MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOHAMED BOUDIAF - M'SILA

FACULTE DES MATHEMATIQUES ET DE L'INFORMATIQUE

DÉPARTEMENT D'INFORMATIQUE

MEMOIRE de fin d'étude

Présenté pour l'obtention du diplôme de MASTER
Domaine : Mathématiques et Informatique
Filière : Informatique
Spécialité : Informatique Décisionnelle et Optimisation

Par : TORKI Karima et MOKHTARI Cheyma

SUJET

Algorithmes génétiques et Recherche Cuckoo pour un problème d'ordonnancement d'atelier Job Shop flexible

Soutenu publiquement le : / /2026 devant le jury composé de :

…… Université de M'silaPrésident

Dr.MouhoubNasser Eddine Université de M'silaRapporteur

…… Université de M'silaExaminateur

Promotion : 2025 /2026

Dédicace

Ô Allah, à Toi la louange, la louange des reconnaissants et de ceux qui T'évoquent, une louange remplissant les cieux et la terre.

À mon premier soutien dans mon parcours, mon appui, ma force et mon refuge après Allah, ma fierté et mon honneur : mon père.

À la lumière de mes yeux, à celle qui illumine mon chemin et embellit ma vie, à celle dont le cœur m'a portée avant même ses bras, et qui a facilité mes épreuves par ses invocations ; à la source de ma force et de ma réussite : ma mère.

À ceux grâce à qui je grandis, sur qui je peux compter, et dont la présence me donne de la force : mes frères (Ayoub, Sameh, Yacine, Abdelhakim et Abderraouf).

À celles dont les paroles bienveillantes et le soutien sincère ont été une source de réconfort et d'encouragement : les épouses de mes frères (Amel et Khouloud).

À la joie de mon cœur, aux beaux sourires qui allégeaient ma fatigue, et à ces petits cœurs remplis d'innocence et de lumière familiale : Ghaïth et Zine.

À celui qui a été le meilleur guide, conseiller, source de confiance et motivation pour persévérer et travailler avec sérieux : mon encadreur (Nasser Eddine Mouhoub).

À la compagne de ce beau parcours, dont la présence a été un véritable soutien : Chaïma.

À celle qui, après Allah, a eu le mérite de me tendre la main ; à ce cœur généreux qui ne m'a jamais privée de son soutien et de ses encouragements : Lamia.

Torki Karima

Dédicace

Je dédie le fruit de mes efforts à l'âme de mon cher père, qu'Allah lui fasse miséricorde. J'aurais tant aimé qu'il soit présent pour être fier de moi. Qu'Allah lui accorde Sa miséricorde et l'accueille dans Son vaste paradis.

À ma chère mère, merci pour ta patience, tes prières et ta tendresse qui ont toujours été une source de force pour moi.

À mon mari, qui a été un véritable soutien tout au long de ce parcours, présent à mes côtés dans les moments de fatigue et de difficulté, et dont l'appui et les encouragements ont grandement contribué à l'aboutissement de ce travail.

À mes sœurs et à mon frère, merci pour vos encouragements et votre soutien constant, et particulièrement à ma sœur Sabah, qui a toujours pris soin de moi avec amour et attention. Que vous restiez toujours mon soutien dans la vie.

Je dédie également ce travail à ma belle-famille, en reconnaissance de leur affection, de leur soutien et de leur bienveillance à mon égard.

J'adresse aussi mes sincères remerciements et toute ma gratitude à toutes les personnes qui m'ont aidée et accompagnée dans la réalisation de ce travail, qui m'ont apporté leur aide, leurs conseils et leurs précieuses orientations ayant largement contribué à l'achèvement de ce mémoire.

Enfin, à tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail, je vous dédie cette réussite avec toute ma reconnaissance et mon affection.

.

MokhtariCheyma

TABLE DES MATIÈRES

LISTES DES FIGURES

LISTES DES ALGORITHMES

LISTES DES TABLEAUX

LES ACRONYMES

AG: Algorithme Génétique.

FJSP: Flexible Job Shop.

RC: Recherche Cuckoo.

C_{max}**:** Makespan**.**

RS : Le Recuit Simulé.

SI: Swarm Intelligence.

EA : Evolutionary Algorithms**.**

RT : Recherche Tabou.

INTRODUCTION GÉNÉRALE

Dans le contexte industriel contemporain, marqué par une compétitivité accrue et des exigences de personnalisation de masse, la gestion des flux de production est devenue un pilier stratégique pour la survie des entreprises. Au cœur de cette gestion se trouve l'ordonnancement, une fonction décisionnelle critique qui consiste à organiser l'exécution de tâches sur des ressources limitées au fil du temps, afin d'optimiser un ou plusieurs critères de performance.

Historiquement, les ateliers de production rigides ont cédé la place à des systèmes de fabrication flexibles pour faire face aux fluctuations du marché. Cette évolution a donné naissance au problème d'ordonnancement d'atelier flexible (Flexible Job Shop SchedulingProblem - FJSP). Contrairement au Job Shop classique où chaque opération est liée à une unique machine, le FJSP introduit une flexibilité où une opération peut être traitée sur plusieurs machines alternatives, potentiellement avec des temps de traitement distincts.

Le FJSP appartient à la classe des problèmes combinatoires fortement NP-difficiles. Sa complexité réside dans l'obligation de résoudre simultanément deux sous-problèmes interconnectés : le sous-problème d'affectation (attribuer chaque opération à une machine compatible) et le sous-problème de séquencement (déterminer l'ordre chronologique de passage des opérations sur chaque machine). L'objectif visé dans ce travail est la minimisation du temps total de complétion (Makespan- C_{max}), un indicateur clé pour maximiser la productivité globale de l'atelier.

Les méthodes de résolution exactes et déterministes se révèlent inefficaces pour résoudre des instances industrielles de grande taille dans des temps de calcul acceptables. C'est pourquoi l'utilisation des métaheuristiques stochastiques s'avère indispensable. Dans le cadre de ce mémoire, nous nous intéressons à deux approches distinctes : l'Algorithme Génétique (AG), une méthode évolutionnaire robuste basée sur la sélection naturelle, et l'Algorithme de Recherche Cuckoo (CuckooSearch - CS), une métaheuristiques bio-inspirée récente fondée sur le parasitisme de couvée et les vols de Lévy.

Dès lors, la problématique centrale de notre recherche peut être formulée ainsi :

Comment concevoir, modéliser et implémenter un système d'aide à la décision basé sur l'Algorithme Génétique et la Recherche Cuckoo pour résoudre efficacement le problème du FJSP, et laquelle de ces deux approches offre les meilleures performances en termes qualité de solution C_{max} et de temps de convergence?

Pour répondre à cette problématique, notre travail est structuré en quatre chapitres complémentaires :

En le premier chapitre pose le cadre théorique en introduisant les concepts fondamentaux de l'ordonnancement, la classification des ateliers et la modélisation mathématique formelle du FJSP ainsi que ses contraintes industrielles.

En le deuxième chapitre dédié à l'étude des deux paradigmes de résolution, ce chapitre explore les fondements biologiques, les principes de fonctionnement et les équilibres entre exploration et exploitation propres à l'AG et au CS.

Le troisième chapitre détaille l'architecture conceptuelle proposée. Il présente le mode de codage dual (séquencement et affectation), la fonction d'évaluation de l'adaptation (Fitness) et l'adaptation discrète des opérateurs stochastiques pour le FJSP.

Le dernier chapitre constitue le volet pratique. Il décrit l'environnement de développement, présente de manière détaillée l'interface utilisateur graphique (GUI) conçue pour piloter le système, et expose l'analyse comparative des résultats expérimentaux illustrés par des diagrammes de Gantt.

CHAPITRE I : *ÉTAT DE L'ART SUR L'ORDONNANCEMENT ET LE FJSSP*

CHAPITRE I ÉTAT DE L'ART SUR L'ORDONNANCEMENT ET LE FJSSP

1. Introduction

L'ordonnancement de la production estunefonctiondécisionnelleclé, visant à optimiserl'allocation des ressourcestemporelles et matérielles au sein des ateliers. L'objet de ce premier chapitreest de poser les fondationsthéoriques indispensables à notreétude.

2. Introduction à l'ordonnancement de la production

L'ordonnancement est une fonction pivot dans la hiérarchie de la gestion de production. Il se définit comme l'art d'organiser la réalisation de tâches sur des ressources limitées dans le but d'optimiser un ou plusieurs critères de performance. Cette étape intervient juste après la planification et avant l'exécution réelle sur le plancher de l'usine. Dans un environnement industriel moderne, l'ordonnancement ne se limite pas à une simple gestion du temps, mais devient un levier stratégique pour minimiser les coûts de production, réduire les encours et respecter les délais contractuels envers les clients. Historiquement, le besoin d'ordonnancement est né avec la révolution industrielle, mais sa complexité a crû de manière exponentielle avec l'introduction de systèmes de production automatisés et flexibles. L'ordonnancement est le processus de prise de décision qui traite de l'allocation des ressources aux tâches sur une période donnée, dont le but est d'optimiser un ou plusieurs objectifs[1].

Au sein du tissu industriel algérien, l'optimisation des calendriers de production est devenue une priorité pour les gestionnaires afin d'améliorer la réactivité face aux fluctuations du marché local et international [2].

3. Classification des problèmesd'ordonnancement

La littérature scientifique distingue plusieurs types d'ateliers en fonction du flux des travaux et de la disposition des machines. Cette classification est essentielle pour choisir la méthode de résolution adéquate.

3.1. Flow Shop

L'atelier de type Flow Shop est caractérisé par un flux linéaire et unidirectionnel. Tous les travaux suivent exactement la même gamme opératoire, passant par les machines dans le même ordre séquentiel. Bien que ce modèle soit efficace pour la production de masse, il manque cruellement de flexibilité[3].

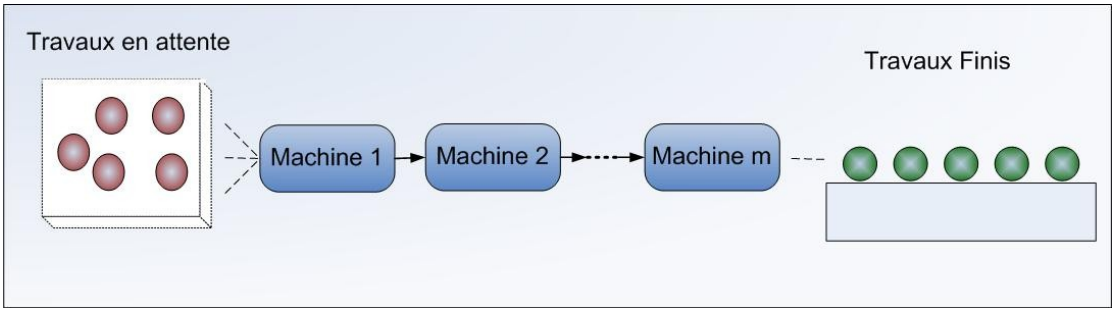


Figure 1: Exemple de Flow Shoptiré de [20].

3.2. Job Shop

Dans un environnement Job Shop, la diversité des produits est plus grande. Chaque travail possède sa propre gamme opératoire unique, ce qui signifie que les travaux peuvent circuler dans l'atelier selon des itinéraires différents. Cependant, dans le modèle classique, chaque opération est strictement liée à une seule machine spécifique[4].

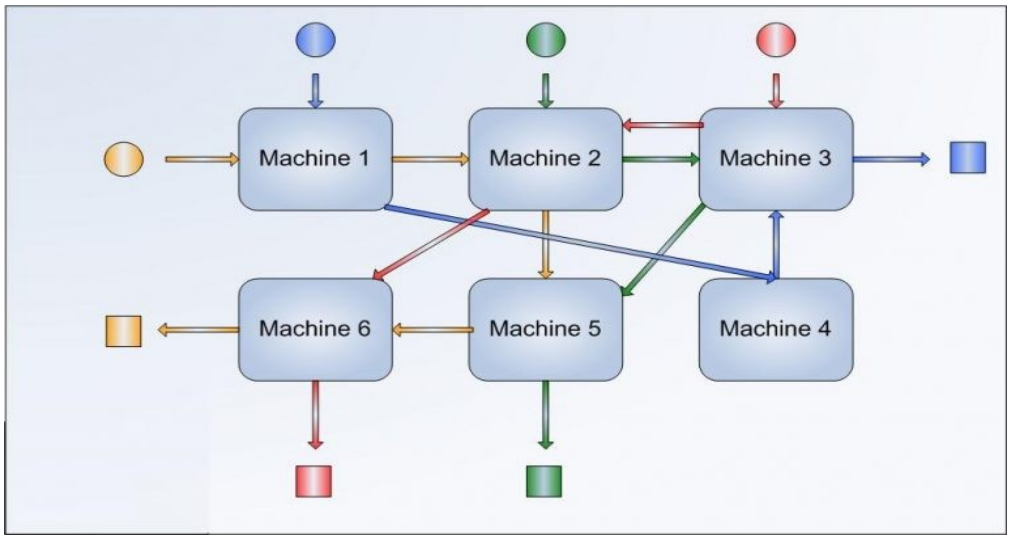


Figure 2: Exemple De Job Shoptiré de [20].

3.3. Flexible Job Shop Scheduling Problem(FJSSP)

Le Flexible Job Shop SchedulingProblem (FJSSP) est une extension directe du Job Shop. Il lève la contrainte d'unicité de la machine. Ici, une opération O_{ij} peut être traitée par n'importe quelle machine appartenant à un ensemble de machines éligibles M_{ij}. Cette flexibilité permet de mieux gérer les charges de travail et les imprévus tels que les pannes [5].

Cette structure est celle que l'on retrouve le plus souvent dans les unités de fabrication mécanique et électronique en Algérie, car elle permet une meilleure résilience opérationnelle [2].

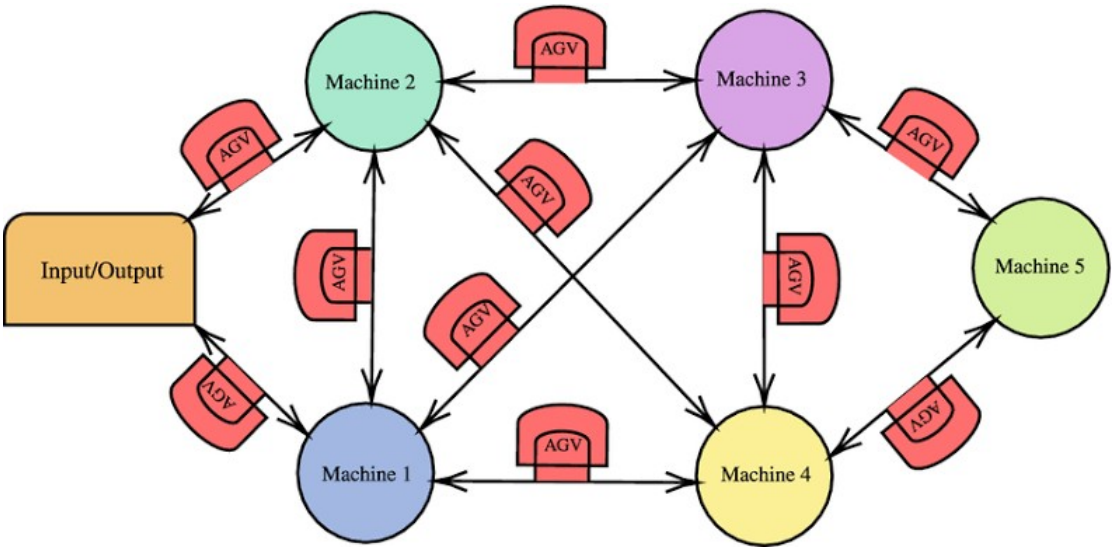


Figure 3: Exemple Flexible Job Shop (FJSSP) tiré de [20].

4. Formulation mathématique du FJSSP

4.1. Variables de décision, contraintes et données d'entrée

La modélisation mathématique rigoureuse est le fondement de toute approche de résolution. Elle permet de traduire les réalités physiques de l'atelier en un langage compréhensible par les algorithmes.

A. Les données d'entrée (Input Data)

Le système est modélisé par un ensemble de *n* travaux et *m* machines. Chaque opération O_{ij} (la $J^{éme}$ opération du travail i) est associée à une liste de machines capables de l'exécuter, ainsi qu'aux temps de traitement correspondants P_{ijk}[6].

B. Les variables de décision

On utilise généralement deux types de variables :

- **Variables d'affectation :** X_{ijk} est une variable binaire qui vaut 1 si l'opération O_{ij} est affectée à la machine M_k, et 0 sinon.

- **Variables de temps :** S_{ij}indique la date de début d'exécution de l'opération O_{ij} , tandis que C_{ij}représente sa date de fin [7].

C. Les contraintes du modèle:

Le respect des contraintes garantit qu'un ordonnancement est "réalisable" (feasible) :

● **Contrainte de précédence :** $S_{ij} \geq C_{i,j-1}$. Une opération ne peut pas commencer avant que la précédente dans le même job ne soit terminée.

● **Contrainte de non-chevauchement :** Une machine ne peut traiter qu'une seule opération à la fois. Si deux opérations sont affectées à la même machine, l'une doit impérativement se terminer avant que l'autre ne commence [3].

4.2. Fonctionobjectif (Optimisation du MakespanC_{max}):

Le critère de performance est la boussole de l'algorithme. Bien qu'il existe plusieurs objectifs (retard total, charge machine, coût), le Makespan reste le critère le plus fondamental. Le Makespan (C_{max}) représente la durée totale nécessaire pour achever l'ensemble des travaux. Mathématiquement, il s'agit de la date de fin de la dernière opération exécutée dans l'atelier : $C_{max} = \max_{1 \leq i \leq n} C_{i,n_i}$

Minimiser le Makespan revient à maximiser le taux d'utilisation global des ressources et à libérer l'atelier le plus tôt possible pour de nouvelles commandes [1].

Dans de nombreux travaux académiques algériens, ce critère est privilégié car il reflète directement la productivité brute de l'usine.

4.3. Complexité du problème (Problème NP-Hard)

Le FJSSP appartient à la classe des problèmes NP-difficiles (NP-Hard). Cette classification signifie que le temps de calcul nécessaire pour trouver une solution optimale croît de manière exponentielle avec la taille du problème (nombre de travaux et de machines). La complexité du FJSSP est double car elle intègre :

- **Un problème d'affectation :** Il faut choisir la machine optimale parmi plusieurs pour chaque opération.

- **Un problème d'ordonnancement :** Il faut déterminer l'ordre chronologique de passage des opérations sur chaque machine. « Le FJSSP est l'un des problèmes d'optimisation combinatoire les plus difficiles à résoudre de manière exacte, ce qui justifie l'exploration de méthodes heuristiques et métaheuristiques. »

L'impossibilité de résoudre des instances industrielles (ex: 20 jobs x 10 machines) en un temps raisonnable par des méthodes classiques pousse les chercheurs à se tourner vers des algorithmes bio-inspirés comme les Algorithmes Génétiques et le CuckooSearch.[7]

5. Conclusion

En conclusion, ce chapitre a permis de clarifier le cadre conceptuel de l'ordonnancement et de formaliser mathématiquement le problème du FJSSP. L'analyse de sa structure montre que la flexibilité, bien qu'avantageuse, engendre une double complexité combinatoire : l'affectation des machines et le séquencement des opérations. Classé comme un problème hautement NP-difficile, le FJSSP ne peut être résolu de manière exacte pour des instances industrielles de grande taille dans un temps raisonnable. Cette limite justifie l'utilisation des méthodes approchées, ce qui nous amène au chapitre suivant.

CHAPITRE 2 : LES METAHEURISTIQUES: ALGORITHME GENETIQUE ET RECHERCHE CUCKOO

CHAPITRE 2 LES MÉTAHEURISTIQUES ALGORITHME GÉNÉTIQUE ET RECHERCHE CUCKOO

1. Introduction

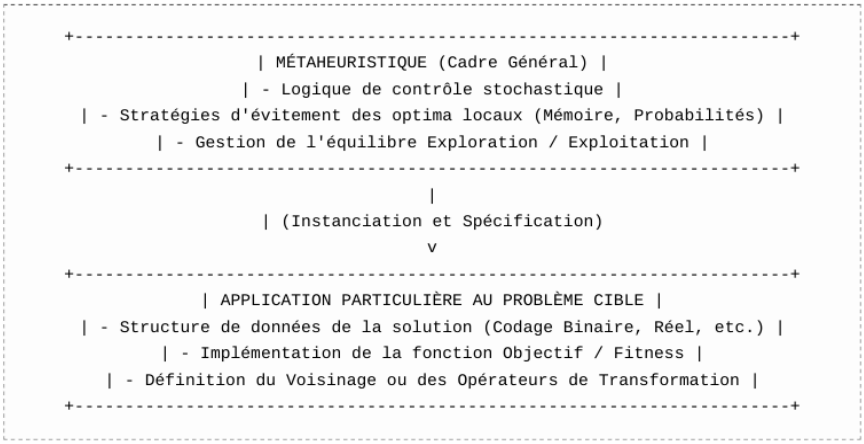
La résolution du problème FJSSP (Flexible Job Shop SchedulingProblem) relève de l'optimisation combinatoire complexe. Ce chapitre expose les fondements théoriques et algorithmiques des deux méthodes retenues : l'Algorithme Génétique (GA) et le CuckooSearch (CS). Nous détaillerons leur logique de fonctionnement, les structures de données utilisées pour la représentation des solutions, ainsi que les opérateurs mathématiques garantissant la convergence vers un optimum global.

2. Les Métaheuristiques

2.1. Définition des métaheuristiques

Le terme "métaheuristique", proposé pour la première fois par Fred Glover en 1986, est construit à partir du préfixe grec meta (qui signifie "au-delà", "à un niveau supérieur") et du verbe heuriskein (qui signifie "trouver", "découvrir"). Une métaheuristique peut être formellement définie comme un processus itératif de haut niveau qui guide et coordonne des heuristiques subordonnées (locales ou spécifiques) afin de concevoir un équilibre optimal entre l'exploration globale de l'espace des solutions et l'exploitation locale des structures prometteuses [8].

Sur le plan structurel, une métaheuristique fonctionne comme un cadre abstrait (un framework applicatif). Elle ne résout pas un problème spécifique par ses propres équations ; elle fournit une logique algorithmique générale qui manipule des représentations de solutions. Il revient à l'ingénieur informaticien d'instancier ce cadre en définissant le codage des solutions, la fonction d'évaluation (fitness) et les opérateurs de transition propres au problème traité [9].



Algorithme 1: Les Méthaheurstiques en cadre général.

2.2. Caractéristiques principales

Bien que les métaheuristiques englobent une grande diversité de concepts de conception, elles partagent un ensemble de caractéristiques fondamentales qui les distinguent des algorithmes déterministes et des techniques d'analyse numérique classiques [9].

- **Nature "Boîte Noire" (Black-Box Optimization) :** Elles n'exigent aucune propriété analytique ou mathématique stricte de la fonction objectif à optimiser. La fonction peut être discontinue, non dérivable, stochastique ou même se présenter sous la forme d'un code de simulation numérique complexe. L'algorithme se contente d'envoyer une solution candidate et de recevoir en retour un score quantitatif.
- **Mécanismes Stochastiques :** Contrairement aux méthodes déterministes, elles intègrent des variables pseudo-aléatoires lors des phases de transition d'états. Cela garantit que deux exécutions distinctes avec des graines aléatoires différentes exploreront des trajectoires différentes, empêchant l'algorithme de se bloquer systématiquement dans la même configuration sous-optimale.
- **Non-exactitude et Approximation :** Elles renoncent explicitement à la garantie mathématique de trouver à coup sûr la solution optimale globale ou de prouver qu'une solution trouvée est l'optimum absolu. Elles visent un compromis pragmatique orienté vers l'efficacité computationnelle.

- **Opérateurs d'Évasion :** Elles possèdent des règles spécifiques permettant d'accepter temporairement des dégradations de la fonction objectif (choisir des solutions moins bonnes). C'est cette propriété fondamentale qui leur permet de franchir des barrières de potentiel et de s'extraire des bassins d'attraction des optima locaux.

L'efficacité d'une métaheuristique repose sur la gestion rigoureuse d'une tension dialectique entre deux forces opposées [10].

- **L'exploration (Diversification) :** Il s'agit de la capacité de l'algorithme à visiter des régions éloignées, hétérogènes et non encore explorées de l'espace de recherche. Une exploration forte évite la convergence prématurée.

-**L'exploitation (Intensification) :** Il s'agit de la capacité à concentrer la recherche autour des zones prometteuses déjà identifiées afin de raffiner localement la qualité des solutions et de converger vers le sommet de l'optimum local associé. Un algorithme qui n'effectue que de l'exploration s'apparente à une recherche purement aléatoire (inefficace), tandis qu'un algorithme axé uniquement sur l'exploitation se comporte comme une descente de gradient locale (suivie d'un blocage rapide).

2.3. Classification des métaheuristiques

Il existe plusieurs critères pour classifier la vaste famille des métaheuristiques (naturelles vs non naturelles, avec ou sans mémoire, dynamiques vs statiques). Cependant, la taxonomie la plus robuste en informatique repose sur la structure de l'état manipulé au cours du cycle de calcul [9].

On distingue ainsi les méthodes à solution unique et les méthodes basées sur une population.

2.4. Méthodes à solution unique

Ces méthodes, également qualifiées de métaheuristiques locales, manipulent une seule solution candidate à la fois au cours du temps. L'algorithme commence à partir d'une configuration initiale et trace une trajectoire continue dans l'espace de recherche en effectuant des sauts successifs de la solution courante vers une solution située dans son voisinage immédiat.

Le principe général repose sur la structure de voisinage. À partir d'une solution S_0 , on définit un ensemble de solutions accessiblesV(S_0) par de petites modifications locales. Les exemples les plus célèbres incluent :

- **Le RecuitSimulé (SimulatedAnnealing - SA) :** Inspiré du processus de recuit en métallurgie, où le refroidissement lent des métaux permet d'atteindre un état cristallin d'énergie minimale. Il accepte des solutions dégradées selon une probabilité régie par un paramètre appelé "Température", qui décroît au fil du temps.
- **La RechercheTabou (TabuSearchTS) :**Conçue par Fred Glover, cette méthode utilise une mémoire à court terme (la liste tabou) pour enregistrer les derniers mouvements effectués. Cela interdit à l'algorithme de revenir sur ses pas, le forçant ainsi à sortir des optima locaux et à explorer de nouvelles zones.
- **La Recherche à VoisinageVariable :** Méthodes basées sur la population Contrairement aux approches de trajectoire, ces méthodes manipulent simultanément un ensemble constitué de plusieurs solutions candidates (appelé une population, un essaim ou une colonie) à chaque itération de l'algorithme. Elles favorisent intrinsèquement l'exploration globale grâce à des mécanismes d'interaction, de transfert d'informations, de coopération ou de compétition entre les différents individus de la population.

Le cycle de vie de ces méthodes consiste à initialiser une population de solutions, puis à appliquer des opérateurs collectifs pour générer une nouvelle population améliorée. Les deux grandes sous-familles sont :

- **Les AlgorithmesÉvolutionnaires (EvolutionaryAlgorithms - EA) :**qui miment les lois de la sélection naturelle et de la génétique. L'Algorithme Génétique en est le représentant le plus emblématique.
- **L'IntelligenceenEssaim (Swarm Intelligence - SI) :**qui modélise le comportement collectif et décentralisé d'agents auto-organisés (ex: l'Optimisation par Essaim Particulaire - PSO, l'Algorithme des Colonies de Fourmis - ACO, et la RechercheCuckoo) [8].

3. L'AlgorithmeGénétique

3.1. Définition et principe général

Développé à l'origine par John Holland à l'Université du Michigan au début des années 1970, et popularisé par son élève David Goldberg,l'Algorithme Génétique (AG) est une métaheuristique évolutionnaire stochastique globale [11]. L'AG repose sur la simulation informatique de l'évolution naturelle d'une population de structures abstraites (appelées chromosomes ou individus) au fil de générations successives. Chaque chromosome encapsule un ensemble de paramètres représentant une solution candidate dans l'espace de recherche du problème à résoudre.

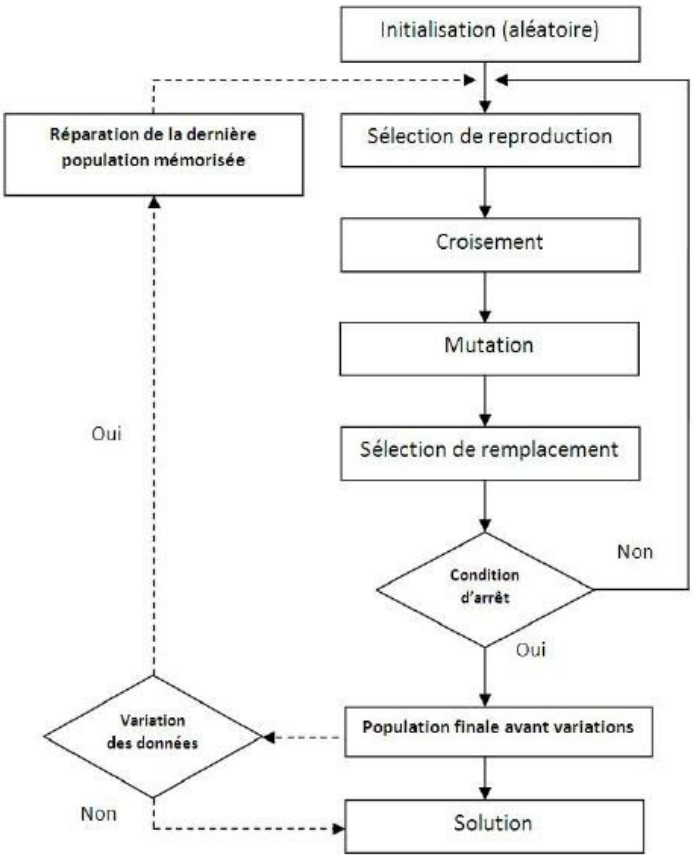


Figure 4: Le diagramme de l'algorithma génétique.

3.2. Il

L'algorithme modélise les concepts fondamentaux de la théorie de l'évolution de Charles Darwin, axée sur la sélection naturelle et la survie du plus apte (survival of the fittest), ainsi que les lois de la génétique moderne établies par Gregor Mendel [11]. Dans la biosphère, les individus les mieux adaptés à leur environnement (ceux possédant des caractéristiques génétiques avantageuses) ont une probabilité statistique plus élevée de survivre face aux prédateurs, de trouver des ressources, de se reproduire et de transmettre leurs gènes à la génération suivante. À l'inverse, les individus moins adaptés meurent prématurément sans descendance, entraînant la disparition progressive de leurs traits génétiques défectueux. Des mutations aléatoires spontanées interviennent à de faibles fréquences, introduisant de nouvelles variations génétiques essentielles pour l'adaptation aux mutations environnementales. [11]

La transposition de ces mécanismes biologiques dans le domaine de l'ingénierie informatique est synthétisée dans le tableau d'analogie formelle suivant :

Concept Biologique Naturel	**Transposition Informatique (Algorithme Génétique)**
Environnement	Espace de recherche (ensemble de toutes les solutions admissibles).
Individu / Organisme	Une solution candidate X au problème d'optimisation.
Chromosome	Vecteur de données codant la solution (structure de données).
Gène	Une composante élémentaire ou variable du vecteur (Allèle = valeur du gène).
Locus	Indice ou position physique d'une variable dans le vecteur.
Aptitude (Fitness)	Valeur quantitative renvoyée par la fonction objectif $\Phi(X)$.
Génération	Une itération temporelle complète de la boucle algorithmique.

Tableau1:Inspiration biologique de l'AG.

3.3. Structure d’un algorithme génétique

L'exécution d'un algorithme génétique canonique obéit à un enchaînement itératif strict et séquentiel d'opérateurs stochastiques. Le pseudo-code de niveau académique est formalisé ci-dessous :

```
Algorithme Génétique Canonique
Début
    t <- 0;
    Initialiser_Population(P(t));
    Évaluer_Population(P(t));
    Tant que (Non Critère_Arrêt()) Faire
        t <- t + 1;
        P'(t)  <- Sélectionner_Parents(P(t-1));
        P''(t) <- Appliquer_Croisement(P'(t));
        P'''(t)<- Appliquer_Mutation(P''(t));
        Évaluer_Population(P'''(t));
        P(t)   <- Remplacer_Population(P(t-1), P'''(t));
    FinTantQue
Fin
```

Algorithme 2: Le pseudo-code de l'algorithme génétique.

a) Population Initiale

La première phase consiste à instancier en mémoire vive un ensemble de N individus, représentant la population initiale notée P(0). Dans la configuration standard, cette génération s'effectue par l'intermédiaire d'un tirage pseudo-aléatoire uniforme au sein des bornes de l'espace de recherche, garantissant une couverture homogène et sans biais de l'espace des solutions [12]. Pour des problèmes fortement contraints (par exemple l'ordonnancement de processus), il est possible d'injecter des solutions générées par des heuristiques gloutonnes spécifiques afin d'accélérer la convergence, bien que cela augmente le risque de convergence prématurée par perte de diversité originelle.

b) Le codage (Représentation)

Le choix du codage est la décision d'architecture logicielle la plus critique. On distingue historiquement le codage binaire, où chaque solution est représentée par une chaîne de bits $\{0, 1\}^L$. Pour les problèmes d'optimisation continue ou d'ingénierie complexe, on préfère le codage réel (vecteurs de nombres à virgule flottante) ou le codage par permutation (utilisé pour le problème du voyageur de commerce, où le chromosome est une liste ordonnée d'indices de villes, ex: [3, 1, 4, 2][9].

c) La Fonction d'évaluation (Fitness)

La fonction de fitness, notée *Φ(X),* agit comme le mécanisme d'interfaçage entre l'algorithme génétique et le problème physique à résoudre. Elle associe à chaque chromosome *X* une valeur scalaire mesurant son adaptabilité ou sa performance.

Si le problème initial est un problème de maximisation d'une fonction objectif *f(X)*, le fitness s'identifie directement à celle-ci : $\Phi(X) = f(X)$. En revanche, si le problème formule la minimisation d'un coût *f(X),* une transformation mathématique monotone décroissante est requise pour garantir que le fitness reste positive et croissante avec la qualité de la solution. Les formulations classiques sont :

$$\Phi(x) = \frac{1}{1+f(x)} \text{ ou } \Phi(x) = C_{max} - f(x)$$

C_{max}est une constante positive estimée représentant une borne supérieure stricte de la fonction de coût au sein de l'espace de recherche [11].

d) La sélection

L'opérateur de sélection applique la pression sélective sur la population. Son but est de dupliquer les individus performants et d'éliminer les chromosomes médiocres, tout en maintenant une probabilité non nulle de survie pour les solutions moyennes afin de préserver le patrimoine génétique. Deux techniques algorithmiques majeures prédominent :

- **Sélection par roulette (Roulette Wheel Selection – RWS)** Également appelée sélection proportionnelle à le fitness. Chaque individu se voit attribuer une probabilité de sélectionP_i strictement proportionnelle à sa valeur de fitness relative par rapport à la somme des fitness de la population globale.

La formulemathématiqueénonceainsi : $$P_i = \frac{\Phi(x)}{\sum_{j=1}^{N} \Phi(x_j)}$$

- **Sélection par tournoi (TournamentSelection)**Cetteméthodeconsiste à extraire de manière aléatoire uniforme un sous-ensemble de k individus (où k désigne la taille du tournoi) de la population courante. Ces k individus entrent en compétition, et celui possédant le fitness la plus élevée gagne le tournoi et se voit sélectionné comme parent. Cette méthode est hautement privilégiée en informatique car elle est insensible aux problèmes d'échelle de la fitness (pas de distorsion si les scores sont très proches) et permet de contrôler finement la pression sélective par le simple ajustement du paramètre entier k [13].

e) Le croisement (Crossover)

Le croisement constitue l'opérateur principal de l'algorithme génétique, matérialisant la phase d'intensification (exploitation). Son rôle est de combiner les informations structurelles hautement adaptées de deux chromosomes parents pour engendrer de nouveaux individus enfants dotés, potentiellement, de caractéristiques supérieures. Cet opérateur est appliqué avec une probabilité probabiliste élevée, notée $P_c \in [0.6\ , 0.95]$.

- **Croisementen un point (Single-point crossover)** Unindice de coupure $c \in \{1, \ldots\ldots, l-1\}$ est sélectionné de manière aléatoire uniforme. Les segments de gènes situés au-delà de ce point de pivot sontintervertis entre les deux structures parentales.

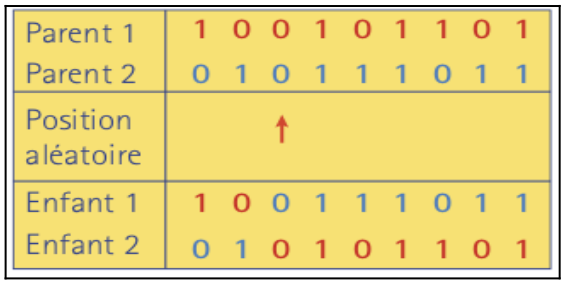
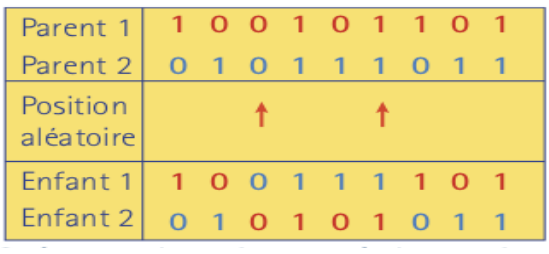


Figure 5: Exemple de Croisement en un pointtiré de [25] .

- **Croisementendeux points** Utilisedeux pivots aléatoires pour extraire et permuter un segment central.



- **Croisementuniforme (Uniform crossover)** Chaque gène de l'enfant est déterminé de manière indépendante en échantillonnant le gène du premier ou du second parent selon un masque binaire aléatoire généré à chaque croisement [12].

Parent 1	1 0 0 1 0 1 1 0 1
Parent 2	0 1 0 1 1 1 0 1 1
Chaîne aléatoire de bits	0 1 1 0 1 0 1 0 0
Enfant 1	1 1 0 1 1 1 0 0 1
Enfant 2	0 0 0 1 0 1 1 1 1

Figure 7: Exemple de Croisement uniformetiré de [25] .

f) Mutation

La mutation est l'opérateur de diversification (exploration). Intervenant après le croisement, elle introduit des altérations aléatoires locales à l'intérieur de la structure d'un chromosome. Son but est d'empêcher le blocage définitif de l'algorithme en réinjectant des allèles qui auraient pu être éliminés par la sélection. Elle est gouvernée par une probabilité très faible, $P_m \in [0.001 \ , \ 0.05]$.

- **Mutation binaire (Flip Bit) :** Si la condition aléatoire est validée pour un locus donné, la valeur du bit est inversée ($0 \rightarrow 1$ *ou* $1 \rightarrow 0$).
- **Mutation réelle (Gaussienne) :** Pour un gène codé par un flottant x_i, la mutation ajoute un bruit blanc gaussien centré en zéro : $x'_i = x_i + N(0, \sigma)$.

g) Le critèred'arrêt

Le bouclage de l'algorithme est interrompu lorsqu'une condition de convergence ou une limite de ressources est atteinte :

-Atteinte d'un nombre maximal prédéfini de générations

-Stagnation de la population : la valeur de fitness de l'individu le plus apte n'enregistre aucune amélioration significative pendant un nombre fixe d'itérations successives (générations stationnaires.

-Convergence de la population : la variance des fitness de la population s'approche de zéro, indiquant une homogénéisation complète des solutions.

3.4. Paramètres de l'algorithme génétique

Le comportement dynamique, la vitesse de convergence et la qualité des solutions trouvées par un AG dépendent d'un réglage fin de ses hyperparamètres de contrôle [14] : Taille de la population *(N)* : Une population trop restreinte ($N< 20$) restreint drastiquement la base génétique disponible, conduisant à une perte rapide de diversité et à un piège local immédiat. Une population surdimensionnée ($N > 500$) assure une excellente exploration mais induit un coût de calcul prohibitif, rendant l'algorithme extrêmement lent. Les valeurs standard se situent généralement entre 20 et 200.

Probabilité de croisement (P_c):Une valeur élevée favorise le brassage et la découverte de nouvelles combinaisons, mais si elle approche de 1.0, elle risque de détruire continuellement les structures de solutions performantes (appelées les blocs de construction ou building blocks) avant qu'elles n'aient pu se propager de manière stable dans la population.

Probabilité de mutation (P_m) : Elle doit rester faible (souvent inversement proportionnelle à la longueur du chromosome : 1/L). Si $P_m$est surévaluée (ex:P_m> 0.3), l'AG perd sa capacité de transmission héréditaire et dégénère en une recherche purement aléatoire (RandomWalk), effaçant la mémoire accumulée au fil des générations.

4. La recherche Cuckoo

4.1.Définition et principe général

Développé par Xin-She Yang et Suash Deb à l'Université de Cambridge en 2009, l'algorithme de la Recherche Cuckoo (CuckooSearch - CS) est une métaheuristiqued'optimisation stochastique globale inspirée des concepts de l'intelligence en essaim et du comportement biologique particulier de reproduction de certaines espèces d'oiseaux [15]. Cet algorithme se distingue par l'utilisation d'un formalisme mathématique rigoureux basé sur les vols de Lévy pour modéliser

le déplacement spatial des agents au sein de l'espace de recherche, offrant ainsi une alternative extrêmement performante aux marches aléatoires gaussiennes traditionnelles.

4.2.Inspiration naturelle des coucous

Dans la nature, plusieurs espèces de coucous (famille des Cuculidae) mettent en œuvre une stratégie de reproduction agressive appelée le parasitisme de couvée. Les femelles coucous n'édifient jamais de nids. Elles surveillent les nids d'oiseaux d'autres espèces (appelés les oiseaux hôtes), et profitent de leur absence momentanée pour *y* déposer discrètement leurs propres œufs. Pour parfaire le camouflage, la femelle coucou retire souvent un œuf d'origine de l'hôte afin de maintenir constant le nombre total d'œufs.

Si l'oiseau hôte s'aperçoit de la supercherie (par détection visuelle d'une divergence de couleur, de texture ou de taille de la coquille), il adopte l'une des deux stratégies de survie suivantes : soit il éjecte directement l'œuf parasite hors du nid, soit il abandonne définitivement son nid souillé et reconstruit une nouvelle couvée ailleurs. Pour optimiser leurs chances de survie, les œufs de coucous ont évolué pour éclore légèrement avant ceux de l'hôte. Dès sa naissance, le poussin coucou, bien que configurationnettement aveugle, déploie un comportement instinctif consistant à pousser et éjecter les œufs non éclos de l'hôte hors du nid, s'appropriant ainsi l'exclusivité des soins et de la nourriture apportée par les parents adoptifs [15].

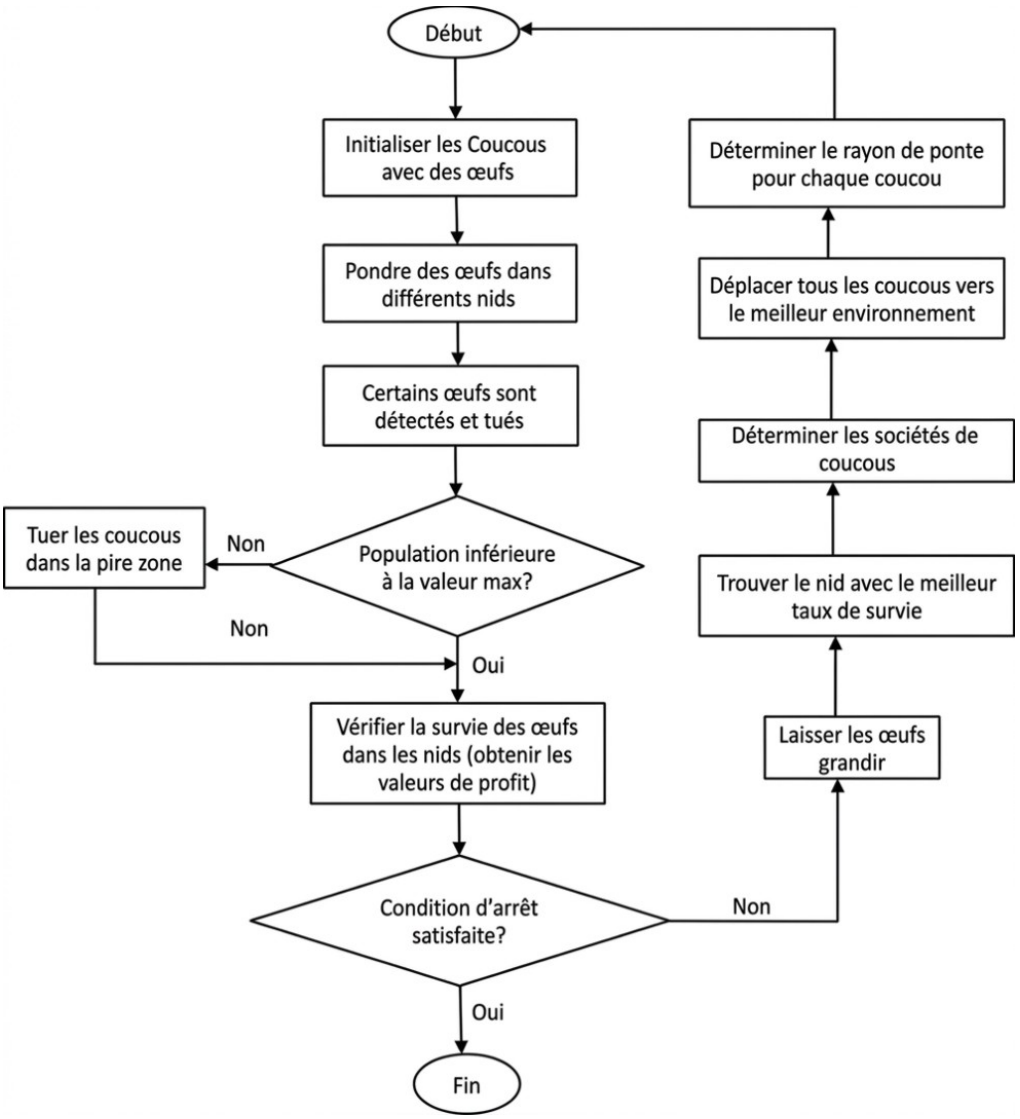


Figure 8: Diagramme de recherche cuckoo.

4.3.Fonctionnement de l'algorithme

Pour transposer cette dynamique comportementale en un algorithme d'optimisation robuste, Yang et Deb ont modélisé le système autour de trois règles idéalisées fondamentales :

-Chaque coucou pond un seul œuf à la fois, qu'il dépose dans un nid hôte sélectionné de manière purement aléatoire.

-Les nids contenant des œufs de haute qualité (représentant les meilleures solutions par rapport à la fonction objectif) sont conservés et transmis intacts à la génération suivante.

-Le nombre total de nids hôtes disponibles est fixé à une constante n. L'œuf parasite déposé par un coucou peut être découvert par l'oiseau hôte avec une probabilité fixe $P_a \in [0, 1]$. Face à cette découverte, l'hôte détruit l'œuf ou reconstruit un nid entièrement neuf dans un nouvel emplacement spatial aléatoire.

-Le pseudo-code formel de la Recherche Cuckoo est structuré de la manière suivante:

```
Algorithme Cuckoo Search (CS)
Début
    Définir la fonction objectif f(x), x = (x1, ..., xd)^T;
    Générer la population initiale de n nids hôtes x_i (i = 1, ..., n);
    Tant que (t < MaxGenerations) ou (Critère_Arrêt) Faire
        Sélectionner un coucou (i) de manière aléatoire;
        Générer une nouvelle solution x_new via un Vol de Lévy;
        Évaluer la fitness F_new = f(x_new);
        Choisir un nid hôte (j) aléatoirement parmi les n nids;
        Si (F_new est meilleure que F_j) Alors
            x_j <- x_new;  // Remplacement de la solution du nid j
        FinSi

        // Phase de découverte des œufs parasites
        Déterminer une fraction (p_a) des moins bons nids;
        Remplacer ces nids découverts par de nouvelles positions (marches
aléatoires locales);
        Conserver les meilleures solutions globales;
        Trier la population et identifier le meilleur nid global G_best;
        t <- t + 1;
    FinTantQue
Fin
```

Algorithme 3: Le pseudo-code de recherche cuckoo.

a) La génération des nids

Chaque nid au sein de la population représente une solution candidate vectorielle $X_i = (X_{i1}, X_{i2}, \ldots\ldots X_{id})$, où d spécifie la dimension géométrique de l'espace de recherche (le nombre de variables du problème). La phase d'initialisation s'opère par une distribution uniforme continue entre les bornes inférieures L_b et supérieures $U_b$définies par les contraintes du problème :

$$X_{ij} = L_{b,j} + rand\ (U_{b,j} - L_{b,j})$$

b) Évaluation des solutions

L'évaluation repose sur la correspondance stricte suivante : un œuf équivaut à une solution candidate, et un nid correspond à un emplacement mémoire physique stockant cette solution. La qualité ou la viabilité de l'œuf dépend directement du score renvoyé par la fonction objectif $f(x_i)$ Le processus algorithmique cherche à substituer systématiquement les œufs de faible qualité logés dans les nids par de meilleurs œufs parasites.

c) Vol de Lévy

Le moteur mathématique fondamental de la phase d'exploration globale dans le CuckooSearch réside dans l'utilisation des vols de Lévy. Contrairement aux marches aléatoires classiques (mouvement brownien) basées sur la distribution de Gauss et caractérisées par des pas courts et homogènes, le vol de Lévy est un processus stochastique non linéaire dont la longueur des pas est régie par une loi de puissance à queue lourde, définie formellement par [16]:

$$L(s) \sim s^{-1-\beta} ou \quad 0 < \beta \leq 2$$

Sur le plan algorithmique, la mise à jour de la position spatiale d'un coucou *i* à l'étape itérative *t+1* s'effectue selon la relation vectorielle suivante :

$$X_i^{(t+1)} = X_i^{(t)} + \alpha \ \mathbin{⨀} Lévy(\beta)$$

où $\alpha > 0$ désigne un paramètre d'échelle lié à la géométrie du problème (généralement fixé à 1), et $\mathbin{⨀}$ représente le produit de Hadamard (multiplication terme à terme).

Pour modéliser numériquement ces pas sans divergence stochastique, le CS implémente l'algorithme mathématique de Mantegna [Mantegna, 1994]. La longueur de pas efficace s est générée par le ratio stochastique :

$$s = \frac{u}{|v|^{1/\beta}}$$

où*u* et *v* sont extraits indépendamment de deux distributions normales centrées :

$$u \sim N\left(0, \sigma_u^2\right) \ , \quad v \sim N\left(0, \sigma_u^2\right)$$

L'écart-type σ_u est calculé analytiquement via l'évaluation de la fonction Gamma (γ) :

$$\sigma_u = \left[\frac{\gamma(1+\beta) \times \sin\left(\frac{\pi\beta}{2}\right)}{\gamma\left(\frac{1+\beta}{2}\right) \times \beta \times 2^{(\beta-1)/2}} \right]^{1/\beta} \ , \ \sigma_v = 1$$

Cette alternance géométrique entre de très nombreux pas courts d'intensification locale et de rares sauts de très grande amplitude (longues distances) confère au CuckooSearch une capacité d'exploration globale exceptionnelle, lui permettant de sauter par-dessus les barrières de potentiel des fonctions de coût multimodales et de s'extraire efficacement des optima locaux [15].

d) Remplacement des mauvaises solutions

Une fois le vol de Lévy exécuté, la nouvelle solution enfant est mise en compétition avec un nid *j* sélectionné aléatoirement. Si l'œuf du coucou présente un fitness supérieure à celle présente dans le nid *j*, la solution existante est écrasée et remplacée.

Simultanément, l'opérateur de découverte intervient : une fraction P_a des nids de la population globale (les moins performants) est identifiée. L'oiseau hôte abandonne ces emplacements. De nouvelles solutions sont alors générées localement via une marche aléatoire basée sur la distance séparant deux nids arbitraires de la population courante :

$$X_i^{(t+1)} = X_i^{(t)} + r \times \left(X_j^{(t)} - X_k^{(t)} \right)$$

Oùest un scalaire uniforme inclus dans [0, 1], tandis que $X_j^{(t)}$ et $X_k^{(t)}$ désignent deux nids distincts sélectionnés aléatoirement.

4.4. Paramètres de la recherche cuckoo

L'un des atouts majeurs du CuckooSearch par rapport aux autres métaheuristiques réside dans sa concision paramétrique. Il nécessite le réglage de très peu d'hyperparamètres de contrôle [15] :

-Nombre total de nids (n) : Représente l'effectif de la population. Des analyses empiriques extensives démontrent qu'une valeur fixe comprise entre n = 15 et n = 40 est amplement suffisante pour converger vers l'optimum global sur la grande majorité des fonctions benchmarks standards.

-Probabilité d'abandon (P_a) : Détermine la vitesse d'élimination des mauvaises solutions. La valeur standard recommandée et validée par la littérature est $P_a = 0.25$.

-Exposant de Lévy (β) : Contrôle la distribution statistique des pas. La valeur de référence est fixée à $\beta = 1.5$.

5. Conclusion

Ce chapitre a permis de formaliser et d'étudier de manière approfondie les mécanismes algorithmiques sous-jacents aux métaheuristiques orientées population à travers deux paradigmes complémentaires. D'une part, l'Algorithme Génétique, fondé sur les lois de l'hérédité génétique et de la sélection naturelle, excelle dans la manipulation de structures discrètes et complexes grâce à la richesse combinatoire de ses opérateurs de croisement.

D'autre part, la Recherche Cuckoo, s'appuyant sur les concepts d'intelligence collective en essaim et la dynamique mathématique avancée des vols de Lévy, offre une puissance d'exploration globale remarquable alliée à une grande simplicité de configuration des paramètres. L'analyse comparative met en évidence qu'il n'existe pas de métaheuristique universellement supérieure à toutes les autres (conformément aux théorèmes du No Free Lunch).

L'Algorithme Génétique offre un cadre hautement structuré adapté aux espaces combinatoires riches mais au prix d'un fardeau de calcul élevé et d'une sensibilité marquée à la calibration de ses hyperparamètres. À l'inverse, la Recherche Cuckoo s'illustre par sa rapidité de convergence et sa capacité d'évasion face aux pièges des optima locaux grâce aux sauts à longue distance de Lévy, bien qu'elle

nécessite des ajustements dynamiques en fin de cycle pour affiner sa précision locale.

Dans l'optique de la résolution de problèmes d'optimisation combinatoire de grande envergure, comme le Flexible Job Shop SchedulingProblem (FJSP), la maîtrise fine de ces deux approches, ou leur hybridation synergique (Algorithmes Hybrides), représente une compétence technologique majeure pour l'ingénieur et le chercheur en informatique, ouvrant la voie au développement de systèmes d'aide à la décision hautement adaptatifs et performants.

CHAPITRE 3 : CONCEPTION ET APPROCHE DE RESOLUTION

CHAPITRE 3 CONCEPTION ET APPROCHE DE RÉSOLUTION

1. Introduction

Le problème d'ordonnancement d'atelier flexible (Flexible Job Shop SchedulingProblem - FJSP) représente l'une des extensions les plus complexes et les plus proches de la réalité industrielle du Job Shop classique [17]. Contrairement au problème traditionnel où chaque opération est affectée à une unique machine prédéfinie, le FJSP introduit une flexibilité totale ou partielle, permettant à une opération d'être exécutée sur plusieurs machines alternatives avec des temps de traitement potentiellement distincts [18].

Cette caractéristique double la complexité du problème, nécessitant à la fois de résoudre un sous-problème d'affectation (attribuer chaque opération à une machine appropriée) et un sous-problème de séquencement (déterminer l'ordre de passage des opérations sur chaque machine) [19]. L'objectif de ce chapitre est de présenter formellement la conception du système d'aide à la décision proposé pour la résolution du FJSP. Nous détaillons d'abord l'analyse et la modélisation mathématique et conceptuelle du problème. Ensuite, nous exposons les approches de résolution conçues : la première basée sur un Algorithme Génétique (AG), une métaheuristique évolutionnaire robuste [20], et la seconde basée sur l'Algorithme de Recherche Cuckoo (CuckooSearch - CS), une métaheuristique récente et performante inspirée de la nature [21]. Enfin, une comparaison théorique de ces deux approches mettra en évidence leurs stratégies respectives d'exploration et d'exploitation de l'espace de recherche .

2. Analyse du Problème Job Shop Flexible

2.1. Description du problème

Le FJSP peut être défini formellement par un ensemble de n jobs, notés J = { $J_1, J_2, \ldots\ldots, J_n$}, devant être exécutés sur un ensemble de m machines physiques, notées M = {$M_1, M_2, \ldots\ldots, M_n$} [17]. Chaque job $J_i$est constitué d'une séquence linéaire et ordonnée de$n_i$opérations, notées O_{ij} (où j représente l'indice de l'opération, j ∈ {1, 2, ...,n_i}).

La flexibilité implique que pour chaque opération O_{ij}, il existe un sous-ensemble de machines compatibles, noté $M_{ij} \subseteq$ M, capables de l'exécuter [18]. Le temps nécessaire pour accomplir l'opération O_{ij} sur une machine compatible $M \in M_{ij}$ est déterministe et noté P_{ijk}. L'ordonnancement consiste alors à trouver une configuration optimale qui affecte chaque opération à une machine de son ensemble M_{ij} et fixe ses dates de début et de fin d'exécution [19]. Ce problème est classé comme NP-difficile, ce qui justifie l'utilisation des métaheuristiques [22].

2.2. Contraintes du problème

Pour garantir la validité des solutions générées, plusieurs contraintes strictes, dites "contraintes dures", doivent être impérativement respectées [17] :

- **Contrainte de précédence (Jobs) :** Les opérations d'un même job J_i doivent être exécutées dans un ordre séquentiel strict. L'opération $O_{i,j}$ ne peut commencer que si l'opération précédente $O_{i,j-1}$ est entièrement terminée.
- **Contrainte d'invisibilité des machines :** Une machine ne peut traiter qu'une seule opération à la fois. Aucune superposition temporelle d'opérations sur une même machine n'est autorisée.
- **Non-préemption :** Une fois qu'une opération O_{ij} a commencé son traitement sur une machine M_k elle ne peut être interrompue ou suspendue ; elle doit être exécutée jusqu'à son achèvement complet.
- **Disponibilité :** Toutes les machines et tous les jobs sont disponibles pour le traitement dès l'instant initial t=0.

2.3. Objectifd'optimisation

Dans le cadre du FJSP on peut répondre à plusieurs critères. Dans le cadre de notre étude, nous nous concentrons sur le critère le plus critique en gestion de production : la minimisation du temps total de complétion (Makespan) [18].

a) Minimisation du Makespan

Le Makespan, noté C_{max}, correspond à la date de fin d'exécution du dernier job de l'atelier [17]. Sa formulation mathématique est la suivante:$C_{max} = \max_{1 \le i \le n}(C_i)$

Où $C_i$représente la date d'achèvement de la dernière opération du job J_i. Minimiser le Makespan revient à maximiser le taux d'utilisation des machines de l'atelier [19].

3. Modélisation du Flexible Job Shop Scheduling

3.1. Représentation des Jobs

Chaque job$J_i$est modélisé comme une entité indépendante caractérisée par un identifiant unique et une liste chaînée ordonnée d'opérations [18]. Un job n'a pas de durée intrinsèque fixe, sa durée totale dépend directement des choix d'affectation des machines pour ses opérations constitutives.

3.2. Représentation des Opérations

Une opération O_{ij}est l'unité élémentaire de travail. Elle est modélisée par un objet contenant ses attributs de précédence et sa matrice de compatibilité listant les machines $M_k \in M_{ij}$et les temps de traitement associés P_{ijk} [19].

3.3.Représentation des Machines

Chaque machine $M_k$est modélisé comme une ressource discrète. Elle conserve un planning de ses plages d'occupation afin de permettre un ordonnancement par insertion, optimisant ainsi les fenêtres de temps libres (gaps) [17].

3.4. Représentation d'une solution

Une solution complète du FJSP doit obligatoirement encapsuler deux informations fondamentales : le vecteur d'affectation (Machine Part) et le vecteur de séquencement (Sequence Part) [19].

4. Approche de résolution par AlgorithmeGénétique

4.1. Principe général

Les Algorithmes Génétiques (AG) sont des méthodes d'optimisation stochastiques fondées sur les mécanismes de la sélection naturelle et de la génétique mendélienne [20]. Le principe repose sur l'évolution d'une population de solutions candidates subissant des transformations (croisement et mutation) afin de converger vers un optimum global [23].

4.2. Codage des solutions

Pour le FJSP, nous concevons un codage chromosomique dual, divisé en deux sections distinctes : un chromosome d'affectation (MS - Machine Selection) et un chromosome de séquencement (OS OperationSequence) basé sur les identifiants de jobs répétitifs [19]. Ce codage élimine nativement le risque d'invalidation des contraintes de précédence lors des variations stochastiques [23].

4.3. Génération de la population initiale

Pour équilibrer la qualité initiale et la diversité génétique de la population, nous concevons une stratégie de génération mixte combinant une initialisation heuristique (60%) pour garantir de bons individus, et une initialisation aléatoire (40%) pour couvrir l'ensemble de l'espace de recherche [19].

4.4. Fonction de fitness

La fonction d'adaptation (Fitness) mesure la qualité d'un chromosome. Le FJSP étant un problème de minimisation, la fonction de fitness F est définie comme l'inverse du MakespanC_{max} afin de maximiser la probabilité de survie des meilleures solutions [23] : F(Chromosome) = $\frac{1}{C_{max}}$

- **Calcul du Makespan**

Le calcul s'effectue en décodant le chromosome : les opérations du vecteur OS sont lues séquentiellement et placées sur la machine cible spécifiée par MS au plus tôt, en respectant la disponibilité de la machine et la fin de l'opération précédente [19].

4.5. Opérateur de sélection

Nous utilisons la Sélection par Tournoi Stochastique. Un sous-ensemble d'individus est choisi au hasard, et celui possédant le meilleur fitness est sélectionné pour devenir parent, ce qui permet de contrôler efficacement la pression sélective [23].

4.6. Opérateur de croisement

En raison de la structure duale de notre codage, deux opérateurs distincts sont appliqués : le croisement POX (PrecedenceOperationCrossover) pour la partie séquencement (OS), qui préserve l'ordre des jobs, et un croisement multipoint (Two-Point Crossover) pour la partie affectation (MS) [19].

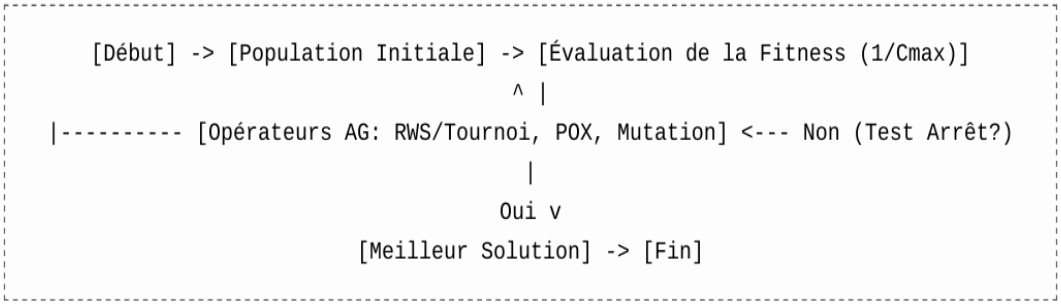
4.7. Opérateur de mutation

La mutation introduit des variations aléatoires locales pour éviter le blocage dans des optima locaux : une mutation par échange (Swap) pour la partie OS, et une modification aléatoire de la machine cible pour une opération donnée dans la partie MS [19].

4.8. Critère d'arrêt

L'algorithme s'arrête lorsqu'un nombre maximal de générations fixé à l'avance est atteint G_{max} , ou en cas de stagnation prolongée de la solution optimale [23].

4.9. Organigramme de l'algorithme génétique



Algorithme 4: Le pseudo-code d AG en FJS .

5. Approche de résolution par Recherche Cuckoo

5.1. Principe général

L'algorithme de Recherche Cuckoo (CS) est une métaheuristique bio-inspired développée par Yang et Deb [2009]. Elle est calquée sur le comportement de parasitisme de couvée de certaines espèces de coucous qui déposent leurs œufs dans les nids d'autres oiseaux, combiné avec le modèle mathématique des Vols de Lévy [21].

5.2. Représentation des nids

Dans notre approche, un nid représente une solution candidate au FJSP. Pour préserver l'équité comparative, la structure interne d'un œuf dans un nid est identique au codage dual (OS et MS) défini pour l'Algorithme Génétique.

5.3. Génération des solutions initiales

La population de N nids est initialisée de manière dispersée dans l'espace de recherche en utilisant le même mécanisme hybride (heuristique et stochastique) pour garantir une base qualitative stable.

5.4. Évaluation des solutions

L'évaluation de la qualité d'un œuf repose directement sur la performance de son calendrier d'ordonnancement. Le Makespan est déterminé par le même algorithme de décodage par insertion temporelle décrit précédemment.

5.5. Lévy Flight

Le pivot de l'exploration dans le CS est le Vol de Lévy. Mathématiquement, il s'agit d'une marche aléatoire dont la longueur des pas est régie par une distribution de probabilité à queue lourde [21]. Pour l'adapter au FJSP discret, la longueur du

pas calculée par l'algorithme de Mantegna [1994] est transposée en un nombre de modifications structurelles (mutations/gaps) à apporter au nid.

5.6. Remplacement des mauvaises solutions

Conformément au modèle naturel, une proportion p_a des œufs découverts par l'oiseau hôte est abandonnée [23]. Dans l'algorithme, les nids présentant les moins bonnes valeurs de Makespan sont éliminés et remplacés par de nouveaux nids construits via des marches aléatoires globales [21].

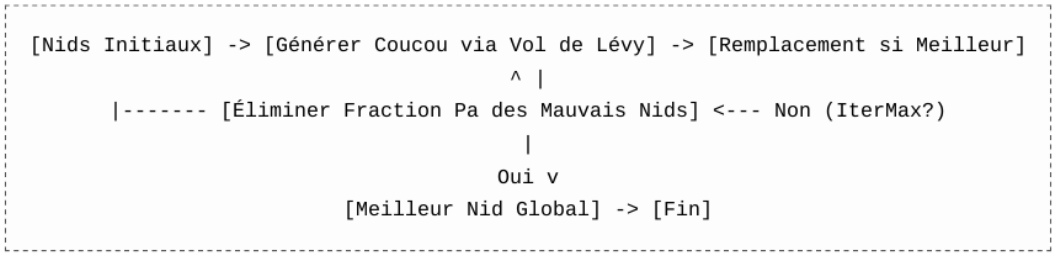
5.7. Sélection de la meilleure solution

À chaque itération, le meilleur nid global (celui possédant le Makespan le plus faible) est conservé sans altération pour la génération suivante (stratégie élitiste) [18].

5.8. Critère d'arrêt

Le processus itératif de la recherche Cuckoo prend fin lorsque le nombre maximal d'itérations ($Iter_{max}$) est complété [19].

5.9. Organigramme de la recherche cuckoo



Algorithme 5: Le pseudo-code de racherche cuckoo en FJS.

5. Comparaison des deuxapproches

6.1. Qualité des solutions

Grâce aux opérateurs combinatoires (comme le POX), l'AG excelle dans l'assemblage de sous solutions de bonne qualité (les schémas génétiques) sur des

instances de grande taille [19]. De son côté, le CS maintient une excellente précision locale grâce à l'ajustement fin des vols de Lévy [21].

6.2. Temps de convergence

La convergence de l'AG peut s'avérer plus lente en raison du traitement lourd requis par la manipulation de populations entières via de multiples opérateurs à chaque cycle [24]. Le CS, possédant moins de paramètres de contrôle, offre souvent une convergence plus rapide vers une solution stable [21].

6.3. Exploration et exploitation de l'espace de recherche

L'équilibre entre exploration et exploitation est géré différemment : l'AG s'appuie sur la diversité de sa population et la mutation [23], tandis que le CS utilise la dualité entre les grands sauts des vols de Lévy (exploration) et l'élimination des mauvais nids via P_a(exploitation) [21]. Cette dynamique comparative est résumée dans les travaux de synthèse sur le FJSP [24].

Caractéristique	Algorithme Génétique (AG)	Recherche Cuckoo (CS)
Exploration (Globale)	Assurée par la taille de la population et les opérateurs de mutation stochastique.	Assurée par les grands pas des Vols de Lévy et le remplacement d'une fraction ***p_a*** de nids.
Exploitation (Locale)	Portée par la pression de sélection du tournoi et la recombinaison des parents d'élite.	Portée par les petits pas de Lévy (recherche locale fine) autour des meilleurs nids conservés.
Sensibilité aux paramètres	Élevée (Taux de croisement, taux de mutation, taille du tournoi).	Faible (Facile à calibrer, repose principalement sur ***p_a***).

Tableau2: Exploration et exploitation de l'espace de recherché.

7. Conclusion

En conclusion, ce troisième chapitre a permis de poser les fondations théoriques nécessaires à la résolution du FJSP. Nous avons formalisé

mathématiquement les contraintes et l'objectif du Makespan. La conception des deux algorithmes (AG et CS) a été détaillée à travers leurs mécanismes de codage et de navigation stochastique. Cette modélisation servira d'ossature rigoureuse pour l'étape d'implémentation logicielle.

CHAPITRE 4 : IMPLEMENTATION ET RESULTATS EXPERIMENTAUX

CHAPITRE 4 IMPLEMENTATION ET RESULTATS EXPERIMENTAUX

1. Introduction

Dans ce chapitre, nous présentons l'implémentation pratique de notre application dédiée à la résolution du problème Flexible Job Shop SchedulingProblem (FJSP) à l'aide de deux métaheuristiques : l'Algorithme Génétique (GeneticAlgorithm - GA) et l'algorithme CuckooSearch (CS).

L'objectif principal de cette implémentation est de comparer les performances des deux approches en termes de qualité de solution, de convergence et de temps d'exécution.

L'application développée permet :

- La génération et le chargement des benchmarks.
- L'exécution des algorithmes d'optimisation.
- Le calcul du Makespan.
- La visualisation des résultats via des diagrammes de Gantt.
- La comparaison expérimentale des performances des algorithmes.

2. Environnement de développement

Le développement de l'application a été réalisé dans l'environnement suivant :

- Langage de programmation : Java.
- Environnement de développement :NetBeans IDE.
- Bibliothèque graphique : Java Swing.
- Plateforme d'exécution: JDK.
- Système d'exploitation: Windows.

NetBeans IDE a été utilisé pour :

- La conception de l'interface graphique.
- La gestion des événements.
- Le développement des algorithmes.
- Le débogage et les tests.

3. Structure du Programme

L'application développée est organisée selon une architecture modulaire afin de faciliter :

- Le développement.
- La maintenance.
- L'exécution des algorithmes.
- La gestion des données.

Le programme est composé de plusieurs classes principales assurant chacune une fonctionnalité spécifique.

Les principales classes utilisées sont :

Classe	Rôle
Splashscreen	La première interface pour nous présenter.
JOBSHOP	Interface principale de l'application qui contient les paramètres à saisir pour exécuter.
Algorithme Génétique	Exécution de l'algorithme génétique.
Recherche cuckoo	Exécution de l'algorithme Recherche Cuckoo.
Diagramme de GANTT	Affichage graphique des solutions.
Comparaison	La comparaison entre les deux algorithme.
Benchmark Loader	Chargement des benchmarks.

Tableau3: Les principales classes utilisées.

L'architecture générale permet :

- La séparation entre l'interface graphique et les algorithmes.
- La manipulation des données des opérations.
- Le calcul du Makespan.
- La visualisation des résultats.

4. Organigrammegénéral de l'aplication

L'organigramme suivant représente le fonctionnement général de l'application développée pour la résolution du problème Flexible Job Shop SchedulingProblem (FJSP).Le processus commence par le chargement des données , puis l'utilisateur choisit l'algorithme d'optimisation à utiliser (GeneticAlgorithm ou CuckooSearch).

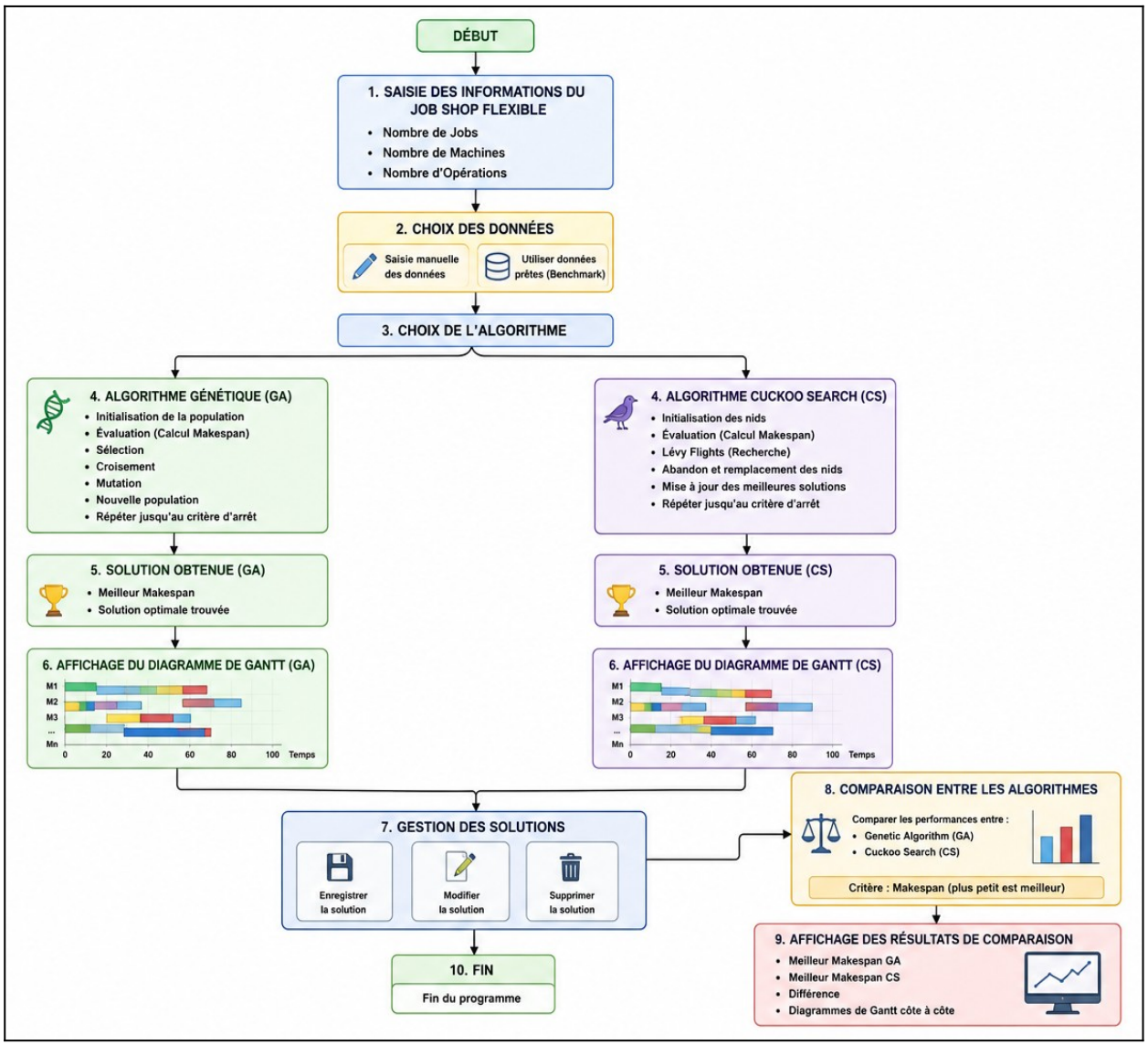


Figure 9: Organigramme de programme.

Ensuite, l'application génère les solutions, calcule le Makespan et affiche les résultats sous forme numérique et graphique.

5. Procédure de Calcul du Makespan

Le calcul du Makespan constitue l'étape principale de l'évaluation des solutions dans notre application.

Cette procédure prend en considération :

- L'ordre des opérations.
- La disponibilité des machines.
- Les temps de traitement.
- Les contraintes du Flexible Job Shop Scheduling Problem.
- Le Makespanreprésente le temps total nécessaire pour terminer toutes les opérations des différents jobs.

Les principales étapes sont :

- Lecture des opérations.
- Sélection de la machine.
- Calcul du temps de début.
- Calcul du temps de fin.
- Mise à jour du Makespan.
- Affichage du résultat final.

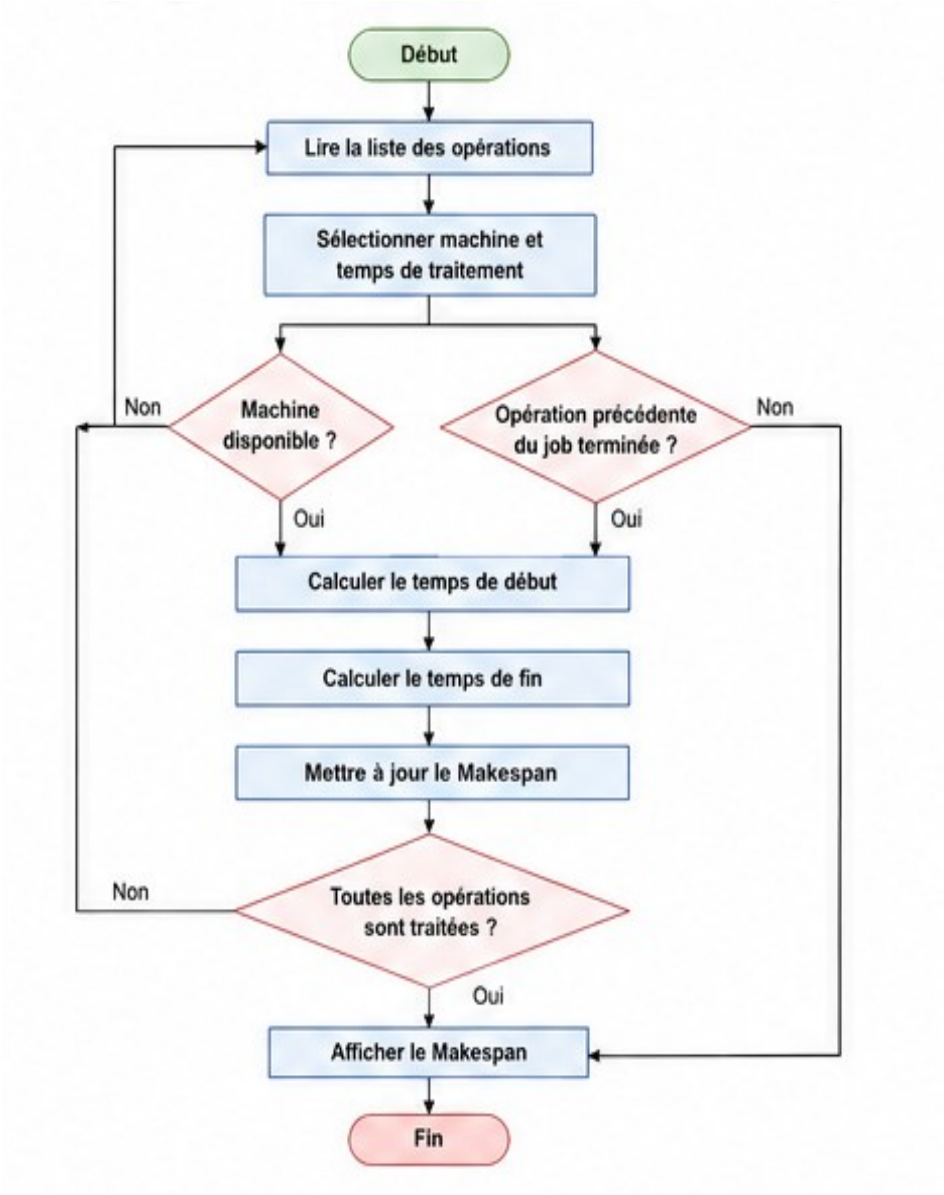


Figure 10: Organigramme Procédure Calcul du Makespan.

6. Procédure de Tri et de Sélection

La procédure de tri et de sélection permet d'identifier les meilleures solutions générées par les algorithmes d'optimisation.

Les solutions sont évaluées selon leur valeur de Makespan afin de sélectionner les solutions les plus performantes.

Dans le cas du GeneticAlgorithm, cette procédure utilise :

- Le tri des solutions.
- La sélection.

- Le croisement.
- La mutation.

Dans le cas du CuckooSearch, elle utilise :

- L'évaluation des nids.
- Le remplacement des solutions faibles.
- L'exploration par Lévy Flights.

Cette étape améliore progressivement la qualité des solutions obtenues.

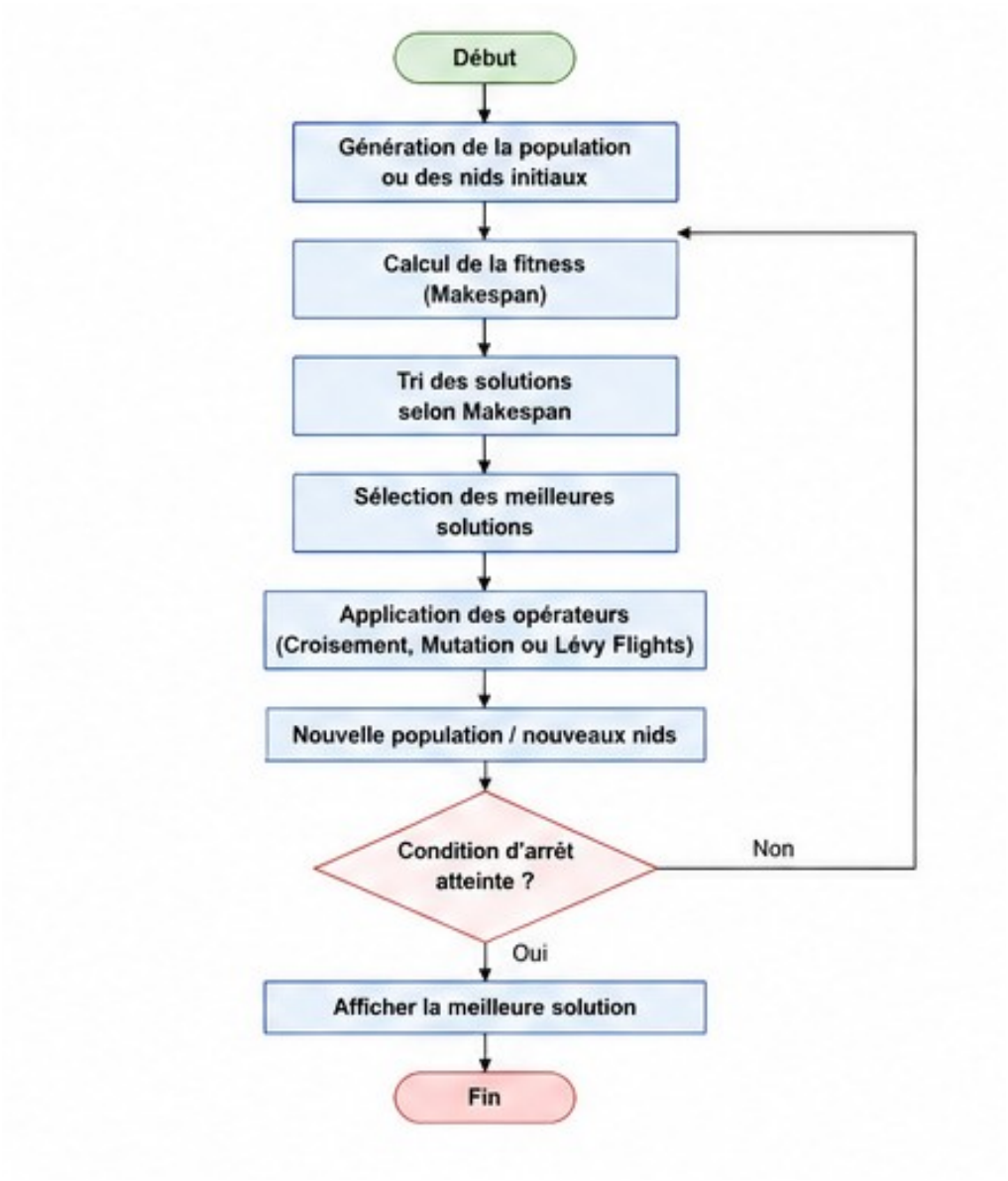


Figure 11:Organigramme de Procédure de Tri et de Sélection.

7. Présentation de l'interfacegraphique

A. SplashScreen

La première interface de l'application

Figure 12: Interface de splash screen.

B. Interface Principale

L'interface graphique développée facilite l'interaction avec l'utilisateur et permet l'exécution des différentes fonctionnalités de l'application.

Elle est composée des éléments suivants :

- Zone des paramètres du Job Shop.
- Zone des paramètres de l'Algorithme Génétique.
- Zone des paramètres de Cuckoo Search.

- Tableau des opérations et machines.
- Boutons d'exécution des algorithmes.
- Zone d'affichage des résultats.
- Diagramme de Gantt.

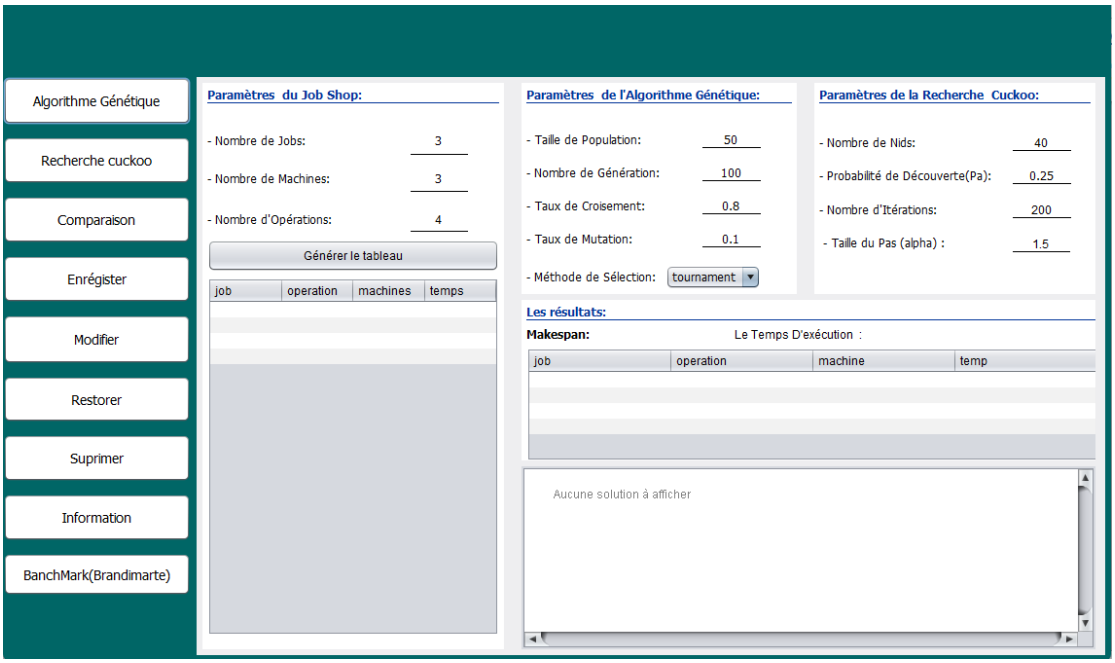
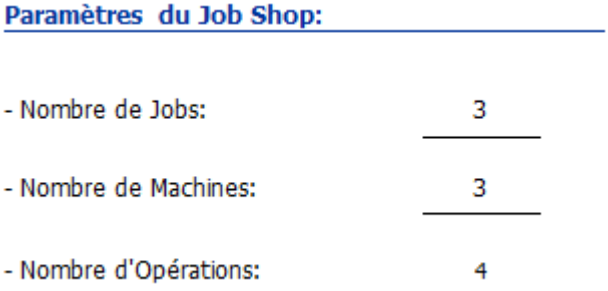


Figure 13: Interface principale.

8. Exemples

a) Paramètres du Flexible Job Shop

Cette partie permet à l'utilisateur de définir :



- Le nombre de Jobs.
- le nombre des machines.
- Le nombre d'Opérations.

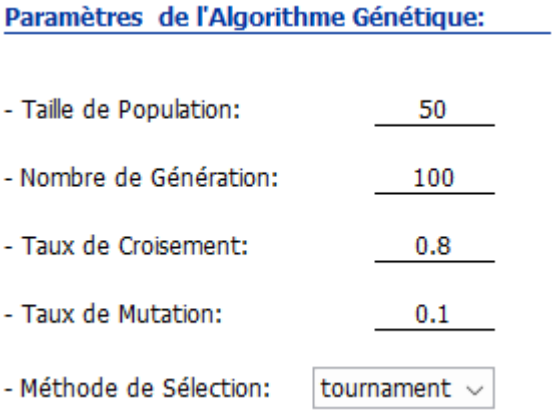


Figure 14: Paramètres du job shop.

b) Paramètres de l'AlgorithmeGénétique

Cette section contient les paramètres nécessaires à l'exécution du Algorithme Génétique :

- Taille de la population.
- Nombre de génération.
- Taux de croisement.
- Taux de mutation.
- Méthode de sélection.

Figure 15: Paramétre de AG.

c) Paramètres de la Recherche Cuckoo

Cette partie permet de configurer les paramètres de l'algorithme Recherche Cuckoo :

- Nomber de Nids.
- Probabilité de découverte.
- Nombre d'itérations.
- Taille du Pas(ALPHA).

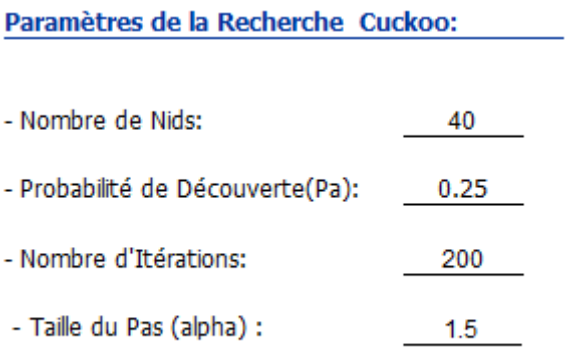


Figure 16: Paramètres de recherche cuckoo.

d) Les Résultats

- Avec l'algorithmeGénétique

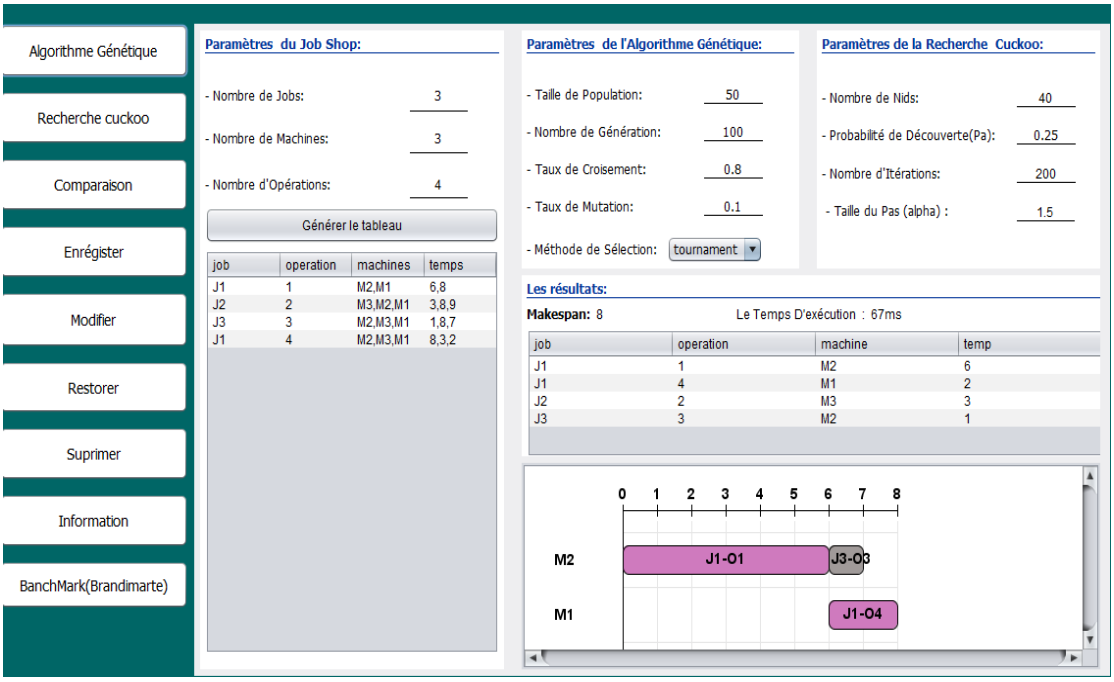


Figure 17: Interface du résultat avec l'algorithme génétique.

- Avec la Recherche Cuckoo

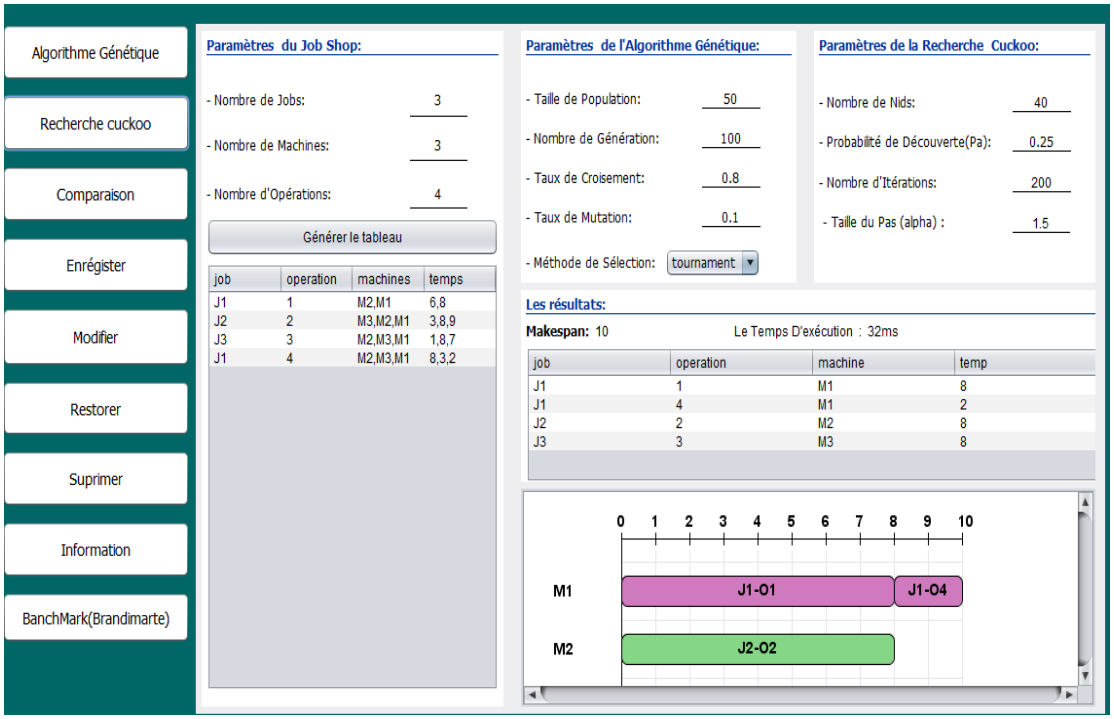


Figure 18: Interface du résultat avec Recherche cuckoo.

e) La comparaison

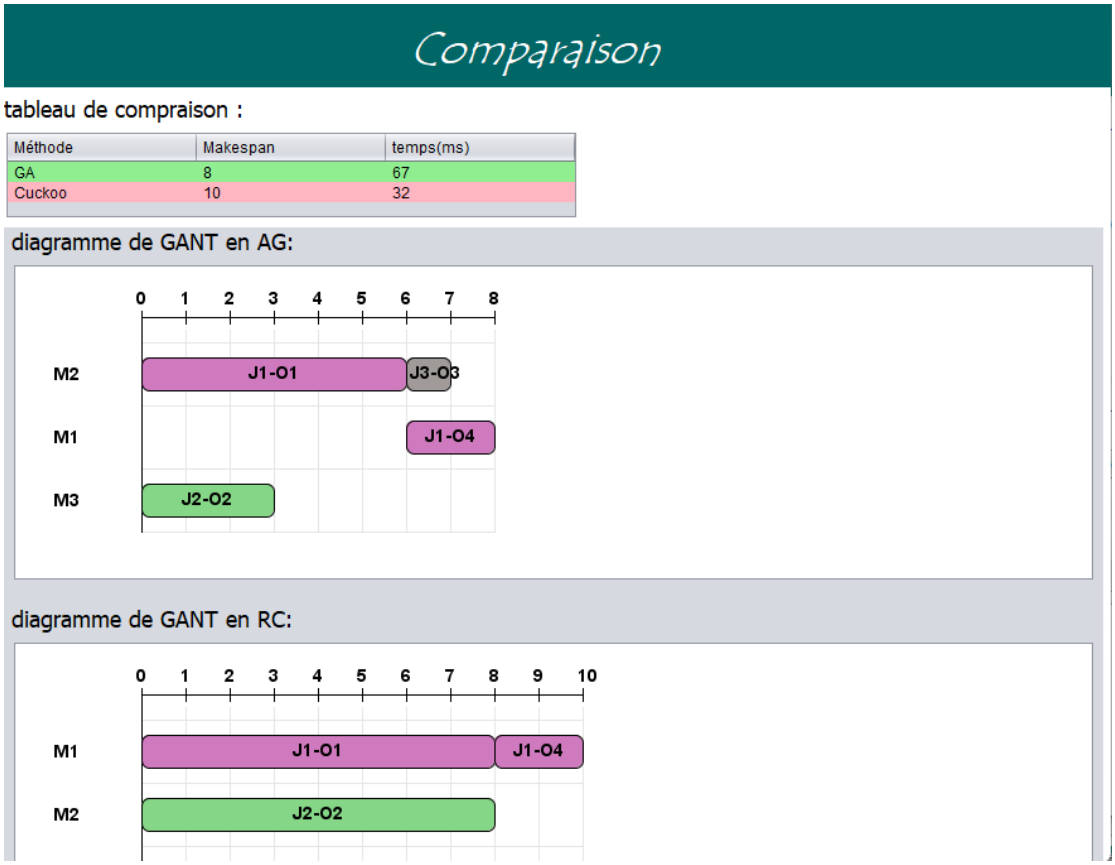


Figure 19: Interface de comparaison.

f) Données de Test (Benchmarks)

Afin d'évaluer les performances des algorithmes, nous avons utilisé des benchmarks standards du problème Flexible Job Shop Scheduling.

Les instances utilisées proviennent des Benchmarks de Brandimarte.

Ces benchmarks contiennent :

- Plusieurs jobs
- Plusieurs machines
- Plusieurs opérations avec différentes alternatives de machines

Les instances permettent d'évaluer :

- La qualité des solutions.
- La capacité de convergence.
- Les performances des métaheutistiques.

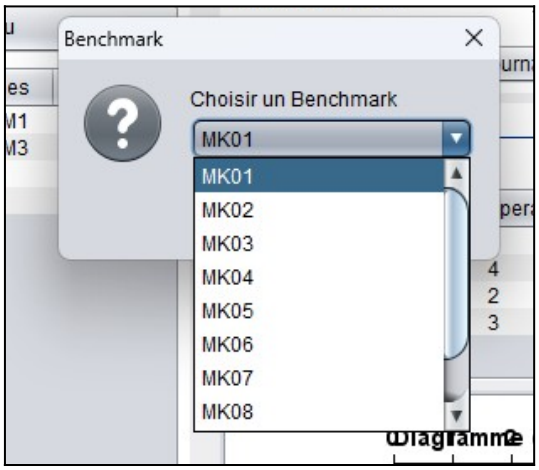


Figure 20 : Interface des Banchemark.

après la sélec

- **Avec l'algorithmeGénétique**

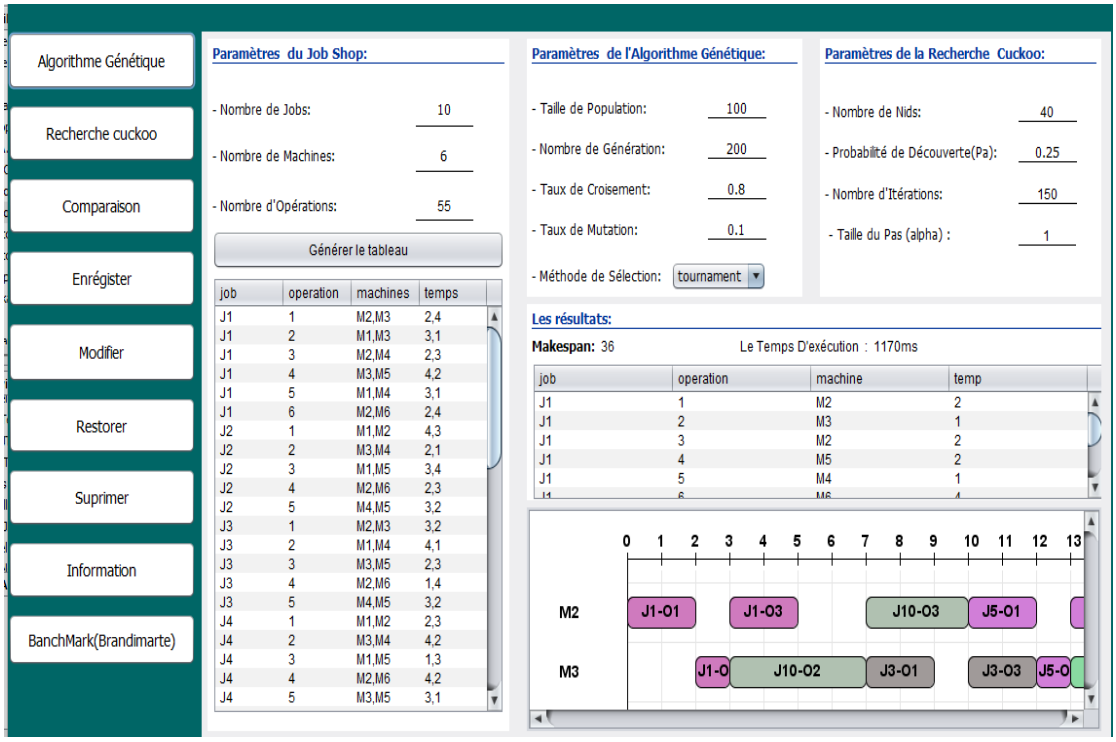


Figure 21: La solution de Banchemark en AG.

- **Avec Recherche Cuckoo**

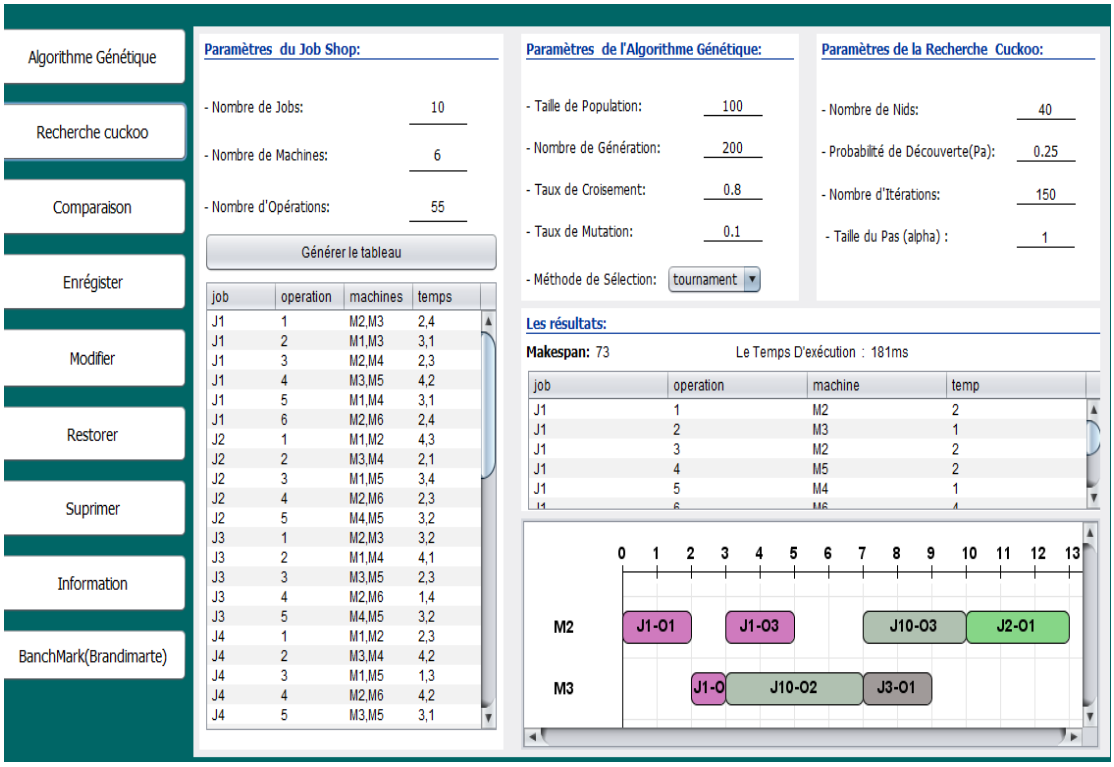


Figure 22: La solution de Banchemark en Recherche cuckoo.

9. Conclusion

Dans ce chapitre, nous avons présenté l'implémentation pratique de l'application développée pour la résolution du Flexible Job Shop SchedulingProblem.

Les expérimentations réalisées ont permis de comparer les performances du GeneticAlgorithm et de CuckooSearch.

Les résultats obtenus montrent que l'Algorithme Génétique fournit de meilleures solutions pour les instances testées, avec un Makespan plus faible et une meilleure convergence.

L'utilisation des diagrammes de Gantt a également permis de visualiser efficacement les solutions obtenues.

CONCLUSION GÉNÉRALE

Ce travail de projet de fin d'étude a été consacré à la modélisation, la conception et l'implémentation d'une plateforme logicielle dédiée à la résolution du problème d'ordonnancement d'atelier flexible (Flexible Job Shop SchedulingProblem - FJSP) en utilisant deux métaheuristiques : l'Algorithme Génétique (AG) et la Recherche Cuckoo (CS). Les objectifs fixés au départ ont été atteints avec succès, permettant d'apporter des réponses concrètes à notre problématique initiale.

Sur le plan conceptuel et théorique, la modélisation mathématique stricte des contraintes d'invisibilité des machines et de précédence des jobs a permis de garantir la validité des plannings générés. De plus, l'adoption d'un codage chromosomique dual (divisé en une partie affectation MS et une partie séquencement OS) s'est révélée particulièrement efficace pour structurer l'espace de recherche et éviter la génération de solutions irréalisables.

Sur le plan pratique et expérimental, le développement de l'application et la conception de son interface graphique ont permis de tester les deux algorithmes sur plusieurs instances et de dégager les conclusions comparatives suivantes :

L'Algorithme Génétique (AG) : Il a démontré une excellente robustesse globale. Grâce à des opérateurs de croisement avancés (tels que le POX), l'AG réussit à recombiner efficacement les sous-solutions performantes pour converger vers un Makespan optimal. Cependant, il requiert un temps de calcul relativement plus lourd et s'avère très sensible au réglage de ses nombreux paramètres (taux de mutation, de croisement, taille du tournoi).

La Recherche Cuckoo (CS) : Cet algorithme s'est distingué par sa rapidité de convergence et sa simplicité structurelle, nécessitant moins de paramètres à calibrer. L'intégration des vols de Lévy (Lévy Flights) a prouvé son efficacité en permettant à l'algorithme d'effectuer de grands sauts dans l'espace de recherche, évitant ainsi les pièges des optima locaux tout en affinant localement les solutions autour des meilleurs nids.

En somme, l'évaluation de notre application montre que la Recherche Cuckoo offre un excellent compromis pour une utilisation industrielle en temps réel grâce à sa rapidité et sa facilité de configuration, tandis que l'Algorithme Génétique demeure une approche puissante et stable lorsque le temps de calcul n'est pas la contrainte prioritaire.

Bien que le système développé soit pleinement fonctionnel et performant, cette étude ouvre la voie à plusieurs perspectives de recherche prometteuses :

L'Hybridation (Algorithme Hybride) : Combiner l'AG et la Recherche Cuckoo au sein d'un même algorithme afin de tirer parti de la capacité d'exploration globale de l'AG et de la puissance d'exploitation locale du CS.

L'Ordonnancement Dynamique : Étendre l'application pour qu'elle puisse réagir en temps réel aux perturbations de l'atelier, telles que les pannes soudaines de machines ou l'arrivée de commandes urgentes.

L'Optimisation Multi-objectif : Intégrer d'autres critères de performance dans l'évaluation, comme la minimisation des coûts énergétiques ou la charge maximale des machines, pour se rapprocher encore plus des réalités de l'usine du futur.

REFERENCES

REFERENCES

1. Michael P., Scheduling: Theory, Algorithms, and Systems, New York, USA
2. KamelB., Optimisation des flux et gestiond'ateliers, Mémoire de Magister, Université de Tlemcen, Algérie, 2012.
3. Stéphane D., Ordonnancementd'atelier : Modèles et algorithmes, Paris, France, 2011.
4. ChristianA., Ordonnancement de la production, Paris, France, 2008.
5. Kacem I., Ordonnancementmulticritère par les algorithmesévolutionnaires, Université de Lille, France, 2003.
6. BlazewiczJ., Scheduling Computer and Manufacturing Processes, Berlin, Allemagne, 2007.
7. FatihaS., Modélisation et résolution du problème de Job Shop Flexible, Mémoire de Fin d'Études, Université de Béjaïa, Algérie, 2021.
8. Glover F.,Future paths for integer programming and links to artificial intelligence , Computers & Operations Research, 1986.
9. Talbi E., Metaheuristics: from design to implementation, 2009.
10. BoussaïdI., A survey on optimization metaheuristics,Information Sciences, 2013.
11. Goldberg D., Genetic Algorithms in Search, Optimization and Machine Learning, 1989.
12. Gen M., & Cheng R., Genetic Algorithms and Engineering Design, 1997.
13. Blickl T., ThieleL.,A mathematical analysis of tournament selection, In Proceedings of the 6th International Conference on Genetic Algorithms, 1995.
14. De Jong K., An analysis of the behavior of a class of genetic adaptive systems, Doctoral dissertation, University of Michigan, 1975.
15. YangX., Deb S.,Cuckoo search via Lévy flights, In 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC),2009.
16. PavlykI., Lévy flights, non-local search and simulated annealing, Journal of Computational Physics, 226(2), 1830-1844, 2007.
17. Brucker P.,Schlie R., Job-shop scheduling with multi-purpose machines, Computing, 45(4), 1990.

18. Kacem I., Hammadi S., Borne P. ,Approach by localization and genetic algorithm for flexible job-shop scheduling, IEEE Transactions on Systems, Man, and Cybernetics, Part C,2002.
19. Pezzella F., Morganti G., CiaschettiG.,A genetic algorithm for the flexible job shop scheduling problem, Computers & Operations Research,2008.
20. Holland J., Adaptation in natural and artificial systems, MIT press, 1992.
21. Yang X., Deb S., Cuckoo search via Lévy flights,World Congress on Nature & Biologically Inspired Computing (NaBIC) , 1972.
22. Garey M.,Johnson D., Sethi R.,The complexity of flow-shop and job-shop scheduling, Mathematics of Operations Research, 1990.
23. Gen M., Cheng R., Genetic algorithms and engineering optimization, 2000.
24. Chaudhry I., Khan A.,A research survey: review of flexible job shop scheduling techniques, International Journal of Advanced Manufacturing Technology, 2016.
25. Fleurent C., Algorithme Génétique, Revue d'intelligence artificielle, vol.30, n°4, 2019.

ملخص

تتناول هذه المذكرة حل مشكلة جدولة ورشة العمل المرنة والتي تُعد واحدة من أعقد مسائل الأمثلية في الأنظمة الصناعية الحديثة، حيث تتطلب المشكلة تحديد التتابع الأمثل للعمليات وتخصيص الآلات المناسبة لها لتقليل الوقت الإجمالي المستغرق للإنتاج نظراً لصعوبة المشكلة من الناحية الحسابية.يقترح هذا العمل منهجاً فعالاً يجمع بين خوارزميتين من خوارزميات الذكاء الاصطناعي والميتاهرستيك هما الخوارزمية الجينية التي تتميز بقدرتها العالية على استكشاف فضاء الحلول على نطاق واسع، وخوارزمية البحث عن الوقواق الفعالة في تحسين الحلول محلياً وتجنب الوقوع في القيم الصغرى المحلية. تم تطوير النظام المقترح وبناء محاكي كامل للمشكلة باستخدام لغة Java وبيئة التطوير NetBeans

كلمات مفتاحية: جدولة ورشة العمل المرنة ، الخوارزمية الجينية، البحث عن الوقواق، الأمثلة التوافقية، لغة جافا.

Résumé

Ce mémoire examine la résolution du problème de planification d'atelier flexible, qui représente l'un des défis d'optimisation combinatoire les plus complexes dans les systèmes industriels modernes. Le problème consiste à déterminer la séquence optimale des opérations et à leur attribuer les machines adéquates afin de minimiser le temps total d'exécution. Étant donné la nature NP-difficile de ce problème, ce travail propose une approche efficace combinant deux métaheuristiques puissantes : l'Algorithme Génétique (GA), reconnu pour sa grande capacité d'exploration de l'espace de recherche, et l'algorithme de Recherche du Cuckou , particulièrement performant pour l'exploitation locale et l'évitement des optimums locaux. Le système proposé a été implémenté et validé en développant un simulateur complet sous l'environnement NetBeans en utilisant le langage Java.

Mots-clés : FJSP, Algorithme Génétique, CuckooSearch, Optimisation Combinatoire, Java, NetBeans.

Abstract

This thesis addresses the resolution of the Flexible Job Shop Scheduling Problem (FJSP), which stands as one of the most challenging combinatorial optimization problems in modern manufacturing systems. The objective is to determine the optimal sequence of operations and allocate them to the appropriate machines while minimizing the total workload completion time. Due to the NP-hard nature of the problem, this work proposes an effective approach combining two powerful metaheuristics: the Genetic Algorithm (GA), known for its strong exploration capabilities across the search space, and the Cuckoo Search algorithm, which excels at local exploitation and avoiding local optima. The proposed framework was fully implemented by developing a dedicated simulation tool using Java within the NetBeans IDE.

Keywords: FJSP, Genetic Algorithm, Cuckoo Search, Combinatorial Optimization, Java, NetBeans.