solutions optimales sont plus accessibles, jusqu'aux grandes instances où la difficulté combinatoire explose, offrant ainsi un terrain d'évaluation robuste et crédible. De plus, le fait que chaque instance est accompagnée de la meilleure solution connue à ce jour en matière de makespan minimal, permet d'avoir un repère de référence pour l'étude. Compte tenu de la difficulté à atteindre les résultats optimaux reportés dans le jeu de données, souvent obtenus par des approches très spécialisées et couteuses en calcul, notre objectif principal n'est pas de surpasser ces solutions, mais d'évaluer l'efficacité et la robustesse de notre approche proposée. Nous cherchons ainsi à situer nos résultats par rapport aux tendances générales de la littérature, en mesurant dans quelle mesure notre méthode se rapproche des performances établies tout en analysant ses points forts et ses limites.

5.2 Tests effectués

Pour chacune des instances de test, nous avons *généré aléatoirement* un ensemble conséquent de combinaisons de paramètres pour l'optimisation par essaims de particules, visant à identifier les réglages les plus prometteurs et à mieux comprendre leur influence sur les résultats finaux (Tableau 7). Les paramètres considérés incluent la taille de l'essaim, la taille des voisinages, l'inertie, le coefficient cognitif, le coefficient social, le taux de diversification ainsi que le nombre maximal d'itérations autorisées. Nous avons en parallèle étudié l'impact de l'injection de positions heuristiques en cas de stagnation.

Le processus d'expérimentation a été mis en œuvre de manière automatisée à l'aide d'un programme écrit en ce sens. Pour chaque configuration, le programme charge les données d'entrée, exécute l'optimisation, puis enregistre soigneusement les paramètres ainsi que les résultats obtenus. Ces résultats comprennent notamment le makespan et le temps d'exécution total. L'ensemble de ces informations est consigné sous forme d'entrées dans un fichier CSV[2], afin de faciliter

[2]Un fichier CSV (Comma-Separated Values) est un format de fichier texte structuré permettant de stocker des données tabulaires, chaque ligne représentant un enregistrement et chaque colonne étant séparée par un caractère délimiteur, généralement une virgule.

une analyse ultérieure. Afin d'assurer une diversité suffisante et obtenir une *couverture statistique* solide, nous avons répété ce processus pour 1000 configurations différentes par instance, ce qui représente au total $30 \times 1000 = 30000$ expériences et entrées dans le fichier CSV. Pour gérer efficacement cette charge computationnelle, les calculs ont été parallélisés à l'aide d'un bassin de processus, ce qui permet de tirer parti des ressources multicœurs disponibles et d'accélérer significativement le temps total de calcul[3].

Une fois les données expérimentales collectées, nous avons entrepris une phase d'analyse. Dans un premier temps, nous avons calculé et visualisé la matrice de corrélation de Spearman entre tous les paramètres et les résultats (sous forme de heatmap), afin d'obtenir une vue d'ensemble des relations entre chaque couple de variables. Cette analyse globale permet de repérer rapidement quels paramètres exercent l'influence la plus marquée sur les performances, en identifiant les corrélations les plus fortes (positives ou négatives), et sert ainsi de guide pour orienter les analyses plus fines. Étant donné que le nombre d'instances est suffisamment grand, afficher directement les nuages de points ne permet pas d'obtenir une vision claire de l'impact des paramètres. Pour approfondir, nous avons donc généré des graphiques de tendance, où la direction générale est représentée par une ligne, tandis que la dispersion des points est illustrée par une bande autour de cette ligne, dont la largeur varie selon l'ampleur de la dispersion. Cette approche permet de mieux percevoir les tendances globales, même en présence de forte variabilité, et d'identifier plus clairement les effets des paramètres sur la qualité des solutions (mesurée par le makespan minimal) et sur le temps de calcul, sans être noyé par le bruit des observations individuelles.

Afin de tirer pleinement parti des données collectées, nous avons isolé, pour chaque instance, les configurations ayant atteint la meilleure adaptation globale, c'est-à-dire celles associées au makespan minimal. À partir de cet ensemble filtré, nous avons calculé des valeurs représentatives pour chaque paramètre : la moyenne arithmétique pour les paramètres continus, afin de refléter leur tendance centrale, et la médiane pour les paramètres discrets, afin d'assurer une meilleure robustesse face aux valeurs extrêmes. Cette approche nous a permis d'extraire une configuration *optimale moyenne*, servant de base à des recommandations concrètes pour le paramétrage de l'optimisation par essaims de particules, adaptées aux spécificités des problèmes d'ordonnancement en ateliers à flux continu. En utilisant cette configuration, nous avons ensuite réalisé des tests sur l'ensemble des instances, compilé les résultats obtenus dans un tableau récapitulatif, et les avons comparés aux meilleurs résultats rapportés dans la littérature afin d'évaluer de manière rigoureuse les performances de notre méthode.

Vu l'*aspect pratique* de la thématique étudiée, nous avons conçu une interface graphique intuitive permettant aux utilisateurs de créer directement, charger, sauvegarder et générer aléatoirement des configurations d'ateliers en flux continu. L'interface intègre spécifiquement l'optimisation par essaims de particules décrite dans cette étude, permettant aux utilisateurs d'expé-

[3]Les calculs ont été réalisés sur une machine équipée d'un processeur Intel Core i7 (1.90 GHz, 4 cœurs, 8 threads), avec 16 GiB de RAM, fonctionnant sous Microsoft Windows 10.

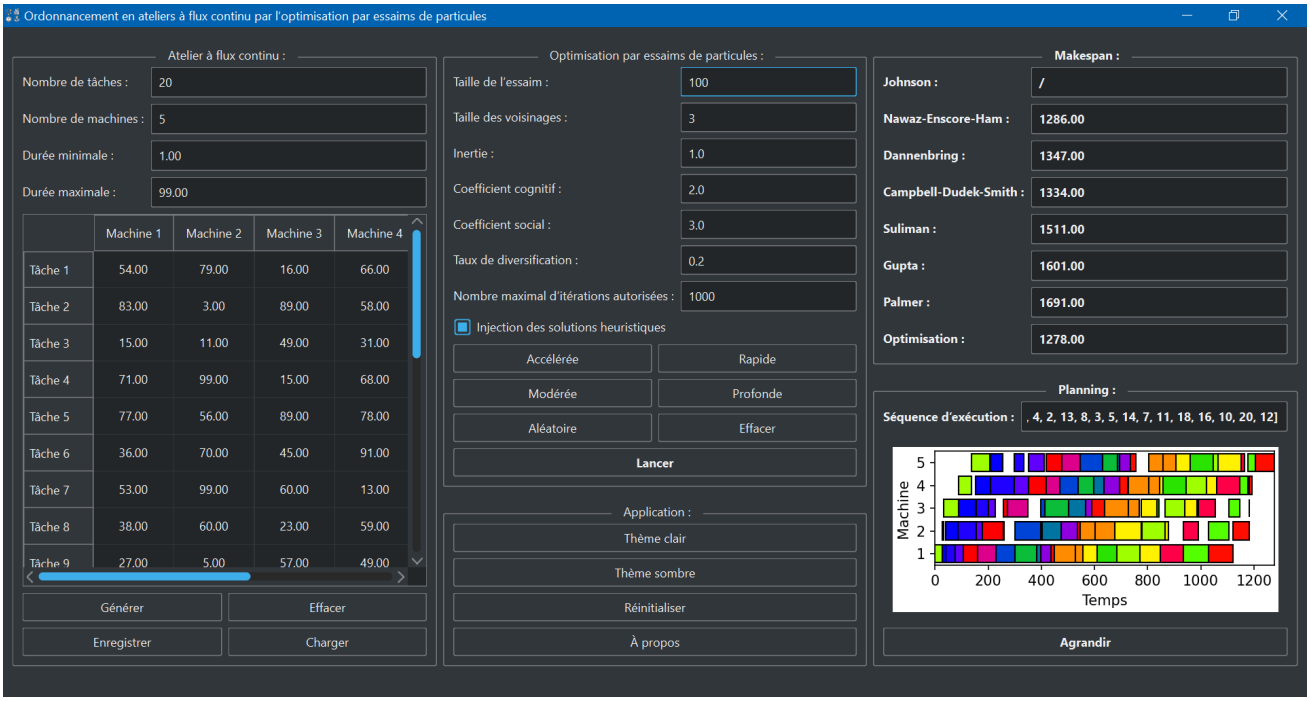

FIG. 11: Traitement de l'instance 20_5_1 en utilisant l'interface graphique.

rimenter avec différentes configurations et d'optimiser les paramètres en temps réel (Figure 11). De plus, l'interface affiche les résultats obtenus à la fois par des heuristiques de base et par l'optimisation par essaims de particules, offrant ainsi une comparaison directe des performances. Les résultats sont visualisés sous forme de diagrammes de Gantt, permettant aux utilisateurs de visualiser clairement le planning et l'organisation des tâches selon chaque configuration testée. Cette approche interactive et flexible offre une manière pratique et accessible d'explorer et d'analyser les performances de différentes configurations.

Nous avons choisi d'utiliser le langage *Python*[4] pour sa simplicité, sa lisibilité et la richesse de ses bibliothèques. Nous avons mobilisé plusieurs *modules standards*, notamment *random* pour générer des valeurs aléatoires et *concurrent.futures* pour paralléliser les tâches lourdes à l'aide de pools de processus, ce qui nous a permis d'améliorer les performances. Pour les calculs scientifiques et le traitement numérique, nous nous sommes appuyés sur *NumPy*[5], qui nous a offert des structures de données efficaces et des fonctions mathématiques avancées. Nous avons manipulé et analysé les jeux de données grâce à *pandas*[6], qui nous a facilité la préparation et l'exploitation des données. Pour visualiser nos résultats, nous avons utilisé *Matplotlib*[7] afin de produire des graphiques et des courbes, et nous avons enrichi ces visualisations avec *Seaborn*[8], qui nous a permis de créer des graphiques statistiques esthétiques et informatifs. Enfin, nous avons développé

[4] https ://www.python.org/.
[5] https ://numpy.org/.
[6] https ://pandas.pydata.org/.
[7] https ://matplotlib.org/.
[8] https ://seaborn.pydata.org/.

l'interface graphique de notre application avec *PySide*[9], qui nous a donné accès à la bibliothèque *Qt*[10] et nous a permis de concevoir une interface riche, interactive et conviviale, intégrant des composants variés comme des boutons, des champs de saisie, des tableaux et des visualisations graphiques.

5.3 Analyse et discussion

Avant de commencer avec les résultats et leur analyse, il est crucial de souligner que l'interaction entre les paramètres étudiés est complexe et que leur influence peut varier en fonction des caractéristiques spécifiques du problème d'ordonnancement traité. La figure 12 présente la matrice de corrélation entre les différents paramètres et les résultats obtenus. Son analyse révèle des liens quantifiables entre les différents paramètres et les résultats, notamment le temps d'exécution et la qualité de la solution (meilleure adaptation globale). Les corrélations les plus saillantes concernent la taille de l'essaim et le nombre maximal d'itérations, qui présentent une corrélation positive significative avec le temps d'exécution. Ceci est intuitivement compréhensible, car un plus grand nombre de particules à traiter et un plus grand nombre d'étapes de calcul augmentent inévitablement la charge computationnelle. De même, le nombre maximal d'itérations montre une corrélation positive modérée avec la meilleure adaptation globale, suggérant qu'allouer plus de temps à l'algorithme pour explorer l'espace de recherche tend à améliorer la qualité de la solution trouvée. Inversement, la plupart des autres paramètres algorithmiques, tels que la taille des voisinages, l'inertie, les coefficients cognitif et social, ainsi que le taux de diversification et l'injection de positions heuristiques, affichent des corrélations très faibles avec les deux mesures de résultat. Cette observation initiale suggère que leur impact linéaire direct sur le temps d'exécution et la meilleure adaptation globale, tel que capturé par la corrélation de Pearson, est minime dans l'ensemble des données analysées.

L'absence de corrélations claires pour certains paramètres ne signifie pas nécessairement qu'ils sont sans importance. Cela s'explique dans notre cas par le fait que les données étudiées englobent une variété de configurations des paramètres, potentiellement dispersées sur un large espace de valeurs. Dans un tel contexte, les effets de certains paramètres sur les résultats pourraient être non linéaires, se manifestant différemment selon les plages de valeurs considérées. De plus, des interactions complexes entre les paramètres pourraient exister, où l'influence d'un paramètre sur la performance de l'algorithme dépend de la valeur d'un ou plusieurs autres. La simple analyse de corrélation bivariée ne permet pas de déceler ces relations plus sophistiquées. Ainsi, les paramètres ayant un impact direct et proportionnel sur la quantité de traitement nécessaire (comme la taille de l'essaim et le nombre d'itérations) sont plus susceptibles de montrer des corrélations claires, tandis que ceux dont l'influence est plus indirecte, contextuelle ou non linéaire

[9] https ://www.qt.io/qt-for-python.
[10] https ://www.qt.io/.

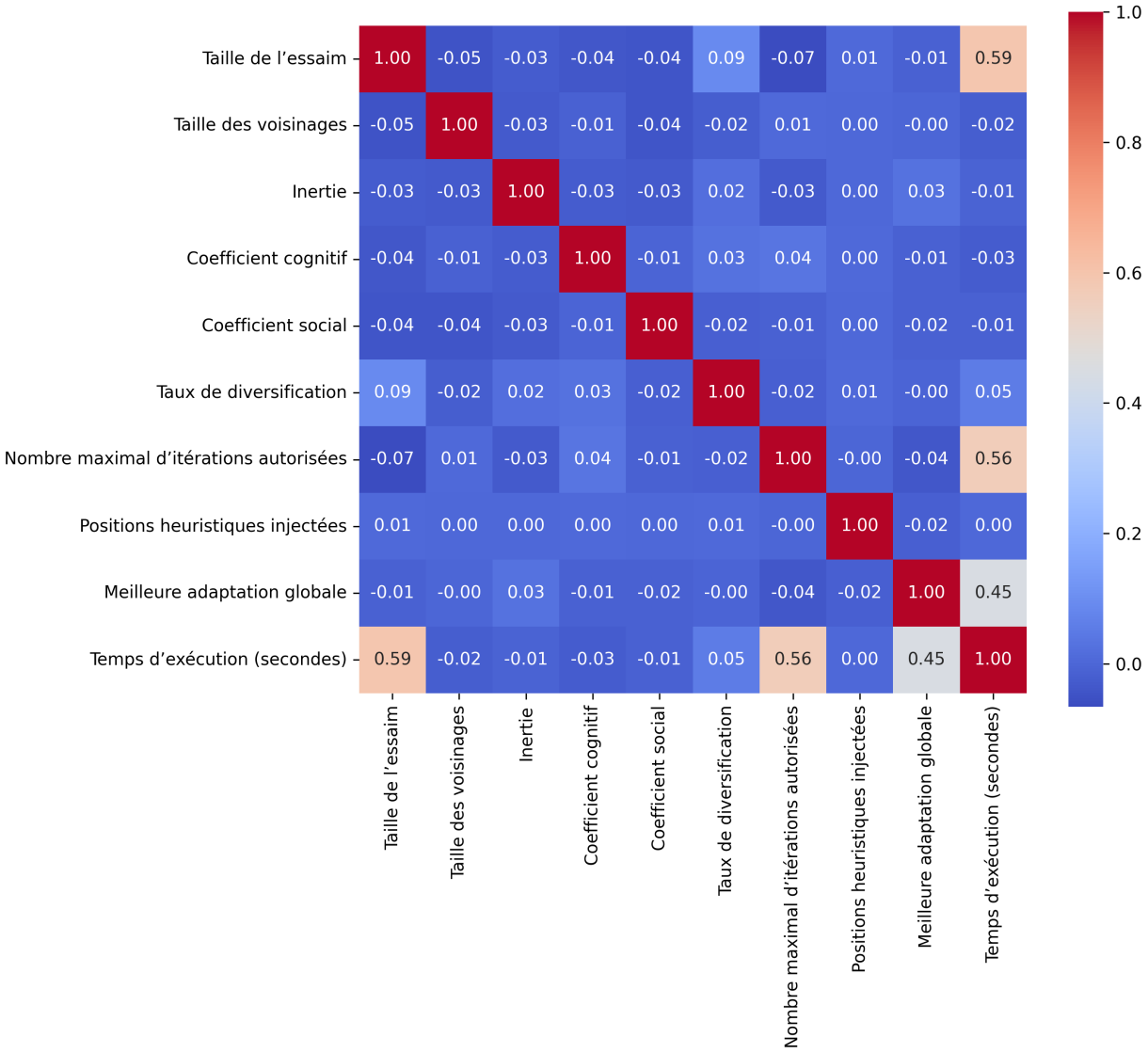

FIG. 12: Corrélations entre les paramètres et les résultats.

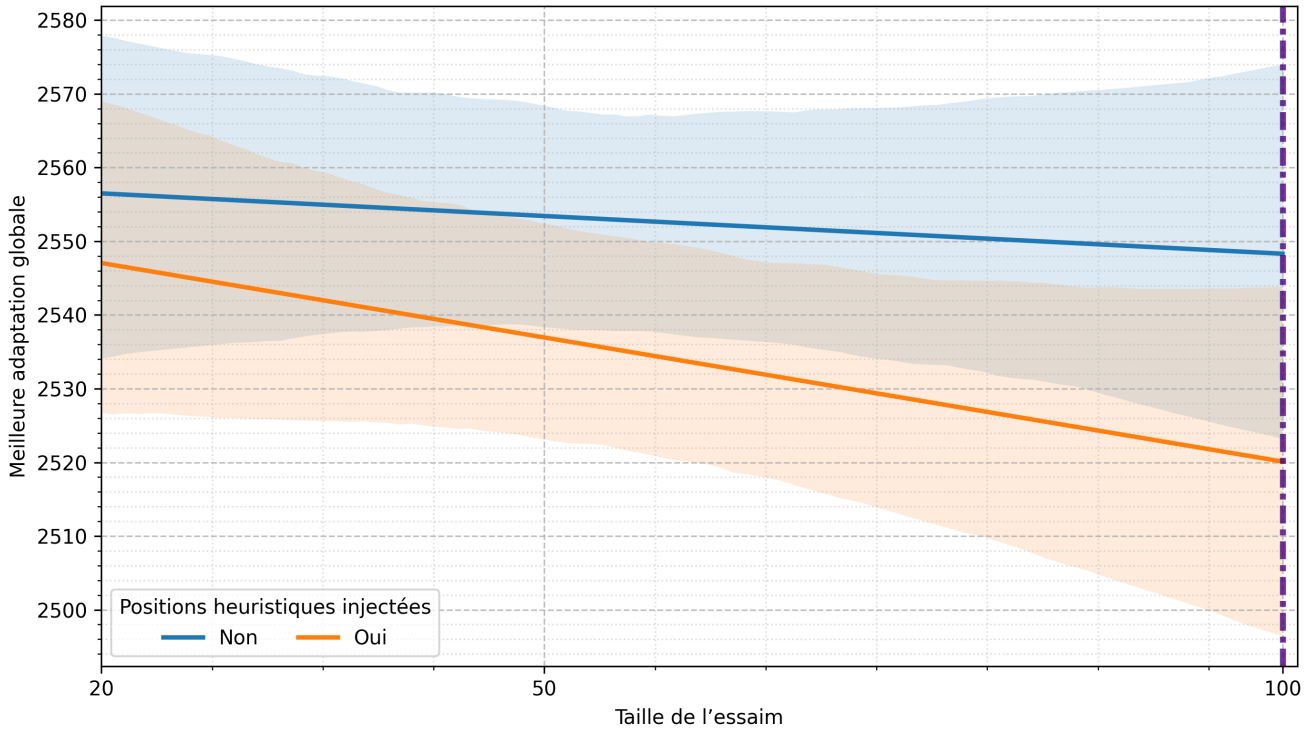

FIG. 13: Influence de la taille de l'essaim sur la meilleure adaptation globale.

pourraient ne pas révéler leur importance à travers cette seule métrique.

Afin d'approfondir cette analyse, les figures 13–26 illustrent l'influence spécifique de chaque paramètre sur le makespan et le temps d'exécution. Ces graphiques de tendance permettent de mieux comprendre la manière dont les variations d'un paramètre donné affectent les performances de l'algorithme, et de déterminer les valeurs optimales ou les plages de valeurs à privilégier. Cependant, il convient de souligner que ces valeurs optimales peuvent varier en fonction de la taille du problème, de la complexité de l'espace de recherche et des contraintes spécifiques du problème d'ordonnancement considéré.

D'après les figures 13–14, les résultats montrent qu'une augmentation de la taille de l'essaim tend à améliorer le makespan, ce qui peut être attribué à une exploration plus exhaustive de l'espace de recherche. Un essaim plus grand est en effet capable de couvrir une portion plus vaste de cet espace, augmentant ainsi les chances de trouver des solutions de meilleure qualité. Toutefois, cette amélioration s'accompagne d'une hausse du temps de calcul, mettant en évidence un compromis inhérent entre la qualité des solutions et l'efficience de l'algorithme. En d'autres termes, il est nécessaire de trouver un équilibre entre la qualité des résultats attendus et les ressources computationnelles disponibles. De plus, il est possible qu'au-delà d'une certaine taille d'essaim, l'amélioration du makespan devienne marginale, tandis que le temps de calcul continue d'augmenter de manière significative.

De manière analogue, l'accroissement de la taille des voisinages semble favoriser l'obtention de solutions de meilleure qualité, probablement en permettant à chaque particule d'intégrer une

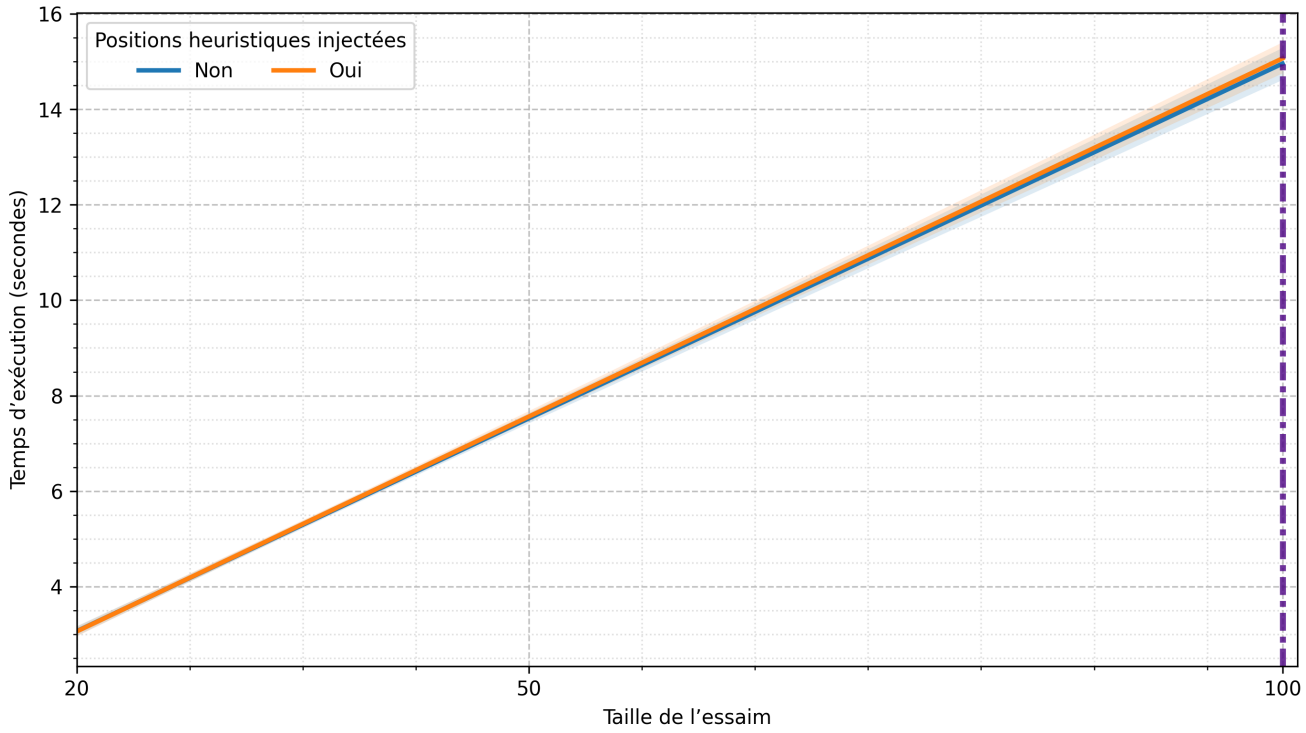

FIG. 14: Influence de la taille de l'essaim sur le temps d'exécution.

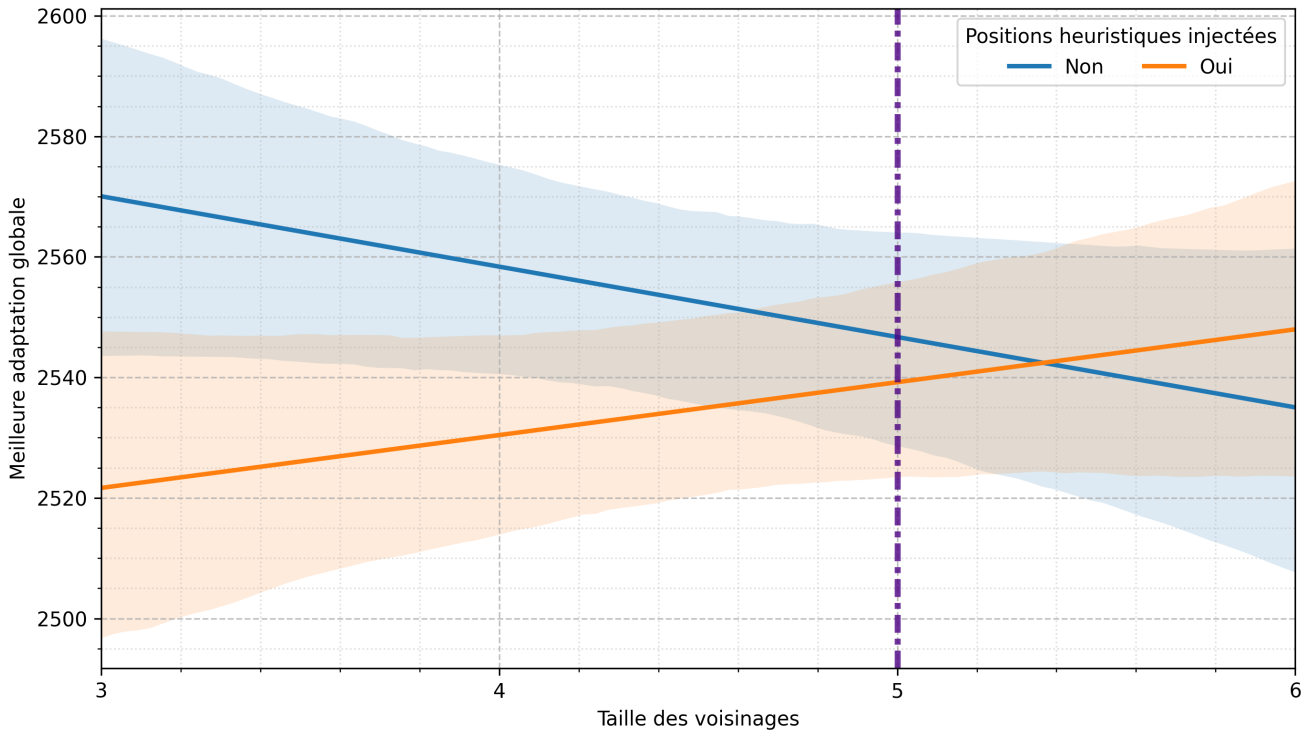

FIG. 15: Influence de la taille des voisinages sur la meilleure adaptation globale.

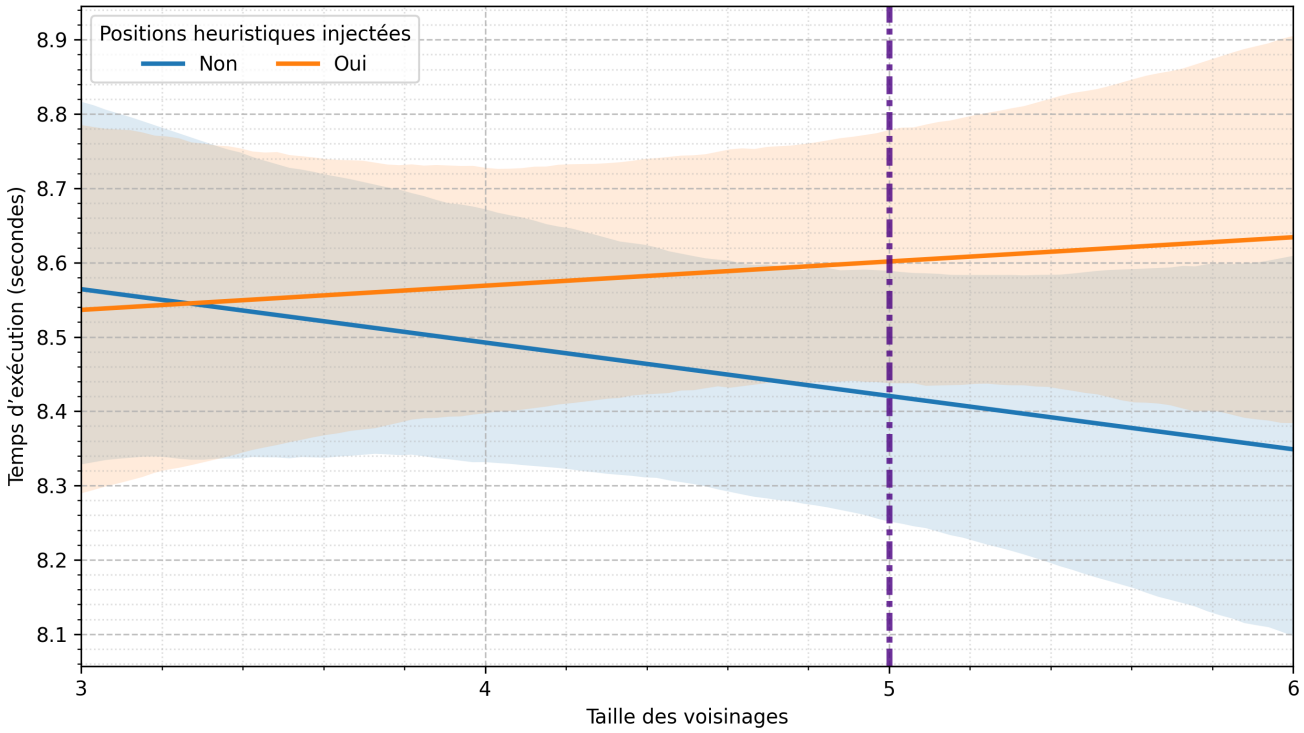

FIG. 16: Influence de la taille des voisinages sur le temps d'exécution.

plus grande quantité d'informations provenant de l'essaim (Figures 15–16). Une taille de voisinage plus importante permet en effet à chaque particule d'interagir avec un plus grand nombre d'autres particules, ce qui peut encourager le partage d'informations et l'émergence de solutions de haute qualité. Cependant, cette stratégie s'accompagne également d'une augmentation du temps de calcul, ce qui suggère l'existence d'une taille de voisinage optimale permettant de concilier efficacement qualité des solutions et temps d'exécution. Il est possible que des voisinages trop réduits limitent la capacité des particules à explorer l'espace de recherche, tandis que des voisinages trop étendus favorisent une convergence prématurée vers des optimums locaux.

L'inertie joue un rôle déterminant dans l'équilibre entre l'exploration et l'exploitation de l'espace de recherche. Une inertie élevée favorise l'exploration en permettant aux particules de maintenir leur trajectoire, tandis qu'une inertie faible encourage l'exploitation en incitant les particules à converger vers les meilleures solutions identifiées. Les résultats montrent que l'augmentation de l'inertie accroit le temps d'exécution et diminue la qualité des solutions en augmentant également le makespan (Figures 17–18). Ils suggèrent qu'une valeur d'inertie intermédiaire permet d'atteindre un compromis adéquat entre ces deux comportements. Cette valeur optimale peut toutefois varier en fonction de la complexité du problème et de la topologie de l'espace de recherche.

Les coefficients cognitif et social contrôlent l'influence relative de l'expérience personnelle et des connaissances sociales de chaque particule. Un coefficient cognitif élevé favorise l'exploration individuelle, en incitant les particules à explorer de nouvelles régions de l'espace de recherche

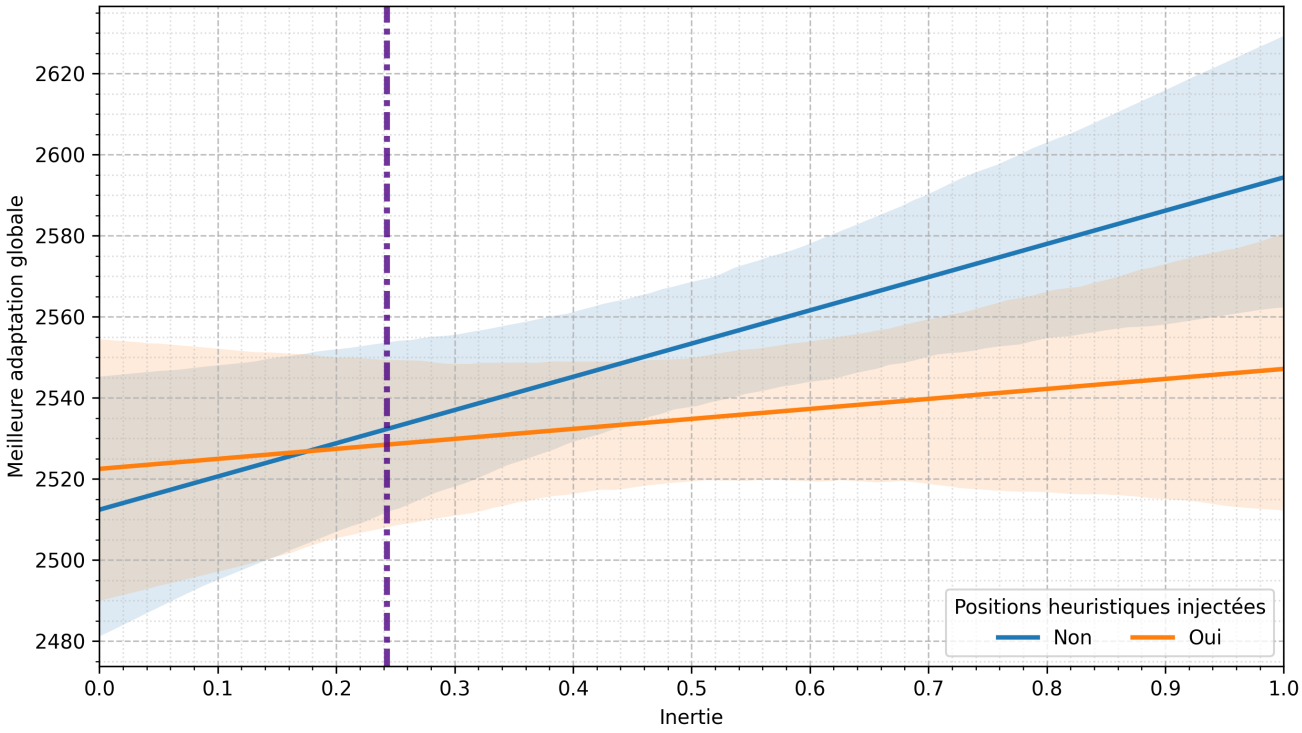

FIG. 17: Influence de l'inertie sur la meilleure adaptation globale.

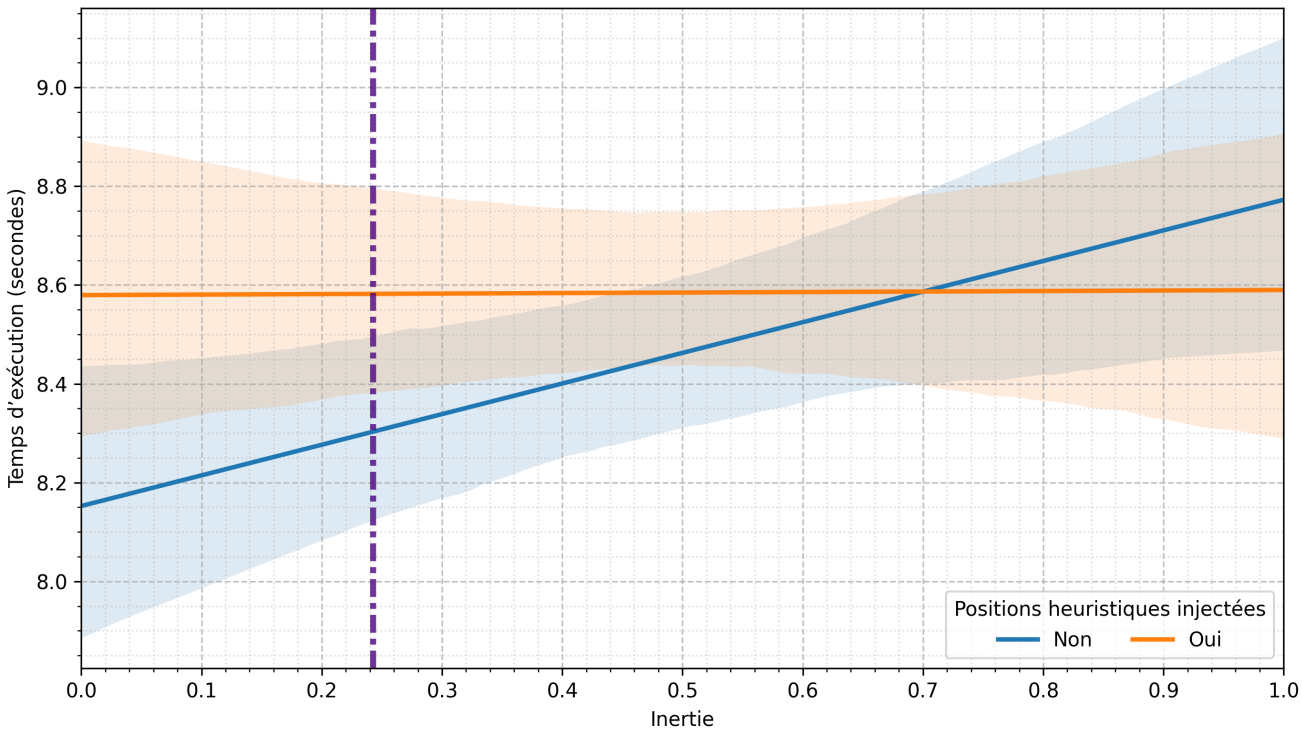

FIG. 18: Influence de l'inertie sur le temps d'exécution.

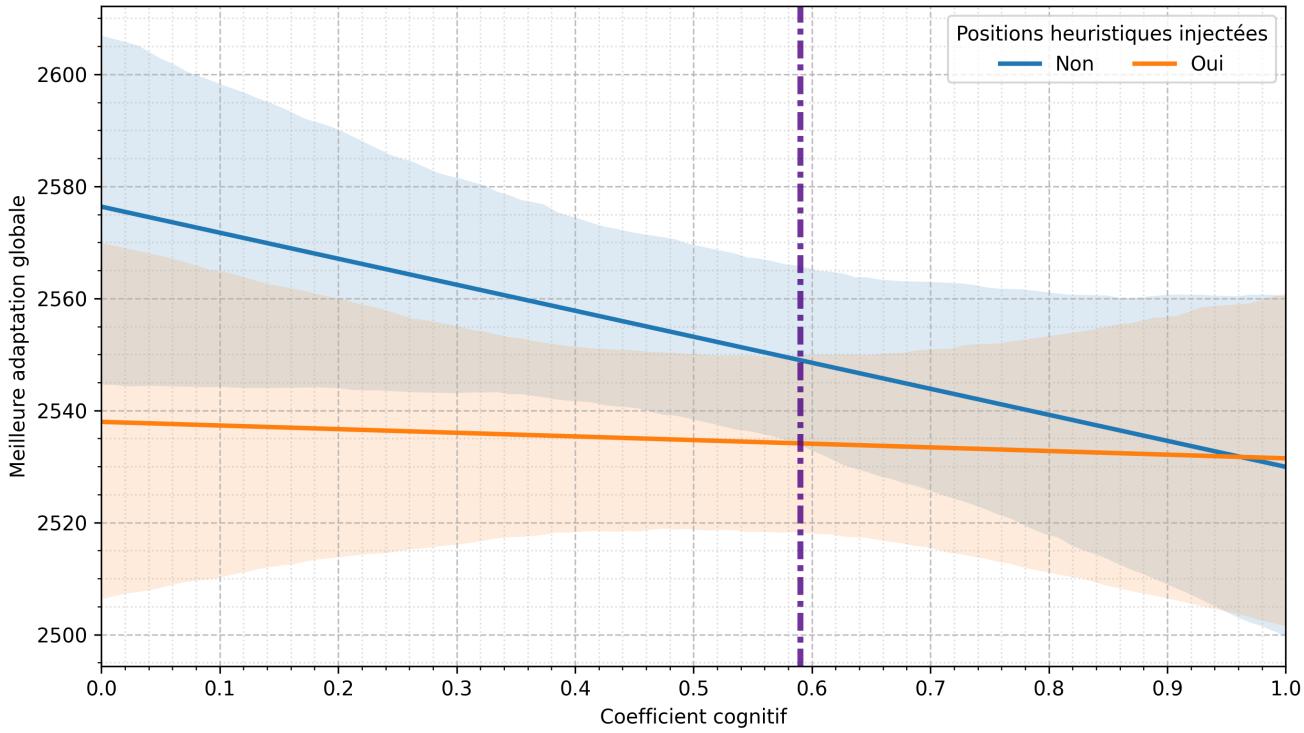

FIG. 19: Influence du coefficient cognitif sur la meilleure adaptation globale.

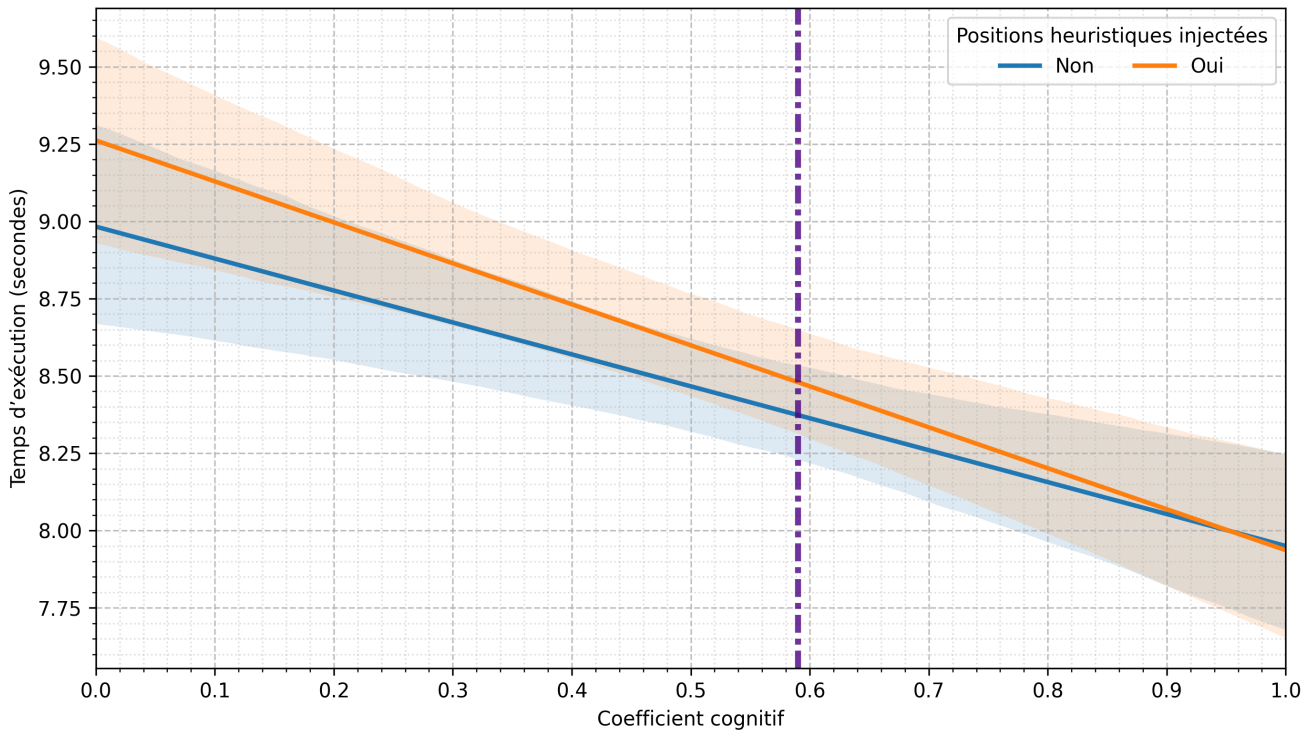

FIG. 20: Influence du coefficient cognitif sur le temps d'exécution.

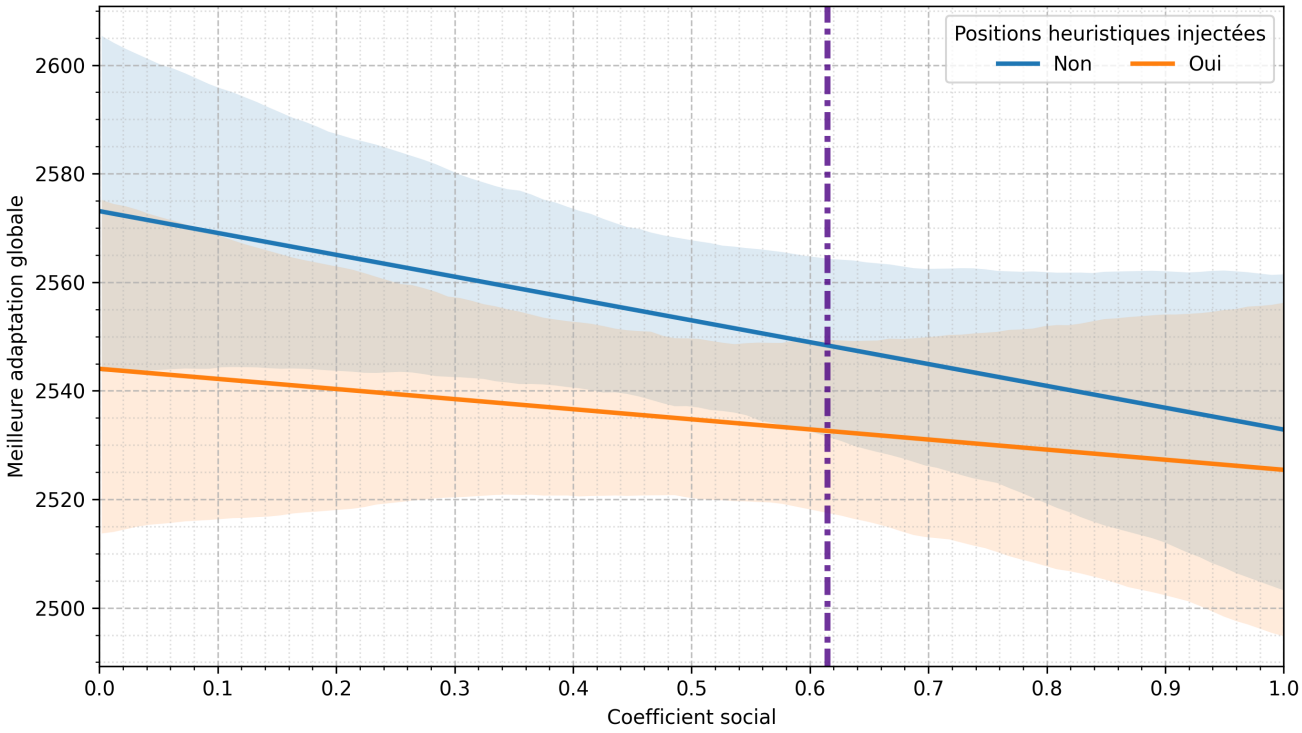

FIG. 21: Influence du coefficient social sur la meilleure adaptation globale.

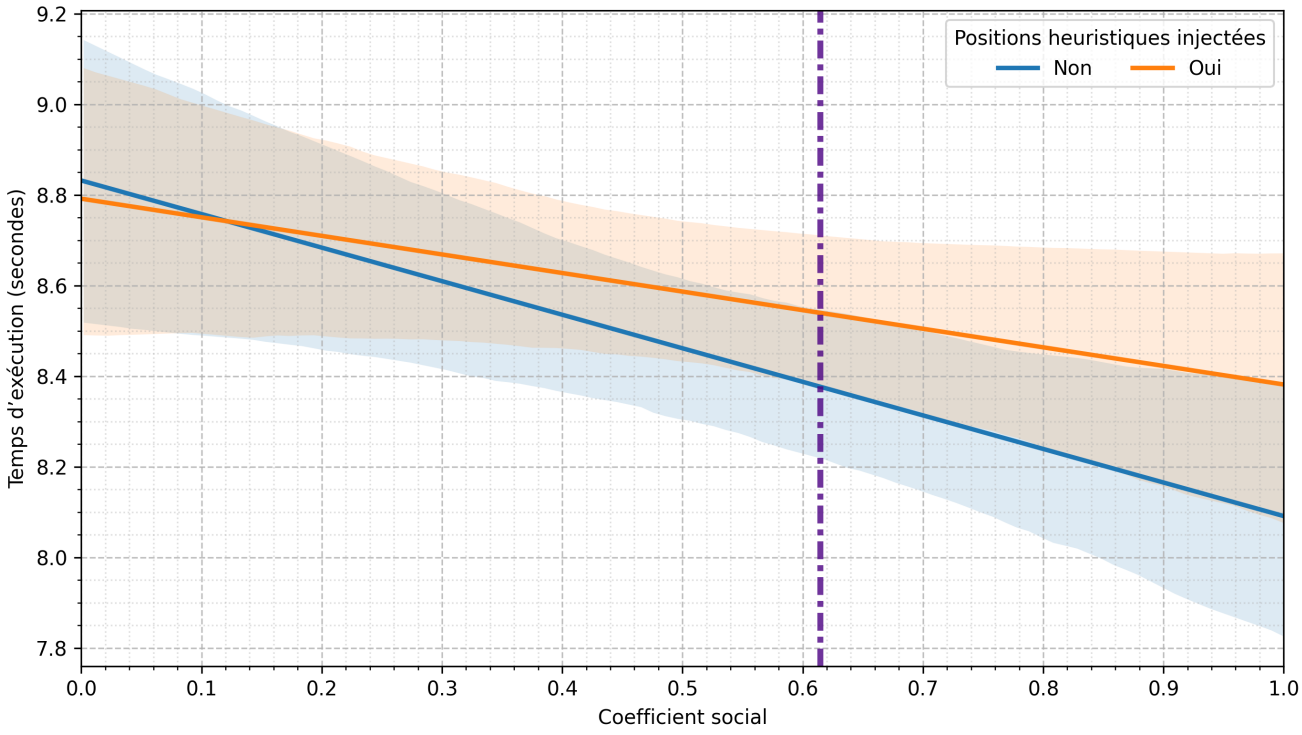

FIG. 22: Influence du coefficient social sur le temps d'exécution.

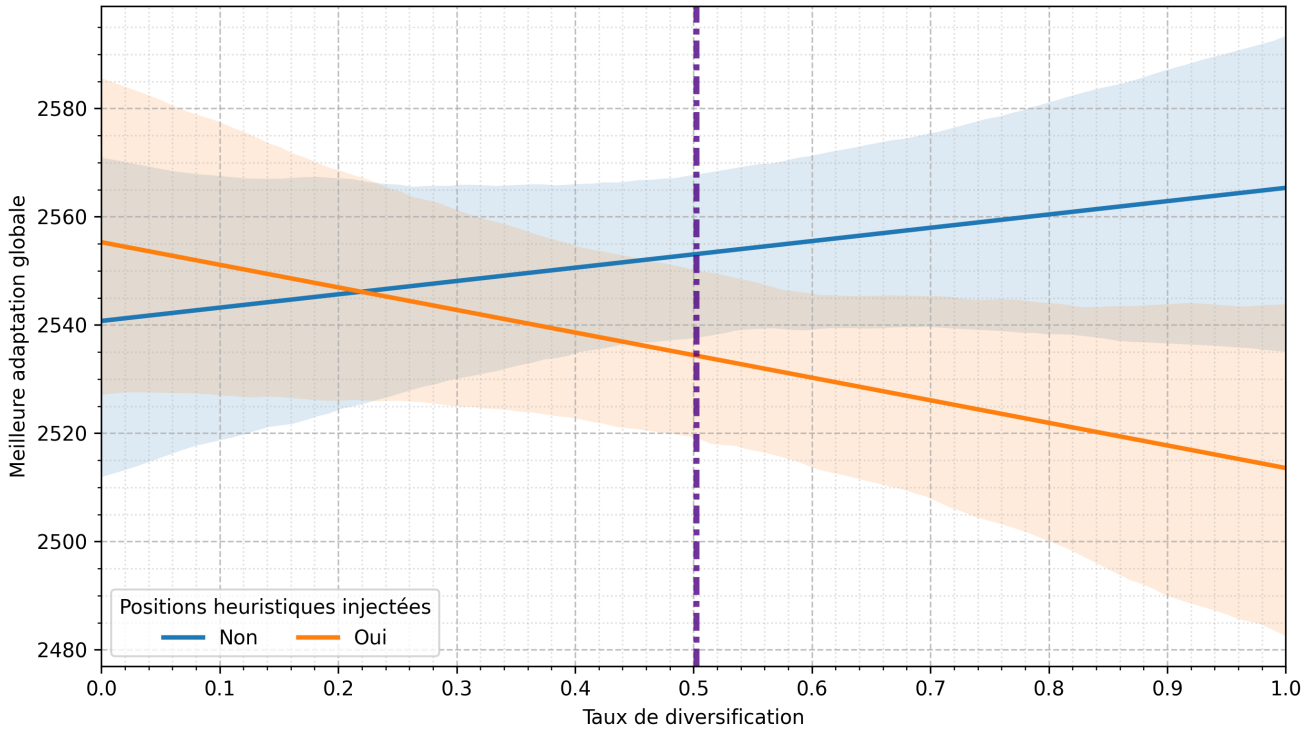

FIG. 23: Influence du taux de diversification sur la meilleure adaptation globale.

sur la base de leurs propres découvertes. À l'inverse, un coefficient social élevé encourage la convergence vers les solutions identifiées par les autres particules, favorisant ainsi l'exploitation des connaissances collectives de l'essaim. Les résultats montrent que l'augmentation de ces deux coefficients permet de réduire à la fois le makespan et le temps d'exécution (Figures 19–22). Ils suggèrent qu'une combinaison équilibrée de ces paramètres est susceptible de conduire à de meilleures performances. Cette combinaison optimale peut toutefois dépendre de la nature du problème et de la phase d'optimisation : une exploration plus poussée peut s'avérer bénéfique au début de la recherche, tandis qu'une exploitation plus intensive peut être préférable en phase finale.

Les figures 23–24 illustrent l'influence du taux de diversification sur la meilleure adaptation globale et le temps d'exécution. Les résultats indiquent qu'un taux de diversification modéré favorise une meilleure exploration de l'espace de solutions, ce qui permet d'améliorer la qualité des solutions obtenues. Cependant, cette exploration accrue s'accompagne d'une augmentation du temps de calcul. Par conséquent, il existe un compromis à trouver entre la qualité de la solution et le temps d'exécution, en ajustant le taux de diversification en fonction des exigences spécifiques du problème d'ordonnancement.

Les figures 25–26 illustrent l'influence du nombre maximal d'itérations autorisées sur la meilleure adaptation globale et le temps d'exécution. Les résultats montrent qu'augmenter le nombre d'itérations permet généralement d'améliorer la qualité de la solution, car l'algorithme a plus de temps pour explorer l'espace de recherche et converger vers de meilleures solutions.

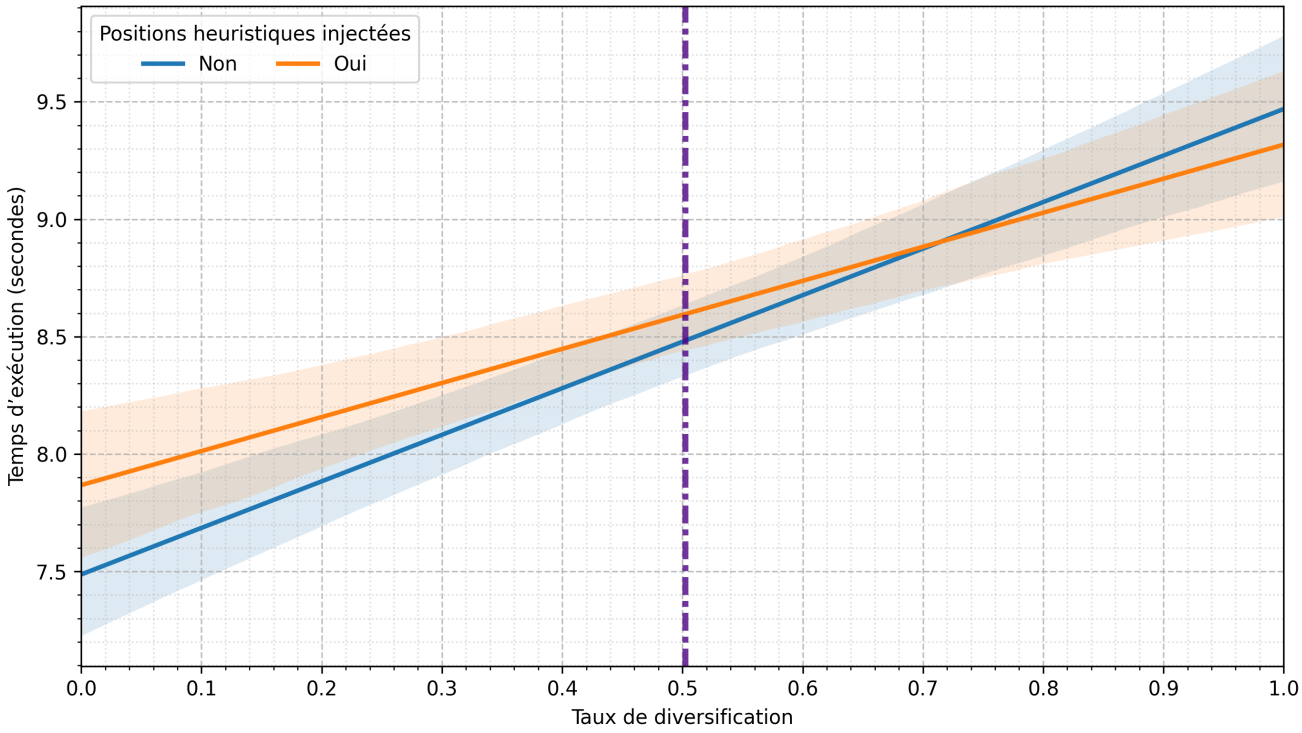

FIG. 24: Influence du taux de diversification sur le temps d'exécution.

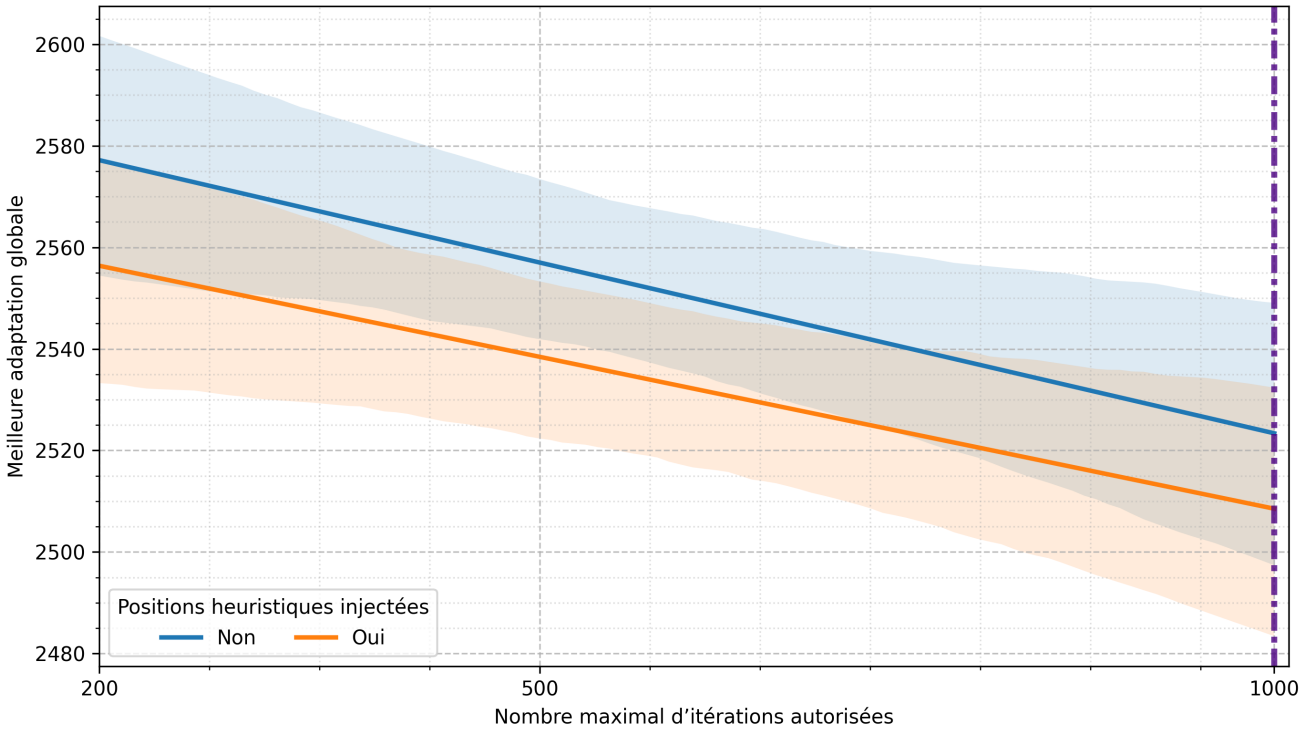

FIG. 25: Influence du nombre maximal d'itérations autorisées sur la meilleure adaptation globale.

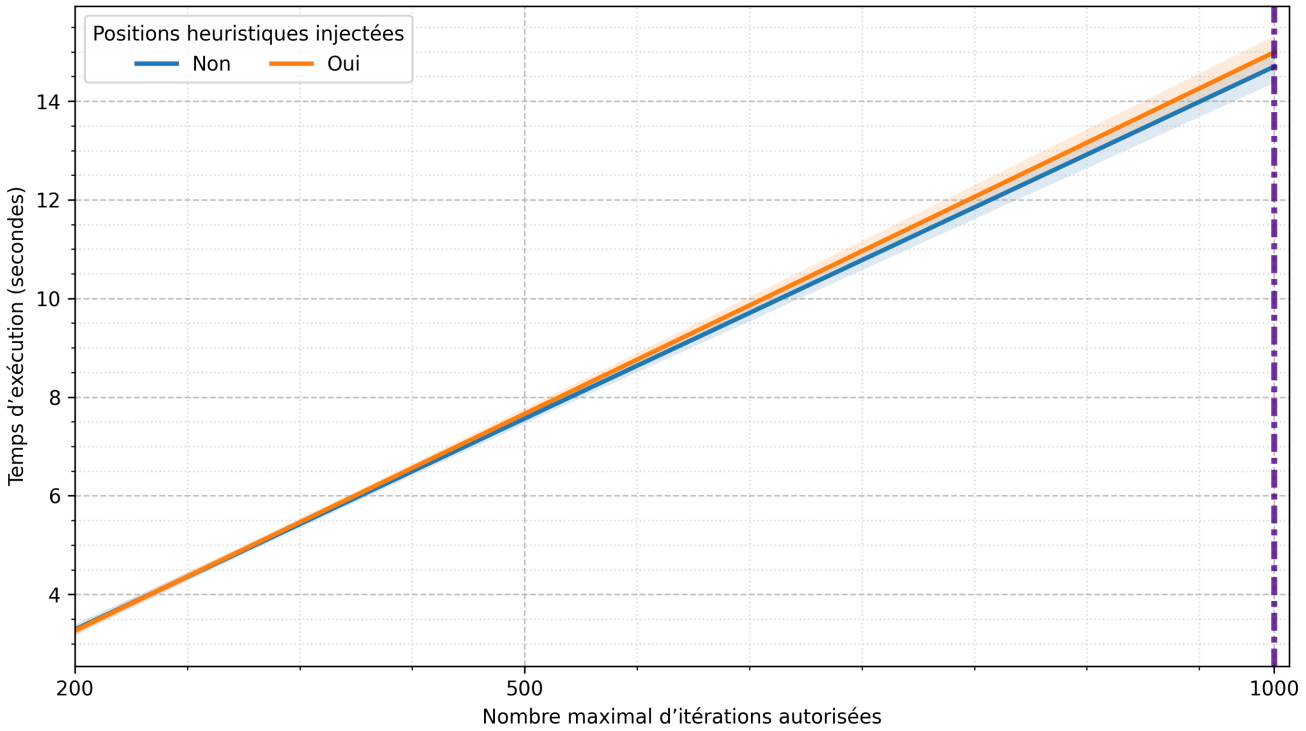

FIG. 26: Influence du nombre maximal d'itérations autorisées sur le temps d'exécution.

Cependant, cette amélioration de la qualité de la solution s'accompagne d'une augmentation du temps d'exécution. Par conséquent, le choix du nombre maximal d'itérations doit tenir compte du compromis entre la qualité de la solution souhaitée et les contraintes de temps de calcul.

L'analyse détaillée des figures 13–26 montre de façon indiscutable que l'injection des solutions heuristiques, notamment dans les situations de stagnation de l'algorithme, exerce un effet bénéfique significatif sur les performances globales de l'approche que nous proposons. Cet effet positif se manifeste très clairement lorsque nous observons les lignes de tendance des graphiques représentant l'influence des différents paramètres étudiés, notamment ceux liés à l'adaptation globale. Nous remarquons en effet que, dans la quasi-totalité des cas, les courbes correspondant aux configurations où les solutions heuristiques sont injectées se situent visiblement en dessous de celles obtenues sans injection, ce qui signifie que les meilleures adaptations globales atteintes avec heuristique surpassent systématiquement celles atteintes sans. Cette observation est particulièrement importante, car elle traduit non seulement une amélioration de la qualité des solutions finales, mais aussi une meilleure capacité de l'algorithme à sortir des zones stagnantes et à éviter les pièges des optimums locaux. En ce qui concerne le temps d'exécution, les figures montrent que les courbes sont souvent proches entre les deux approches, voire légèrement inférieures dans certains cas avec l'injection heuristique, ce qui indique que cette amélioration qualitative ne se fait pas au prix d'un cout computationnel accru. Bien au contraire, nous constatons même dans certaines configurations un gain léger en efficacité temporelle, ce qui démontre que l'injection des solutions heuristiques permet non seulement de guider plus efficacement la recherche,

mais aussi d'optimiser le rapport entre qualité et rapidité des résultats. Ainsi, les résultats illustrés confirment pleinement notre hypothèse selon laquelle l'utilisation des solutions heuristiques comme mécanisme de relance en cas de stagnation contribue à renforcer la robustesse et l'efficience globale du processus d'optimisation, en apportant simultanément des bénéfices en matière de performance, de stabilité et de cout en temps de calcul. Ces constats viennent donc conforter l'intérêt de l'intégration des solutions heuristiques au sein des métaheuristiques appliquées à l'ordonnancement, en mettant en évidence leur rôle clé dans l'amélioration globale des résultats obtenus.

Dans un volet complémentaire de cette étude, le tableau 8 présente une synthèse des performances obtenues par la technique développée, en appliquant les paramètres optimaux déterminés lors de la phase de tests. Ce tableau confronte les résultats de notre approche aux meilleures valeurs connues dans la littérature pour un ensemble représentatif d'instances du problème d'ordonnancement en ateliers à flux continu. Les paramètres retenus correspondent à ceux ayant donné les meilleures performances globales, comme indiqué par les lignes en pointillés dans les figures 13–26.

Chaque instance est définie par un nombre précis de tâches et de machines, et les résultats sont présentés sous la forme de deux colonnes : la colonne *Optimisation* indique le makespan obtenu à l'aide de l'optimisation par essaims de particules, tandis que la colonne *État de l'art* fournit le makespan de la solution optimale ou quasi optimale identifiée dans les travaux antérieurs. Globalement, les résultats montrent que l'algorithme parvient à atteindre, voire à égaler, les meilleures performances pour plusieurs instances de taille modérée (notamment les cas 20_5 et 50_5), témoignant ainsi de son efficacité et de sa robustesse. Toutefois, des écarts plus significatifs sont observés pour les instances plus complexes, notamment celles de grande dimension (50_10 et 50_20), suggérant des pistes d'amélioration.

Ces écarts peuvent s'expliquer par plusieurs facteurs. Tout d'abord, les paramètres utilisés sont fixés de manière uniforme pour l'ensemble des instances, sans ajustement spécifique selon la taille ou la complexité du problème. Une stratégie adaptative, capable de moduler dynamiquement ces paramètres en fonction des caractéristiques de chaque instance, aurait pu améliorer les performances. L'intégration de stratégies hybrides, de métaheuristiques coopératives, ou de mécanismes d'autoadaptation constitue ainsi des pistes pertinentes pour les travaux futurs, en vue d'accroitre la généricité et la performance de l'approche proposée.

L'analyse des figures 27–28 nous permet de mieux comprendre le comportement dynamique de l'algorithme d'optimisation par essaim de particules appliqué à l'ordonnancement en ateliers à flux continu. La ligne en pointillés figurant sur les deux graphiques marque l'itération correspondant à la meilleure solution obtenue. Ainsi, la figure 27 montre l'évolution de la longueur moyenne des vélocités au sein de l'essaim au fil des itérations. Nous observons une réduction graduelle des vélocités, accompagnée de certaines fluctuations : au début, cette réduction semble lente, mais elle s'accélère progressivement au fil des itérations. Ce comportement reflète parfaitement le mé-

Tab. 8: Résultats obtenus en utilisant les paramètres extraits des tests.

Instance	**Optimisation**	**État de l'art**	**Écart**
20_5_1	1278	1278	0
20_5_2	1359	1359	0
20_5_3	1088	1081	7
20_5_4	1293	1293	0
20_5_5	1235	1236	-1
20_10_1	1583	1582	1
20_10_2	1664	1659	5
20_10_3	1511	1496	15
20_10_4	1384	1378	6
20_10_5	1420	1419	1
20_20_1	2312	2297	14
20_20_2	2111	2100	11
20_20_3	2350	2326	24
20_20_4	2234	2223	11
20_20_5	2303	2291	12
50_5_1	2724	2724	0
50_5_2	2838	2834	4
50_5_3	2621	2621	0
50_5_4	2753	2751	2
50_5_5	2863	2863	0
50_10_1	3082	3025	57
50_10_2	2976	2892	84
50_10_3	2917	2864	53
50_10_4	3113	3064	49
50_10_5	3060	2986	74
50_20_1	3978	3875	103
50_20_2	3827	3715	112
50_20_3	3802	3668	134
50_20_4	3851	3752	99
50_20_5	3772	3635	137

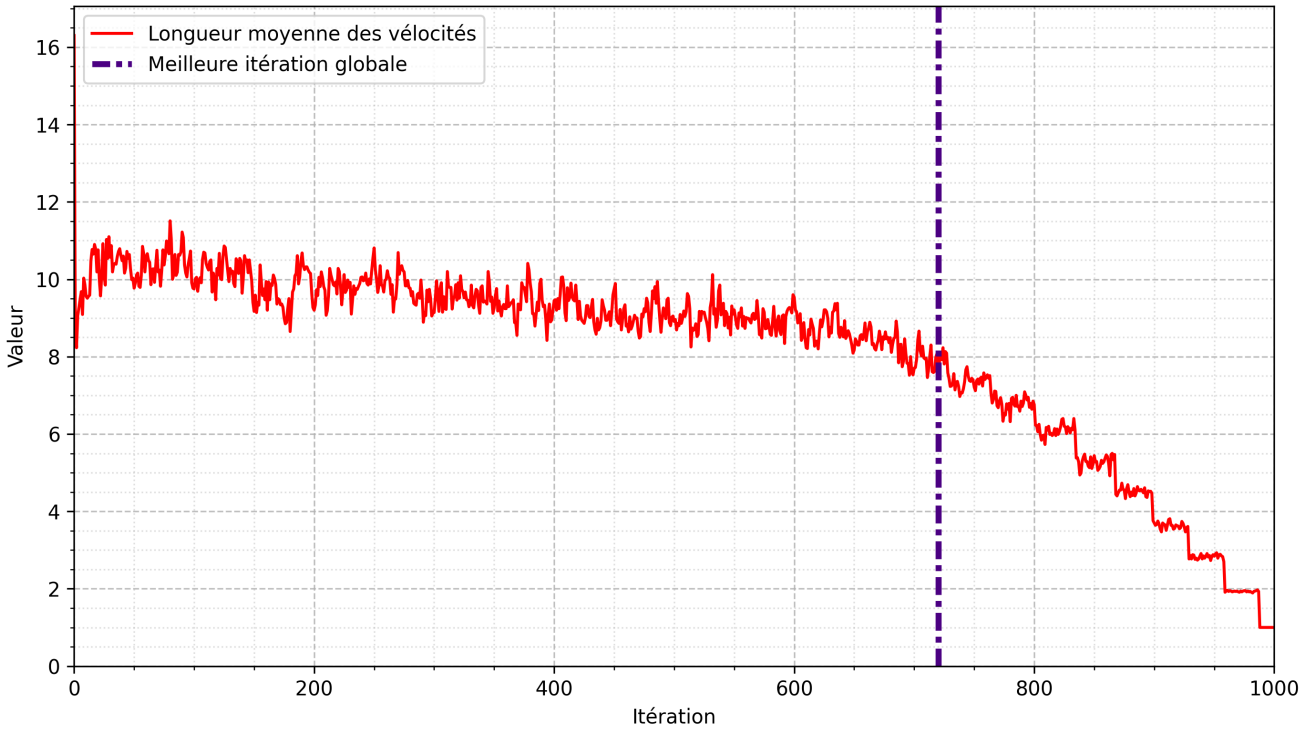

FIG. 27: Évolution de la longueur moyenne des vélocités des particules de l'essaim.

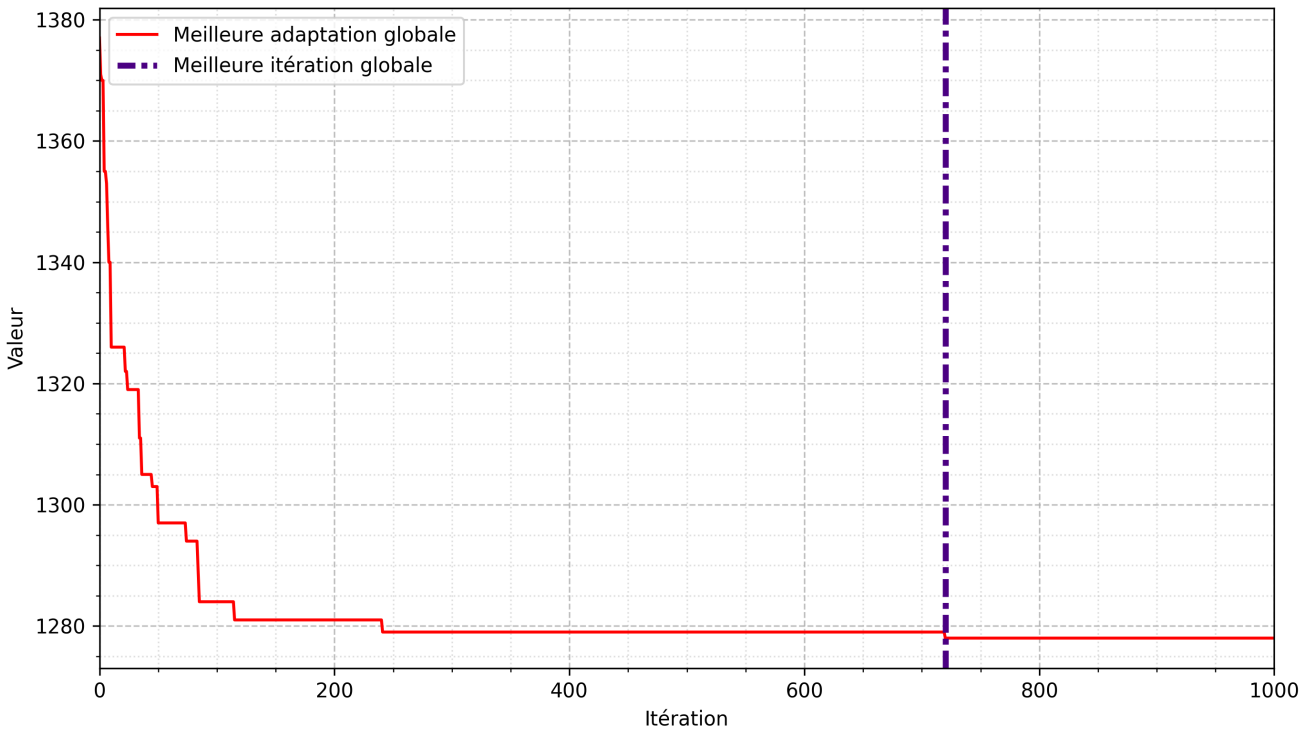

FIG. 28: Évolution de la meilleure adaptation globale de l'essaim.

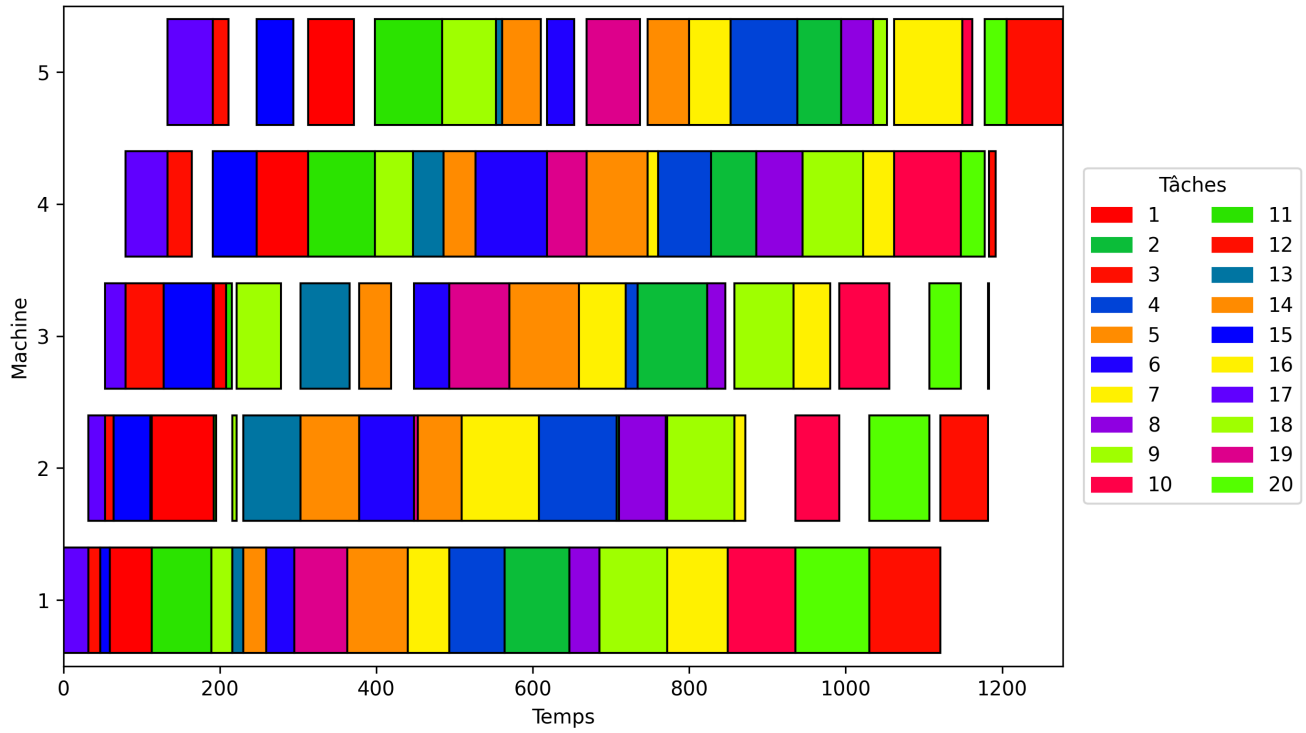

FIG. 29: Diagramme de Gantt associé au planning trouvé pour l'instance 20_5_1.

canisme implémenté, qui donne aux particules une certaine liberté de déplacement au début de l'optimisation afin de garantir une exploration large et efficace de l'espace des solutions, avant de restreindre progressivement cette liberté en autorisant uniquement des mouvements courts, encourageant ainsi l'exploitation fine des meilleures zones identifiées. Les fluctuations observées traduisent un compromis essentiel entre exploration et exploitation : bien que la tendance générale soit à la réduction, des ajustements ponctuels (dus aux forces cognitives et sociales de l'algorithme ou aux mécanismes de diversification) permettent à l'essaim d'éviter de se figer trop tôt dans des optimums locaux. La figure 28, qui trace l'évolution de la meilleure adaptation globale, illustre également cette double dynamique : les gains de performance sont notables au départ, traduisant une phase exploratoire efficace, puis tendent à se stabiliser, indiquant une exploitation intensive des solutions prometteuses déjà découvertes. Ces observations soulignent l'importance d'un calibrage précis des paramètres guidant la transition entre exploration et exploitation, et suggèrent l'intérêt de stratégies adaptatives capables de relancer l'exploration en cas de stagnation.

Le diagramme de Gantt associé au planning trouvé pour l'instance 20_5_1 représente visuellement la répartition des tâches sur les cinq machines au fil du temps (Figures 29). Chaque barre horizontale illustre l'intervalle temporel d'exécution d'une tâche spécifique sur une machine donnée. En l'analysant, nous constatons une bonne occupation des machines, avec peu de temps morts, ce qui indique une utilisation efficace des ressources. Nous repérons également clairement les séquences critiques, c'est-à-dire les chaines de tâches dont le décalage retarderait l'ensemble

du planning. Cette visualisation nous permet d'identifier les goulets d'étranglement, notamment aux étapes où plusieurs tâches attendent le passage sur une même machine. De plus, elle met en lumière l'équilibrage des charges : certaines machines sont sollicitées presque en continu, tandis que d'autres présentent de brèves périodes d'inactivité.

Au-delà de la simple performance numérique, cette représentation graphique joue un rôle essentiel pour évaluer la qualité structurelle des plannings générés, notamment en matière d'équilibrage des charges, de fluidité des enchainements et de robustesse face aux aléas. Elle nous permet de visualiser des aspects souvent invisibles dans les seules métriques chiffrées, tels que la répartition harmonieuse des tâches entre les machines, l'identification des goulets d'étranglement et la capacité du système à amortir d'éventuelles perturbations (retards, pannes ou réajustements de dernière minute). En croisant ces observations qualitatives avec les indicateurs quantitatifs issus de l'évolution des métriques d'adaptation (par exemple, le makespan ou les taux d'inactivité), nous disposons d'une vue plus complète et nuancée, indispensable pour valider et affiner les solutions proposées. Par ailleurs, cette approche intégrée ouvre des perspectives prometteuses pour enrichir les objectifs d'optimisation à l'avenir, en y intégrant des critères plus complexes et multidimensionnels, tels que l'impact énergétique, les couts opérationnels, la résilience face aux aléas ou encore la satisfaction des contraintes environnementales et sociales. Ces extensions permettraient de rapprocher davantage les modèles utilisés des réalités industrielles, en les alignant plus finement avec les priorités stratégiques des entreprises et les défis de durabilité et de compétitivité.

Les analyses menées dans ce chapitre ont permis de mieux comprendre l'impact des paramètres de l'optimisation par essaims de particules sur les performances en matière de makespan et de temps de calcul, en s'appuyant sur un ensemble consistant d'instances et de configurations expérimentales. Nous avons mis en évidence que certains paramètres, tels que la taille de l'essaim et le nombre d'itérations, jouent un rôle prépondérant, tandis que d'autres présentent une influence plus subtile qui pourrait être explorée par des analyses plus fines. La comparaison avec les résultats de la littérature montre que, sans chercher à dépasser les meilleures solutions connues, notre méthode offre un compromis intéressant entre simplicité, adaptabilité et qualité des solutions. Enfin, le développement d'une interface graphique interactive complète l'approche en offrant un outil pratique et accessible pour appliquer ces méthodes à des problèmes concrets. Ces résultats ouvrent la voie à plusieurs perspectives, notamment l'exploration de variantes hybrides, l'optimisation multiobjective ou encore l'adaptation dynamique des paramètres au cours de l'optimisation.

Chapitre 6

Conclusion

Ce travail a montré la pertinence et la flexibilité de l'application de l'optimisation par essaims de particules à l'ordonnancement en ateliers à flux continu. En dépit des résultats positifs obtenus, de nombreuses opportunités d'amélioration et d'élargissement subsistent, que ce soit sur le plan méthodologique ou pratique. Ce chapitre conclut ainsi le mémoire en rappelant les principaux résultats obtenus et en proposant plusieurs pistes de recherche futures afin de mieux cerner la portée et les perspectives de cette étude. Nous espérons que ces résultats contribueront à enrichir ce domaine et à offrir des solutions pratiques et efficaces aux défis industriels actuels et à venir.

6.1 Principaux résultats

Ce mémoire a exploré le problème d'ordonnancement en ateliers à flux continu, un défi majeur dans la gestion de la production industrielle. Ce problème, qui consiste à déterminer la séquence optimale de traitement des tâches sur une série de machines afin de minimiser une fonction objectif donnée, souvent le temps de fabrication total, est connu pour être NP-difficile, ce qui signifie qu'il n'existe pas d'algorithme polynomial capable de le résoudre de manière exacte pour des instances de grande taille. Face à cette complexité, les méthodes heuristiques et métaheuristiques, qui fournissent des solutions approchées en un temps de calcul raisonnable, sont devenues des outils incontournables.

Dans ce travail, nous avons proposé une approche basée sur l'optimisation par essaims de particules, une métaheuristique inspirée du comportement social des oiseaux ou des poissons à la recherche de nourriture. Cette métaheuristique, initialement conçue pour l'optimisation de problèmes à variables continues, a été adaptée ici aux spécificités combinatoires de l'ordonnancement en ateliers à flux continu. Cette adaptation a nécessité la définition d'une représentation appropriée des solutions, une permutation des tâches, et la conception d'opérateurs de déplacement et de mise à jour des particules qui tiennent compte des contraintes du problème. Les résultats expérimentaux ont démontré l'efficacité de cette approche. L'algorithme proposé rivalise avec les meilleures méthodes de l'état de l'art sur des instances de taille modérée, tout en

offrant une bonne performance sur des instances plus complexes, pour lesquelles les méthodes exactes deviennent rapidement impraticables.

Afin d'évaluer notre approche, nous avons mis en œuvre et testé l'algorithme proposé sur un ensemble de données de référence standardisées, et nous avons comparé ses performances avec celles d'autres algorithmes de l'état de l'art. Cette évaluation expérimentale a permis de valider l'efficacité de notre approche et de la situer par rapport aux méthodes existantes. Nous avons utilisé des instances de différentes tailles et difficultés pour évaluer la robustesse et la scalabilité de l'algorithme. Les résultats expérimentaux montrent que l'algorithme proposé est compétitif avec les meilleures méthodes existantes, en particulier pour les instances de petite et moyenne taille. Pour les instances plus grandes, l'algorithme offre également de bonnes performances, bien qu'il puisse être surpassé par certains algorithmes spécialisés. Cela montre le potentiel de l'optimisation par essaims de particules adaptée pour résoudre l'ordonnancement en ateliers à flux continu dans une variété de contextes.

6.2 Améliorations et perspectives

Cette recherche ouvre plusieurs perspectives pour les travaux futurs. Il serait intéressant d'étudier les pistes d'amélioration suivantes :

- **Renforcement de la métaheuristique de base :** l'un des axes d'amélioration consiste à enrichir la métaheuristique initiale en intégrant des mécanismes supplémentaires de gestion de la stagnation afin d'éviter le piégeage prématuré dans des optimums locaux. Le contrôle dynamique du nombre de particules pourrait également être exploré, permettant d'ajuster la taille de l'essaim en fonction de la progression de la recherche pour un meilleur équilibre entre exploration et exploitation. De plus, la parallélisation des calculs et la distribution des traitements au sein de l'essaim peuvent améliorer l'efficacité computationnelle, notamment pour des instances de grande taille. Enfin, des hybridations avec d'autres heuristiques ou métaheuristiques (par exemple, recherche locale, algorithmes génétiques) pourraient être mises en œuvre pour renforcer la robustesse et la performance globale de l'algorithme.

- **Intégration de l'apprentissage automatique :** une autre piste prometteuse réside dans l'incorporation de techniques d'apprentissage automatique pour améliorer les performances de la métaheuristique. Cela pourrait inclure la mise en place d'un système de réglage automatique des paramètres clés de l'algorithme à travers une phase d'apprentissage préalable, permettant d'adapter dynamiquement les paramètres en fonction des caractéristiques spécifiques des instances traitées. L'apprentissage pourrait également être utilisé pour guider l'évolution de la recherche en identifiant des schémas récurrents dans les trajectoires des particules ou en adaptant les stratégies de mise à jour selon l'état courant de l'optimisation. De plus, des techniques prédictives pourraient être exploitées pour anticiper la qualité

des solutions partielles et orienter l'exploration vers les régions les plus prometteuses de l'espace des solutions.

- **Améliorations méthodologiques :** il serait également pertinent de renforcer la méthodologie expérimentale afin d'obtenir une évaluation plus exhaustive et rigoureuse de l'algorithme. Cela passe par l'augmentation de la taille et de la diversité des ensembles de données de test pour mieux refléter les conditions variées rencontrées en pratique. L'introduction de métriques complémentaires au makespan, telles que le temps moyen de passage ou encore le taux de respect des délais, permettrait d'apporter une vision plus complète des performances. Des tests paramétriques plus poussés devraient être menés pour mieux comprendre l'influence de chaque paramètre sur le comportement de l'algorithme. Enfin, l'intégration d'analyses statistiques plus avancées, telles que le calcul des p-values, des intervalles de confiance, ou encore l'utilisation de tests ANOVA, renforcerait la validité des conclusions tirées des expérimentations.

- **Amélioration de la visualisation et de l'interprétabilité :** pour faciliter l'adoption pratique de l'algorithme, il serait utile de développer des outils de visualisation avancés permettant de suivre en temps réel l'évolution de l'optimisation, la répartition des particules dans l'espace des solutions, ainsi que la convergence vers les meilleures solutions. De plus, la mise en place de rapports visuels clairs et interactifs, incluant des diagrammes de Gantt ou des graphes dynamiques d'ordonnancement, permettrait aux utilisateurs de mieux comprendre et valider les résultats. Enfin, des mécanismes visant à expliquer les décisions clés de l'algorithme contribueraient à renforcer la confiance des utilisateurs industriels dans son fonctionnement et ses performances.

- **Applications pratiques :** une direction importante pour la suite des travaux consiste à rechercher des applications concrètes en collaborant avec des entreprises industrielles et des usines afin d'identifier des cas d'usage réels. Un tel partenariat permettrait de mieux comprendre les contraintes spécifiques, les priorités opérationnelles et les besoins réels des acteurs du secteur. Ces collaborations offriraient également l'opportunité de tester et d'ajuster l'algorithme dans des environnements de production authentiques, ce qui permettrait de valider son applicabilité sur le terrain. Par ailleurs, cette démarche pourrait ouvrir la voie à l'adaptation de l'approche à des variantes du problème standard, incluant par exemple des contraintes spécifiques aux ressources humaines, des exigences de qualité ou des délais stricts de livraison.

- **Extension vers des problématiques adjacentes :** une autre piste intéressante consiste à étendre cette approche à des problématiques connexes en optimisation combinatoire. Cela pourrait inclure des variantes plus complexes de l'ordonnancement, telles que les ateliers flexibles, les ateliers avec machines parallèles ou les ateliers avec contraintes de maintenance et de disponibilité variable. Par ailleurs, l'algorithme pourrait être adapté à d'autres

domaines industriels présentant des similarités, comme la planification des tournées de véhicules, l'affectation de ressources, ou la gestion des chaines logistiques. Ces extensions permettraient d'explorer la polyvalence de l'approche et d'évaluer son efficacité dans des contextes différents, mais structurellement proches.

Références

K. R. Baker and D. Trietsch. *Principles of Sequencing and Scheduling*. Wiley, 2nd edition, 2019. doi:10.1002/9781119262602.

J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, M. Sterna, and J. Weglarz. *Handbook on Scheduling: From Theory to Practice*. Springer Cham, 2nd edition, 2019. doi:10.1007/978-3-319-99849-7.

M. Bóna. *Combinatorics of Permutations*. Chapman & Hall/CRC, 3rd edition, 2022. doi:10.1201/9780429274107.

M. Bóna. *Introduction to Enumerative and Analytic Combinatorics*. Chapman & Hall/CRC, 3rd edition, 2025. doi:10.1201/9781003304272.

H. G. Campbell, R. A. Dudek, and M. L. Smith. A heuristic algorithm for the n job, m machine sequencing problem. *Management Science*, 16(10):B:579–697, 1970. doi:10.1287/mnsc.16.10.b630.

D. G. Dannenbring. An evaluation of flow shop sequencing heuristics. *Management Science*, 23(11):1149–1259, 1977. doi:10.1287/mnsc.23.11.1174.

M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, 1999a.

M. Dorigo and G. Di Caro. Ant colony optimization: A new meta-heuristic. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC 1999), Washington, DC, United States of America, July 6–9, 1999*, volume 2, pages 1470–1477, 1999b. doi:10.1109/cec.1999.782657.

M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999. doi:10.1162/106454699568728.

R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science (MHS 1995), Nagoya, Japan, October 4–6, 1995*, pages 39–43, 1995. doi:10.1109/mhs.1995.494215.

H. Emmons and G. Vairaktarakis. *Flow Shop Scheduling: Theoretical Results, Algorithms, and Applications*. Springer New York, 2013. doi:10.1007/978-1-4614-5152-5.

T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989. doi:10.1016/0167-6377(89)90002-3.

H. L. Gantt. *Organizing for Work*. Harcourt, Brace and Howe, 1919.

M. Gendreau and J.-Y. Potvin, editors. *Handbook of Metaheuristics*. Springer Cham, 3rd edition, 2019. doi:10.1007/978-3-319-91086-4.

F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977. doi:10.1111/j.1540-5915.1977.tb01074.x.

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986. doi:10.1016/0305-0548(86)90048-1.

J. N. D. Gupta. A functional heuristic algorithm for the flowshop scheduling problem. *Operational Research Quarterly*, 22(1):39–47, 1971. doi:10.1057/jors.1971.18.

J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, 2nd edition, 1992. doi:10.7551/mitpress/1090.001.0001.

S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954. doi:10.1002/nav.3800010110.

J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks (ICNN 1995), Perth, Western Australia, Australia, November 27–December 1, 1995*, volume 4, pages 1942–1948, 1995. doi:10.1109/icnn.1995.488968.

J. Kennedy and R. C. Eberhart. A discrete binary version of the particle swarm algorithm. In *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation (ICSMC 1997), Orlando, Florida, United States of America, October 12–15, 1997*, volume 5, pages 4104–4108, 1997. doi:10.1109/icsmc.1997.637339.

S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. doi:10.1126/science.220.4598.671.

J. Li, B. Zhang, and D. Zhang. *Information Fusion: Machine Learning Methods*. Springer Singapore, 2022. doi:10.1007/978-981-16-8976-5.

R. Martí, P. M. Pardalos, and M. G. C. Resende, editors. *Handbook of Heuristics*. Springer Cham, 2018. doi:10.1007/978-3-319-07124-4.

N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997. doi:10.1016/s0305-0548(97)00031-2.

M. Nawaz, E. E. Enscore, Jr, and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983. doi:10.1016/0305-0483(83)90088-9.

D. S. Palmer. Sequencing jobs through a multi-stage process in the minimum total time—A quick method of obtaining a near optimum. *Operational Research Quarterly*, 16(1):101–107, 1965. doi:10.1057/jors.1965.8.

M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Cham, 6th edition, 2022. doi:10.1007/978-3-031-05921-6.

J. R. Raol. *Data Fusion Mathematics: Theory and Practice*. CRC Press, 2015. doi:10.1201/b18736.

P. Siarry, editor. *Metaheuristics*. Springer Cham, 2016. doi:10.1007/978-3-319-45403-0.

S. M. A. Suliman. A two-phase heuristic approach to the permutation flow-shop scheduling problem. *International Journal of Production Economics*, 64(1–3):143–152, 2000. doi:10.1016/s0925-5273(99)00053-5.

É. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993. doi:10.1016/0377-2217(93)90182-m.

E.-G. Talbi. *Metaheuristics: From Design to Implementation*. Wiley, 2009. doi:10.1002/9780470496916.

D. H. Wolpert and W. G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997. doi:10.1109/4235.585893.

M. Zambrano-Bigiarini, M. Clerc, and R. Rojas. Standard particle swarm optimisation 2011 at CEC-2013: A baseline for future PSO improvements. In *Proceedings of the 2013 IEEE Congress on Evolutionary Computation (CEC 2013), Cancun, Mexico, June 20–23, 2013*, pages 2337–2344, 2013. doi:10.1109/cec.2013.6557848.