République Algérienne Démocratique et Populaire

Ministère de l'enseignement supérieur et de la recherche scientifique

Université 20 Aout 1955 – Skikda

Faculté des Sciences

Département d'Informatique

Mémoire en vue de l'obtention du diplôme de Master

Filière : Informatique

Option : Systèmes Informatiques

Thème

Étude et Mise en Œuvre d'Algorithmes d'Optimisation

Encadré par :

Pr. REDJIMI Mohammed

Présenté par :

M^me^ BOUAKBA Nora

JUIN 2025

Dédicace

JE DÉDIE CE TRAVAIL

A ma famille, qui a fait de moi ce que je suis aujourd'hui :

Spécialement ma défunte mère et mon défunt père que le bon dieu les accueille dans son vaste paradis. Ce sont eux, qui m'ont apporté leur soutien, que ça soit moral ou financier, et sans leur apport je n'aurais jamais pu poursuivre mes études ou faire quoi que ce soit dans cette vie, jalonnée de difficultés multiples.

A mes frères et sœurs, qui m'ont toujours soutenu et encouragé

A mon mari, qui a été un grand soutien, un apport psychologique et surtout un partenaire qui m'a beaucoup aidé dans la rédaction de cette thèse.

A mes deux enfants (Slimane et Tinhinane), qui me procurent la force et la joie de vivre

d

Remerciements

Je *tiens d'abord à remercier mon directeur de thèse,*

Le *Professeur* ***Mohammed Redjimi*** *pour avoir accepté de proposer, diriger, conseiller et surtout, montrer une patience hors pair pour mener à bien ce travail.*

Mes *remerciements, s'adressent également au professeur* ***Mazouzi Ismail*** *pour son et soutien.*

Mes remerciements vont également a monsieur ***Mansouri ziad****, maitre-assistant au département de technologie qui n'a pas hésité un seul instant à apporter sa précieuse contribution dans la réalisation de ce travail*

Sont *également destinataires de mes remerciements, messieurs les membres du jury qui n'ont pas hésité un seul instant à répondre présent pour être parmi nous le jour de la soutenance et surtout d'avoir consacré un temps très précieux dans la lecture de cette thèse, malgré leurs nombreuses occupations, surtout, pendant cette période de l'année.*

Je *ne veux surtout pas oublier les enseignants que j'ai eu l'honneur de suivre leur cours tout au long de mon parcourt, spécialement, messieurs* ***Bourmel, Boucheham et madame Hazmoune*** *ainsi que tous les enseignants et personnel administratif qui ont rendu possible ce privilège de présenter mon travail devant un parterre d'enseignants et de professeurs*

Résumé

Ce mémoire s'inscrit dans le cadre de l'optimisation, discipline centrale dans la résolution de problèmes complexes où il s'agit de rechercher une solution optimale respectant un ensemble de contraintes prédéfinies. L'optimisation joue un rôle déterminant dans divers domaines tels que la planification, l'affectation des ressources ou encore l'organisation de systèmes, notamment lorsqu'il est nécessaire de concilier efficacité, performance et faisabilité. Afin de résoudre ces problèmes, plusieurs algorithmes d'optimisation ont été développés. Ce travail se concentre sur deux approches complémentaires : la méthode AGL (Algorithme Glouton Local), qui construit une solution étape par étape en choisissant localement les meilleures options, et AGE (Algorithme Génétique Évolutionnaire) un méta heuristique inspiré des processus biologiques de sélection naturelle, permettant une exploration approfondie de l'espace des solutions. Ces deux approches sont évaluées en termes de performances, de temps de calcul, et de capacité à satisfaire des contraintes.

L'étude a été appliquée au cas concret de la génération automatique des emplois du temps universitaire, un problème de nature combinatoire, souvent **NP-difficile**, nécessitant la prise en compte de nombreuses contraintes : indisponibilités des enseignants, limitation du nombre de cours par jour, incompatibilités d'horaires ou de salles, pauses pédagogiques, etc.

Nous avons conçu un outil permettant d'évaluer et de maitre en œuvre deux techniques d'optimisation dans le but de produire un emploi du temps (EDT) satisfaisant les contraintes académiques

Les résultats obtenus mettent en évidence que, si la méthode gloutonne permet une génération rapide de solutions acceptables dans des cas simples, les algorithmes génétiques offrent de meilleures performances en matière de qualité de solution et de satisfaction des contraintes dans des scénarios plus complexes.

Cette étude montre l'intérêt des approches heuristiques et évolutives dans la résolution efficace de problèmes organisation efficace en milieu universitaire.

Mots-clés : Optimisation, complexité algorithmique, algorithme glouton, algorithme génétique, emploi du temps universitaire, planification de tâches, contraintes.

Summary

This thesis is part of the field of optimization, a key discipline in solving complex problems that involves searching for an optimal solution while satisfying a set of predefined constraints. Optimization plays a crucial role in various fields such as planning, resource allocation, and system organization, especially when efficiency, performance, and feasibility need to be balanced.

To address these problems, several optimization algorithms have been developed. This work focuses on two complementary approaches: the Local Greedy Algorithm (LGA), which builds a solution step by step by choosing the best local options, and the Evolutionary Genetic Algorithm (EGA), a metaheuristic inspired by biological processes of natural selection, allowing for an in-depth exploration of the solution space. These two approaches are evaluated in terms of performance, computation time, and their ability to satisfy constraints.

The study was applied to the real-world case of automatic university timetable generation a combinatorial, often NP-hard problem that requires consideration of many constraints: teacher's unavailability, limit on the number of classes per day, scheduling and room conflicts, pedagogical breaks, etc.

We developed a tool to evaluate and implement both optimization techniques with the goal of producing a timetable that satisfies academic constraints.

The results highlight that while the greedy method allows for quick generation of acceptable solutions in simpler cases, genetic algorithms provide better performance in terms of solution quality and constraint satisfaction in more complex scenarios.

This study demonstrates the value of heuristic and evolutionary approaches in the efficient resolution of organizational problems in academic settings.

Keywords: Optimization, algorithmic complexity, greedy algorithm, genetic algorithm, university timetable, task scheduling, constraints.

الملخّص

يندرج هذا البحث ضمن مجال الأمثلة، وهو فرع محوري في معالجة المشكلات المعقّدة التي تستدعي إيجاد حلول مثلى ضمن إطار من القيود المحددة سلفًا. وتكتسي تقنيات الأمثلة أهمية بالغة في العديد من المجالات التطبيقية كالتخطيط، وتوزيع الموارد، وتنظيم الأنظمة، خصوصًا عندما يستوجب الأمر تحقيق توازن دقيق بين الكفاءة، والأداء، وقابلية التنفيذ.

من أجل معالجة هذا النوع من المشكلات، تم تطوير عدد من الخوارزميات المتخصصة في الأمثلة. ويركّز هذا العمل على تحليل ومقارنة نهجين متكاملين: الخوارزمية الجشعة المحلية(AGL) التي تعتمد على بناء الحل تدريجيًا من خلال اختيار الحلول المثلى محليًا في كل مرحلة، والخوارزمية الجينية التطورية(AGE) التي تندرج ضمن الخوارزميات الميتا-إرشادية، وهي مستلهمة من آليات الانتقاء الطبيعي في علم الأحياء، وتتيح استكشافًا موسّعًا لفضاء الحلول الممكنة. وقد تم تقييم هذين النهجين بناءً على معايير الأداء، وزمن المعالجة، ومدى القدرة على احترام القيود المفروضة

طُبّقت الدراسة على حالة تطبيقية واقعية تتمثّل في التوليد الآلي لجداول التوقيت الجامعية، وهي مشكلة ذات طبيعة تركيبية، تصنّف غالبًا ضمن فئة المشكلات-NP الصعبة، وتقتضي مراعاة عدد كبير من القيود كعدم توفر الأساتذة، وتحديد عدد الحصص اليومية، والمعارضات الزمنية والمكانية، وفترات الراحة البيداغوجية، وغيرها من المتطلبات

لقد تم تطوير أداة برمجية لتقييم وتطبيق النهجين المعتمدين في هذه الدراسة، بهدف توليد جداول زمنية تستجيب للقيود الأكاديمية المفروضة

وقد أظهرت النتائج التجريبية أنّ المنهج الجشع يسمح بالحصول على حلول مقبولة بسرعة في الحالات البسيطة، في حين تبرز الخوارزميات الجينية كخيار أكثر فعالية في إنتاج حلول ذات جودة أعلى وقدرة أفضل على تلبية القيود في السيناريوهات المعقدة

تُبرز هذه الدراسة الجدوى العملية لاعتماد المقاربات الإرشادية والتطورية في معالجة المشكلات التنظيمية المعقّدة، لاسيّما في السياقات الأكاديمية الجامعية

الكلمات المفتاحية: الأمثلة، التعقيد الخوارزمي، الخوارزمية الجشعة، الخوارزمية الجينية، الجداول الزمنية الجامعية، تخطيط المهام، القيود

Table des Matières

CHAPITRE II : Les Méthodes heuristiques

CHAPITRE III : Contribution

Liste des Tables et Figures

Liste des Tableaux

Acronymes

Symbole	Désignation
AGE	Algorithme Génétique
MGL	Méthode Gloutonne
ETU	Emploi du Temps Universitaire
TSC	Taux de Satisfaction des Contraintes
PSO	Optimisation Par essaim de particules
NP	Non Déterministe Polynomial
MATLAB	Matrix Laboratory

Introduction Générale

Introduction Générale

Dans le domaine de la résolution de problèmes, il est essentiel de distinguer entre les problèmes simples et ceux qui sont plus complexes. Pour les problèmes simples, des méthodes directes peuvent suffire, tandis que les problèmes plus complexes exigent, souvent, des approches plus sophistiquées. L'objectif est toujours de trouver la solution optimale qui répond aux besoins de tous les utilisateurs, ce qui peut impliqué d'explorer plusieurs solutions possibles avant de prendre la décision finale.

L'optimisation est une branche des mathématiques appliquées et de l'informatique qui consiste à rechercher la meilleure solution possible à un problème donné, parmi un ensemble de solutions admissibles, tout en respectant certaines contraintes. Elle vise généralement à minimiser un coût ou à maximiser un gain, selon une ou plusieurs fonctions objectives **[1]**.

L'optimisation est appliquée dans des domaines variés tels que la production industrielle, la logistique, les réseaux ou encore l'intelligence artificielle. Elle constitue un levier fondamental pour améliorer l'efficacité des systèmes complexes. De ce fait, plusieurs algorithmes ont été développés pour répondre à différents types de problèmes, qu'ils soient linéaires, non linéaires, combinatoires ou dynamiques.

Ces algorithmes diffèrent selon plusieurs critères, tels que le choix des données, les objectifs à atteindre et la nature des problèmes à résoudre, certains privilégient la rapidité d'exécution, comme la méthode gloutonne, tandis que d'autres, plus complexes, comme les algorithmes génétiques ou les colonies de fourmis, cherchent à explorer plus largement l'espace des solutions pour obtenir des meilleures solutions afin d'obtenir de meilleurs résultats.

Toutefois, la performance théorique d'un algorithme ne garantit pas à elle seule son efficacité dans des situations réelles ; c'est pourquoi la mise en œuvre concrète de ces algorithmes revêt une importance particulière. Elle permet de tester leurs comportements face à des contraintes réelles, d'ajuster les paramètres et de développer des interfaces aux utilisateurs finaux. La mise en œuvre constitue une étape essentielle pour passer de la théorie à la pratique.

Le travail entrepris, dans ce mémoire, consiste à étudier les différences entre deux méthodes de résolution de problèmes à travers l'analyse des fonctions objectives projetées.

Plus précisément, l'objectif a été de comparer deux approches d'optimisation : la méthode Gloutonne et l'algorithme Génétique. Ces deux techniques ont été choisies pour leur nature contrastée, l'une étant simple et directe, l'autre inspirée de mécanismes évolutifs.

Cette comparaison permet de mettre en évidence leurs performances respectives dans la résolution du problème de l'emploi du temps universitaire, en s'éloignant volontairement de la méthode manuelle traditionnelle, souvent longue et peu efficace face à la complexité croissante des contraintes académiques. L'un des problèmes d'optimisation les plus concrets et récurrents est celui de la génération automatique d'emplois du temps universitaires. Cette mise en œuvre nécessite notamment la définition précise des ressources humaines et matérielles : enseignants, groupes, niveaux, horaires, préférences et salles.

Il s'agit d'un problème combinatoire de type **NP-difficile**, dans lequel il faut affecter des cours selon des créneaux horaires et des salles, en tenant compte de nombreuses contraintes : la disponibilité des enseignants, l'absence de conflits entre cours, le nombre limité de salles, la cohérence pédagogique, les préférences dans les horaires, etc. La résolution manuelle de ce problème est chronophage (prends beaucoup de temps) et souvent sujette à des erreurs ou à des compromis insatisfaisants.

Une implémentation réussie permet non seulement d'obtenir des solutions réalisables, mais aussi d'analyser la performance comparative des différentes approches en termes de qualité, de rapidité et de taux de satisfaction des contraintes pédagogiques, humaines et logistiques.

L'objectif est de développer une solution capable de générer automatiquement un emploi du temps cohérent, tout en comparant l'efficacité de différentes approches algorithmiques, notamment la méthode Gloutonne (**GL**) et l'algorithme Génétique(**GE**).

D'après notre expérience ainsi que plusieurs autres de la littérature, les résultats montrent que la gestion des contraintes constitue un point de divergence majeur entre la méthode gloutonne et la méthode génétique.

La méthode gloutonne, bien qu'efficace pour respecter des contraintes simples comme l'absence de chevauchement de cours ou la limitation du nombre de séances par jour, atteint rapidement ses limites dès que le problème devient plus complexe (préférences d'enseignants, contraintes temporelles strictes, ou optimisation globale).

En revanche, la méthode génétique s'adapte mieux à des contraintes multiples et hiérarchisées grâce à sa capacité d'exploration de solutions diverses au fil des générations. Dans plusieurs

études **[2]** la méthode génétique a permis de satisfaire plus de 90 % des contraintes, y compris les préférences individuelles, là où la méthode gloutonne n'en respectait qu'environ 70 à 80 %. Ainsi, plus le système d'emploi du temps contient de règles complexes, plus l'avantage de l'approche génétique ne devient évident.

Un autre aspect fondamental dans la comparaison des algorithmes d'optimisation est le temps d'exécution. En effet, au-delà de la qualité des solutions produites, un algorithme doit également être capable de fournir une solution dans un délai raisonnable, surtout lorsqu'il est destiné à être utilisé dans un cadre pratique et imminent. Dans la génération d'un emploi du temps, la méthode gloutonne se distingue souvent par sa rapidité, puisqu'elle suit une logique de choix simple et immédiat. En revanche, l'algorithme génétique, bien que généralement plus performant en termes de qualité des résultats, nécessite plus de temps pour converger, car il repose sur des itérations successives de sélection, croisement et mutation. Le choix entre ces deux approches dépend ainsi d'un compromis entre la précision des solutions et les contraintes temporelles du système cible.

Ce mémoire s'inscrit dans cette problématique et propose la mise en œuvre de deux types d'algorithmes d'optimisation : la méthode gloutonne et l'algorithme génétique, un méta heuristique inspirée du processus de sélection naturelle. Ces deux approches sont comparées dans le cadre de la génération d'emplois du temps universitaire, à travers une application développée sous **MATLAB R2017a**, dotée d'une interface graphique intuitive pour la saisie, la visualisation et l'évaluation des résultats.

Problématique

Dans les établissements d'enseignement supérieur, la planification manuelle des emplois du temps présente une tâche complexe, fastidieuse et sujette aux conflits.

Cette complexité augmente avec le nombre de matières, d'enseignants, les groupes étudiants et autres contraintes institutionnelles ou humaines à prendre en compte. Face à ces défis, se pose la question centrale suivante :

Comment concevoir un système d'optimisation capable de générer automatiquement un emploi du temps universitaire réaliste, tout en respectant un maximum de contraintes pédagogiques et logistiques ?

Ce questionnement se décline à travers les interrogations suivantes :

Quelle est la performance comparative entre une méthode simple comme la méthode gloutonne et une approche plus sophistiquée comme l'algorithme génétique dans ce contexte applicatif ?

Quel est le taux de satisfaction des contraintes pour chaque méthode ?

Comment concevoir un outil logiciel interactif, utilisable par un personnel non spécialiste, pour générer, modifier et évaluer des emplois du temps ?

À travers l'étude et la comparaison de ces deux approches, l'objectif de ce travail est de proposer une solution algorithmique et logicielle efficace, fiable et adaptée aux besoins des établissements universitaires.

Organisation du document

Ce mémoire est organisé selon quatre chapitres comme suit :

- Le premier chapitre : présentation générale du contexte et de la problématique des méthodes optimisation, et ces algorithme et de la génération des emplois du temps, ainsi que les objectifs de l'étude.

- Le deuxième chapitre : Revu bibliographique des différentes méthodes d'optimisation, tel que le classement d'algorithme d'optimisation comme les méthodes exactes et heuristique

- Le troisième chapitre : Description de la méthodologie adaptée, incluant la modélisation du problème, les contraintes prises en compte, et les algorithmes utilisées (AGE, AGL)

- Le quatrième chapitre : Présentation des résultats obtenus, analyse comparative des performances d'algorithme appliqué, et discussion.

CHAPITRE I

Généralités

1- Introduction

Dans le monde réel, avec l'évolution de la technologie, les problèmes sont passés de simples a de plus en plus complexes ; les choses se compliquent au point où il n'existe parfois pas de solutions exactes et cela est dû aux exigences sans cesse croissantes de la vie courante.

Le but de l'optimisation est de minimiser des coûts, des temps, la consommation d'énergie, ou maximiser un profit, une performance, une efficacité, selon des contraintes techniques, économiques ou environnementales éventuelles.

Ces problèmes d'optimisation sont confiés à des algorithmes qui jouent un rôle majeur dans le monde réel. Dans les approches usuelles, les solutions sont codées et les algorithmes vont chercher à trouver des solutions consistant à maximiser ou minimiser des fonctions dont les objectifs répondent à un ensemble de contraintes liées aux problèmes à résoudre.

Ce premier chapitre, donne un aperçu le plus détaillé possible concernant les problèmes d'optimisation en insistant sur les notions de base ainsi que sur les aspects les plus importants et certaines des solutions les plus adaptées aux problèmes posés.

Enfin, nous mettrons en évidence l'importance de ce travail dans plusieurs domaines, notamment celui de la sante, du génie civil et de l'informatique en général.

2- Définitions

Nous présentons, ici, quelques termes que nous jugeons utiles pour la compréhension du travail qui sera abordé dans la suite :

2-1 L'optimisation : consiste à trouver la meilleure solution possible parmi un ensemble de solutions, en maximisant ou en minimisant une fonction objectif, tout en respectant des contraintes spécifiques.

2-2 Les variables de décision : Ce sont les variables recherchées dans le problème. Ces variables forment un vecteur qui s'appelle vecteur de décision.

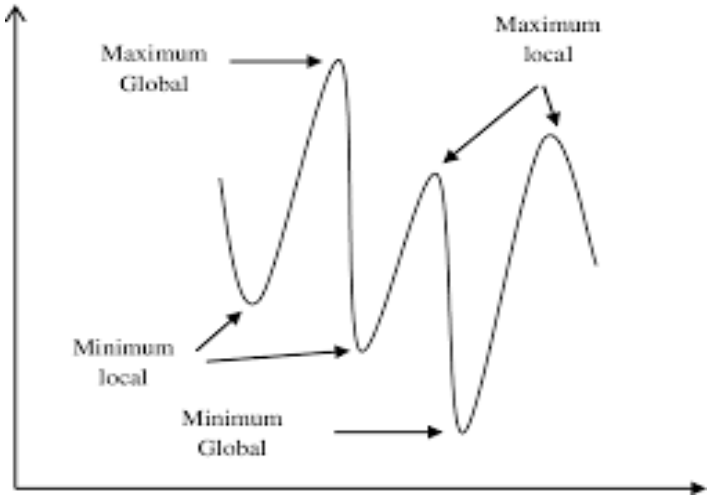
2-3 La dimension du problème : En général, le nombre de composantes du vecteur de décision constitue la dimension du problème à résoudre **[3].**

2-4 L'espace de recherche et les solutions admissibles : Une solution admissible est une affectation de toutes les variables de décision, qui répond aux contraintes du problème. L'ensemble X de dimension n contient les valeurs que le vecteur X peut prendre. Ce dernier est

connu sous différentes appellations : ensemble admissible, espace de solutions ou encore espace de recherche **[3]**.

2-5 <u>Les contraintes</u> : Une contrainte est une condition que doivent respecter les vecteurs de décision du problème dans l'espace de recherche X est limité par quelques règles **[3]**.

2-6 <u>Solution optimale</u> : C'est la meilleure solution au problème posé, sachant qu'il existe différentes solutions selon la fonction objective. Dans le cas de la minimisation ou de la maximisation, on trouve des solutions locales ou globales **[3]**. Comme illustré sur la **Figure 1.1** ci-dessous :



<u>Figure 1.1</u> : Max et Min locaux et globaux [3].

2-7 <u>Fonction objectif</u> :

Une telle fonction sert de critère pour déterminer la meilleure solution au problème à valeur scalaire. Elle est appelée fonction objectif (fonction fitness, critère, etc.). Elle attribue à chaque solution $x \in X$ de l'espace de recherche, un nombre réel indiquant sa valeur (son score).

Si on cherche à minimiser la fonction objectif ; c'est à dire que l'on cherche le vecteur x qui est une borne inferieure dans l'ensemble X, la fonction objectif est connue comme une fonction de coût. Par contre si on cherche à maximiser la fonction objectif, tel que x soit une borne supérieur de X, dans ce cas la fonction objectif est connu comme une fonction d'utilité ou de profit Une telle fonction sert de critère pour déterminer la meilleure solution du problème à

valeur scalaire. Elle est appelée fonction objectif (fonction fitness, critère, etc.). Elle attribue à chaque solution x ∈ X de l'espace de recherche, un nombre réel indiquant sa valeur (son score). Si on cherche à minimiser la fonction objectif s'est à dire que l'on cherche le vecteur x qui est une borne inferieure dans l'ensemble X, la fonction objectif est connue comme une fonction de coût.

Par contre si on cherche à maximiser la fonction objectif, tel que x soit une borne supérieur de X, dans ce cas la fonction objectif est connu comme une fonction d'utilité ou de profit **[4]**.

2-8 Voisinage : Le voisinage de la solution x est l'ensemble des solutions voisines de x et il est défini comme une légère modification de la solution. Cette modification est appelée aussi mouvement.

3- Définition de L'optimisation

L'optimisation est un domaine riche et varié qui offre de nombreuses méthodes pour résoudre des problèmes complexes.

L'optimisation est un processus qui consiste à rendre quelque chose aussi efficace, performant ou fonctionnel que possible. Elle vise à améliorer les résultats en utilisant les ressources de manière plus efficace dans divers domaines **[4].**

4- L'importance de L'optimisation:

Efficacité accrue: L'optimisation permet d'améliorer l'utilisation des ressources, qu'il s'agisse de temps, d'argent, de main-d'œuvre ou de matériaux. Cela peut conduire à des économies significatives.

Amélioration des performances : En optimisant un processus ou un produit, on peut améliorer ses performances, que ce soit en termes de vitesse, de qualité ou de fiabilité.

Prise de décision éclairée : L'optimisation aide à analyser différentes options et à choisir la meilleure solution en fonction de critères spécifiques, ce qui conduit à des décisions plus informées.

Satisfaction du client : En optimisant les produits ou services, les entreprises peuvent mieux répondre aux besoins et attentes de leurs clients, ce qui peut améliorer la satisfaction et la fidélité.

Compétitivité: Dans un environnement commercial concurrentiel, l'optimisation peut offrir un avantage concurrentiel en permettant à une entreprise de se démarquer par ses coûts, sa qualité ou son innovation.

Innovation: Le processus d'optimisation peut également conduire à de nouvelles idées et solutions en remettant en question les méthodes existantes et en cherchant des moyens d'améliorer les résultats.

Durabilité: L'optimisation peut contribuer à des pratiques plus durables en réduisant le gaspillage et en améliorant l'efficacité énergétique, ce qui est de plus en plus important dans le contexte des préoccupations environnementales.

5- Les différents Domaine qui utilisent de l'optimisation :

L'optimisation est largement utilisée dans des domaines variés pour améliorer les performances, réduire les couts et prendre des décisions efficaces face à diverses contraintes telles que :

-**Économie** : En minimisant les coûts de production et maximisant le profit d'une entreprise.

-**Finance** : Optimisation de portefeuille (Maximisation) et minimisation du coût d'un emprunt.

- **Marketing** : Maximisation du retour sur investissement publicitaire et le chiffre d'affaires ou le bénéfice en choisissant le bon prix.

- **Logistique** : Minimiser le coût total de transport entre plusieurs entrepôts et clients

- Minimisation des tournées de livraison en minimisant la distance totale parcourue ou le temps

5.1 Domaine de la santé :

- des traitements médicaux, par exemple, en déterminant la meilleure combinaison de médicaments pour un patient.

- Planification du personnel médical (Minimisation).

- Optimisation de la dose de médicament (Minimisation).

- Affectation des patients aux lits (Minimisation).

-Planification : Optimisation des horaires de travail, de production ou d'événements pour maximiser l'efficacité.

5.2 <u>Domaine informatique</u> : L'optimisation dans le domaine informatique est cruciale pour plusieurs raisons :

- Performance : L'optimisation permet d'améliorer la vitesse d'exécution des programmes et des algorithmes. Cela est particulièrement important dans des applications où le temps de réponse est critique, comme dans les systèmes en temps réel, les jeux vidéo, ou les applications financières.

- Utilisation des ressources : Une bonne optimisation aide à réduire l'utilisation des ressources matérielles, comme la mémoire et le processeur. Cela peut conduire à des économies de coûts.

- Expérience utilisateur : Une application optimisée offre une meilleure expérience utilisateur, avec des temps de chargement plus rapides et une interface plus réactive. Cela peut augmenter la satisfaction des utilisateurs et leur fidélité.

- **Durabilité** : L'optimisation peut également contribuer à la durabilité en réduisant la consommation d'énergie des systèmes informatiques **[5]**.

- **Sécurité** : Dans certains cas, l'optimisation peut également renforcer la sécurité des systèmes. Par exemple, des algorithmes optimisés peuvent réduire les vulnérabilités en limitant les points d'attaque potentiels.

- **Coût de développement** : L'optimisation peut également réduire le coût de développement à long terme.

5.3 <u>Domaine du génie civil</u> :

- **Conception efficace des infrastructures** : L'optimisation des conceptions structurelles permet de créer des bâtiments et des infrastructures plus sûrs, durables et économiques. Cela inclut l'utilisation de matériaux appropriés et la minimisation des déchets.

- **Gestion des projets** : L'optimisation des plannings et des ressources dans les projets de construction permet de respecter les délais et les budgets, tout en garantissant la qualité des travaux.

- **Amélioration de la durabilité** : L'optimisation des processus de construction et des matériaux utilisés contribue à réduire l'impact environnemental des projets de génie civil, en favorisant des pratiques plus durables.

- **Analyse des risques** : L'optimisation des modèles d'analyse des risques permet d'identifier et de minimiser les dangers potentiels associés aux projets de construction, garantissant ainsi la sécurité des travailleurs et des usagers.

6- Les algorithmes d'optimisation

6-1 Définition des algorithmes d'optimisation

Les algorithmes d'optimisation sont des méthodes mathématiques et informatiques utilisées pour trouver la meilleure solution possible à un problème donné, souvent sous certaines contraintes.

Cela peut signifier souvent en maximisant ou en minimisant une fonction objective **[6]**.

6-2 Historique des algorithmes d'optimisation

L'optimisation est un domaine en constante évolution, avec des applications dans de nombreux secteurs, y compris l'ingénierie, l'économie, la logistique et l'intelligence artificielle.

L'histoire des algorithmes d'optimisation remonte à plusieurs siècles et a évolué avec les avancées des mathématiques, de l'informatique et des sciences appliquées **[7].**

Voici un aperçu des principales étapes et développements dans ce domaine :

- Avant les années ***1940***, la première base pour l'optimisation et d'origine mathématique comme par exemple l'optimisation des fonctions continues avec ***Newton et Leibniz*** où pour des problèmes simples ils sont utilisés des méthodes analytiques classiques, comme par exemple maximiser une aire, minimiser une distance, etc.
- Entre les années ***1940*** jusqu'à ***1950***, ils y a la naissance des méthodes exactes comme la programmation linéaire par la méthode du simplexe inventée par ***George Dantzig*** en *1947* et en 1950 le début de la programmation dynamique inventée par ***Bellman.*** Pour les problèmes entiers par ***Branch and Bound.*** **[8].**

  Dans les années ***1960*** à ***1970,*** on constate une amélioration des algorithmes exacts et les limites apparaissent pour les problèmes NP-difficiles (combinatoires), c'est le début de l'apparition des algorithmes **heuristiques**.

Par exemple la recherche locale et ***Gloutonne*** **[8].**

- Durant les années ***1980***, on a assisté à la naissance de l'heuristique avancée comme le recuit simulé, la recherche ***Tabou*** et la première approche hybride qui combine plusieurs techniques.
- Dans les années ***90*** il y'a eu l'apparition des Méta heuristique inspiré de la nature.
- Et enfin, depuis l'année ***2000*** et jusqu'à aujourd'hui, l'utilisation de méthode issues de intelligence artificielle et de l'optimisation hybride sont appliquées aux problèmes d'optimisation **[8].**

6-3 Comment choisir un algorithme d'optimisation:

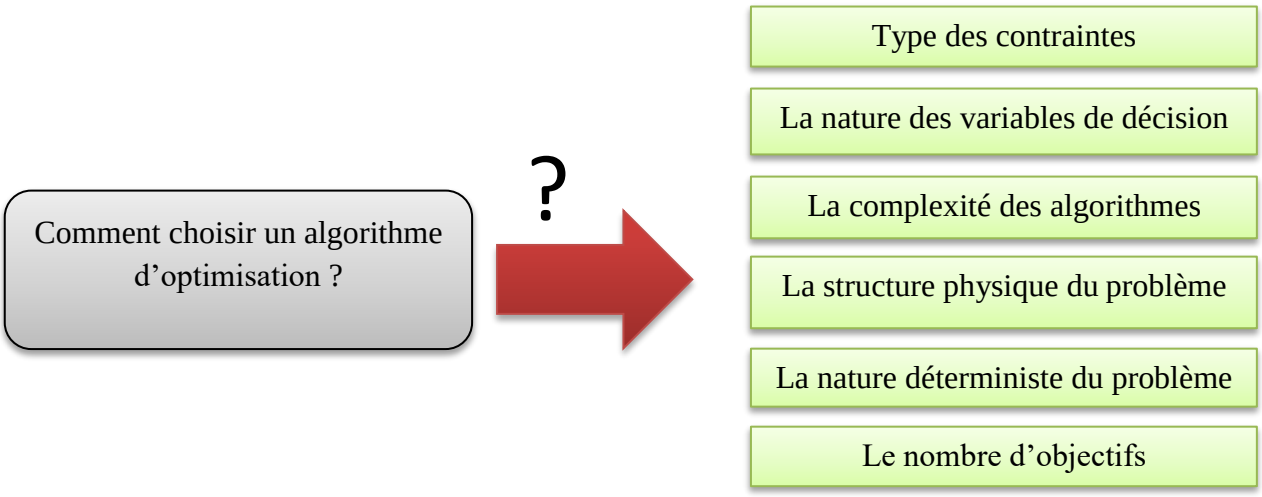


Figure 1.2 : comment choisir un Algorithme d'optimisation

6-3-1 Classification des Problèmes :

On peut classer tout problème dans l'une des trois classes : **P, NP ou NP-Complet**, Sachant qu'un problème possède en général plusieurs solutions, nous parlons de :

- **Classe P** : Un problème est dans P s'il possède un algorithme déterministe et d'ordre polynomial par rapport à la taille des données **n**. On qualifie alors le problème de polynomial. La classe **P** est la classe des problèmes dits faciles **[9].**
- **Classe NP** : Est la classe des problèmes dont on ne connait pas l'existence d'un algorithme déterministe polynomial, mais possédant un algorithme non-déterministe permettant la vérification d'un cas de solution en un temps polynomial par rapport à la taille des données **[9].** Notez que NP signifie « non-déterministe polynomial » (et non pas non-polynomial)

- **Classe NP-Complet** : est composée des problèmes dits les plus difficiles de NP. Ils sont dits aussi : Noyau dur des problèmes de NP.

On dit qu'un problème est NP-Complet si sa résolution en temps polynomial entraînerait la résolution en temps polynomial de tout problème de NP [9].

6-3-2 La complexité des algorithmes

La complexité exprime le coût d'un algorithme ou bien la quantité de ressources consommées par l'algorithme en fonction de la taille des données.

Il existe deux types de la complexité :

1- **Complexité temporelle** : dépend du temps consommé par algorithme,
2- **Complexité spatiale** : dépend des quantités mémoire consommées par cet algorithme.

On calcule la complexité comme suit :

1- **La complexité pratique d'un algorithme :** par la mesure du temps d'exécution ou l'espace mémoire ou bien par la nature des données utiliser,
2- **La complexité théorique d'un algorithme :** on la mesure en fonction de la taille des données pour estimer le temps, coût ou la mémoire consommée.

Dans ce cas-là, on utilise la technique de calcul des coûts théoriques ou pratiques ou bien les principales règles de comptabilisation associées aux structures algorithmiques fondamentales **[10]**.

6-3-3 Types des contraintes : Le schéma si dessous montre les différent type des contraintes :

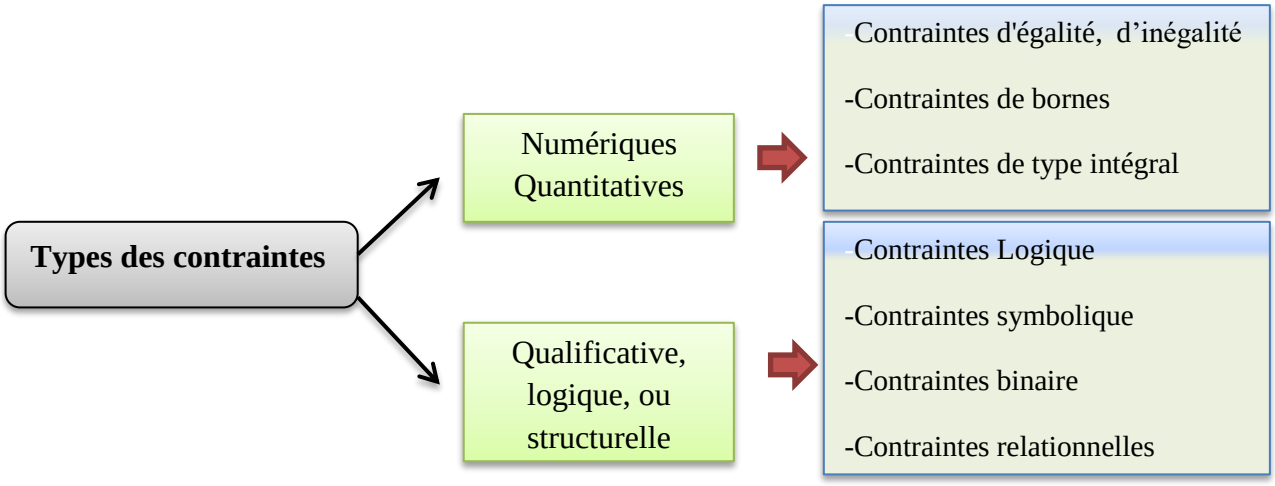


Figure 1.3: les différents types des contraintes

Les algorithmes d'optimisation utilisent différentes types de contraintes pour modéliser les problèmes qu'ils cherchent à résoudre.

Ces contraintes se sont des règles ou limites qu'une solution doit respecter, des ressources ou en assurent la compatibilité entre ces contraintes et la fonction objective.

Les algorithmes d'optimisation, comme la programmation linéaire, la programmation non linéaire, ou les algorithmes génétiques, utilisent ces contraintes pour trouver des solutions optimales.

Les algorithmes d'optimisations peuvent classées selon plusieurs manières la figures ci-dessus résume le principale classe des méthodes d'optimisations.

7- Classification des méthodes d'optimisation

En peu classer les méthodes d'optimisations selon plusieurs critères comme la figure ci-dessus nous le montre :

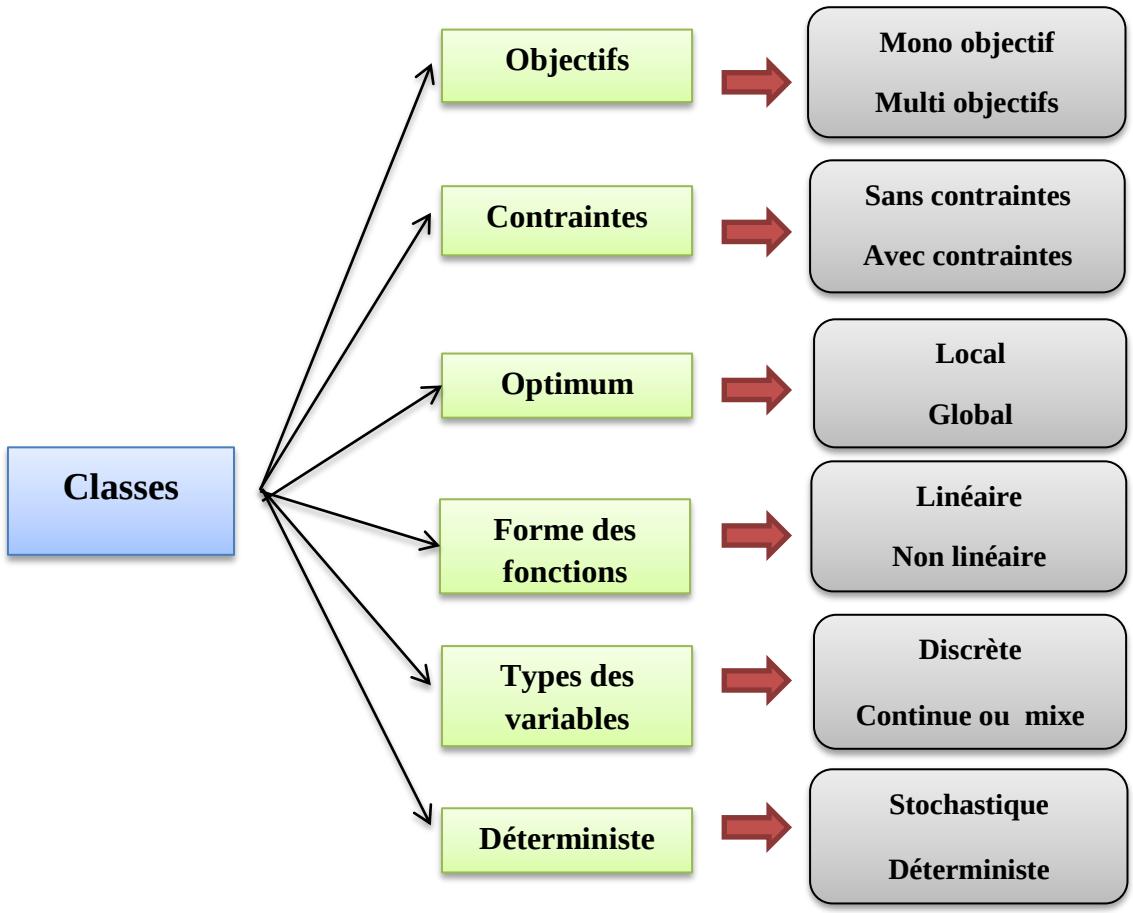


Figure1.4: Classification des problèmes d'optimisations **[11]**

On va citer quelques méthodes d'optimisations avec leurs algorithmes utilisés pour résoudre ces problèmes **[11].**

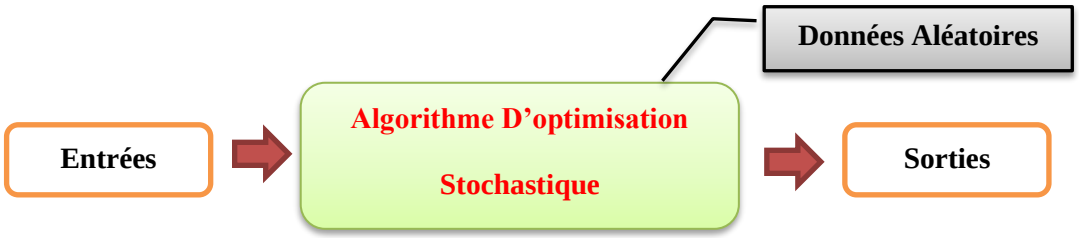
7-1 <u>Déterministe ou Stochastique</u> :

7-1-1 Déterministe :

Un algorithme déterministe est un algorithme qui produira toujours les mêmes sorties en passant toujours par la même séquence d'états d'exécution **[12].**

Exemple d'algorithmes : **Dijkstra, Simplexe**, etc.

7-1-2 <u>Stochastique</u> :



Modèle qui a un caractère aléatoire soit la fonction objective est aléatoire ou les contraintes sont aléatoires, ou en obtient comme résultat des décisions probabilistes. On utilise cette méthode lorsqu'on a des données bruitées ou incertaines, ou lorsque le problème est trop complexe.

Exemple d'algorithmes: Recuit simulé, Génétique, etc.

7-2 <u>Mono objectifs ou multi objectif</u> : ces deux approches sont fondamentales en optimisation.

Lorsqu' on cherche à optimiser une seule fonction objectif donc en cherche à résoudre un problème dit mono objectif, dans les autres cas ce sont des problèmes multi objectifs ou multicritères qui reposent sur plusieurs fonction objectifs.

<u>Exemple d'algorithmes Mono Objectifs</u> : algorithme de ***Dijkstra***, dans le cas de la recherche du chemin le plus court entre deux nœuds dans un graphe pondéré [13].

Exemple d'algorithmes Multi Objectifs : Optimisation par essaim particulaire, recuit simulé pour explorer l'espace des solutions en tenant compte de plusieurs objectifs.

7-3 Uni modale ou multimodale :

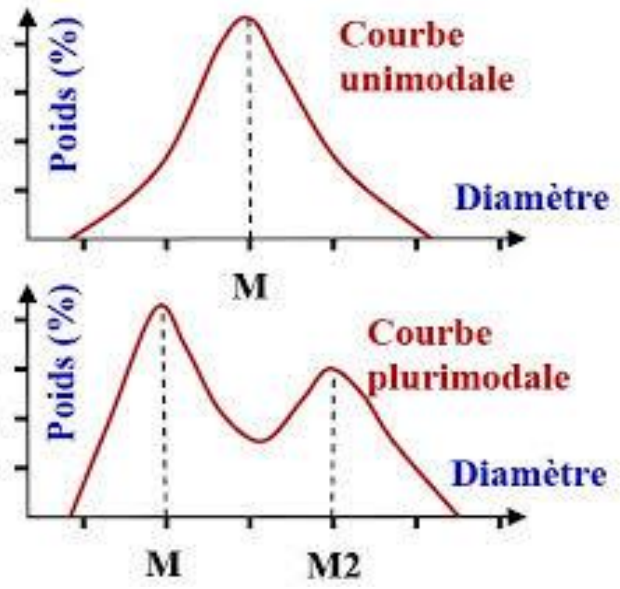


Figure1.5 : Courbe exprimant les fonctions monomodales et multimodales [14]

Une fonction est dite uni modale si elle possède un seul extremum local (max ou min) sur un intervalle donné (donc il y a une seule bosse, et multimodale ou plurimodale à deux ou plusieurs bosses comme présenté **Figure 1.5).**

7-4 Les données : Il existe trois types des données : Discrète, Continues et Mixte.

Les données discrètes prennent des valeurs entières. Par contre les données continues peuvent prendre n'importe quelles valeurs réelles. Les données mixtes sont formées de données discrètes et continues à la fois.

7-5 Avec contraintes ou sans contraintes :

7-5-1 Optimisation avec contraintes

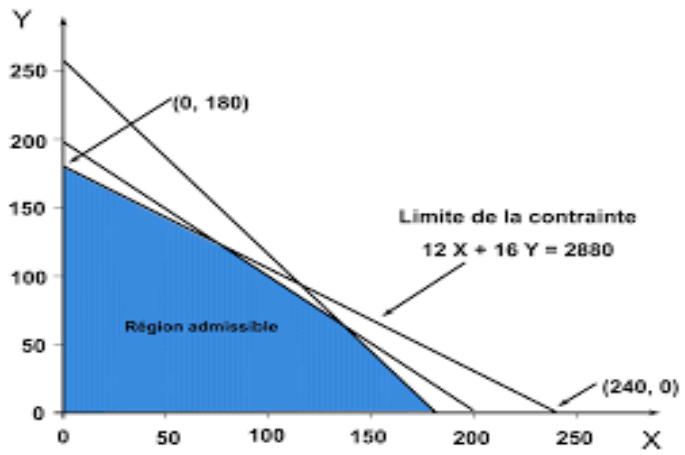


Figure 1.6 : Optimisation avec contrainte [15]

La fonction objective où la solution est limitée par des contraintes qui doivent être respectées.

Exemple : La **Figure1.6**, montre qu'il existe trois contraintes, qui sont représentées par des droites linéaires, une des trois contraintes est décrite par la fonction linéaire 12x+16y= 2880,

7-5-2 Optimisation sans contraintes

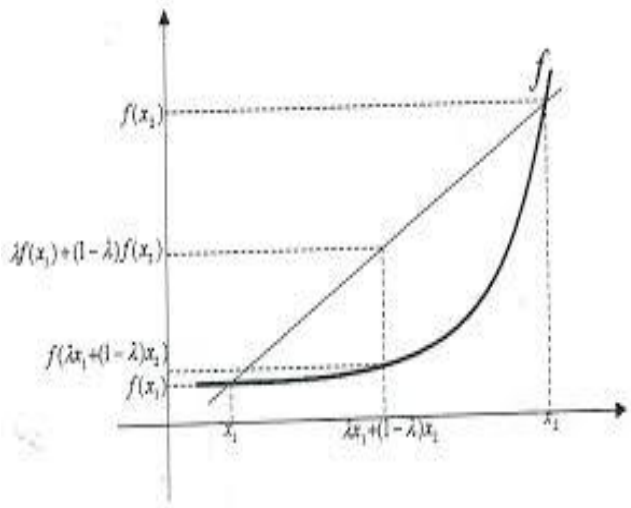


Figure 1.7 : Optimisation sans contrainte [16]

Dans l'optimisation sans contraintes, les problèmes cherchent des solutions sans aucune restriction sur les variables, telle que la meilleure solution dans toute l'espace possible (**Figure 1.7).** Exemple d'algorithme qui n'exige pas de contraintes : Descente de gradient,…

7-6 Optimisation linéaire et non Linéaire

L'optimisation linéaire a pour but de maximiser ou minimiser une fonction objective linéaire sous des contraintes linéaires (exemple schématisé (**Figure 1.6**).

La forme des solutions est présentée dans un polygone convexe qui englobe la solution optimale. Exemple « y= 3∗x+6 ». C'est méthodes sont rapides et efficaces pour trouver une solution optimale. Par contre dans le cas de l'optimisation non linéaire, où la fonction objectif est non linéaire, on doit respecter les contraintes de non linéarité.

Ces problèmes sont généralement plus complexes et nécessitent des méthodes spécifiques pour trouver des solutions optimales, exemple : **f(x)=** $x^2 + sinx$, comme nous le montre la **Figure 1.7**

8- Classification des méthodes d'optimisation

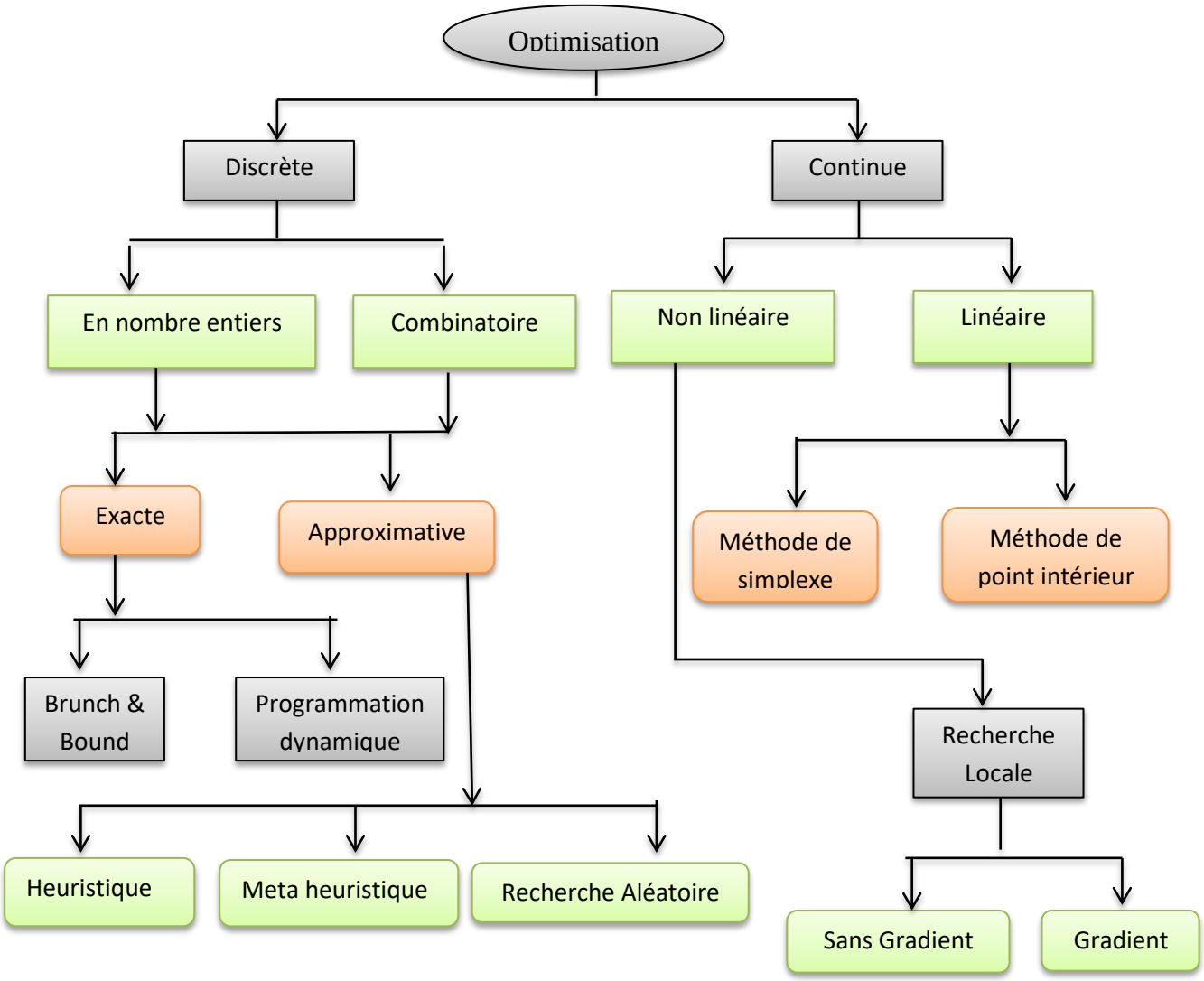


Figure 1.8 : Classification des méthodes d'optimisation [17]

Il existe de nombreux algorithmes d'optimisation dans la littérature, aucun algorithme unique ne convient à tous les problèmes, le choix de la méthode d'optimisation pour un problème donnée est d'une importance cruciale et le choix d'algorithme pour une tache d'optimisation dépondra largement des critères cités dans la **Figure 1.2**.

Généralement, les algorithmes d'optimisation peuvent être classés selon plusieurs manières.

La **Figure 1.8** résume les principales classes des méthodes d'optimisation.

En constate que l'on part de deux types de méthodes d'optimisation : l'optimisation continue et la deuxième et l'optimisation discrète.

Les méthodes d'optimisation continue traitent les problèmes dont les variables sont autorisées à prendre n'importe quelle valeur réelle, par contre la méthode discrète traite des problèmes où certaines ou toutes les variables du modèle doivent contenir un ensemble discret.

Dans la partie discrète, il existe deux types de méthodes :

- La première concerne la programmation en nombres entiers, où l'ensemble discret est un sous ensemble entier.
- La deuxième est l'optimisation combinatoire où l'ensemble discret est un ensemble d'objets, ou de structures combinatoires fini tel que les affectations, les horaires, les séquences, etc.…

Les méthodes discrètes peuvent être exactes ou approximatives, ces méthodes sont capables de trouver une solution optimale a un problème d'optimisation exemple de ce type de méthodes ***Brunch & bround*** et ***la programmation dynamique.***

Les méthodes d'optimisation approximative sont des algorithmes qui trouvent des solutions approximatives aux problèmes d'optimisation, dans ce type de méthodes, on distingue trois sous classes :

- ✓ Les heuristiques qui sont des méthodes approximatives spécifiques à un problème donné.
- ✓ Les métas heuristiques qui sont des méthodes indépendantes du problème et peuvent être appliquées à plusieurs classes de problèmes.
- ✓ La recherche aléatoire est une méthode qui tend de trouver l'optimum en testant la valeur de la fonction objective sur une série des valeurs aléatoires.

Les méthodes continues peuvent être classées en linéaire ou non linéaire.

- Les méthodes linéaires sont conçues pour l'optimisation d'une fonction linéaire et peuvent être soumises à des contraintes linéaires, exemple de ces méthodes : méthode du simplex et ses variantes, la méthode du point d'intérieur [17].

- Alors que les méthodes d'optimisation non linéaires sont conçues pour l'optimisation d'une fonction non linéaire ou une fonction linéaire avec des contraintes non linéaires, elles se composent des méthodes d'optimisation globale qui consistent à trouver la valeur optimale d'une fonction donnée parmi toutes les solutions possibles, exemple : algorithmes génétiques.

Tandis que dans la recherche locale, on cherche la valeur optimale dans l'ensemble voisin d'une solution candidate.

Les méthodes de recherche locales peuvent être basées sur le gradient et ce sont des méthodes itératives qui utilisent largement l'information du gradient de la fonction objectif au cours des itérations, exemple la méthode de ***Newton***, ***Descente de Gradient***.

L'autre méthode locale n'utilise pas le gradient ou sans gradient exemple ***recuit simulé***, ***recherche aléatoire***, ***recherche Tabou***, etc.

9- Les étapes à suivre pour exécuter un algorithme d'optimisation

Choisir un algorithme d'optimisation pour un problème donné nécessite de prendre en compte plusieurs facteurs dont les suivantes :

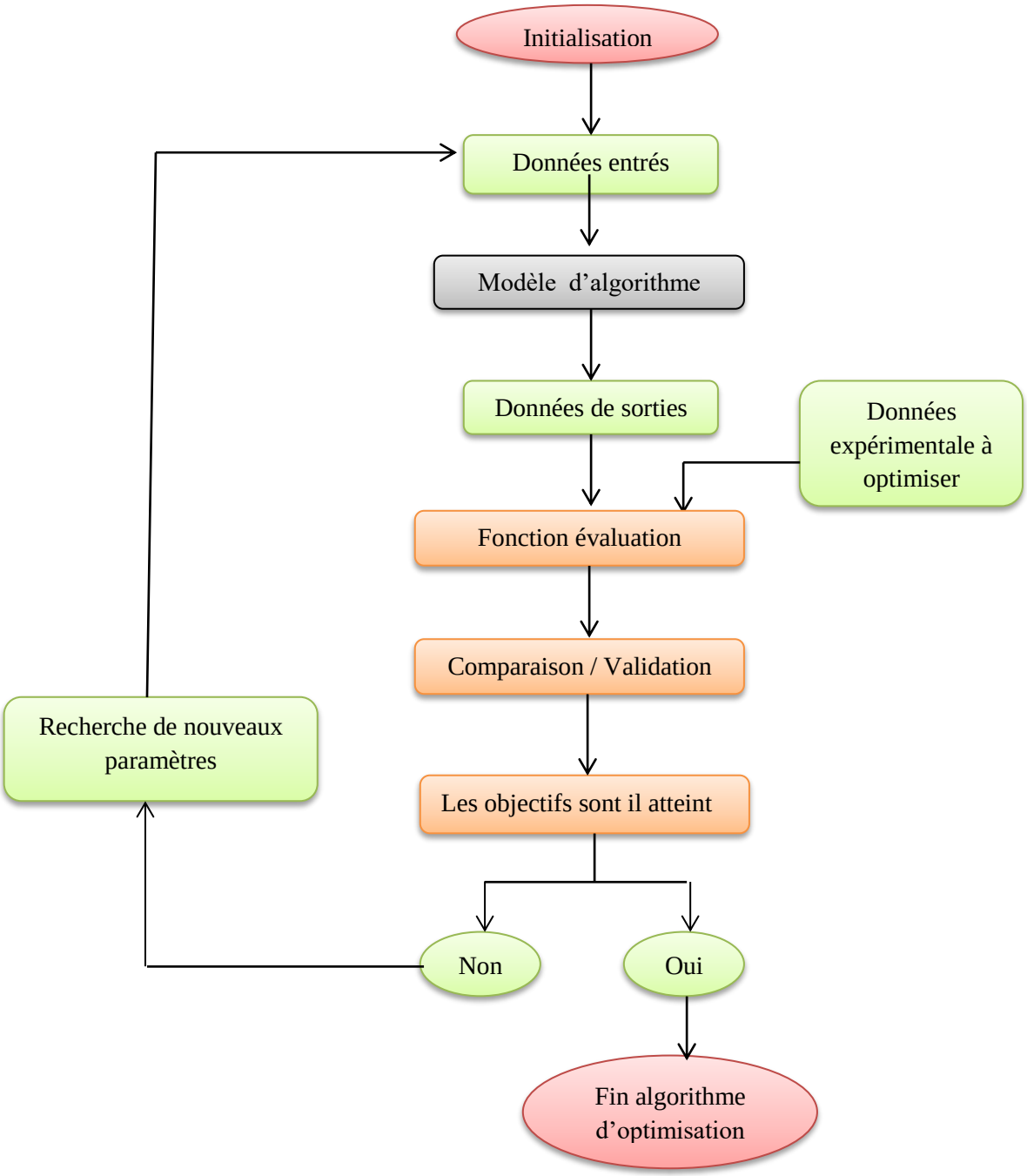


Figure 1.9 : Etape pour choisir un algorithme d'optimisation [17]

9-1 Comprendre le problème : c'est à dire identifier les contraintes du problème et déterminer les objectifs que nous souhaitons atteindre, maximiser ou minimiser les fonctions objectives.

9-2 Considérer la taille et la complexité du problème : Donc pour choisir un algorithme en prend en considération la taille du problème. Par est exemple, Pour des problèmes de petite taille, des méthodes exactes comme la programmation linéaire entière peuvent être envisagées. Pour des problèmes plus grands, des méthodes heuristiques ou méta heuristiques peuvent être plus appropriées.

Évaluation de la complexité du problème consiste à considérer sa classe. Exemple si le problème est NP-difficile, il peut être nécessaire d'accepter des solutions approximatives plutôt que de chercher une solution optimale.

9-3 Évaluer les algorithmes d'optimisation : Choisir quelque algorithme qui est compatible avec mon problème. Ou utiliser des algorithmes sans utilise dans des autre problèmes qui sont similaire à mon problème.

9-4 Test et évaluation

Prototypage : Implémenter plusieurs algorithmes et les tester sur des instances du problème. Puis comparer les résultats en termes de :

-Qualité de la solution, de temps de calcul et de satisfaction des contraintes.

- Feedback : Obtenir des retours générés pour ajuster les algorithmes et les paramètres, c'est à dire désigner le retour d'information fourni par un système , un logiciel ou un utilisateur comme il peut être visuel, sonore ou textuel et permet d'informer l'utilisateur sur l'état du système ,la réussite ou l'échec d'une opération pour ajuster ses paramètre en fonction des retours reçus **[17]** .

9-5 Itération

Amélioration continue : L'optimisation est souvent un processus itératif. Utilisez les retours et les résultats des tests pour affiner notre approche et améliorer les algorithmes choisis.

10- Conclusion

Dans ce chapitre, nous avons exploré certaines notions de base sur la mise en œuvre des algorithmes d'optimisation et on a posé des critères pour mieux choisir un algorithme qui réponde à nos besoins pour obtenir des solutions exactes ou même approximative.

D'après notre recherche, on a constaté que cette discipline est très riche et vaste puisque les problèmes deviennent de plus en plus complexes. L'optimisation est importante parce qu'elle touche tous les aspects dans le monde réel tel que l'ingénierie, l'économie, la logistique, la santé et surtout l'intelligence artificielle.

L'optimisation est un domaine important pour garantir l'efficacité des systèmes, réduire les coûts et optimiser les ressources afin d'atteindre la fonction objectif et surtout de gagner du temps.

CHAPITRE II

Méthodes Heuristiques

1- Introduction

La complexité des problèmes peut s'observer à différents niveaux des plus simples au plus complexes (P, NP, NP complet) en fonction des ressources nécessaires pour leur résolution. Les méthodes exactes sont utilisées pour la résolution de problèmes simples et permettent d''obtenir des solutions optimales, exactes et dans un temps limité.

Cependant, les méthodes exactes ne permettent pas, généralement, de générer les meilleures solutions pour des problèmes de grande complexité. Ces méthodes ne peuvent en donner que des solutions approximatives et assez vagues, voire non adaptées tout en consommant beaucoup de ressources et des temps de calcul parfois prohibitifs. De ce fait, elles sont jugées inefficaces et non fiables pour cette classe de problèmes.

Sachant que notre but est de trouver les solutions les plus appropriées et optimales consommant le minimum de ressources et de temps de calcul, les choix des algorithmes d'optimisation adéquats s'imposent ave rigueur. Les méthodes permettant d'approcher le plus possible des solutions optimales pour les problèmes complexes sont appelées méthodes heuristiques ou Meta heuristiques. Ainsi, ces méthodes dites heuristiques se substituent aux méthodes exactes pour approcher le plus possible une solution proche de l'optimalité dans un temps raisonnable avec une complexité raisonnable et une simplicité d'implémentation.

Ce chapitre est consacré aux principales familles d'algorithmes heuristiques ainsi que leurs applications dans la résolution de problème complexes.

2- Historique des méthodes heuristiques :

Les méthodes heuristiques existent depuis les années 50 et elles ont été utilisées surtout dans le domaine militaire ainsi que pour les jeux d'échecs notamment.

- En ***1957*** : ***George Polya*** *publie* « ***How to Solve It*** » pour résoudre les problèmes mathématiques, il a utilisé des stratégies heuristiques.
- En ***1960*** : La première apparition informatique de la méthode heuristique dans l'IA (intelligence Artificielle).
- En ***1970*** : Introduction des méthodes ***Gloutonnes (Greedy)*** et des algorithmes de recherche locale (comme ***Hill Climbing***).
- Apres 1970 : Apparition des grandes familles de méthodes bio-inspirées de phénomènes tel que l'algorithme Génétique ***(AGA)*** qui est introduit par ***John Holland.***

- En ***1983 Kirkpatrick, Gelatt*** et ***Vecchi*** s'inspirent du processus de refroidissement des métaux pour proposer la méthode du ***Recuit simulé*** (***Simulated Annealing SA***).
- En ***1986*** Recherche ***Tabou*** (***Tabu Search***) est proposé par ***Fred Glover.***
- C'est ensuite qu'apparaissent les nouvelle approches bio-inspirées : Optimisation par colonies de fourmis (**Ant Colony Optimization**) proposée par ***Dorigo*** en *1992*, pour résoudre les problèmes d'optimisation combinatoire et l'Optimisation par Essaims particulaires **(Particle Swarm Optimization**) proposée par ***Kennedy et Eberhart*** en ***1995.*** D'autres méthodes bio-inspirées ont été proposées par la suite telles que celles inspiréesdes abeilles, lucioles, chauves-souris, etc.
- Enfin et depuis ***2010***l'intelligence artificielle et à son apogée et propose beaucoup d'outils heuristiques permettant de développer des méthodes plus efficaces et pouvant résoudre des problèmes de plus en plus complexes dans des domaines variés [18].

3-Définitions et concepts de base

L'heuristique ou euristique (du grec ancien **Heuriskein**, qui signifie « **trouver ou découvrir** ») est « l'art d'inventer, de faire des découvertes »

En effet, une heuristique est spécifique à un problème donné pour le résoudre dans un temps raisonnable et avec des solutions approchées non optimales. Une heuristique est une stratégie de recherche de solution satisfaisante à un problème donné qui se base sur l'expérience (utilise des résultats déjà obtenus) et l'intuition.

Une Méta heuristique est une méthode générique pouvant optimiser une large gamme de problèmes différents, ses algorithmes d'optimisation sont stochastiques itératifs, hybridés. (Méta est un suffixe signifiant 'au –delà').

Les Méta heuristiques sont généralement des algorithmes qui progressent vers l'optimum global d'une fonction, par échantillonnage d'une fonction objectif.

Le terme Méta est donc pris au sens où les algorithmes peuvent regrouper plusieurs heuristiques **[19]**.

4- Caractéristiques des algorithmes heuristiques :

- **Approximatifs** : Ils ne garantissent pas la solution optimale, mais visent à trouver une solution proche de l'optimum dans un temps raisonnable.

- **Rapides** : Ils sont souvent plus rapides que les algorithmes exacts, car ils ne passent pas par toutes les possibilités.
- **Adaptatifs** : Ces algorithmes peuvent être modifiés en fonction du problème spécifique à résoudre.
- **Basés sur des expériences** : Ils s'appuient sur des règles de décision, souvent inspirées de l'intuition, des expériences antérieures ou de l'observation des problèmes similaires.

5- Intérêt des méthodes heuristiques par rapports des méthodes exactes

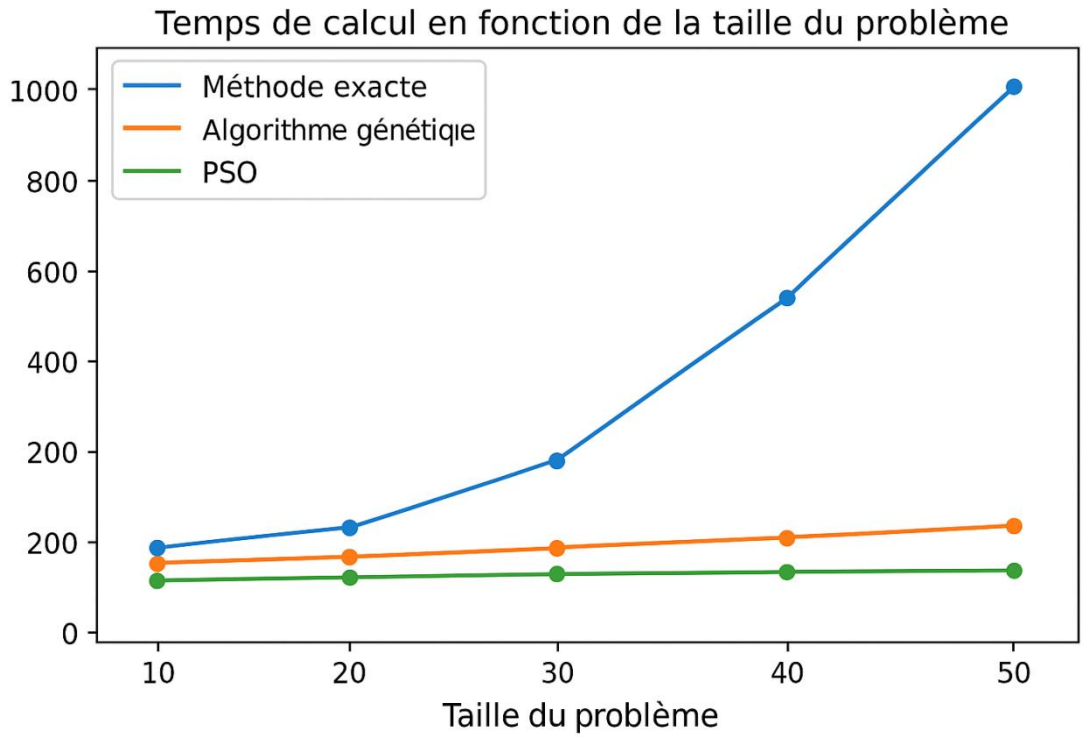


Figure 2-1 : comparaison entre méthodes exacte et heuristique selon le temps [20]

Selon des études et des recherches sur les méthodes heuristiques, une comparaison a été faite entre les méthodes exactes et heuristiques. D'après la **Figure 2-1** présente la comparaison entre des méthodes exactes et des méthodes heuristiques. Dans cet exemple, on utilise l'algorithme génétique et PSO (Optimisation par Essaim de Particules) **[20]**

Cette comparaison prend en considération le temps de calcul en fonction de la taille du problème. On y voit clairement que les deux courbes concernant les méthodes heuristiques sont des lignes

droite horizontales ; le temps pratiquement stable à 180 s, alors que la méthode exacte est suit une courbe paraboliques démontrant une augmentation exponentielle. On en conclu que la méthode exacte devient très lente quand la taille du problème augmente tandis les heuristiques restent rapides et stables.

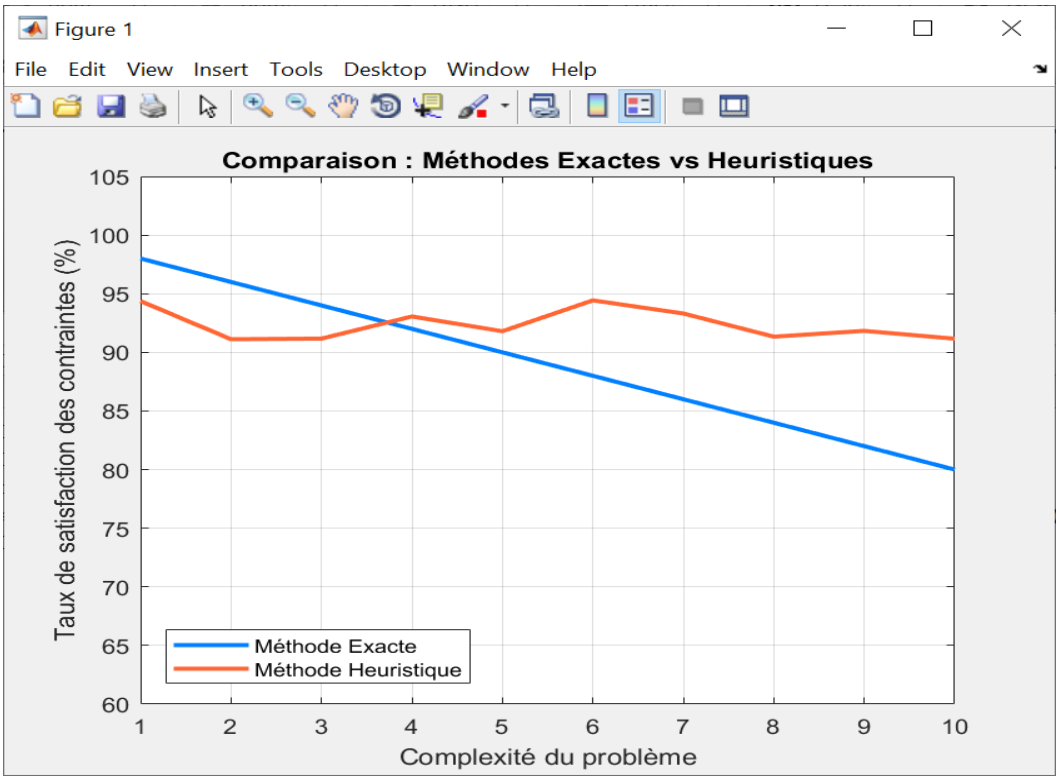


Figure 2-2 : Comparaison entre méthodes exacte et heuristique selon le TSC [20]

L'efficacité des méthodes heuristiques par rapport aux méthodes exactes dépend de la complexité du problème où le temps est un facteur limitant et elles permettent de trouver de bonnes solutions approchées et rapides sachant que dans les méthodes exactes le temps de calcul peut devenir exponentiel surtout pour les problèmes de grande taille.

Les heuristiques sont souvent plus simples à mettre en œuvre que des algorithmes exacts [21].

Elles offrent une flexibilité et une résilience supérieures aux méthodes exactes dans la résolution de problèmes complexes. Si l'on veut intégrer de nouvelles contraintes ou objectifs, les méthodes heuristiques peuvent être rapidement adaptées à ces changements, on combine d'autres méthodes pour améliorer les résultats, ce qui revient à l'hybridation entre plusieurs méthodes.

C'est pour ces raisons que les méthodes heuristiques sont privilégiées par rapport aux méthodes exactes lorsque les problèmes deviennent de plus en plus complexes et de grandes tailles.

6- Classification des méthodes heuristiques

Sachant que les problèmes diffèrent selon la nature des données et de leurs complexités, le choix des algorithmes heuristiques peut se faire selon plusieurs critères tels que :

- ✓ leur approche de résolution,
- ✓ leur dépendance aux données
- ✓ leur capacité à explorer l'espace des solutions
- ✓ la stratégie de recherche
- ✓ le comportement de recherche (Déterministe ou Stochastique)
- ✓ le niveau d'abstraction
- ✓ La nature de l'exportation de l'espace de recherche (recherche locale ou globale)
- ✓ leur inspiration (biologique, physique)

Dans le cas du classement selon le niveau d'abstraction, on distingue les deux types suivants :

- **Heuristiques simples ou classiques** : on utilise une stratégie locale simple, dans un temps raisonnable, donc c'est une méthode directe pour des problèmes moins complexes. Ce genre d'heuristique se caractérise par sa rapidité d'exécution par ce qu'elle trouve rapidement la solution, et par la simplicité, donc facile à coder et adapter et en peu modifiés selon les contraintes du problème.

Dans le cas où on classe les méthodes heuristique selon la stratégie de recherche des méthodes on distingue trois grandes catégories :

- **Les heuristiques constructives** : sont des méthodes où l'on ajoute des éléments selon une certaine règle pour construire une solution pas à pas, elles se caractérisent par la rapidité de l'exécution et sont basées sur des règles simples mais elles ne garantissent pas l'optimalité.

Exemples :

- ➢ Algorithme du plus proche voisin pour le voyageur de commerce (**KNN**).
- ➢ Algorithme glouton pour le sac à dos.

Les heuristiques de recherche locale : Elles utilisent la solution initiale en explorant son voisinage pour atteindre un optimum local. Exemple :

- ➢ Algorithme de descente (***Hill Climbing***)
- ➢ Recherche en voisinage variable (***VNS***)

Les méthodes méta heuristiques : ce sont des méthodes plus générales utilisées pour résoudre différents type de problèmes complexes. Ce sont des stratégies de haut niveau conçues pour guider d'autres heuristiques inspirées de phénomènes naturels ou biologiques, elles sont plus flexibles et robustes et peuvent échapper aux optima locaux.

7- Comparaison entre les différentes méthodes heuristiques

7-1 Recherche Locale : Cette méthode commence par une solution initiale et explore les solutions voisines pour trouver une meilleure solution.

Avantage : Cette méthode est Simple à mettre en œuvre et efficace pour des problèmes de petite et de taille moyenne.

Inconvénient : par contre elle peut se retrouver bloquée dans des optima locaux et ne garantit pas une solution optimale.

7-2 Recuit Simulé (Simulated Annealing) : Inspiré du processus de refroidissement des métaux, cette méthode permet d'accepter des solutions moins bonnes à certaines étapes pour échapper aux optima locaux.

Avantage : de cette méthode capable de trouver des bonnes solutions même dans des espaces de recherche complexes.

Inconvénient : La performance dépend fortement de la température initiale et du schéma de refroidissement.

7-3 Recherche Tabou : Utilise une mémoire pour éviter de revisiter des solutions déjà explorées, ce qui aide à échapper aux optima locaux.

Avantage : Efficace pour des problèmes combinatoires complexes et peut fournir des solutions de haute qualité.

Inconvénient : La gestion de la mémoire tabou peut être complexe et nécessite un bon réglage.

7-4 Colonies de Fourmis (Ant Colony Optimization) : Inspirée du comportement des fourmis, cette méthode utilise des agents (fourmis) pour explorer des solutions et communiquer via des phéromones.

Avantage : Particulièrement efficace pour les problèmes de cheminement et d'optimisation combinatoire.

Inconvénient : Peut-être lent à converger et nécessite un bon réglage des paramètres.

7-5 Optimisation par Essaim de Particules (Particle Swarm Optimization) : Basée sur le comportement social des oiseaux ou des poissons, cette méthode utilise un groupe de solutions (particules) qui se déplacent dans l'espace de recherche.

Avantage **:** Simple à mettre en œuvre et efficace pour de nombreux types de problèmes.

Inconvénient : Peut souffrir de convergence prématurée et nécessite un bon réglage des paramètres.

8- Etude de quelques méthodes utilisées

8.1 Algorithme Glouton (Greedy Allgorithm) : C'est une méthode de résolution de problème d'optimisation qui construit une solution en effectuant le choix local optimal, étape par étape, sans revenir sur les décisions présidentes.

Si la solution proposée n'est pas optimale, on parle d'heuristique gloutonne.

8.1.1 Historique de la méthode de gloutonne :

En anglais ***Greedy Algorithm*** est une méthode d'optimisation ancienne et fondamentale en informatique et en mathématique appliquées.

Cette méthode gloutonne a commencé à être formalisée dans le contexte de la programmation mathématique et de l'optimisation combinatoire dans les années 60.

Elle est bien adaptée à un ensemble de problèmes comme le problème de sac à dos (***Knapsack problem***) et la recherche du plus court chemin (algorithme de ***Djikstra***), ainsi que pour l'affectation de taches et la planification **[22]**.

8.1.2 Principe fondamental de la méthode

- ✓ La méthode gloutonne prend le meilleur choix local, à chaque étape, dans l'espoir d'obtenir une solution optimale globale.
- ✓ suit une stratégie simple, qui commence avec une solution vide ou partielle et à chaque étape, sélectionne l'option disponible qui semble la meilleure selon un critère local (exemple, le cout minimal ou le gain maximal).
- ✓ Une fois le choix effectué, il n'est pas remis en question
- ✓ Répéter le processus jusqu'à ce qu'une solution complète soit obtenue

  La **Figure 2-2** résume les étapes

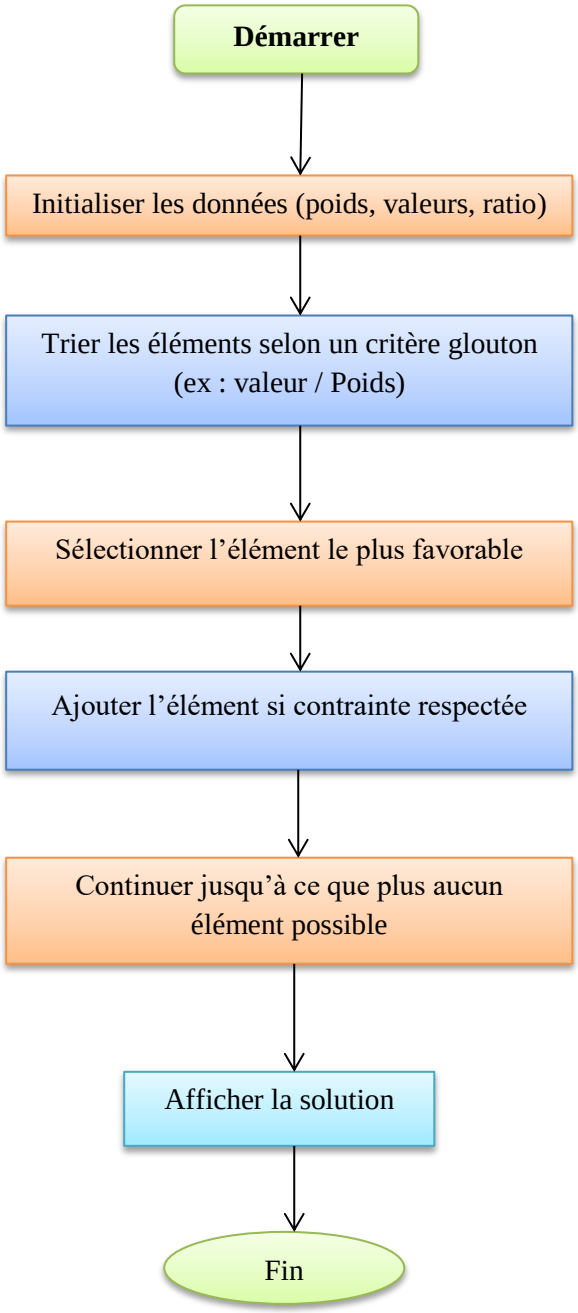


Figure 2.3 Organigramme de la méthode Gloutonne **[22]**

8.1.3 Optimisation utilisant la méthode de Glouton :

- Construction itérative sans retour arrière
- Critère de sélection local (optimum)

- Invariant (pour un algorithme glouton : la solution partielle construite est un préfixe d'une solution optimale).
- Heuristique gloutonne : pas de propriété d'optimum global mais intuition d'une solution pas trop mauvaise.
- Elle fonctionne bien pour certains problèmes, comme la monnaie rendue, le problème de sac à dos.

Exemple :

- Problème d'allocation
- Coloriage de graphes
- Arbres couvrants de poids minimum dans un graphe (algorithme de ***Prim et Kruskal***)
- Arborescence des plus courts chemins (algorithme de ***Djikistra***)

8.2 Les algorithmes génétiques évolutionnaires (AGE) :

Ils ont été imaginés par Holland dans les années ***70*** pour imiter les phénomènes d'adaptation des êtres vivants.

Les **AGE** s'inspirent des principes de la sélection naturelle et de l'évolution pour rechercher des solutions optimales ou quasi-optimales à des problèmes donnés.

Le principe de cette méthode est de trouver le Max ou le Min de la fonction objective à partir d'un ensemble d'éléments au hasard.

Dans les **AGE**, on utilise l'idée des chromosomes pour représenter une solution possible au problème.

Chromosome = une solution codée sous forme binaire.

Chaque Gène du chromosome correspond à une partie de la solution.

Une population = un ensemble de chromosomes.

Le nombre total de solutions possibles est $\mathbf{2}^{NbObjet}$ ($NbObjet$: nombre d'objets).

8.2.1 Bref historique sur les algorithmes génétiques

Les algorithmes génétiques sont inspirés de l'évolution naturelle. Ils sont été formalisé dans les années ***1970 par John Holland*** (Chercheur à l'Université du *Michigan* qui formalise les algorithmes

génétiques dans son livre " Adaptation in Natural and Artificial Systems") tout en montrant comment utiliser les opérations de sélection, croisement, et mutation pour résoudre les problèmes complexes **[23]**.

Puis en *1980*, les AGE commencent à être appliqués dans divers domaines comme l'optimisation, la robotique, la planification, l'intelligence artificielle, etc.

Avec l'augmentation de la puissance de calcul, on assiste à une forte expansion de l'utilisation des AGE, depuis 1990. Avec des nombreux ajustements de cette méthode.

Depuis l'année 2000 et jusqu'à ce jour, les GA touchent d'autre domaines, tels que la finance, la bio-informatique, les jeux vidéo,… Souvent, on trouve des mélanges d'algorithmes génétiques et d'autres techniques telles que les AG hybridés avec les réseaux de neurone [24].

8.2.2 Les étapes principales d'un AG :

Exemple un individu= un objet / entité.

1- **Initialisation de la population** : créer une population initiale de solutions aléatoires.
2- **Evaluation** : pour chaque solution en va évaluer une fonction de performance (Fitness).

La fonction «fitness» est une répartition par pourcentage de ses performances.

3- **Sélection** : choisir les meilleures solutions pour se reproduire
4- **Croissement** : combiner les individus sélectionnés pour créer de nouveau individus.

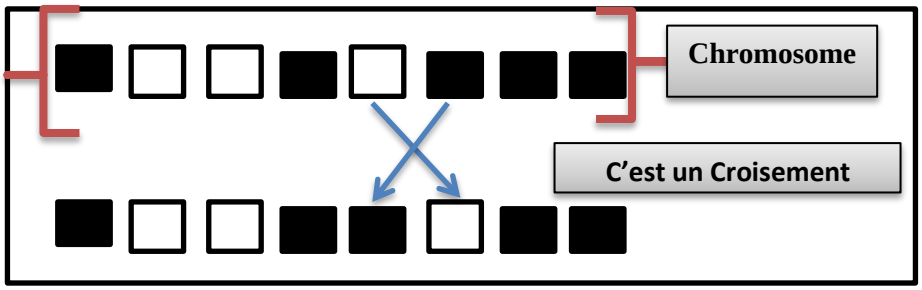


Figure 2-4 : Le croisement entre deux solutions [25]

But : combiner deux parents pour produire un ou deux enfants

Exemple :

Parent 1 : 11011

Parent 2 : 00101

Le croisement se passe après le 2eme Gène en obtient deux enfants

Enfant 1 : 11100

Enfant 2 : 00010

C'est la nouvelle génération (nouveaux solutions)

5- **Mutation** : Modifier aléatoirement certaines parties de solutions créées lors du croisement.

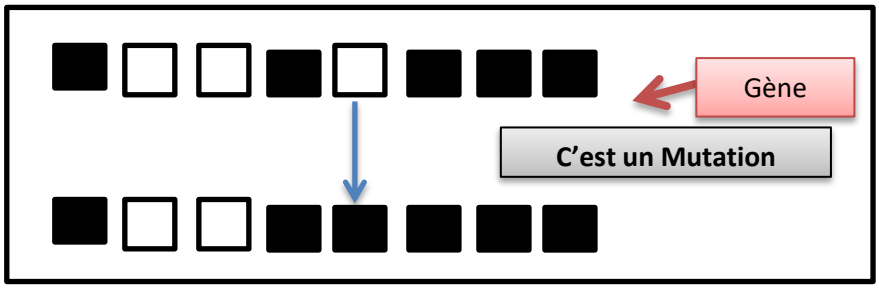


Figure 2-5 : La mutation entre deux solutions [25]

Si en fait la mutation de Enfant 1 : 11**10**0, en obtient par exemple 11**01**0

6- **Evaluation de la nouvelle population** : calculer la fonction de fitness pour les nouveaux individus

7- **Remplacement** : former une nouvelle génération en remplaçant certains anciens individus par les nouveaux.

8- **Arrêt** : vérifier si la solution optimale est atteinte ou si le nombre d'itérations a été atteint si non répéter les étapes 2, 6, jusqu'à atteindre un certain critère d'arrêt (exemple : nombres d'itération, ou qualité suffisante) [25].

Les étapes sont résumées par organigramme ci-dessous **(Figure 2-5)**

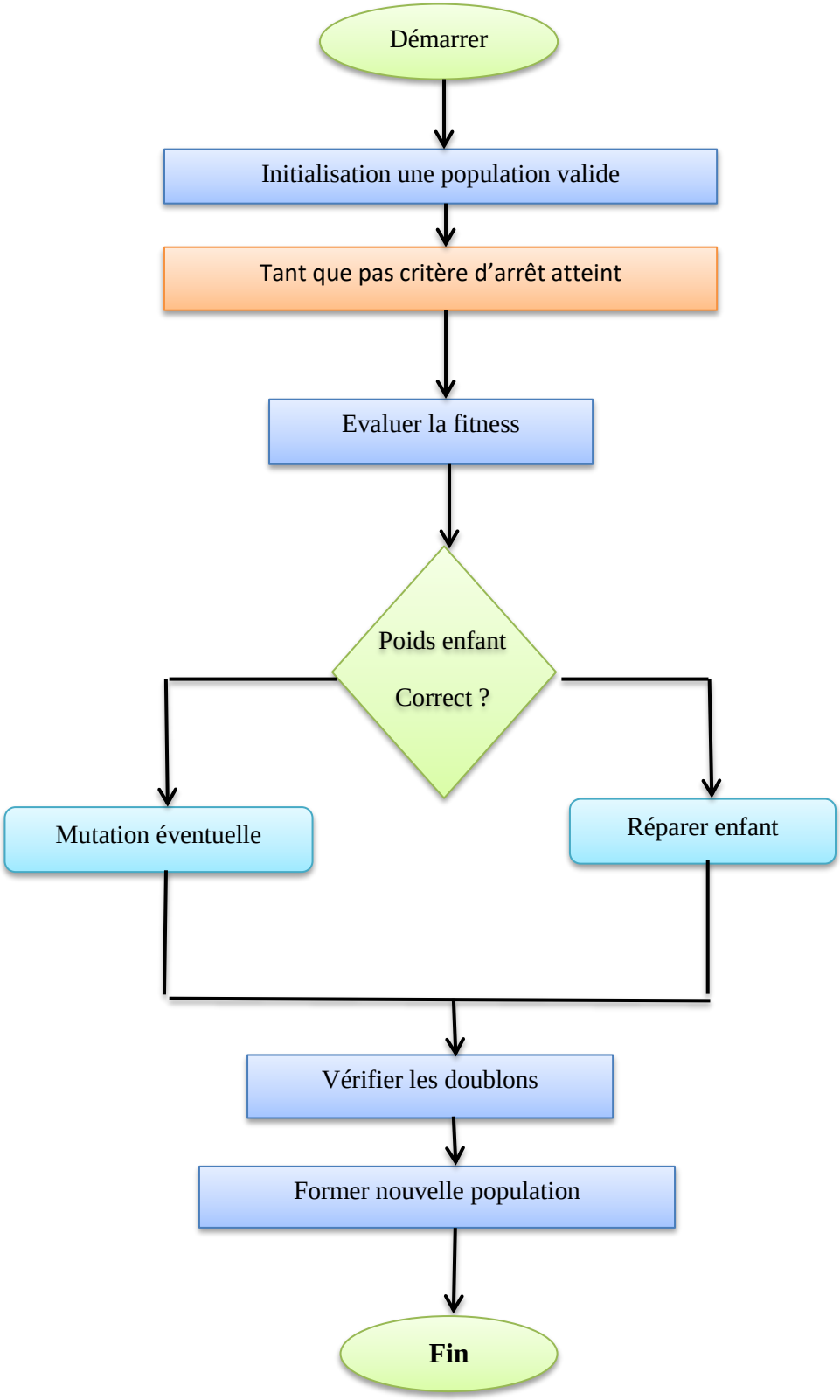


Figure 2-6 Organigramme des étapes de l'algorithme génétique [26]

Remarque :

- ❖ l'algorithme s'arrêtera après avoir atteint le nombre de génération (cycles). Même si la solution n'est pas trouvée. (Cas d'un critère d'arrêt basé sur le nombre de générations).
- ❖ Si le critère d'arrêt se base sur un seuil de performance, l'algorithme sarrète dès qu'il trouve une solution qui dépasse ce seuil même avant d'avoir atteint le nombre de générations.

8.2.3 Problèmes d'optimisation utilisant les algorithmes génétiques

Exemple :

- Affectation de fréquences en téléphonie.
- Le problème du sac-à-dos.
- Couverture d'ensemble.
- Routage de véhicules.
- Le voyageur de commerce.
- Gestion de ressource.
- Horaire de train, emploi du temps.

8-3 Exemple d'application

Exemple problème de Sac à Dos (Knapsack)

Données: On a un sac à dos qui peut contenir au maximum 15 kg, et des objets avec chacun un poids et une valeur. On peut prendre des fractions d'objets, dans le tableau ci-dessous les valeurs des objets utilisées dans cette expérience :

Objet	Poids (kg)	Valeur (Da)
A	**4**	**12**
B	**2**	**2**
C	**6**	**14**
D	**1**	**1**
E	**4**	**10**

Problème: Le but de cette expérience de maximiser la valeur total dans le sac.

8-3-1 méthode Gloutonne

Etapes de l'algorithme Glouton :

La fonction objective du problème à sac à dos : gagner la valeur (da) sous contrainte : le poids ne dépasse pas à 15kg

1- Calcul du Ratio =valeur/ poids pour chaque objet :

A : 12/4 = 3

B : 2/2 = 1

C : 14/ 6= 2.3

D: 1/1 = 1

E : 10/4 = 2.5

2- **En va trier les objets par ratio décroissant**

A (3), E (2.5), C (2.5), B (1), D (1)

3- **Remplir le sac avec les objets dans cet ordre** :

-Prendre tous **A (4, 12Da),** Reste **(15 -4= 11)** donc **11**

-Prendre tous **E (4, 10Da),** Reste **(11 -4= 7)** donc **7**

-Prendre tous **C (6, 14Da),** Reste **(7 -6= 1)** donc **1**

-Prendre tous **B (2, 2Da),** **2 >1** donc **B** est refuser

-Prendre tous **D (1, 1Da),** Reste **(1 -1= 0)** donc **0**

Résultat final :

Objets pris : A, E, C, D

Poids total : 15 kg (c'est la limite du sac donc en a respecter la contrainte)

Valeur total : 37 Da (c'est la fonction objectif)

8-3-2 Résolution à l'aide des algorithmes génétiques

Etape 1 : Définir les paramètres

1-Les objets :

Objet	Poids (kg)	Valeur (Da)
A	**4**	**12**
B	**2**	**2**
C	**6**	**14**
D	**1**	**1**
E	**4**	**10**

2- Capacité du sac : 15 kg

3-Population : Taille de la population (par exemple, 4 individus)

4-Taux de mutation : Par exemple, 0.1 (10%)

5-Critere d'arrêt c'est le Nombre de générations : Par exemple, 2

Étape 2 : Représentation des individus

Le nombre de solution se calcule par la formule Nb S= 2^NbObjet.

Chaque individu dans la population est une solution et peut être représenté par un tableau binaire où chaque bit représente la présence (1) ou l'absence (0) d'un objet dans le sac. Par exemple, pour 5 objets, un individu pourrait être représenté comme suit : [1, 0, 1, 0,1]

Chaque solution (individus) est un chromosome binaire de 5 Genès (un pour chaque objet) :

[1, 0, 1, 0,1] implique en prend les objets A pris, B non, C pris, D non, E pris.

Donc lorsque on additionne les poids des **A, C et E** en obtient **4+6+4= 14 kg**

Etape 3 : Fonction Fitness :

Si le poids total **<=15kg ; Fitness = somme des valeurs**

Sinon, Fitness=0 (ou une pénalité forte) ;

Etape 4 : Population initial (Les propositions) = 4

Individus	Chromosome	Poids (kg)	Valeurs (da)
1	[1, 1, 1, 0, 0]	4+2+6 = 12	12+2+14 =28
2	[0, 1, 1, 1, 0]	2+6+1 = 9	2+14+1 =17
3	[1, 0, 1, 0, 1]	4+6+4 = 14	12+14+10 = 36
4	[1, 0, 1, 1, 0]	4+6+1 = 11	12+14+1 =27

Tableau 2-1 La population initiale

Remarque

On remarque sur le tableau ci-dessus que l'on obtient une performance optimale à la ligne 3.

Etape 5 : le nombre de croisements se calcule par la formule C= 4!/ 2!(4-2)! = 6

Donc on peu faire 6 croisements, chaque croisement nous donne deux enfants (nouveaux individus) ce qui nous donne 12 nouveaux individus. Cependant, le croissement dépend des gènes, ce qui nous donne :

Individus	Croisement	Chromosome	Objet a combinés	Poids (kg)	Valeurs (da)
1	Entre1 et 2	[1, 1, 1, 1, 0]	A, B, C, D	4+2+6+1 = 13	12+2+14+1 =29
2		[0, 1, 1, 0, 0]	B, C	2+6 = 8	2+14 = 16
3	Entre1 et 3	[1, 1, 1, 0, 1]	A, B, C, E	4+2+6+4 = 16	12+2+14+10 = 38
4		[1, 0, 1, 0, 0]	A, C	2+6 = 8	12+14 = 26
5	Entre1 et 4	[1, 1, 1, 1, 0]	A, B, C, D	4+2+6+1 = 13	12+2+14+1 =29
6		[1, 0, 1, 0, 0]	A, C	2+6 = 8	12+14 = 26
7	Entre2 et 3	[0, 1, 1, 0, 1]	B, C, E	4+2+6 = 12	2+14+10= 26
8		[1, 0, 1, 1, 0]	A, C, D	2+6+1 = 9	12+14+1 = 27
9	Entre2 et 4	[0, 1, 1, 1, 0]	B, C, D	2+6+1 = 9	2+14+1 = 17
10		[1, 0, 1, 1, 0]	A, C, D	2+6+1 = 9	12+14+1 = 27
11	Entre3 et 4	[1, 0, 1, 1, 0]	A, C, D	2+6+1 = 9	12+14+1 = 27
12		[1, 0, 1, 0, 1]	A, C, E	2+6+4 = 12	12+14+10 = 36

Tableau 2-2 : la population après les croisements

En remarque du tableau 2-2 **:**

- Il existe des duplicata **1-5, 4-6, 8-10-11** (mêmes individus)
- Contrainte de surpoids (déplacement du poids maximum autorise, pour individus 3.
- On remarque que le max des poids atteint est 13 pour l'individu 1. . Il est possible d'arrêter à la première population.

Solution possible: Si l'on n'arrive pas au poids optimal dans la première population, le tableau **2-2** sera modifié comme suit

- Après le croisement, on applique la mutation qui change quelque gènes (0vers 1 ou 1vers 0)
- On peut, aussi, réparer les individus, de la façon suivante :

- Respecter les contraintes (enlever des objets au hasard jusqu'à atteindre le poids maximal.
- Éviter la duplication, on refuse un individu qui existe déjà dans la même population.
- Faire un croisement uniforme (choisir à chaque gène aléatoire l'un des deux parents).

8-3-4 La convergence de l'algorithme Génétique

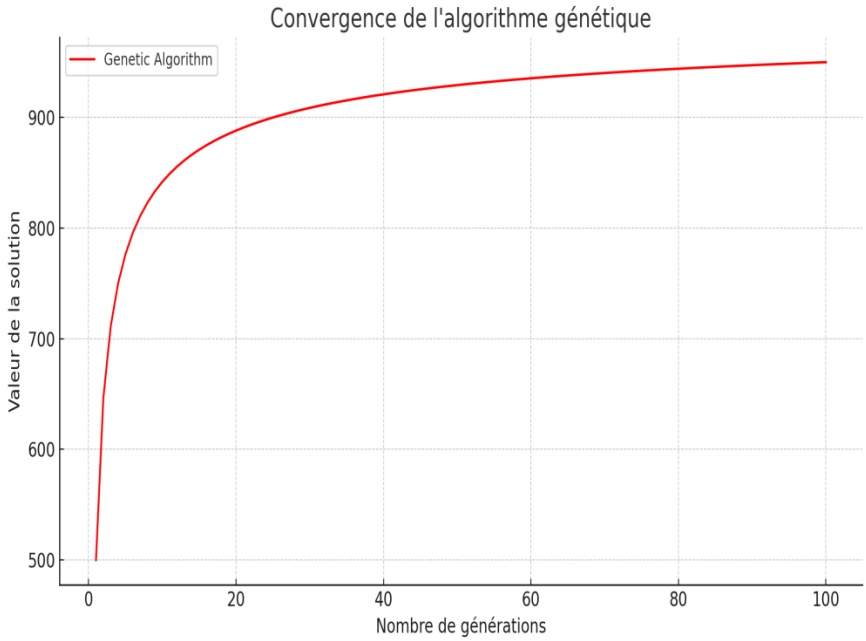


Figure 2-7 : La convergence d'algorithme heuristique [27]

En remarque d'après la **Figure 2-6** qui représente la qualité de la solution en fonction du nombre de générations d'algorithme génétique, que lorsque le nombre de générations augmente entre 0 et 40 générations (population) la valeur de la solution augmente rapidement, et après certaines valeurs. La solution devient très lente et presque constante.

9- Comparaison entre quelques méthodes heuristiques

La comparaison entre les méthodes heuristiques consiste à évaluer leurs performances respectives en termes de qualité des solutions, temps de calcul, robustesse, et facilité d'implémentation.

Les méthodes diffèrent les unes par rapport aux autres en fonction du temps d'exécution, de la qualité de la solution, de la taille et de la complexité du problème.

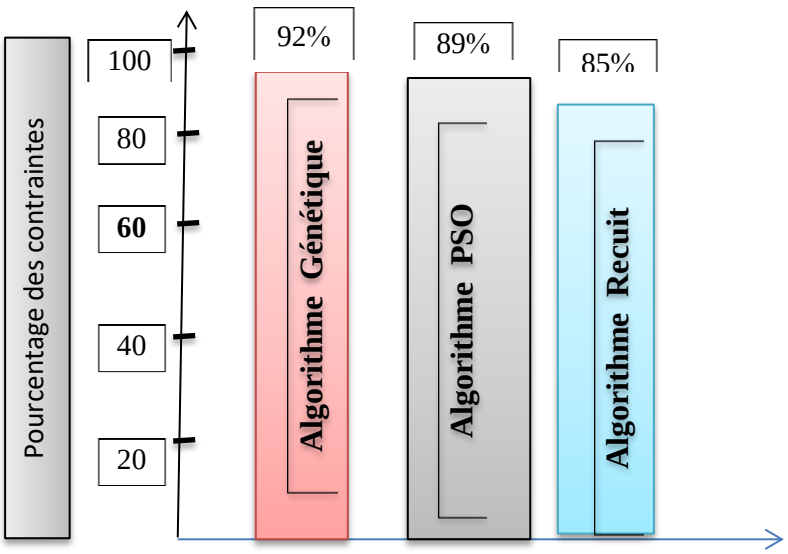


Figure 2.8 : Taux de satisfaction des contraintes de différentes méthodes [28]

La **Figure 2-7** présente le taux de satisfaction des contrainte pour les trois méthodes heuristiques : pourcentage de la satisfaction des contraintes qui est de 92% pour la méthode génétique, 89% pour la méthode PSO et 85¨% pour le recuit Simulé. On remarque que la meilleure méthode qui peut satisfaire presque la majorité des contraintes est l'algorithme génétique.

Ces résultats peuvent changer en fonction des critères suivants :

- Nombre d'itérations
- Temps d'exécution
- Taille de la population / essaim.
- Pour l'algorithme glouton et dans le cas du problème sac à dos, il est rapide, simple, et efficace pour certains cas simples, mais peut mener parfois à un mauvais choix de d'optimum.
- L'algorithme génétique utilise une population de solutions qui évolue par croisement, mutation et sélection. Cependant il est souvent très lent, moins prévisible (résultats pas toujours exactement optimaux), et pour arriver à la solution il faut régler certains paramètres comme la taille de la population et le taux de mutation.

La **Figure 2.8** présente le temps d'exécution selon la taille du problème pour les deux méthodes concernant le problème de Sac à Dos.

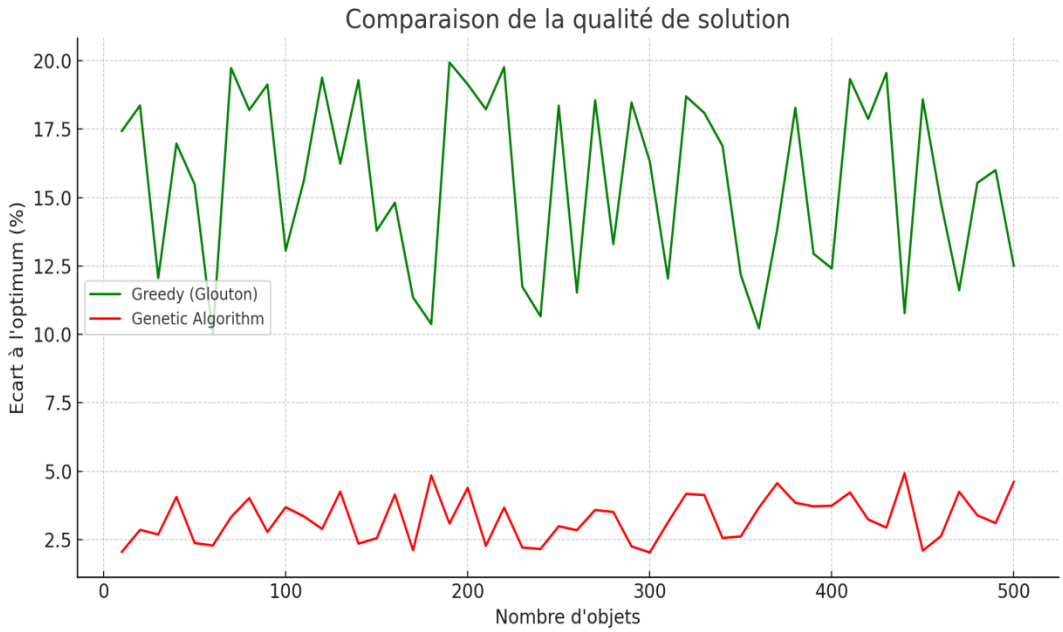


Figure 2-9 Comparaison entre les deux méthodes de la meilleure solution en fonction du nombre d'objets [28]

On remarque d'après la figure 2-8 que la méthode Glouton donne des solutions rapides, mais elles peuvent être lointaines de l'optimum avec une erreur comprise entre entre10 % et 20% (grand écart).

Par contre la méthode Génétique donne des solutions proches de optimum avec une marge d'erreur est entre 2% et 5% (petite erreur).

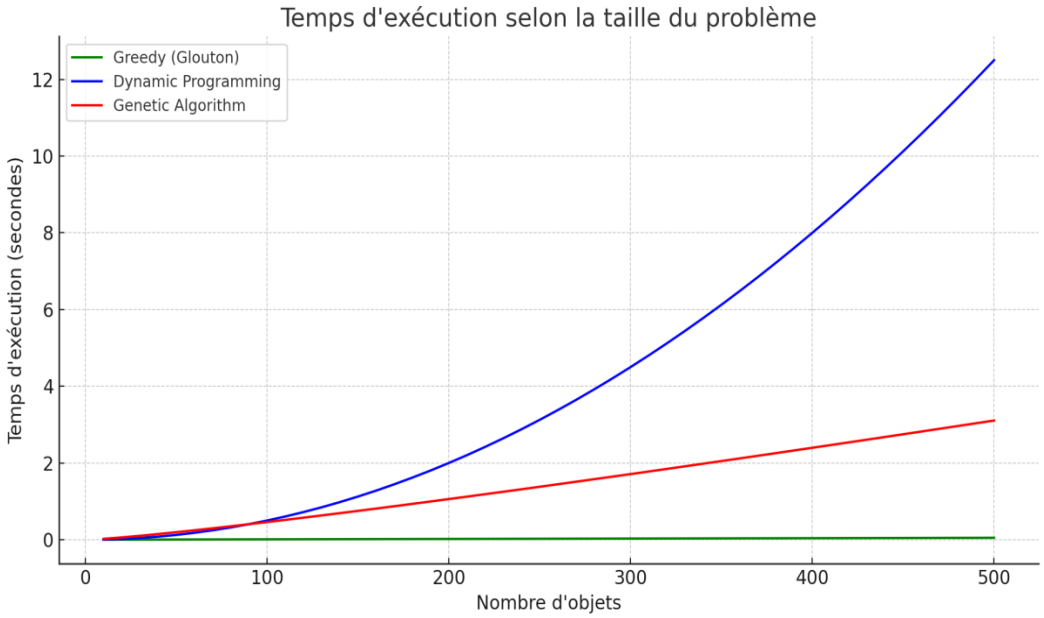


Figure 2-10 : Comparaison entre quelque méthode heuristique selon le temps d'exécution en fonction la taille du problème [29].

La analyse pour le problème de sac à dos, d'après les chercheur en conclue que :

On observe **Figure 2-9** que la courbe qui concerne l'algorithme Glouton (***Greedy*)** (de couleur verte) est presque une ligne droite, ce qui implique qu'elle reste très rapide (croissance).

Tandis que pour la courbe d'algorithme programmation dynamique (de couleur bleu) est de croissance quadratique et avec l'augmentation des nombres d'objets, elle devient lente.

Enfin la troisième courbe qui concerne l'algorithme génétique est plus lente que l'algorithme Glouton.

10- Quelques domaines d'applications utilisant les méthodes heuristiques :

- **Planification et ordonnancement** : Exemple de création d'emplois du temps universitaires.

Le problème du **scheduling** (ou d'ordonnancement) a pour objectif de trouver un emploi du temps qui respecte un ensemble de contraintes en optimisant certains critères. Exemples : l'utilisation des salles, la disponibilité des professeurs et leurs heures de disponibilité.

Algorithmes utilisés : algorithme génétique, recuit simulé

- **Transport et logistique** : comme l'optimisation des tournées de livraison (problème de voyageur de commerce),

Algorithmes utilisés : Colonies de fourmis, recherche tabou,

- **Réseaux et télécommunication** : Attribution efficace des fréquences dans un réseau sans fil pour éviter les interférence, les perturbations ou la dégradation du signal de communication causées par des sources extérieures ou d'autres signaux qui utilisent des fréquences similaires qui entraine une perte de qualité du signal, une réduction de la vitesse de transmission ou une déconnection des dispositifs.

Algorithmes utilisés **:** algorithmes génétiques ou recuit simulé.

- **Conception et ingénierie :** Optimisation de la forme d'une structure en génie civil pour réduire le poids tout en gardant la résistance des matériaux

Algorithmes utilisés : Recuit simulé, algorithmes génétiques

- **Finance et économie** : Optimisation de portefeuille d'investissement sous plusieurs contraintes de risque (risque de crédit, de marché). On minimise les pertes possibles pour optimiser le rendement.

Algorithme utilise **:** Algorithmes heuristique hybrides.

- **Intelligence artificielle et apprentissage automatique :** Optimisation des paramètres que l'on définit avant l'entrainement du modèle tels que le nombre d'époque, le nombre de couches, la fonction d'activation, etc. d'un réseau de neurones

Algorithmes utilisés **:** recherche aléatoire, optimisation par essaim particulaire

11- Conclusion

Dans ce chapitre, nous avons présenté une description des méthodes heuristiques par rapport à la méthode exacte, notamment pour les problèmes-difficile comme le problème du sac à dos.

Les méthodes heuristiques sont des approches puissantes et flexibles contrairement aux méthodes exactes, qui garantissent une solution optimales mais avec un coût plus élevé.

Chaque méthode présente des avantages et des inconvénients. Les méthodes les plus populaires sont, certainement, les approches heuristiques comme la méthode de gloutonne et les algorithmes génétiques. Nous avons traité le problème du sac à dos qui demeure l'un des problèmes d'optimisation classiques.

Nous avons constaté que chaque méthode possède ses propres caractéristiques et que son efficacité dépend de l'objectif visé pour résoudre un problème complexe donné.

Si l'on cherche une solution rapide, on peut utiliser la méthode de gloutonne car elle est simple, rapide à exécuter et donne souvent une solution acceptable ; bien que souvent non optimale

Mais si l'on cherche une solution optimale ou proche, on utilisera la méthode génétique ou une autre, car elles explorent un plus grand espace de solutions au prix d'un temps de calcul plus élevé.

Cette étude peut aider à choisir des méthodes adaptées au problème posé et montre la possibilité de trouver des solutions efficaces.

CHAPITRE III

Contribution

1- Introduction

Les méthodes heuristiques utilisent des techniques de recherche qui visent à trouver des solutions satisfaisantes à des problèmes complexes dans un délai raisonnable sans garantir nécessairement l'optimalité. Ces méthodes sont particulièrement adaptées aux problèmes combinatoires, comme dans le contexte de l'enseignement où la gestion efficace des emplois du temps est un enjeu majeur pour les établissements scolaires et universitaires.

Le problème d'emploi du temps est un problème où la complexité croissante vues les contraintes liées à l'affectation des cours, la disponibilité des enseignants et des salles de classe, les préférences des enseignants et des étudiants ainsi que la disponibilité de locaux et autres contraintes qui sont obligatoires ou flexibles.

Dans ce chapitre, nous explorons plusieurs approches heuristiques ; notamment les algorithmes génétiques, les algorithmes de colonies de fourmis et les méthodes de recherche locale en les adaptant spécifiquement aux contraintes et aux exigences du problème d'emploi du temps.

Nous nous focalisons, par la suite, sur deux méthodes heuristiques, en illustrant chacune par des exemples concrets.

Ce travail a pour ambition de donner une idée sur la conception d'un système capable de générer automatiquement des emplois du temps pour l'ensemble des classes d'un département universitaire. Notre approche consiste à modéliser ce problème grâce à un processus d'optimisation combinatoire. On pourra noter, ainsi, de façon concrète que l'utilisation des méthodes heuristiques offre des solutions pratiques et adaptées aux besoins pédagogiques selon la complexité du problème d'emploi du temps.

2- Présentation du problème

Le problème de l'emploi du temps universitaire est un problème **NP-difficile** et un enjeu crucial qui nécessite une approche systématique et collaborative pour trouver des solutions efficaces, tel que l'organisation des cours, des examens et des activités académiques au sein d'une institution d'enseignement supérieur.

Vu le nombre croissant des effectifs dans le secteur de l'enseignement : Étudiants, niveaux, enseignants, locaux, etc…L'élaboration des emplois du temps peut prendre beaucoup de temps alors que son application doit se faire dans des délais bien déterminés. Il est, de ce fait, impératif que cette tâche soit effectuée dans les délais impartis les plus courts possibles.

Pour répondre à ces impératifs, il est nécessaire de bien cerner les contraintes qui représentent le facteur majeur du problème. On peut dénombrer plusieurs types de contrainte dans le cas de l'élaboration des emplois du temps don les suivants :

- **Complexité des Cours et des Programmes**

- Les universités offrent une grande variété de cours qui peuvent avoir des prérequis, des niveaux de difficulté différents et des horaires variés.

- Les programmes d'études peuvent inclure des cours obligatoires et optionnels (en présentiel ou en ligne), ce qui peut compliquer la planification.

- **Conflits d'Horaires**

- Les étudiants peuvent être inscrits à plusieurs cours qui peuvent se chevaucher, rendant impossible la participation à tous les cours souhaités.

- Les enseignants peuvent également avoir des conflits d'horaires, surtout si l'enseignant est engagé dans plusieurs emplois du temps.

A- Ressources Limitées

- Les ressources matérielles concernent les salles de classe, les laboratoires et d'autres locaux ainsi que le matériel d'enseignement (data shows, ordinateurs, paillasses de travaux pratiques,…) qui sont en nombre limité.

- Les ressources humaines, comme les enseignants, les assistants, et autre personnel de soutien doivent également être gérées de façon optimale.

B- Préférences des Étudiants et des Enseignants

- Les étudiants ont souvent des préférences pour certains horaires (par exemple, éviter les cours tôt le matin ou tard le soir).

- Les enseignants peuvent également avoir des préférences ou des contraintes personnelles qui influencent leur disponibilité.

C- Changements Fréquents

- Les emplois du temps peuvent changer d'une année à l'autre, voire d'un semestre à l'autre, en raison de l'évolution des programmes, des changements de personnel ou d'autres facteurs.

D- Technologie et Outils de Planification

- De nombreuses universités utilisent des logiciels de gestion des emplois du temps pour aider à résoudre ces problèmes mais ces outils peuvent avoir leurs propres limitations et nécessitent une mise à jour régulière, sachant que les ressources se diffèrent d'université à une autre et d'un pays ce qui peut causer des problèmes de compatibilité et de mise à jour.

3- Objectifs de l'optimisation de l'emploi du temps

Les universités doivent équilibrer les besoins des étudiants, des enseignants et des ressources disponibles pour créer un emploi du temps qui favorise l'apprentissage et le succès académique. L'optimisation de l'emploi du temps vise à maximiser l'efficacité et la productivité tout en minimisant le stress et la fatigue.

Ce problème peut être à objectif global, selon comment en formules le problème et qu'est-ce que l'on cherche à optimiser.

Exemples :

- On peut optimiser l'emploi du temps pour tous les groupes, tous les enseignants, toutes les salles en même temps.
- on peut optimiser l'emploi du temps pour toutes les facultés, parce qu'on trouve quelques enseignants qui donnent des cours dans plusieurs facultés et qu'il faut, donc, éviter la contrainte de chevauchement pour les enseignant ou les salles, en optimise un emploi du temps central. On peut dire que ce problème a un objectif local car en peut optimiser juste pour un seul groupe, un prof, ou un département, un niveau.
- On peut chercher à optimiser une solution globale approximative, puis on améliore localement certaines parties exemple préférences d'un prof ou d'un groupe.

Citons quelques objectifs spécifiques de l'emploi du temps :

A- Maximiser l'utilisation du temps : S'assurer que chaque minute est utilisée de manière productive, en évitant les temps morts et les activités non essentielles.

B- Équilibrer les tâches : Répartir les tâches de manière équilibrée tout au long de la journée ou de la semaine pour éviter la surcharge de travail à certains moments que ce soit pour les étudiants ou les enseignants.

C- **Prioriser les activités** : Identifier les tâches les plus importantes et urgentes pour s'assurer qu'elles sont réalisées en premier par exemple les cours essentielles avant les cours facultatives.

D- **Améliorer la concentration** : Créer des plages horaires dédiées à des tâches spécifiques pour favoriser la concentration et réduire les distractions.

E- **Intégrer des pauses** : Planifier des pauses pour les étudiants et pour les enseignants pour éviter la saturation et prendre leur déjeuner.

F- **Faciliter la gestion des imprévus** : Prévoir des marges de manœuvre pour gérer les imprévus sans perturber l'ensemble de l'emploi du temps. Comme exemple un enseignant.

En résumé, l'optimisation de l'emploi du temps vise à créer un cadre de travail qui favorise l'efficacité tout en préservant le bien-être des individus.

4- Le choix de la méthode

Le choix de la méthode dépend en grande partie des contraintes prises en compte pour atteindre la fonction objectif dans un temps limité, si on le compare ce problème avec la méthode exacte et les approches heuristiques nous constatons :

Si on utilise une méthode exacte (programmation linéaire, méthode du simplexe), on se heurte aux problèmes suivants :

- Explosion combinatoire : sachant que le problème emploi du temps est un problème de satisfaction de contraintes, il devient de plus en plus complexe si on a plusieurs salles, de nombreux enseignants, différentes matières, plusieurs classes/ niveaux ou des contraintes spécifiques (climatiques, régionales, préférence, disponibilité…). Les méthodes exactes doivent les évaluer toutes ou presque, ce qui coute énormément en temps de calcul.

- Temps de calcul prohibitif : pour les méthodes exactes ; nous pouvons arriver à des solutions optimales, mais si le problème dépasse une certain taille, les méthodes deviennent trop lentes et le temps dépensé dans le cas de l'emploi du temps est prohibitif.

- Flexibilité limitée : sachant que les contraintes de emplois du temps sont souples (préférences, confort, et peuvent évoluer par exemple ajouter de nouvelles règle en cours de route), les méthodes exacte ne sont pas toujours adaptées à ces changement.

- ❖ Solution acceptable ou optimale : Dans les problèmes de l'emploi du temps, on cherche, souvent, des solutions faisable et pratiques et pas nécessairement la solution parfaite parce que généralement, on ne peut ne pas satisfaire toutes les contraintes.
- ❖ En ce qui concerne les contraintes, les méthodes exactes utilisent des variables réelles mais pour ce problème précis, on utilise souvent des variables binaires.

En résumé les méthodes exactes pour ce problème ne sont pas efficaces car elles sont trop lentes, peu flexible, et peu adaptées à la complexité réelle du problème.

Le problème de l'emploi du temps est un problème heuristique d'optimisation combinatoire /discrète très complexe parce qu'il y a de nombreuse contraintes à satisfaire et on n'arrive pas toujours à une solution unique optimale, mais selon les critères choisis on peut trouver plusieurs solutions dont certaines sont meilleures et d'autres acceptable ou satisfaisantes.

Nous allons utiliser différentes méthodes comme les algorithmes génétiques et colonie de fourmi pour traiter ce problème.

5- La complexité de l'emploi du temps

- **Meilleurs des cas (Best case)** : Si les contraintes sont faibles : exemple peu de cours, beaucoup des salles avec un nombre d'enseignants suffisants, peu de niveaux,… une solution peut être trouvée rapidement en temps linéaire ou quadratique selon les algorithmes utilisés. Ce cas rare en réalité à cause du manque de ressources et beaucoup de contraintes que ce soit humaines ou matérielles (On aura des contraintes de type C = O(n) ou O (n^2)).
- **Pire des cas (Worst Case)** : Si en a N cours, R ressources, T créneaux horaires, le nombre de combinaison possibles peut être de l'ordre O((⟦R *T)⟧ ^N) donc exponentiel.
- **Moyen des cas (Average Case)** : dans ce cas la complexité est souvent proche du pire des cas. C'est le cas rencontré dans la plupart des cas réels.

6- Mise en œuvre d'un algorithme d'optimisation de l'emploi du temps

Dans ce travail, nous proposons de concevoir un système capable de générer automatiquement les emplois du temps pour l'ensemble des classes d'un département universitaire. Notre approche consiste à modéliser ce problème comme un processus d'optimisation combinatoire.

Ce processus peut être résumé en cinq étapes principales, que nous allons détailler ci-dessous :

1. Définir la structure d'une solution.

2.Élaborer une méthode de génération de solutions aléatoires mais plausibles (pour l'exploration).

3. Identifier les contraintes permettant de comparer et de hiérarchiser les solutions.

4. Définir une fonction objective pour évaluer la qualité des solutions.

5. Concevoir un algorithme capable d'explorer efficacement l'espace de recherche à la recherche d'un optimum.

Nous allons examiner en détail chacun de ces éléments

6.1 La structure de la solution

Dans un département universitaire, on trouve généralement les composantes suivantes :

- **Les niveaux :** (Domaine/Filière/Spécialité)
  - Première année Licence MD
  - Deuxième année Licence LMD
  - Troisième année Licence LMD
  - Première année Master LMD
  - Deuxième année Master LMD
- **Les groupes et les sections :**
  Chaque section appartient à un niveau, et chaque groupe à une section.
- **Les matières :**
  Chaque niveau possède un ensemble défini de matières par semestre, réparties en trois types : cours, travaux dirigés (TD) et travaux pratiques (TP).
- **Les salles :**
  Elles sont classées selon le type d'activité :
  - Amphithéâtres pour les cours magistraux,
  - Salles ordinaires pour les TD,
  - Laboratoires pour les TP.
- **Les enseignants :**
  Chacun peut enseigner un ensemble défini de matières selon les niveaux, avec une charge horaire limitée, qui dépend du type de matière (cours, TD ou TP).
- **La plage horaire :**
  La semaine scolaire s'étend du samedi au jeudi, avec 6 séances quotidiennes de1h30, de 08h00 à 17h00.

Le système que nous développons vise à produire, pour chaque groupe, une affectation cohérente des matières, des enseignants, des salles et des horaires, en évitant les conflits et en respectant un maximum de contraintes.

Chaque configuration complète d'emploi du temps (même très vaste) est considérée comme une solution dans l'espace de recherche.

Exemple de représentation

Examinons un Exemple d'un emploi du temps aléatoire d'un groupe donné :

Jours/Heure	08:00 - 09:30	09:30 - 11:00	11:00 - 12:30	12:30 - 14:00	14:00 - 15:30	15:30 - 17:00
Samedi	**Matiere1 Salle1 Enseigant1**					
Dimanche						
Lundi		**Matiere2 Salle2 Enseigant2**	**Matiere3 Salle3 Enseigant3**			
Mardi					**Matiere4 Salle4 Enseigant4**	
Mercredi	**Matiere5 Salle5 Enseigant5**					
Jeudi						

Figure 3.1 Exemple d'un emploi du temps

Cependant, cette représentation est peu efficace en termes d'espace mémoire, en raison des nombreuses cases vides. Pour optimiser le stockage, nous proposons une structure linéaire fondée sur une liste de 36 créneaux (6 jours × 6 séances par jour).

Liste 1	Liste 2	Liste 3	…	Liste 36

Chaque créneau (liste) peut contenir plusieurs quadruplets (classe, salle, matière, enseignant). Par exemple, la liste 1 correspond au samedi de 08h00 à 09h30, la liste 3 au samedi de 11h00 à 12h30, etc.

Donc la forme générale d'une solution peut être résume comme suit :

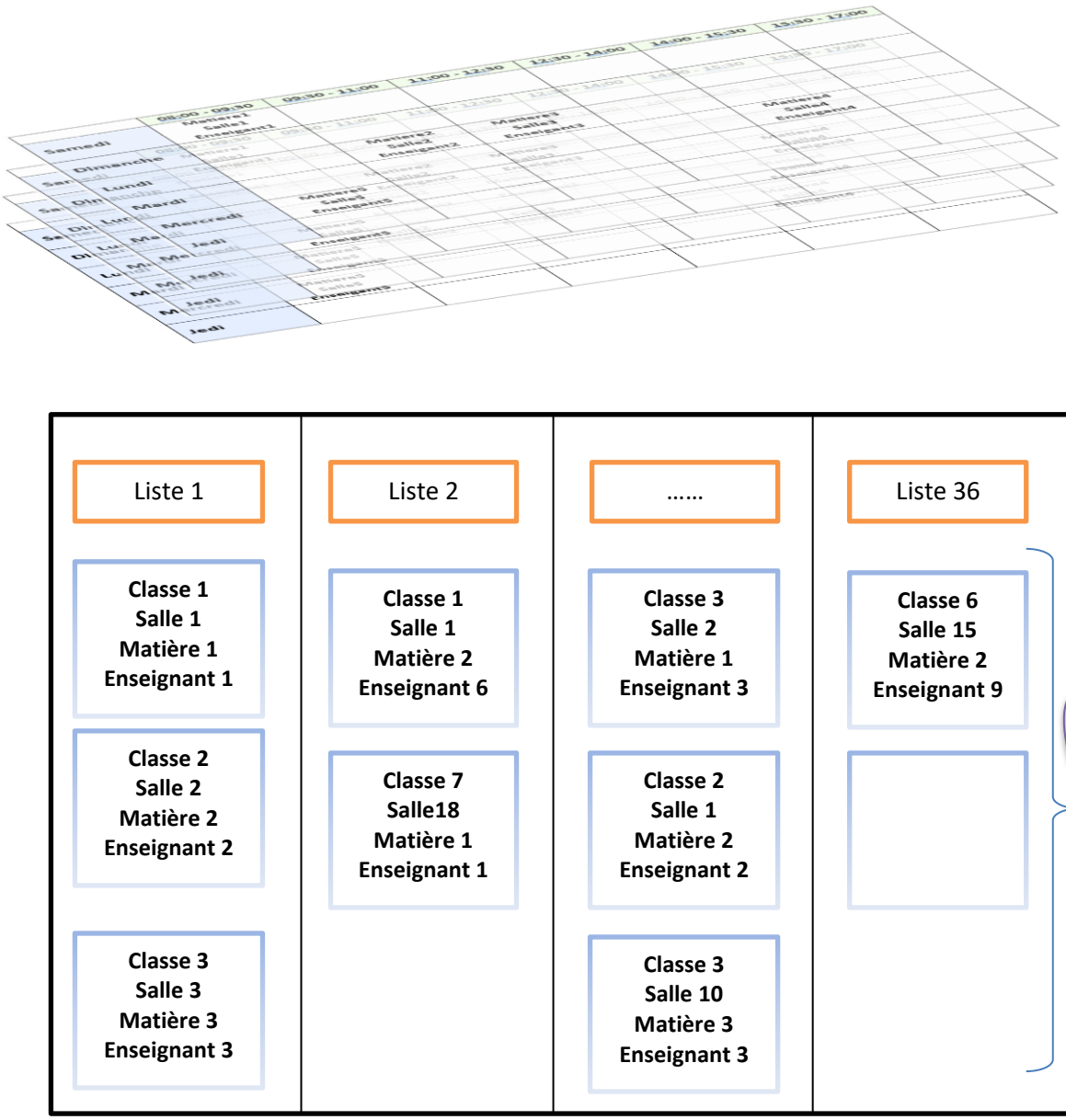


Figure 3.2 Forme générale d'une solution

6.2 Génération des solutions aléatoires :

La plupart des algorithmes d'optimisation commencent par explorer aléatoirement l'espace de recherche. Il est donc crucial de pouvoir générer des solutions aléatoires mais valides, respectant les contraintes dures (celles qui ne peuvent être violées) tout en préparant le terrain pour l'évaluation des contraintes souples (préférences).

6.3 Identifier les contraintes

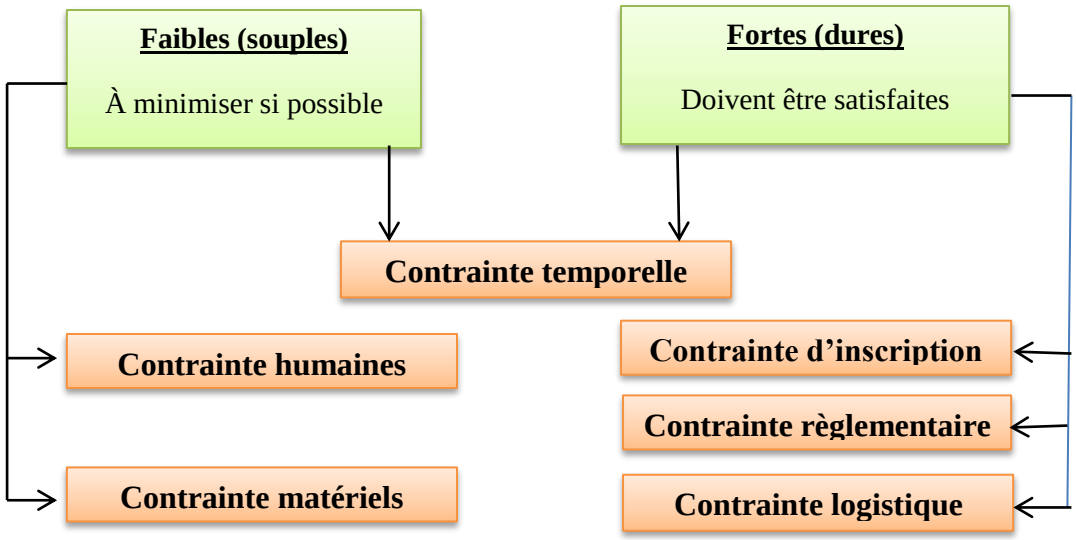


Figure 3.3 : Différents types des contraintes

En va sites quelque contrainte :

6 .3 .1 Contraintes fortes doivent être satisfaites :

Contrainte temporelle

- Pas de conflits horaires : un enseignant, un groupe d'étudiants, ou une salle ne peut être affecté à deux cours en même temps.
- Durée des cours respectée : chaque cours, TP ou TD a une durée fixe qui doit être respectée.
- Heure de pose : il peut être nécessaire de prévoir une pause après les trois premières séances.
- La charge horaire de chaque enseignant ne doit pas être dépassée.

Contraintes d'inscription

- Capacité des salles : une salle doit pouvoir contenir tous les étudiants du cours.
- Disponibilité des enseignants : à certain moment, certains enseignants ne sont pas disponibles.
- Limites sur le nombre d'étudiants par cours : certaines salles ont une capacité maximale, et les cours ne peuvent pas dépasser ce nombre.
- Besoins spécifiques pour certains cours (laboratoires, équipements spéciaux, etc.) qui nécessitent des salles spécifiques.

Contraintes règlementaires :

- Respect des réglementations universitaires concernant les horaires de cours, les jours de repos (weekend), et d'autres normes institutionnelles.
- Repartir les cours de manière équilibrée sur la semaine.
- Disponibilité des enseignants : chaque enseignant a un emploi du temps qui doit être respecté, incluant les heures de cours qu'ils peuvent donner.
- Une matière doit être affectée à un enseignant qualifié et disponible.

Contraintes logistiques :

- Besoins spécifiques pour certains cours (laboratoires, équipements spéciaux, etc.) qui nécessitent des salles spécifiques.

6.3.2 Contraintes faible à minimiser si possible

Contraintes humaines :

- Pour les étudiants, éviter la charge insoutenable pendant la journée, et assurer que les enseignants puissent avoirs des journées en repos.
- Pour enseignant préfère avoir en minimum 2 cours dans la journée, et maximum 2 jours dans la semaine.

Contraintes Temporelles :

- Eviter les trous, le temps libre non désiré entre deux cours dans une journée pour un groupe ou enseignant.
- Certains groupes préfèrent des jours sans cours, et certains enseignants préfèrent ne pas enseigner tôt ou tard l'après-midi.

6.4 Définir de La fonction objective

La création d'un emploi du temps universitaire nécessite une approche systématique qui prend en compte à la fois les objectifs d'optimisation et les contraintes variées. Cela implique souvent l'utilisation d'outils d'optimisation, de programmation linéaire ou d'algorithmes heuristiques pour trouver des solutions viables qui répondent aux besoins de toutes les parties.

Donc la fonction objectif c'est minimiser les conflits et maximiser la qualité de emploi du temps.

L'emploi du temps est constitué d'un ensemble d'affectation représentée par la fonction T a six paramètres **: T (J, P, C, E, M, S)** où :

• **J** : représente le jour de la semaine ∈ [1, 6]. (Samedi ... Jeudi)

• **P** : représente la période de la journée ∈ [1, 6]. (08:00 - 09:30 ... 15:30 - 17:00)

• **C** : représente la classe concernée ∈ [1, CMAX]. CMAX el le nombre total des classes.

• **E** : représente l'enseignant concerné ∈ [1, EMAX]. EMAX el le nombre total des enseignants.

• **M** : représente la matière concernée ∈ [1, MMAX]. MMAX el le nombre total des matières.

• **S** : représente la salle concernée ∈ [1, SMAX]. SMAX el le nombre total des salles.

Pour désigner une affectation donnée on écrit :

$$T(J_i, P_j, C_k, E_l, M_m, S_n) = \begin{cases} 1, & \textbf{si une affectation existe avec ces parametres} \\ 0, & \textbf{sinon} \end{cases} \quad \textbf{(1)}$$

Ou **T (i∈J, j∈P, k∈C, l ∈E, m∈M, n∈S)** est une variable de décision

Pour calculer le nombre d''affectation d'une journée donnée pour une classe donnée on écrit :

$$T(J_i, C_k) = \sum_{j=1}^{6} \sum_{l=1}^{EMAX} \sum_{m=1}^{MMAX} \sum_{n=1}^{SMAX} T(J_i, P_j, C_k, E_l, M_m, S_n) \quad \textbf{(2)}$$

De la même manière on peut calculer le nombre des cours données par un enseignant donnée en mettant :

$$T(E_l) = \sum_{i=1}^{6} \sum_{j=1}^{6} \sum_{k=1}^{CMAX} \sum_{m=1}^{MMAX} \sum_{n=1}^{SMAX} (J_i, P_j, C_k, E_l, M_m, S_n) \quad \textbf{(3)}$$

La fonction objective ***f*** est une combinaison de sous fonctions f_i que chacune traite une des contraintes précédemment présentées en ajoutant des pénalités aux solutions ayant un mauvais respect des contraintes.

1. Contrainte des bonnes partitions des charges ***f1*** : (répartir harmonieusement les séances dans la semaine pour éviter des journées surchargées ou trop vides).

$$E1(C_k) = \max_i(T(J_i, C_k)) - \min_{\neq 0}\left(T(J_i, C_k)\right) \quad (4)$$

$$\text{Donc } f1 = \sum_k^{CMAX} E1(C_k) \quad (5)$$

2. Contrainte des pauses déjeuner ***f2*** : (Chaque classe doit avoir un vide de 11:00 – 12:30 ou de 12:30 – 14:00).

$$E2(C_k) = \sum_{i=1}^{6} T(J_i, P_3, C_k) \ . \ T(J_i, P_4, C_k) \quad (6)$$

$$Donc\ f2 = \sum_k^{CMAX} E2(C_k) \quad (7)$$

3. Contrainte de vide pour le jeudi après-midi ***f3*** : (éviter autant que possible les séances le jeudi après-midi

$$E3(C_k) = T(J_6, P_5, C_k) + 2\,T(J_6, P_6, C_k) \quad (8)$$

$$Donc\ f3 = \sum_k^{CMAX} E3(C_k) \quad (9)$$

4. Contrainte de cinq jours d'étude ***f4*** : (chaque classe devrait avoir un jour de repos par semaine).

$$E4(C_k) = \begin{cases} 1, & si\ \min_i\left(T(J_i, C_k)\right) \neq 0 \\ 0, & sinon \end{cases} \quad (10)$$

$$Donc\quad f4 = \sum_k^{CMAX} E4(C_k) \quad (11)$$

La fonction objective ***f Global*** est donc :

$$f = \alpha.f1 + \beta.f2 + \gamma.f3 + \delta.f4 \quad (14)$$

Les paramètres $\alpha, \beta, \gamma, \delta, \varepsilon$ contrôlent l'influence des contraintes par rapport aux autres.

<u>Remarque</u> :

-Si on utilise l'optimisation linéaire ou entière, on sépare la fonction objective et ces contraintes

-Si on utilise des métas heuristiques, on peut tout regrouper dans une seule équation avec des pénalités

6 .5 La distance entre deux solutions :

La plupart des algorithmes d'optimisation utilise une fonction pour calculer la distance entre deux solutions distinctes afin de guider le processus d'exploration et d'exploitation.

C'est ainsi que nous définissons la fonction ***D*** qui calcule la distance entre deux emplois du temps distincts pour chercher une solution optimale (un emploi du temps optimal).

Remarque :

Si la distance est très petite, on dira que les deux emplois du temps sont corrélés, sinon ils sont éloignés et sont, donc, différents.

Il existe différentes méthodes de calcul des distances :

- **La distance euclidienne** dans le cas où les résultats sont numériques :

  D (A, B) = $\sqrt{\sum_{i=1}^{n}(A_{i-}B_i)^2}$

- **Visualisation des solutions avec la méthode de ACP** (analyse en composants principale) qui utilise la méthode de réduction de dimension 2D ou 3D et permet de tracer chaque emploi du temps comme un point.
- **Distance de Hamming** : pour les vecteur binaire comme suit :

  D (A, B) = $\sum_{i=1}^{n} |A_i - B_i|$

6.6 Les algorithmes d'optimisation :

6.6.1 Algorithme Glouton

Début

Entrées :

- Listes des cours à planifier (chaque cour : matière, groupe, enseignant, durée)
- Disponibilité des enseignants
- Disponibilités des groupes
- Liste des créneaux horaires (jours, heures)
- Liste des salles disponibles avec capacité

Initialise :

Emploi Du Temps[$jours$] créneau]=vide

Pour chaque Matière dans la liste :

Pour chaque jour :

Pour chaque créneau du jour :

Si enseignant, classe et salle sont disponibles à ce moment :

Affecter le cours pour ce créneau et salle
Marquer comme enseignant, la classe, et la salle comme occupés
Passer au cours suivant
Fin si
Fin pour
Fin pour
Si aucun créneau trouvé :
Ajouter a une liste d'échecs
Fin pour
Retourner : emploi du temps rempli + cours non affectes
Fin

6.6.2 Algorithme d'optimisation par Colonie de fourmis (Ant Colony Optimization, ACO)

L'**ACO** est inspiré du comportement des colonies de fourmis dans la nature.

- Sachant que le but des fourmis c'est chercher de la nourriture pour consommer pendant l'hiver, par la recherche des chemins entre la nature et la fourmilière.
- Elles déposent des phénomènes sur les chemins les plus courts.
- Les chemins les plus efficaces sont renforcés par les phéromones.

Termes utilisés :

- **Fourmi :** présente un emploi du temps (agent virtuel).
- τ[Cours, créneau] : quantité de phéromones associés au fait d'attribuer un cours pour un certain créneau (heure +salle +prof).
- + = : Ajout d'une quantité de phéromones.
- Q : Constante qui contrôle combien de phéromones sont déposées.
- Nb_ conflit : le nombre de conflits dans les solutions (exemple : deux cours en même temps dans la même salle).
- Q / (1+nb_conflits) :
  Plus il y a de conflits, plus la valeur est petite, moins de phéromones sont déposées.
  Moins il y a de conflits, la valeur est grande, plus de phéromones (donc plus attirante pour les prochaines fourmis).

Qu'est-ce que la phéromone ?

Dans la nature, les vraies fourmis déposent une substance chimique appelée phéromone sur le sol pour indiquer un bon chemin à leurs congénères.

Dans l'informatique, la phéromone est une valeur numérique qui représente la qualité d'un choix fait dans le passé, donc elles servent de mémoire collective.

A quoi sert cette phéromone ?

Quand une fourmi doit choisir ou placer un cours, elle regarde :

- La quantité de phéromone sur chaque créneau.
- Une règle heuristique (est-ce sue prof libre, salle libre).
- Plus la phéromone est élever, plus la fourmi est attirée par ce choix.
- Apres que toutes les fourmis ont construit un emploi du temps, en fait la mise à jours des phéromones par :

- Evaporation : on diminue toutes les phéromones un peu **($\tau=\tau*(1-\rho)$).**
- Renforcement : les meilleures solutions déposent de la phéromone.

Paramètres ACO :

α : Influence des phénomènes

β: Influence d'heuristique (disponibilité, conflits)

ρ : Taux d'évaporation de la phéromone

N : Nombre d'itérations

M : Nombre de fourmis

Nœuds (Etats) : combinaison [cours, prof, salle, heure]

Chemins : choix possibles pour affecter chaque cours

Algorithme

Début

//**Entrées** :

- Listes des cours à planifier (chaque cour : matière, groupe, enseignant, durée)
- Disponibilité des enseignants
- Disponibilités des groupes
- Liste des créneaux horaires (jours, heures)
- Liste des salles disponibles avec capacité

// **Initialiser phéromones τ**[cours, créneau horaire possible] a une petite valeur

Pour chaque itération de 1 à max itération :

Pour chaque fourmi Faire

Initialiser une solution vide, Emploi du temps = {}

Initialisation compteur de conflit = 0

Pour chaque cour Faire

Générer la liste des créneaux valides (salle, heure, prof) possible :

Pour chaque créneau possible Faire

Calculer la probabilité :

P= τ[cours, créneau] α^∗ heuristique disponibilité β^

Fin pour

Choisir un créneau en fonction de **P** (roulette ou max P)

Si pas de conflit (salle libre, prof libre,..) Alors

Ajouter le créneau a l'emploi du temps

Si non Ajouter à l'emploi du temps (solution partielle)

Incrémenter nb conflits

Fin Si

Fin pour (chaque cours)

Sauvegarde la solution construire par cette fourmis

Mettre à jours les phénomènes

Pour chaque case τ**[cours, créneau]** Faire

τ= τ∗**(1 - ρ) //** ρ est le taux d'évaporation

Fin Pour

Pour chaque fourmi avec bonne solution (peu de conflits) Faire

Pour chaque cours attribué dans sa solution Faire

τ[Cours, créneau] += Q / (1+nb_conflits)

Retourner la meilleure solution trouvée

Fin pour

Fin pour // pour chaque fourmis

Mémoriser la meilleure solution de cette itération

Fin pour // itérations

Retourner la meilleure solution globale trouvée.

Fin

L'algorithme de colonie de fourmis est un algorithme stochastique, donc les résultats peuvent varier d'une exécution à l'autre. Il peut être utile de tester différentes configurations et d'analyser les résultats pour trouver la meilleure approche.

6.6.3 Algorithme Génétique

Une population = un ensemble d'emploi du temps différents

Individu (un Chromosome) = emploi du temps complet

Gène = (cours i, Enseignant i, Salle i, jour i, Heure i)

N : taille de la population

G : Nombre de génération

Pc : Taux de croissement

Pm : Taux de mutation

Fitness = score maximal – total des pénalités

Pour chaque individu, on calcule un score de qualité basé sur :

- Conflits horaire (Pénalité)
- Respect des disponibilités (bonus ou pénalité)
- Répartition équilibré des cours (pénalité)

Algorithme

Début

//Entrées :

- Listes des cours à planifier (chaque cour : matière, groupe, enseignant, durée)
- Disponibilité des enseignants
- Disponibilités des groupes
- Liste des créneaux horaires (jours, heures)
- Liste des salles disponibles avec capacité

// 1.Représentation d'un individu

Individu = [(Cours1, Enseignant1, Salle1, Créneau1), (Cours2, Enseignant2, Salle2, Créneau2), (Cours3, Enseignant3, Salle3, Créneau3), … (Cours Mmax, Enseignant Emax, Salle Smax , Créneau Pmax)],

// 2.Initialisation une population P de N individus

N individus :

Pour chaque individu :

Pour chaque cours :

- Choisir aléatoirement un enseignant autorisé
- Choisir une salle disponible
- Choisir un créneau horaire libre

Ajouter l'individu a P

Fin pour

// 3.Fonction Fitness

Pour chaque individu :

Calculer le nombre de conflits :

// Contrainte de conflits

Calculer bonus :

// Contrainte souple

Cours bien repartis dans la semaine

Respect des préférences d'enseignant

Fitness = ScoreMax – Pénalité

Fin pour

Fin pour (pour chaque individus)

// **4.Selection**

Utiliser la méthode de roulette ou de tournoi :

Choisir les meilleurs individus pour reproduction

//5. Croissement

Pour chaque paire de parents :

Choisir un point de croisement aléatoire

Enfant1= 1ere partie de parent1+ 2eme partie de parent2

Corriger les incohérences (si besoin)

Ajouter l'enfant a la nouvelle population

//6.Mutation

Pour chaque enfant :

Pour chaque cours :

Avec une probabilité Pm :

Changer l'enseignant, ou changer la salle, ou changer le créneau

//7.Remplacement

Remplacer les individus les moins performants par les enfants

//8. Critère d'arrêt

Si : nombre de génération atteint, ou Emploi du temps sans conflit trouvé Alors

Retourner le meilleur individu comme solution finale

Sinon retour à l'étape 3

Fin

7. Conclusion

Dans ce chapitre, nous avons présenté en détail le problème des emplois du temps universitaires, une problématique complexe de nature combinatoire (parce qu'il y a un nombre de combinaison possible est énorme), avec un grand nombre de contraintes.

Nous avons proposé trois algorithmes pour résoudre ce problème de façon optimale : la méthode Gloutonne (algorithme de Glouton), les agents génétique et l'optimisation par colonie de fourmis qui sont des paradigmes différents et permettent d'aborder le problème sous plusieurs angles, en tenant compte à la fois des contraintes strictes (pédagogique et logistique) et des objectifs d'optimisation.

La différence entre ces trois méthodes inclue la rapidité et la qualité des solutions obtenues.

Les techniques utilisées dans les différentes méthodes constituent des outils complémentaires pour la résolution du problème de l'emploi du temps.

Une mise en œuvre comparative sera abordée dans le chapitre suivant et permettra d'identifier la méthode la plus pertinente selon les critères de performances définies.

CHAPITRE IV

Implémentation

1- Introduction

Ce chapitre se concentre sur l'application de méthodes heuristiques pour résoudre le problème de l'emploi du temps, en mettant en avant l'utilisation de MATLAB comme outil de modélisation et de simulation.

2- L'environnement de développement

Ce travail est implémenté en utilisant l'environnement de développement Matlab version 7.10 sous Windows 10 Professionnel. Nous avons utilisé un PC DELL avec les propriétés suivantes:

Processeur	Intel(R) Core (TM) i7-7600U CPU @ 2.80GHz 2.90 GHz
Mémoire RAM installée	8,00 Go (7,88 Go utilisable)
Type du système	Système d'exploitation Windows 10 - 64 bits, processeur x64

A propos de WINDOWS

ÉditionWindows :	10 Professionnel
Version :	22H2
Installé le :	01/01/2025
Experience Windows Feature Experience: Pack 1000.19061.1000.0	

Figure 4.1 : les propriétés de notre PC

3- Matlab « Mat (matrix) et laboratory (lab.) »

3-1 Présentation

Matlab est un langage de programmation de quatrième génération et un environnement de développement ; il est utilisé à des fins de calcul numérique. Développé par la société The Math Works, MATLAB permet de manipuler des matrices, d'afficher des courbes et des données, de mettre en œuvre des algorithmes, de créer des interfaces utilisateurs, et peut s'interfacer avec d'autres langages comme le C, C++, Java, et Fortran. Les utilisateurs de MATLAB (environ un million en 2004) sont de milieux très différents comme l'ingénierie, les sciences et l'économie dans un contexte aussi bien industriel que pour la recherche. Matlab peut s'utiliser seul ou bien avec des Tools box (« boîte à outils »).

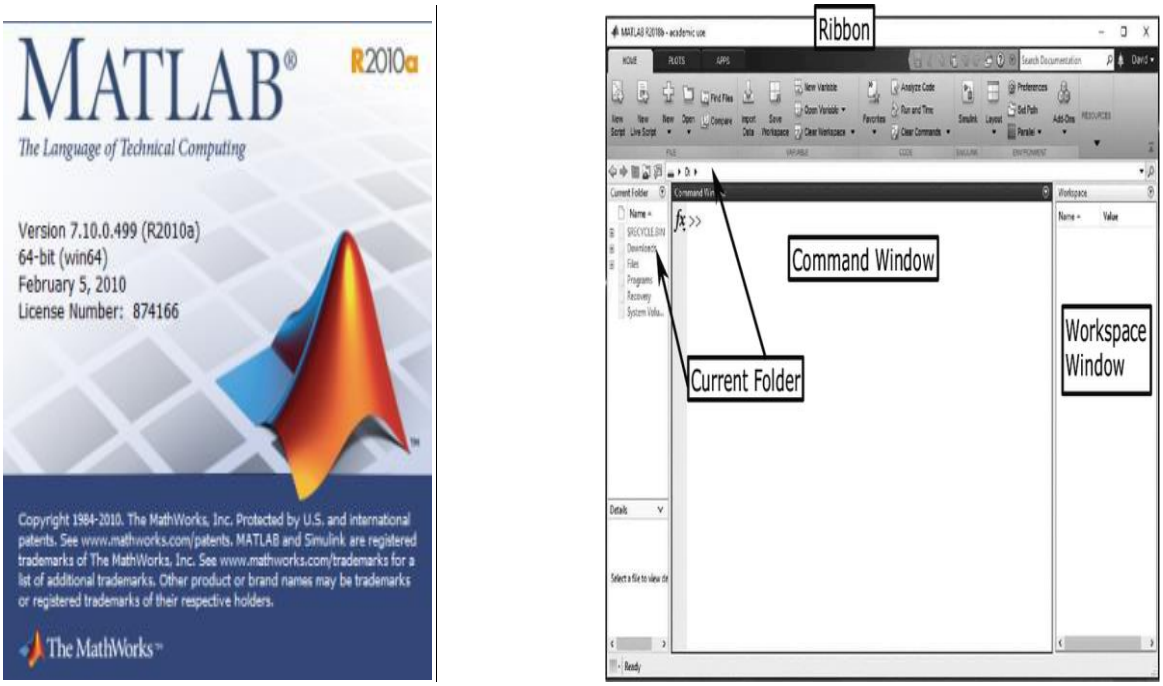


Figure 4 -2: Logo MATLAB (version R2010a)

3-2 Historique de *Matlab*

Matlab a été développe à la fin de l'année ***1970*** comme une premier version par ***Cleve Moler***, professeur de mathématique, pour permettre à ses bibliothèque ***LINPACK*** et ***EISPACK*** sans avoir à programmer avec ***Fortran***.

La première version commerciale de ***Matlab*** a été lancée en ***1984***, et la plus récente en R2025a sortie en mars 2025 où on trouve des améliorations par l'intelligence artificielle, des outils avancée pour la simulation et une meilleure intégration avec Python, ainsi q une interface plus moderne et des performances accrues.

Entre ces deux versions, il existe d'autres tel qu'en 1990 où Simulink devient un outil clé pour la modélisation comme, introduction des matrices, structure,…

En ***2000***, apparition de l'environnement ***Matlab Desktop*** (interface avec fenêtre), développent orienté objet.

En ***2004***, nouvelle gestion de mémoire ; meilleur performance, ajout de nombreux ***Toolbox*** spécialises.

En ***2008***, introduction du préfixe « R » dans les versions exemple ***R2008.***

Entre ***2020*** et **2024,** la version ***Matlab R2020a-b*** jusqu'à ***R2024a-b***, il y a une grande amélioration en IA, visualisation en 3D, outils Cloud, développement en ligne via ***Matlab Online***.

3-3 Les différents usages de MATLAB

Les cas d'usage de ***Matlab*** sont nombreux. Le langage de programmation est notamment utilisé dans [3]:

- Système de contrôle.
- Machine Learning.
- La maintenance prédictive.
- Le traitement du signal et les séries temporelles.
- L'automatisation des tests.
- Les systèmes de télécommunication.
- La robotique.

4. Implémentation

4. Interface principale

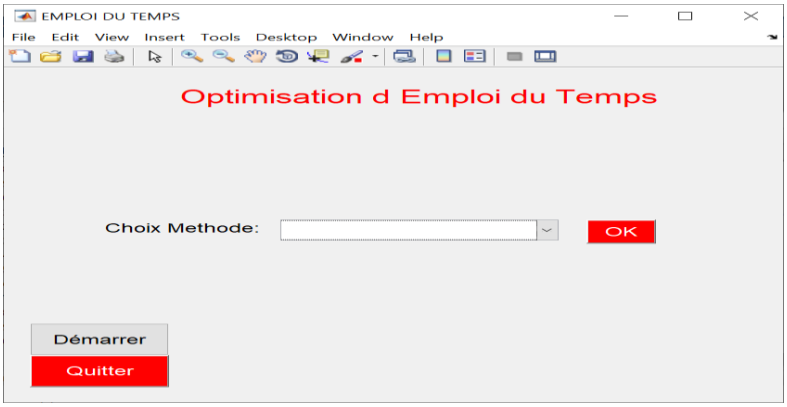
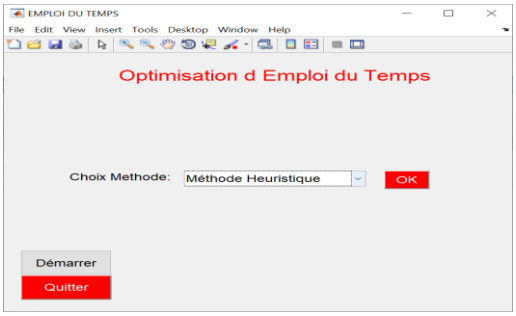


Figure 4 -3: Interface principale de l'application

L'interface principale ci-dessus (Figure 4-3) est une interface graphique concernant la gestion de l'emploi du temps. Elle est composée d'un bouton **" Choix Méthode"**, après le clic du bouton **"Démarrer".**

L'utilisateur peut choisir l'une des deux méthodes **("Méthode Manuelle"** ou **"Méthode Heuristique"**).



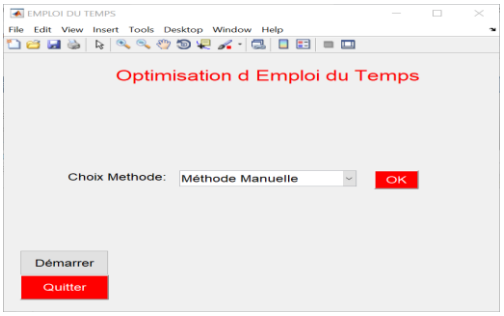


Figure 4 -4: Choix de méthode

4.2 Méthode manuelle

Dans ce cas, une interface graphique interactive permet à l'utilisateur de sélectionner manuellement les données liées à l'emploi du temps, telles que l'enseignant, la matière, la salle, le jour et l'heure.

Après chaque sélection, l'information est automatiquement affichée dans un tableau représentant l'emploi du temps.

Le système intègre un mécanisme de vérification en temps réel des contraintes. Ainsi, si une affectation viole une contrainte (par exemple, un enseignant doublement occupé, une salle déjà prise, un créneau interdit, ou une limite hebdomadaire dépassée), un message d'erreur clair s'affiche, empêchant l'ajout du cours.

Bien que cette méthode manuelle permette un contrôle précis de chaque affectation, elle présente deux inconvénients majeurs ; d'une part la tâche devient complexe à mesure que le nombre de données augmente et d'autre part le temps nécessaire à la complétion de l'emploi du temps est élevé surtout lorsque le nombre d'enseignants, de matières ou de créneaux augmente. Cela souligne la nécessité d'automatiser la génération de l'emploi du temps à l'aide d'algorithmes comme la méthode gloutonne ou génétique.

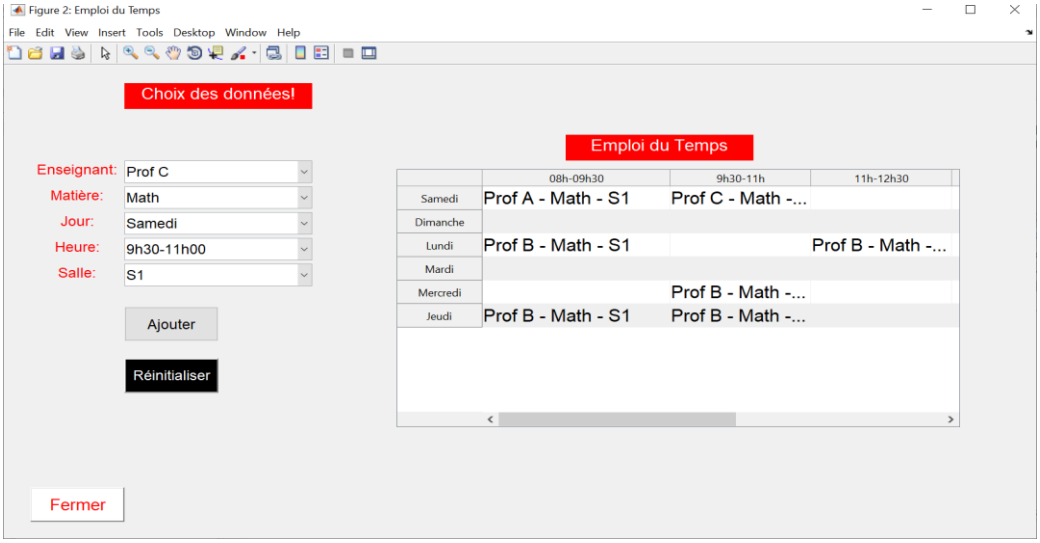


Figure 4 -5: Emploi du temps avec la méthode manuelle

Exemple de contraintes utilisées

- Chaque enseignant ne peut assurer que présente 3 cours par jours au maximum,
- 5 séances par semaine,
- Absence de conflits de salle ou horaires,
- Pas de cours le jeudi à partir de 14heure.

La figure ci-dessous présente un emploi du temps manuel avec des contraintes non satisfaites : un message d'erreur qui s'affiche :

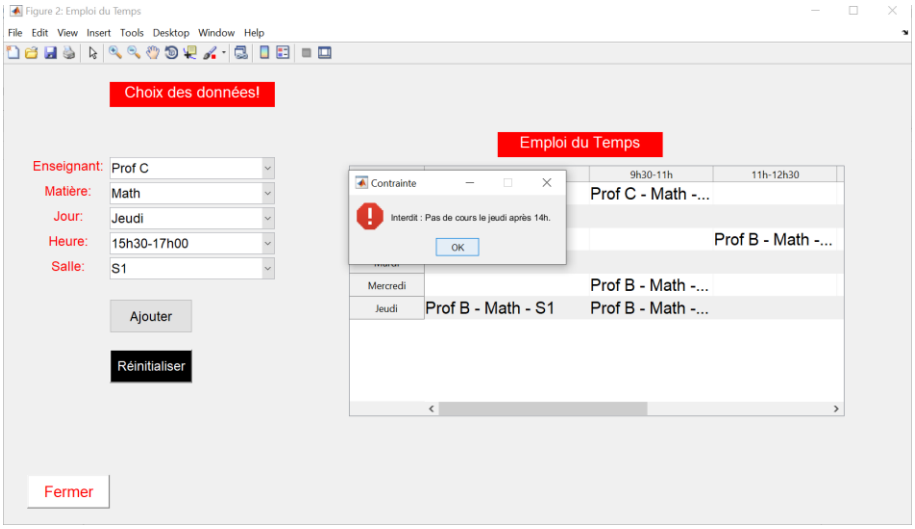


Figure 4 -6 : Message d'erreur dans application manuelle

4.3 Implémentation d'emploi du temps utilisant la méthode heuristique

4.3.1 Description de l'interface Utilisateurs

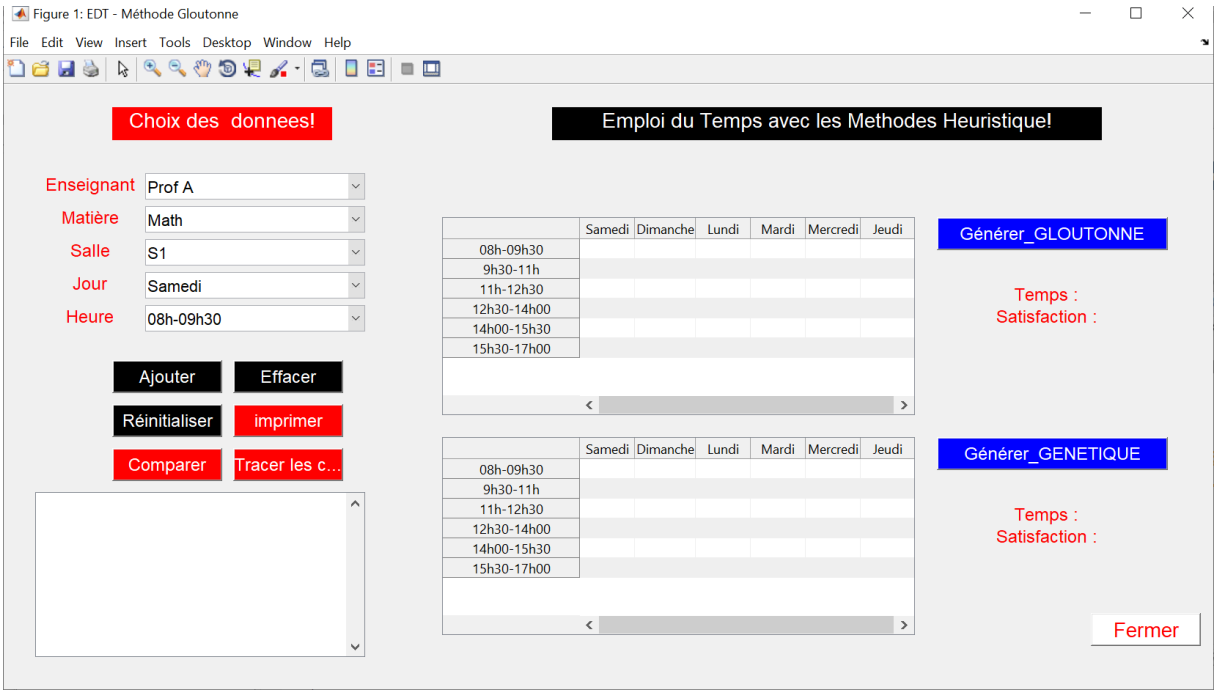


Figure 4 -7 : Interface graphique avec méthode heuristique

L'interface développée pour la génération d'un emploi du temps repose sur une interaction progressive et intuitive avec l'utilisateur. Elle permet de saisir des données nécessaires à la planification des cours en suivant une série d'étapes structurées.

L'interface se compose des **Popupmenu,** des **Boutons,** deux **Tableaux** 6X6 (jours / heures), et une **Listbox** comme indiqué ci-dessous :

4.3.2 Etapes de construction de l'emploi du temps

4.3.2.1 Saisie des informations via des menus déroulants (Popupmenu) :

L'utilisateur sélectionne successivement les différentes composantes d'un créneau d'enseignant à travers des menus déroulants (cet emploi du temps concerne un seul niveau).

Ces éléments comprennent le nom de l'enseignant ; la matière enseignée, la salle attribuée, le jour ainsi que horaire souhaité

Enseignants = {'Prof A' , 'Prof B' , 'Prof C' , 'Prof D' }

Matières = {'Math', 'Informatique', 'Option', 'Recherche opérationnel', 'GL'}

Salles = {'S1','S2','S3'}

Jours = {'Samedi' , 'Dimanche' , 'Lundi' , 'Mardi' , 'Mercredi' , 'Jeudi'}

Heures = {'08h-09h30','9h30-11h','11h-12h30','12h30-14h00','14h00-15h30','15h30-17h00'}

4.3.2.2 Ajout des données dans une liste récapitulative (Listebox)

Une fois tous les champs renseignés, l'utilisateur valide son entrée en cliquant sur le bouton **"Ajouter".**

Cette méthode de saisie guidée permet de limiter les erreurs de saisie et garantir la cohérence des collectées.

L'information est alors enregistrée et affichées sous forme de ligne dans une " **List box"**

Cela permet à l'utilisateur de suivre en temps réel les combinaisons déjà saisies.

Exemple : **Ens A – Math – Salle1 –Samedi – 8h00-9h30**

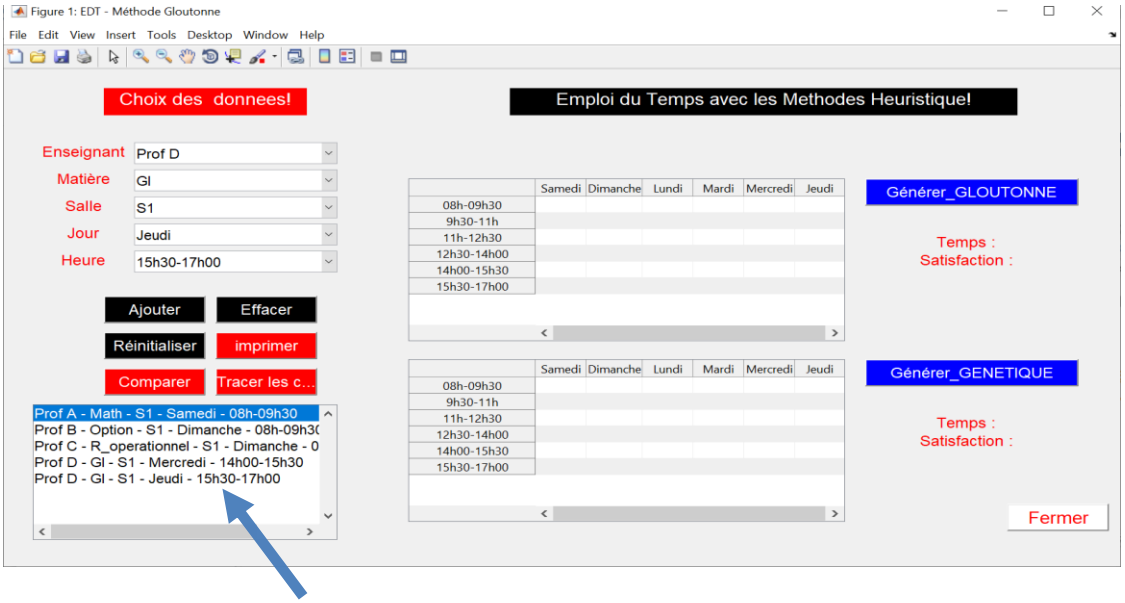


Figure 4 -8 : Interface graphique avec des données dans liste box

Avant de lancer la génération de l'emploi du temps l'utilisateur a la possibilité de consulter, voire, de supprimer ou modifier certaines entrée affichées dans " **Liste box**" et peut effacer une ligne à l'aide du bouton Effacer.

4.3.2.3 Lancement de la génération d'un emploi du temps

Lorsque toutes les données sont saisies, l'utilisateur déclenche la génération automatique de l'emploi du temps en cliquant sur le bouton Générer (On a prévu deux **Boutons** ("**Générer _GE**" ou "**Générer _GL**").

Le système traite alors les informations enregistrées en tenant compte de différentes contraintes imposées.

Exemple de contraintes utilises

- Chaque enseignant peut assurer 3 cours au maximum par jour.
- 5 séances par semaine
- Absence de conflits de salle ou d'horaires.
- Jeudi pas de cours à partir de 14heure

La génération se fait selon le choix de l'utilisateur entre la méthode gloutonne ou génétique

4.3.2.4 Affichage du résultat final

Une fois le traitement effectué, l'emploi du temps généré est affiché dans deux tableaux structurés permettant une lecture claire et ordonnée de la répartition des cours pour les deux méthodes (génétique ou gloutonne).

Au final, l'utilisateur peut exploiter, enregistrer ou modifier les résultats selon ses besoins.

Après avoir obtenu les résultats, l'algorithme utilisé calcule le temps d'exécution d'une méthode de la génération et aussi le taux de satisfaction des contraintes est affiché dans l'interface exemple (**Figure 4.9**).

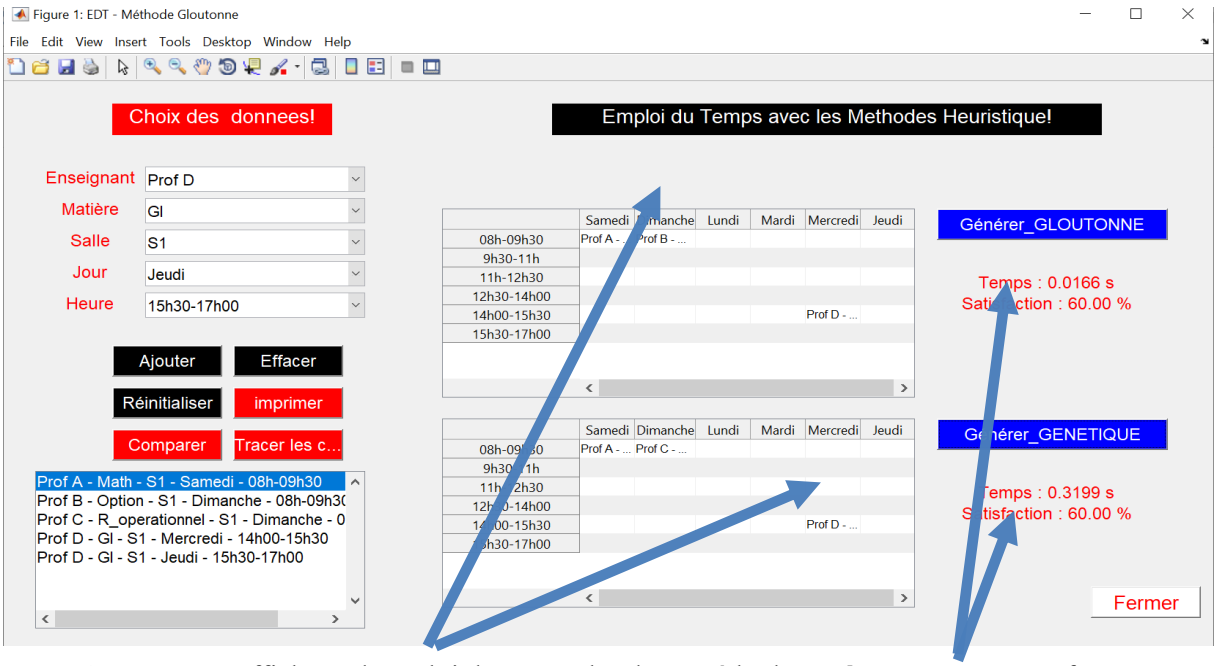


Figure 4 -9 : Affichage d'emploi du temps des deux méthodes et le temps avec satisfactions

Ainsi, en cas de conflits ou d'erreurs, un message est émis par le système.

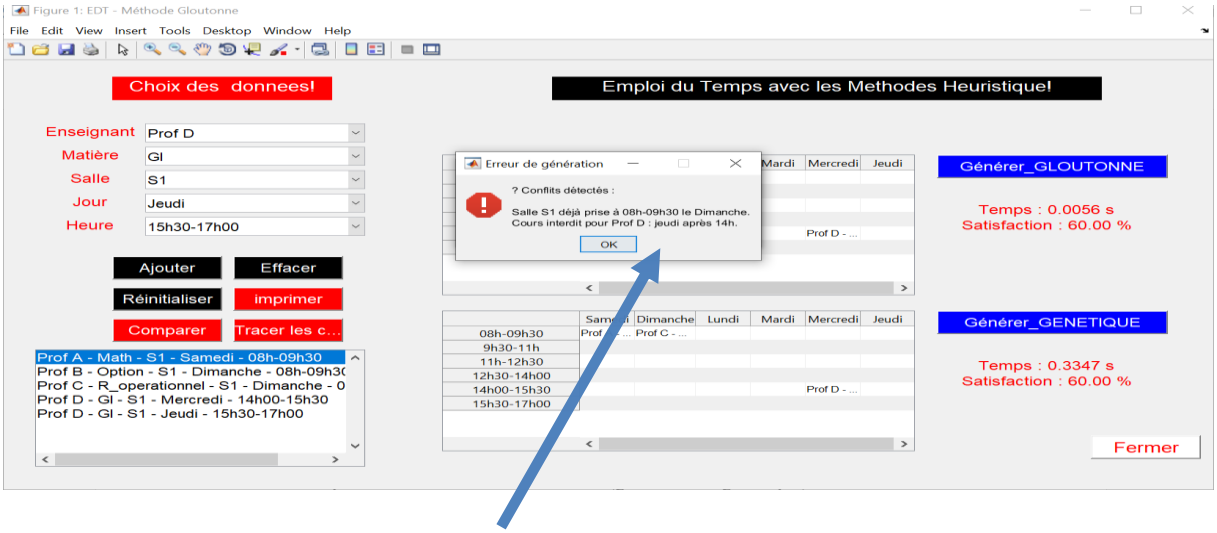


Figure 4 -10 : Message d'erreur pour la méthode de Gloutonne

4.3.3 Statistiques

Dans l'interface graphique, nous avons prévu un bouton **"Comparer",** et un autre **"Tracer les courbes".**

La comparaison entre les deux méthodes (**gloutonne et génétique**) se basant sur les deux critères suivants :

A. Le temps de génération (en secondes)
B. Le taux de satisfaction des contraintes (en pourcentage)

Le système

- vérifie que les deux méthodes ont bien été exécutées.
- crée une nouvelle fenêtre (figure) avec deux graphiques côte à côte :

Il affiche aussi les résultats dans la console MATLAB avec **fprintf.**

- Affichage des graphes de comparaison (taux & temps).
- Affichage d'un message automatique indiquant la méthode la plus performante.
- Sauvegarde des résultats dans un fichier texte comparaison_resultats.txt dans le dossier courant.
- Affichage de l'algorithme le plus rapide (Figure 4-11).

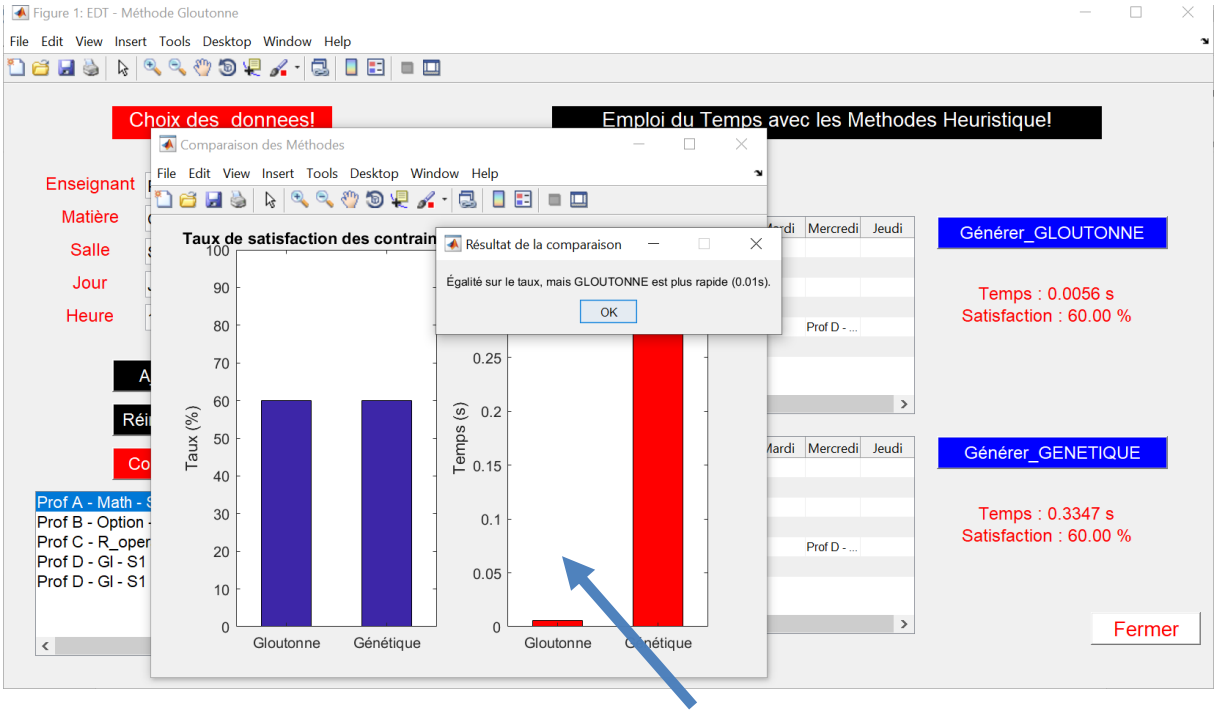


Figure 4 -11 : Figure de comparaison

- Le bouton ‘Tracer les courbes’ a pour objectif de générer l’emploi du temps à plusieurs reprises.

5. Fonctionnement des algorithmes

5.1 Méthode de Gloutonne

Initialisation :

- **edt** : matrice vide représentant l’emploi du temps (heures × jours).
- **Conflits** : liste des messages d’erreur ou conflits détectés.
- **Enseignants List** : liste unique des enseignants.
- **Cours par ENS jour** : matrice de comptage du nombre de cours par enseignant et par jour.

Boucle principale : pour chaque cours :

- Extraction des données du cours : enseignant, matière, salle, jour, heure.
- Conversion du jour et de l’heure en indices numériques.

Vérification des contraintes (dans l’ordre) :

- ❌ Pas de cours jeudi après 14h (heures après la 4e indexée).
- ❌ Max 3 cours par jour pour un enseignant.
- ❌ Pas de double réservation :
  - Si un enseignant est déjà pris à ce créneau.
  - Si la salle est déjà occupée à ce créneau.

2. Affectation du cours :
   - Si aucune contrainte n'est violée, on ajoute le cours dans la cellule **edt{h,j}.**
   - Mise à jour du compteur **cours par ENS jour**.
3. Résultats finaux :
   - Calcul du temps d'exécution (temps glouton).
   - Calcul du taux de satisfaction : proportion des cours sans conflit.
   - Affichage de l'emploi du temps dans la table (t1.Data).
   - Message :
     - ✅ Si tout est correct : boîte de dialogue de succès.
     - ❌ S'il y a des conflits : boîte de dialogue d'erreurs avec détails.

5.2 Fonctionnement de l'algorithme génétique:

5 .2.1 Initialisation

- nb individus = 50 : nombre de solutions (emplois du temps) dans chaque génération.
- nb générations = 100 : nombre d'itérations pour faire évoluer la population.
- meilleurs fitness : meilleur taux de satisfaction trouvé jusqu'à présent.
- Meilleur edt : meilleure solution (emploi du temps) sauvegardée.

5.2.2 Création de la population initiale

- Chaque **individu** (solution) est une **permutation aléatoire** des cours existants.

5.2.3 Évaluation (Fitness)

Pour chaque individu (emploi du temps) :

- Création d'un **emploi du temps temporaire** (edt temps).
- Application des **contraintes** :
  - ❌ Pas de cours **jeudi après 14h**.
  - ❌ Pas plus de **3 cours par jour par enseignant**.

- ❌ Pas de **double réservation** : même enseignant ou même salle à la même heure.

- Chaque contrainte non respectée ajoute une pénalité.
- Le **taux de satisfaction** (fitness) est calculé comme :

5.2.4 Sélection

- L'individu avec la **meilleure fitness** est sauvegardé

5.2.5 Croisement

- Les individus sont appariés deux à deux aléatoirement.
- Un **point de croisement** est choisi pour échanger une partie de leur contenu (comme un échange de gènes).
- Cela génère de nouveaux individus (enfants) à partir des parents.

5.2.6 Mutation

- Avec **10% de probabilité**, on échange aléatoirement deux cours dans un individu, pour favoriser la **diversité génétique**.

5.2.7 Répétition

- Le processus (évaluation + croisement + mutation) se répète pendant `nb=50 générations` fois.

5.2.8 Résultats finaux

- L'emploi du temps avec la **meilleure satisfaction** (meilleur edt) est affiché.
- Le **temps d'exécution** et le **taux de satisfaction** sont calculés et affichés.
- Un message est affiché :
  - ✅ Si toutes les contraintes sont satisfaites.
  - ❌ Sinon, une boîte d'erreur indique des conflits restants.

6. Analyse entre les deux méthodes (Heuristique)

6.1. 1ers tests enregistrement sans conflit

Nb enreg	2 enregistrements		6 4enregistrements		10 enregistrements	
Méthodes	Temps (s)	Taux %	Temps (s)	Taux %	Temps (s)	Taux %
Gloutonne	0.0008	100	0.0036	100	0.0332	100
Génétique	0.1246	100	2098	100	0.7379	100
Résultat	Gloutonne mieu (temps)		Génétique mieux en fonction de taux de satisfaction des contraintes			
			Gloutonne mieux en fonction de temps d'exécution			

Tableau 4-1 : Test sans conflit

Les résultats montrent, quel que soit le volume de données, la méthode gloutonne est systématiquement plus rapide que la méthode génétique. De plus, comme il n'avait aucun conflit dans les emplois du temps générer, les deux méthodes ont atteint un taux de satisfaction des contraintes de 100%.

6.2. 2eme tests enregistrement avec un conflit

Nb enreg	2 enregistrements		6 4enregistrements		10 enregistrements	
Méthodes	Temps (s)	Taux %	Temps (s)	Taux %	Temps (s)	Taux %
Gloutonne	0.0032	50	0.0056	83.33	0.0083	90
Génétique	0.1137	50	0.3132	100	0.4363	100
Résultat	Gloutonne mieu (temps)		Génétique mieux en fonction de taux de satisfaction des contraintes			
			Gloutonne mieux en fonction de temps d'exécution			

Tableau 4-2 : Test Avec un seul conflit

Dans le deuxième test, un conflit est apparu pour deux données.

La méthode gloutonne est restée plus rapide que la méthode génétique, avec un taux de satisfaction équivalent. Cependant, à mesure que le nombre de données augmente (notamment de 6 et 10 enregistrements), le temps d'exécution de la méthode gloutonne augmente également, bien qu'il reste inférieur à celui de la méthode génétique. En revanche, côté efficacité, la méthode génétique maintient un taux de satisfaction des contraintes de 100 %, tandis que celui de la méthode gloutonne diminue légèrement.

6.3. 3eme Test : avec 30 enregistrements et avec des conflits

Lors du test avec 30 enregistrements comportant des cas de chevauchement, la méthode gloutonne a affiché un temps d'exécution très court (0,0050 s), mais avec un taux de satisfaction des contraintes seulement 56,67 % et des messages de conflits détectés. En revanche, la méthode génétique, bien que plus lente (1,14 s), a atteint un taux de satisfaction plus élevé (66,67 %) sans générer de conflits.

Nb enreg	30 enregistrements	
Métohdes	Temps (s)	Taux %
Gloutonne	0.0050	56.67
Génétique	1.1432	66.67
Résultat	Gloutonne mieu (temps)	
	Genetique mieu (contrainte)	

Tableau 4-3 : Test Avec plusieurs conflits

6.4 Comparaison graphique entre méthode gloutonne et méthode génétique

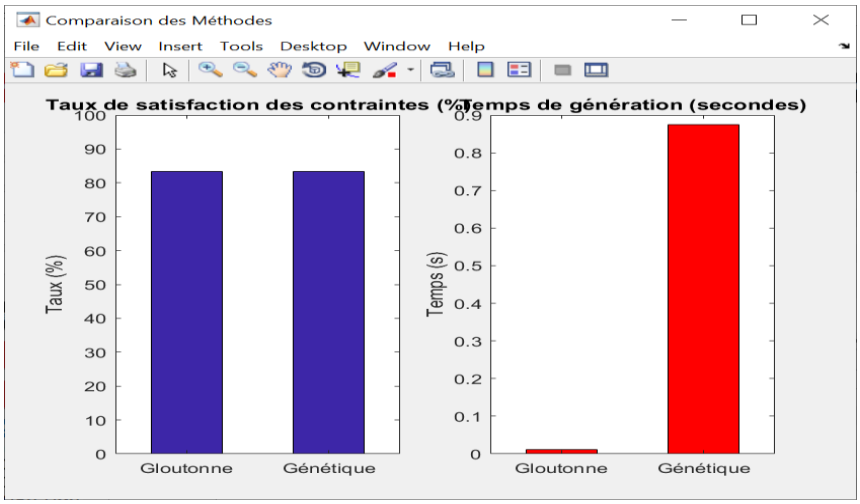


Figure 4-12 : Comparaison graphique entre les deux méthodes

Un graphique sous forme de barres rectangulaires a été réalisé pour comparer les performances de la méthode gloutonne et de la méthode génétique selon deux indicateurs : le temps de génération de l'emploi du temps et le taux de satisfaction des contraintes comme montre la Figure 4-12.

- Avec la couleur bleu : le taux de satisfaction des contraintes.
- Et avec la couleur rouge : le temps de génération.

Les résultats montrent clairement que la méthode gloutonne est plus rapide, avec un temps d'exécution très faible, même lorsque le nombre d'enregistrements augmente. Cependant, elle atteint parfois un taux de satisfaction plus faible, notamment en cas de conflits ou de contraintes complexes.

En revanche, la méthode génétique, bien que plus lente, parvient à générer des emplois du temps avec un meilleur taux de satisfaction, grâce à son processus évolutif basé sur la sélection et l'amélioration progressive des solutions.

Ce graphique met donc en évidence le compromis entre rapidité et qualité selon la méthode utilisée.

6.5 Analyse de la méthode Génétique en fonction du Nb Génération et Nb des Individus

Dans l'algorithme génétique, deux paramètres essentiels sont définis dès le départ :

- Le nombre d'individus, qui représente le nombre de solutions (emplois du temps) dans chaque génération.
- Le nombre de générations, qui correspond au nombre d'itérations pendant lesquelles l'algorithme améliore la population.

Nb Générations	**50**	**30**	**5**
Nb Individus	**30**	**10**	**10**
Temps execution	**0.69**	**0.153**	**0.37**
Taux de satisfaction	**60%**	**86.67%**	**80%**

Tableau 4.4 : Test Avec différents nb de génération et nb individus

À chaque génération, l'algorithme conserve une population fixe (par exemple, 30 individus), qu'il fait évoluer par des opérations de sélection, croisement et mutation. Ce processus est répété pendant un certain nombre de générations (ex. : 50), permettant une amélioration progressive des solutions. Ainsi, au total, l'algorithme explore un grand nombre de combinaisons tout en respectant les contraintes.

Dans le **Tableau 4-4** dans le premier cas en a 30 emploi du temps sont conservés, croisées et mutées pour former une nouvelle génération. Ou en trouvent des emplois du temps éliminé et d'autre accepté pour rester dans le même nombre pondant 50 itérations

Les tests réalisés avec 15 enregistrements montrent que le temps d'exécution diminue quand on réduit le nombre d'individus ou de générations.

Cependant, le taux de satisfaction des contraintes n'évolue pas de manière linéaire : dans certains cas, un petit nombre d'individus peut suffire à générer de bonnes solutions. Le meilleur taux (86,67 %) est obtenu avec seulement 10 individus et 39 générations, pour un temps de calcul très faible (0,15 s).

Cela indique que le réglage fin des paramètres (nombre d'individus et de générations) permet d'optimiser à la fois la qualité de l'emploi du temps et le temps de génération.

7. Discutions des résultats :

Au cours des différents tests réalisés avec des volumes de données croissants (de 2 à 30 enregistrements ou données) et en introduisant progressivement des conflits (chevauchements), il a été observé que :

✓ La méthode gloutonne se distingue par sa rapidité d'exécution, même avec un grand nombre de données. Cependant, sa stratégie d'affectation immédiate sans retour en arrière la rend moins efficace face à des cas complexes avec contraintes multiples. Elle génère souvent des conflits (salles ou enseignants doublés) et son taux de satisfaction diminue à mesure que les données deviennent plus conflictuelles.

✓ La méthode génétique, quant à elle, est plus lente car elle évalue plusieurs combinaisons de solutions, mais elle est plus performante pour respecter les contraintes complexes. Elle parvient à éviter les conflits dans la majorité des cas, avec un taux de satisfaction des contraintes stable et élevé (souvent 100 %), même avec des données nombreuses.

Cela s'explique par la nature des algorithmes : la méthode gloutonne applique une affectation immédiate sans retour arrière, ce qui peut entraîner des conflits lorsque les ressources sont limitées, tandis que la méthode génétique explore plusieurs solutions et optimise globalement, ce qui lui permet d'éviter les conflits au prix d'un temps de calcul plus long. La méthode gloutonne est non seulement suffisante mais aussi plus efficace en termes de temps d'exécution.

8. Validation

8.1 Comparaison le taux de satisfaction des contraintes entre les deux méthodes avec d'autres expériences

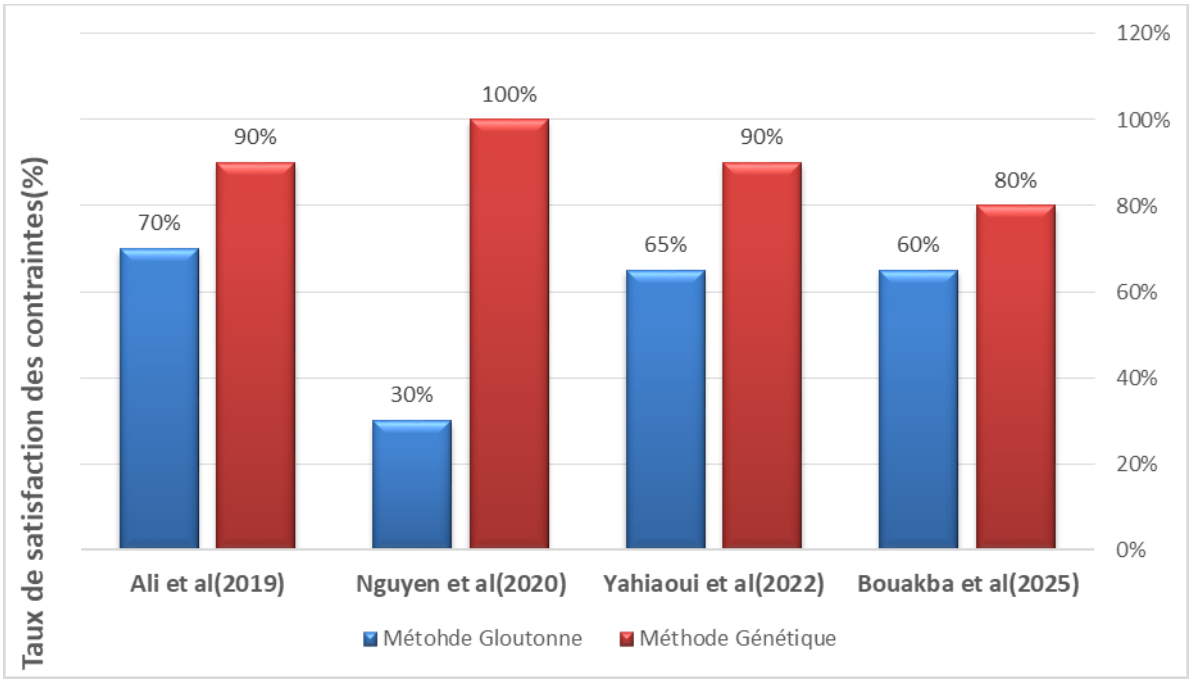


Figure 4-13 : comparaison entre les deux méthodes avec d'autres expériences

Histogramme dans la **Figure 4-13** est un comparatif des taux de satisfaction des contraintes pour chaque expérience, selon la méthode utilisée (Gloutonne et Génétique).

Il met clairement en évidence la supériorité de la méthode Génétique dans la plupart des cas, notamment des vastes des données complexes.

Les différentes expériences analysées montrent que la gestion des contraintes constitue un point de divergence majeur entre la méthode gloutonne et la méthode génétique. La méthode gloutonne, bien qu'efficace pour respecter des contraintes simples comme l'absence de chevauchement de cours ou la limitation du nombre de séances par jour, atteint rapidement ses limites dès que le problème devient plus complexe (préférences d'enseignants, contraintes temporelles strictes, ou optimisation globale). En revanche, la méthode génétique s'adapte mieux à des contraintes multiples et hiérarchisées grâce à sa capacité d'exploration de solutions.

8.2 Comparaison du temps d'exécutions entre les deux méthodes avec d'autres expériences

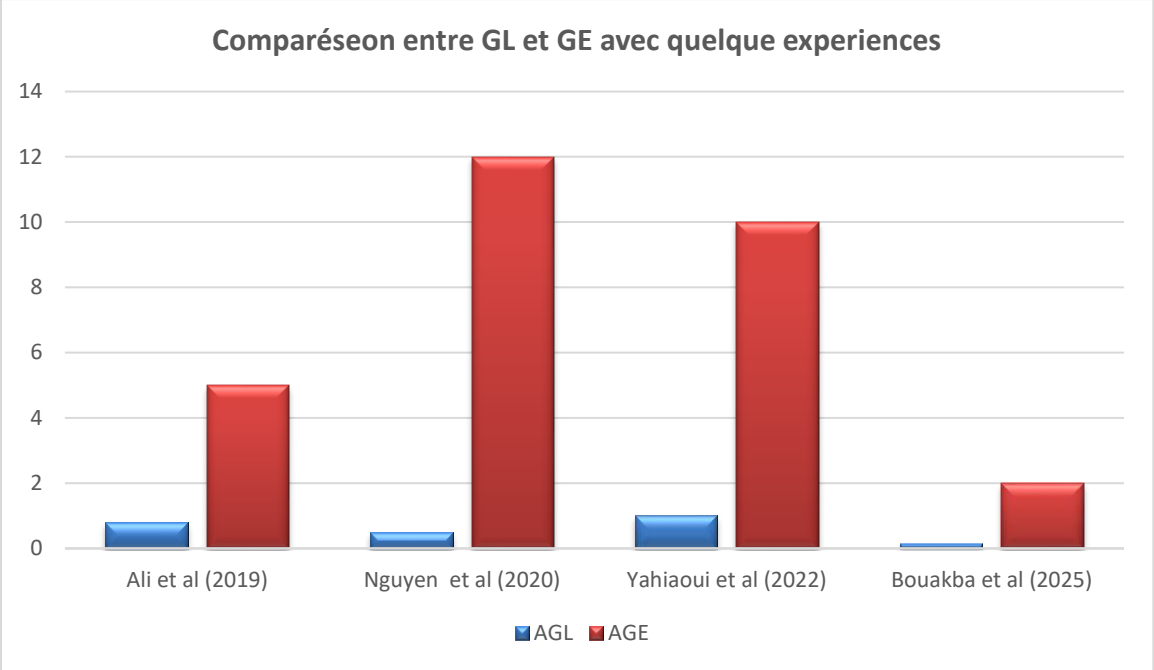


Figure 4.14 : Comparaison de GE et GL avec quelques expériences

Histogramme présente par la **Figure 4.14** compare les temps de génération des emplois du temps obtenue par les méthodes Gloutonne et Génétique dans différentes expériences tirées de la littérature ainsi que de mon propre application

On observe que la méthode Gloutonne présente systématiquement des temps d'exécution très courts allant de 0.5 à 1 Seconde, ce qui confirme son efficacité en termes de rapidité, surtout dans des contextes ou les contraintes sont simples ou peu nombreuses

En revanche, la méthode génétique affiche des temps plus élevés, variant de 5 à 15 secondes selon les études

Cette différence s'explique par la nature même d'algorithme génétique, qui explore un grand nombre de solutions potentielles avant de converger vers une solution optimale, ce qui rend couteux en temps de calcul

Ainsi, cet histogramme met en évidence un compromis clair entre la vitesse d'exécution de la méthode gloutonne et la qualité optimisée offerte par la méthode génétique.

9. Conclusion

Nous avons décrit dans ce dernier chapitre l'implémentation de notre solution et la production d'une solution logicielle écrite sous Matlab. Deux possibilités sont, ainsi, offertes aux utilisateurs : une méthode manuelle qui est efficace lorsque le nombre de contraintes n'est pas trop élevé et qui permet une grande liberté dans la gestion des plannings, la deuxième méthode est plus avantageuse ; elle automatise la gestion de l'emploi du temps et elle est très efficace pour un nombre élevé de contraintes. Dans cette deuxième méthode, l'utilisateur peut choisir entre une méthode qui utilise l'algorithme glouton ou une seconde utilisant les algorithmes génétiques. Le système assiste l'utilisateur dans le développement de l'emploi du temps tout en générant un ensemble de messages en cas de conflits ou d'erreurs. Des statistiques permettant d'assurer l'une ou l'autre des méthodes sont effectuées par le système.

La résolution automatique du problème complexe de génération d'emplois du temps universitaires nécessite des approches robustes capables de prendre en compte un grand nombre de contraintes pédagogiques, humaines et logistiques. Dans ce mémoire, nous avons exploré deux techniques d'optimisation contrastées, la méthode gloutonne et l'algorithme génétique afin d'évaluer leur efficacité respective dans un contexte réel.

La mise en œuvre concrète de ces algorithmes, au-delà de leur formulation théorique, a représenté un enjeu central de ce travail. En effet, la performance d'un algorithme ne se juge pas uniquement sur le papier, mais également à travers sa capacité à s'adapter à un environnement réel, à intégrer des préférences utilisateurs, à respecter des contraintes diverses et à produire des résultats utilisables en un temps raisonnable. La mise en œuvre sous MATLAB R2017a, avec une interface graphique conviviale, a permis non seulement d'évaluer la qualité des solutions générées, mais aussi de faciliter l'interaction avec des utilisateurs non spécialistes.

La méthode gloutonne, simple à implémenter, s'est montrée efficace pour des scénarios peu contraints, notamment grâce à sa rapidité d'exécution. Cependant, elle atteint rapidement ses limites lorsque le problème devient plus complexe. L'algorithme génétique, en revanche, bien qu'exigeant davantage de ressources computationnelles, a permis d'obtenir de meilleures solutions, notamment en termes de satisfaction des préférences et de réduction des conflits.

La mise en œuvre de ces deux méthodes a mis en lumière l'importance de la phase de modélisation, du choix des paramètres et de l'équilibre entre performance algorithmique et convivialité de l'outil. Ces aspects sont essentiels pour que les solutions d'optimisation soient effectivement adoptées dans des contextes opérationnels comme celui de la planification universitaire.

À travers cette étude, il apparaît que le recours aux algorithmes d'optimisation, et en particulier au méta heuristique, constitue une réponse efficace aux problèmes complexes de planification. Leur mise en œuvre soignée, adaptée aux contraintes du terrain, ouvre la voie à des outils intelligents, automatisés et flexibles, capables de faciliter les tâches traditionnellement lourdes et répétitives.

Perspective

Dans la littérature scientifique, on trouve des approches dites **"Hybrides"** qui combinent des algorithmes heuristiques avec d'autre méthodes, par exemple plusieurs article parlent de l'utilisation de approche **AGL** avec **AGE** pour arriver à une solution efficaces. Comment ça se passe ?

Les résultats génères par l'approche **AGL** ont permis d'obtenir des solutions initiales rapides, bien que parfois sous optimale, cette dernière est utiliser comme une solution initiale pour **AGE**.

Cette combinaison vise à bénéficier à la fois de la rapidité de la méthode **AGL** et de la capacité exploratoire de **AGE**, afin de tendre vers une solution plus optimale tout en respectant les contraintes définies.

La méthode proposée consiste à appliquer, dans un premier temps, **AGL** a une base de données préalablement définie. Ce traitement initial permet de détecter et d'éliminer les conflits ainsi que les contraintes non souhaitées. On peut ainsi considérer cette étape comme une forme d'analyse et de filtrage des données, visant à produire une solution initiale cohérente.

Cette solution, obtenue de manière constructive, est ensuite utilisée comme population initiale pour **AGE**, qui pourra l'exploiter à travers les mécanismes de mutation, de sélection et de remplacement, dans le but de converge vers une solution optimale.

En effets en fournissant à **AGE** une base déjà partiellement satisfaisante à la qualité des solutions explorées tout en évitant les configurations incohérentes dès les premières générations.

Cette approche hybride permet non seulement de réduire le temps nécessaire à la convergence vers une solution acceptable, mais aussi d'augmenter le taux de satisfaction des contraintes pédagogique et logistique.

Les résultats obtenus montrent que cette complémentarité entre méthode constructive **AGL** et méthode évolutive **AGE** constitue une solution pertinente pour traiter ce type de problème complexe, ou la recherche d'un équilibre entre faisabilité et optimalité est essentielle.

Bibliographie

[1] BIERLAIRE Michel, *Introduction à l'optimisation*, EPFL Press (Presses polytechniques et universitaires romandes Lausanne), 2009.

[2] Rezzoug A et al, Extended grey wolf optimization–based adaptive fast nonsingular terminal sliding mode control of a robotic manipulator, 2022.

[3] Brahimi Souha. Les Problèmes d'ordonnancement Multi-Objectifs dans un Système de Production, 2022

[4] https://elearning-facsci.univ-annaba.dz/mod/resource/view.php?id=751

[5] https://www.studysmarter.fr/resumes/mathematiques/mathematiqueappliquees/optimisation/

[6] https://fr.surveymonkey.com/mp/improve-customer-loyalty-retention/

[7] CIGREF (Club Informatique des Grandes Entreprises Françaises) « Généalogie de l'algorithme… ses ancêtres ! » septembre 2017.

[8] Bensaid, H. (s.d.). *Conception des algorithmes* [cours, diaporama PDF], Département RIM, Université X, en ligne sur Academia.edu, consulté le 11 juin 2025,

[9] Bétréma, J. Classes de complexité P et NP Chapitre 8 [En ligne, 2023].

[10] https://www.bibmath.net/dico/index.php?action=affiche&quoi=./c/complexite.

[12] Dupont, A. Introduction à l'optimisation et la classification des méthodes d'optimisation. Document pédagogique, 4 février 2022.

[13] Kaaniche, M. *Algorithmique et structures de données* [cours en ligne], INSA Toulouse, 2021.

[14] Rioual, E. Algorithme de **Dijkstra,** calculer le chemin le plus court entre deux sommets d'un graphe. [En ligne]. Études Tech, 27 mai 2023.

[15] BAHSINA, N. Réalisation de la carte paléogéographique d'une région. Cours Sciences de la Vie et de la Terre 1ère Bac, a été publiée le **19 septembre 2022**.

[16] Esparza, J., Kiefer, S. & Luttenberger, M. *Computing the Least Fixed Point of Positive Polynomial Systems*, **SIAM Journal on Computing**, vol. 39, n° 6, 2010.

[17] https://arxiv.org/abs/1001.0340,

[18] Kirkpatrick, S., Gelatt, C.D. & Vecchi, M.P. *Optimization by Simulated Annealing*, Science, vol. 220, n° 4598, 1983, p. 671–680. DOI : 10.1126/ a été publié dans la revue *Science* le **13 mai 1983**.

[20] https://fr.pendo.io/glossary/user-feedback/.

[19] Daniel Gelatt.C, Optimization by simulated annealing. Science. 1983 May 13; 220 (4598):671-80.

[20] Anonyme. *Méthodes d'Optimisation : différence entre Heuristique et Méta heuristique* [vidéo en Ligne], YouTube, 17 déc. 2020, consulté le 11 juin 2025. Disponible sur : https://www.youtube.com/watch?v=80K1kTJc2JU

[21] Sioud, A. Approches hybrides pour la résolution d'un problème d'ordonnancement industriel. Thèse de doctorat, Université du Québec à Chicoutimi, mai 2011.

[22]https://www.reddit.com/r/MCAT2/comments/gze1jd/lt%C3%AE%C3%BA%C5%A1_practice_exam_2_37_heuristic_vs_algorithm/?tl=fr

[23] Wack, B. Algorithmes gloutons – Coloration de graphe. Cours Algorithmique et Analyse d'Algorithmes, L3 Informatique, Université Grenoble Alpes, 2025.

[24] https://www.dukeupress.edu/an-other

[25] Dahmani, K., Notton, G. Dizène, R., & Paoli, C. État de l'art sur les réseaux de neurones artificiels appliqués à l'estimation du rayonnement solaire. Revue des Énergies Renouvelables, 15(4), 687–702.

[26] Goldberg, E. Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.

[27] https://u.demog.berkeley.edu/~jrw/Biblio/Eprints/DFLE/Switzerland_Wanner.pdf

[28] Reeves, C. R. Modern Heuristic Techniques for Combinatorial Problems. Blackwell, 1993.

[29] Goldberg, D. E. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, 1989.