République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohammed Seddik Ben Yahia de Jijel

Faculté des Sciences Exactes et Informatique

Département d'Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme de master

En : Informatique

Option : *Systèmes d'Information et **A**ide à la **D**écision (**SIAD**)*

Thème

Métaheuristique à base de populations appliquée à la génération d'un emploi du temps d'un département à l'université.

Présenté par :

BOUKELMOUNE Houssam

Encadré par :

Dr. KARA Messaoud

Promotion 2021

Résumé

Le problème de l'emploi du temps à l'universitéest un problème NP difficile. Il fait partie des problèmes d'ordonnancement.

Le processus de planification pour chaque semestre doit être effectué fréquemment, ce qui est une tâche difficile et longue.

Le problème de l'emploi du temps à l'université est complexe, car il prévoit la programmation de (cours, travaux dirigés et travaux pratiques) entre professeurs et étudiants dans des salles de diverses natures, amphithéâtres, salles de travaux dirigés et salles de travaux pratiques pendant une période de temps déterminée. Et cela, en prenant compte plusieurs contraintes spécifiques aux professeurs et aux étudiants appelées contraintes dure et contraintes souples, où toutes les contraintes dures doivent être atteintes tout en respectant le plus grand nombre de contraintes souples pour assurer une solution acceptable.

Dans ce travail, nous nous sommes intéressés à la résolution automatique du problème de l'emploi du temps en utilisant une métaheuristique à base de population.Pour cela, nous avons développé une application basée sur les algorithmes génétiques.

Mots clés : *Le problème de l'emploi du temps, Timetabling, University Course Timetabling, Genetic Algorithms.*

Abstract

The university timetable problem is a difficult NP problem. It is a part of the scheduling problems.

The planning process for each semester has to be done frequently, which is a difficult and time consuming task.

The problem of timetabling at the university is complex, because it provides for programming different kind of courses (courses, directed works and practical works) between professors and students in rooms of various kinds: amphitheatres, classrooms and practical work rooms within fixed periods of time. And this, taking into account several specific constraints to teachers and students called hard constraints and soft constraints, where all the hard constraints

must be met while achieving the greatest number of soft constraints to ensure an acceptable solution.

In this work, we are interested in the automatic resolution of the timetabling problem using a population-based metaheuristic, for this reason we have developed an application based on genetic algorithms.

***keywords**: The Timetable Problem, Timetabling, scheduling problem,University Course Timetabling, Genetic Algorithms.*

ملخص

مشكلةإنجاز الجداول الزمنية (البرامج الأسبوعية) في الجامعة هي مشكلةكثيرة الحلول ولا يمكن الوصول لحل محدد لها. وهي جزء من مشاكل الجدولة، إذ يجب أن تتم عملية التخطيط لكل فصل دراسي بشكل متكرر، وهي مهمة صعبة وتستغرق وقتًا طويلاً.

مشكلة جدولة الوقت في الجامعة معقدة لأنها تتطلّب برمجة (الدروس، الأعمال الموجهة والأعمال التطبيقية) بين الأساتذة والطلاب في قاعات مختلفةالأنواع (المدرجات، قاعات الأعمال الموجعة ومخابر والأعمال التطبيقية) و في فترات زمنية محددة. وهذا، مع الأخذ بعين الاعتبار العديد من القيود الخاصة بالمدرسين والطلاب والتي تسمى القيود الصعبة والقيود المرنة، حيث يجب تلبية جميع القيود الصعبة مع تحقيق أكبر عدد من القيود المرنة لضمان حل مقبول.

في هذا العمل ، نحن مهتمون بالحل التلقائي لمشكلة جدولة الوقت باستخدامطرق خوارزميات ارشاديةمرتكزة على السكان ، ولهذا السبب قمنا بتطوير تطبيق يعتمد على الخوارزميات الجينية.

***كلمات مفتاحية**:مشكلة الجداول الزمنية ، الجدول الزمني، الجدول الزمني للدرسة في الجامعة، الخوارزميات الجينية.*

Remerciements

Ce travail n'aurait jamais été mené à terme sans le soutien de DIEU le tout puissant qui m'a donné la force et la volonté pour le réaliser.

J'exprime mes remerciements les plus sincères à ***Dr. KARA Messaoud*** *d'avoir assuré la direction de ce travail, de m'avoir encadré et de m'avoir guidé.*

Je tiens à remercier les membres du jury Mr. ***LATER Azzedine*** *et Mr.* ***ALIOUCHE Abdelaziz*** *d'avoir accepté d'examiner ce mémoire.*

J'adresse ma gratitude à tous ceux qui ont contribué, de près ou de loin à réaliser ce mémoire par leur présence et leurs conseils.

Table des matières

Table des figures

Liste des tableaux

Liste desabréviations

ACO	Ant Colony Optimisation
EA	Evolutionary Algorithms
EPDT	EmPloi Du Temps (Nom de l'application développée)
GA	Genetic Algorithm
GC	Graph Coloring
GSGA	Guided Search Genetic Algorithm
IP	Integer Programming
LP	Linear Programming
LS	Search Local algorithm
MA	Memetic Algorithm
RIICN	Randomized Iterative Improvement with a Composite Neighboring algorithm
SA	Simulated Annealing
TS	Tabu Search algorithm
UCTTP	University Course TimeTabling Problem
VNS	Variable Neighborhood Search algorithm

Introduction générale

Les responsables de nombreux domaines de la vie professionnelle sont confrontés au problème de la planification du temps de travail. Dans les hôpitaux, les horaires de travail doivent être organisés entre médecins et infirmiers afin d'assurer un meilleur respect des règles de travail à l'hôpital. Dans les entreprises, des sources spécifiques telles que le nombre de véhicules et d'employés avec d'autres contraintes telles que le timing lorsqu'il y a une condition pour terminer une tâche à un moment précis. Donc, la planification est très importante afin de réduire les coûts de production, d'améliorer la qualité de service pour les clients et d'assurer une bonne qualité de vie pour les employés.

La planification du temps est un ensemble sur mesure de processus, d'outils, de techniques et de méthodes. La planification du temps est généralement une nécessité dans tout projet de développement car elle détermine le temps et la portée du projet, de façon à satisfaire au mieux un ensemble d´objectifs tels que l´amélioration de la qualité de service et l´amélioration des conditions de travail.

Parmi les problèmes de planification présentés, on retrouve le problème de planification de l'emploi du temps dans les établissements éducatifs, notamment les universités, où il consomme beaucoup de ressources humaines et financières, où la mauvaise gestion du temps se traduit sur l'acquisition scientifique des étudiants.

Le problème de l'emploi du temps est un problème difficile et le résoudre à la main demande beaucoup de temps, d'efforts et de personnes. En cas de problème dans la plannification, il est difficile de modifier des données.

Face à ces difficultés, des recherches ont été menées pour contribuer à l'amélioration de la qualité de l'ordonnancement. Où l'idée d'utiliser l'ordinateur est apparue pour faciliter cette tâche, grâce à des outils basés sur de puissants algorithmes d'optimisation à base de population.

Parmi ces algorithmes, on trouve les algorithmes génétiques, qui sont l'une des solutions aux problèmes de lemploi du temps, où ils partent de la population initiale et tentent de l'améliorer en suivant les étapes de l'algorithme.

Dans ce mémoire, nous nous sommes intérrésés à la résolution du problème de l'emploi du temps à laide des algorihtmes génétiques.

Organisation du mémoire

Ce mémoire est organisé en 4 chapitres présentés comme suit :

- ✓ ***Chapitre 1 : Etat de l'art.***

  Dans ce chapitre, nous présentons d'abord les méthodes utilisées pour résoudre le problème de l'emploi du temps.

  Ensuite, nous présentons les méthodes métaheuristiques à base de populations utilisées pour résoudre le probléme de l'emploi du temps.

- ✓ ***Chapitre 2 :Les algorithmes génétiques pour la résolutiondu problème de l'emploi du temps***

  Dans ce chapitre, nous nous sommes intéressés aux algorithmes génétiques proposés pour résoude le problème de l'emploi du temps. Dans la première partie, nous décrivons les principes des algorithmes génétiques. Tandisque, dans la seconde partie, nous présentons quelques algorithmes génétiques utilisés pour résoudre le problème de l'emploi du temps et nous choisissons l'un d'eux.

- ✓ ***Chapitre 3 : Description de l'algorithme choisi***

  Dans ce chapitre, nous commençons d'abord par une présentation de notre probleme. Ensuite, nous présentons l'algorithme génétique sélectionné pour l'implémentation.

- ✓ ***Chapitre 4 : Implémentation et résultats***

  Dans ce chapitre, nous allons tout d'abord présenter l'application EPDT(EmPloi Du Temps) que nous avons développé et qui implémente l'algorithme génétique. Ensuite,nous faisons une expérimentation de notre application et nous discutons les résultats obtenus.

- ✓ Le mémoire se termine par une conclusion générale.

Chapitre 1 : Etat de l'art

1.1 Introduction

La confection d'un emploi du temps dans les établissements scolaires est un travail très important, difficile à réaliser ; c'est typiquement un problème de résolution de contraintes, NP-complet, dont la solution n'est pas, a priori connue dans le cas général (1).

Pour fournir une solution, il est nécessaire d'être capable de s'adapter aux changements dynamiques de l'environnement en tenant compte de la diversité des contraintes et l'interdépendance des programmes d'enseignement, la multitude des matières étudiées et les contraintes sur ces matières (cours, TD, TP, …), la durée des cours, les contraintes de disponibilité des enseignants, la disponibilité limitée des salles (2).

C'est un problème qui peut être défini comme un problème qui fait assigner quelques évènements dans un nombre limité de périodes. Il peut être divisé en deux catégories principales : la confection d'horaires des cours et la confection d'horaires des examens.

Ces problèmes sont soumis à beaucoup de contraintes qui sont d'habitude divisées en deux catégories : « ***les contraintes dures*** » et « ***les contraintes souples*** » (3).

La confection de plannings d'horaires est donc une tâche très difficile et sa solution manuelle peut exiger beaucoup d'effort ce qui a attiré énormément l'attention de la communauté scientifique. Dans la mesure où notre travail se rapporte au problème de gestion d'emploi du temps au sein de l'université, il semble naturel de faire une synthèse des différentes études retrouvées dans la littérature et afférentes à ce thème.

1.2 Les méthodes utilisées pour résoudre le problème de l'emploi du temps

Un grand nombre de méthodes diverses ont été proposées dans la littérature pour résoudre les problèmes d'horaires.

Ces méthodes proviennent d'un certain nombre de disciplines scientifiques telles que la recherche opérationnelle, l'intelligence artificielle et l'intelligence computationnelle. Elles peuvent être divisé en plusieurs catégories :

1) Méthodes de Recherches opérationnelles : Ce sont des techniques basées sur la théorie de de la programmation entière / linéaire (IP / LP), la méthode de coloration de graphes (GC), et la programmation de satisfaction de contraintes (CSP) (4).

2) Méthodes de clusters, dans les quelles le problème est divisé en un certain nombre d'ensembles d'événements.Chaque ensemble est défini de manière à satisfaire toutes les contraintes dures. Ensuite, les ensembles sont affectés à des intervalles de temps réels pour satisfaire également les contraintes souples (5).
3) Méthodes méta-heuristiques, telles que les algorithmes génétiques (AG), le recuit simulé (SA) (6), la recherche tabou (TS) et d'autres approches heuristiques (7), qui sont principalement inspirées de la nature, et appliquent des processus de type nature à des solutions ou des populations de solutions, afin de les faire évoluer vers l'optimalité. Les AG sont utilisées pour résoudre les problèmes d'horaire depuis 1990 (3).

1.2.1 Les méthodes de recherche opérationnelle

1.2.1.1 Définition

La recherche opérationnelle peut être définie comme l'ensemble des méthodes et techniques rationnelles orientées vers la recherche du meilleur choix dans la façon d'opérer en vue d'aboutir au résultat visé ou au meilleur résultat possible. Elle fait partie des « aides à la décision » dans la mesure où elle propose des modèles conceptuels en vue d'analyser et de maitriser des situations complexes pour permettre aux décideurs de comprendre, d'évaluer les enjeux et d'arbitrer ou de faire les choix les plus efficaces. Ce domaine fait largement appel au raisonnement mathématique (logique, probabilités, analyse des données) et à la modélisation des processus. Il est fortement lié à l'ingénierie des systèmes, ainsi qu'au management du système d'information (8).

1.2.1.2 La théorie de la coloration de graphes

1.2.1.2.1 Définition

En théorie des graphes, la coloration de graphe consiste à attribuer une couleur à chacun de ses sommets de manière que deux sommets reliés par une arête soient de couleur différente. On cherche souvent à utiliser le nombre minimal de couleurs, appelé nombre chromatique. La coloration fractionnaire consiste à chercher non plus une mais plusieurs couleurs par sommet et en associant des coûts à chacune. Le champ d'applications de la coloration de graphe couvre notamment le problème de l'attribution de fréquences dans les télécommunications, la conception de puces électroniques ou l'allocation de registres en compilation (9).

1.2.1.2.2 La coloration de graphe et l'emploi du temps

La première définition du problème de l'emploi du temps a été introduite par Gotlib (1963) comme trois ensembles d'enseignants, de salles de cours et de plages horaires (10). Le premier

problème d'horaire a été résolu en utilisant une méthode de coloration de graphes (11). Cependant, cette méthode présentée n'a pas été en mesure de résoudre les problèmes lorsqu'il y avait des sessions pré-attribuées. De Werra (1985) avait décrit la méthode de coloration des graphes pour modéliser un problème d'horaire en utilisant un graphe non directionnel (12). Ici, le but est de colorer le graphe en utilisant un nombre donné de couleurs où aucun sommet (noeud) adjacent n'a la même couleur. Le calendrier résultant ne devrait pas avoir de conflit et il devrait être coloré avec le moins de couleurs possible. Dans le problème d'horaire basé sur la théorie de la coloration des graphes, les événements, les contraintes (de préférence les contraintes dures) et les intervalles de temps sont considérés comme des noeuds, des arêtes et des couleurs, respectivement. Les noeuds, les arêtes et les couleurs dans cette théorie sont représentés sur la figure 2. Le nombre chromatique est le plus petit nombre de couleurs nécessaire pour colorer les noeuds et les arêtes du graphique. Les noeuds colorés sont équivalents au nombre de plages horaires. Ainsi, nous pouvons programmer les événements dans un créneau horaire en une journée en utilisant cette coloration; en d'autres termes, si deux noeuds adjacents ont la même couleur, un conflit de temps se produira. Toutefois, Selim (1988) a introduit l'idée de sommets de graphes séparés pour réduire le nombre chromatique de graphes et l'a appliquée à des étudiants séparés (13). Ici, la séparation d'un sommet est similaire à la séparation des étudiants dans un cours. La méthode decoloration des sommets d'un graphe en deux parties a également été menée par Hafizah et Zaidah (2010) afin de réduire le nombre de pénalités et de créer des emplois de temps de haute qualité par rapport aux emplois de temps manuels (14). L'ordonnancement des salles de classe a également été réalisé en utilisant la méthode de coloration des graphes de Dandashi et Al-Mouhamed (2010) où les sommets et les arêtes représentent respectivement les cours communs et les étudiants (15), et le but est de présenter une approche heuristique afin de :

1) Promouvoir la répartition uniforme des parcours sur les couleurs.
2) Équilibrer le nombre de parcours pour chaque tranche horaire par rapport aux salles existantes.

Une autre approche hybride pour résoudre le problème UCTTP utilisant la coloration génétique a été proposée par Asham, et Ramadan (2011) où cette méthode réduitle coût de recherche du nombre minimum de couleurs requises pour colorer un graphe (16).

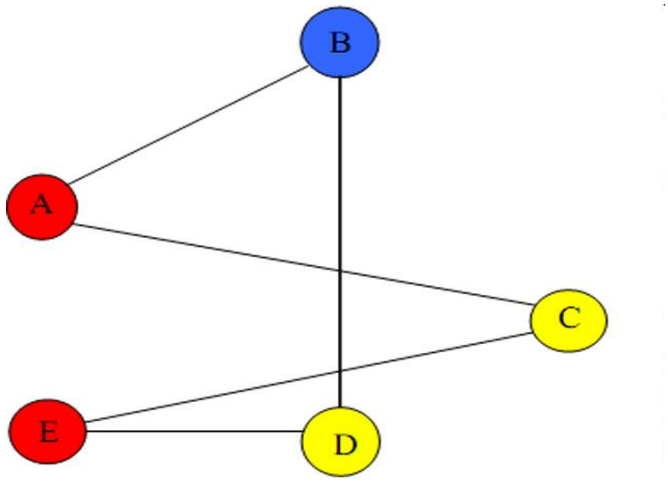

Figure 1 : Représentation de problème de l'emploi du temps par la méthode de coloration de graphe

(Événement) cours : A, B, C, D, E.

Arêtes (contraintes) : { (A,B),(A,C),(B,D),(C,E),(D,E)}.

Couleurs (plages horaires) : A, E : rouge. B :bleu. D, C : jaune.

1.2.1.3 La méthode de programmation linéaire

1.2.1.3.1 Définition

Selon William J. BAUMAUL, la programmation linéaire est une technique mathématique d'optimisation (maximisation ou minimisation) de fonction à objectif linéaire sous des contraintes ayant la forme d'inéquations linéaires. Elle vise à sélectionner parmi différentes actions celle qui atteindra le plus probablement l'objectif visé.

Robert DORFMAN et Paul Samuelson, ajoutent que la programmation linéaire est une méthode de détermination du meilleur plan d'action pour réaliser des objectifs donnés dans une situation où les ressources sont limitées. C'est donc une méthode de résolution du problème économique, soit dans le cadre d'une économie globale, soit dans celui du secteur public, soit dans une entreprise particulière.

1.2.1.3.2 Programmation linéaire/entière (IP/LP) et emploi de temps

Feiring, avait définit la programmation linéaire (LP) comme un sous-ensemble de la programmation mathématique où la solution est trouvée par l'attribution efficace de ressources limitées aux activités spécifiées afin de maximiser l'intérêt et de minimiser le coût. Bunday, avait également définit la programmation entière (IP), ici tout ou partie des variables pouvaient être définies comme des valeurs entières non négatives (2).

La méthode IP / LP est une méthode mathématique complète et a été appliquée en fonction de la structure et du type de faculté. Exploitant la méthode IP / LP entretiens avec le directeur et le doyen de la faculté. Les solutions du problème UCTTP utilisant cette méthode sont différentes pour les instituts d'enseignement; la taille de l'institut a une grande influence sur la résolution de ce problème de sorte qu'il est difficile d'obtenir les solutions optimales à la fin de l'exécution d'une méthode IP / LP pour un problème UCTTP. Cette méthode est principalement utilisée séparément sans combinaison avec d'autres approches; Cependant, nous pourrions utiliser des heuristiques constructives dans cette méthode qui facilite l'analyse des contraintes. Par conséquent, la structure générale de la méthode IP / LP dans le problème UCTTP est présentée ci-dessous :

La formulation en IP / LP est effectuée sur la base de différents regroupements en deux classes sur le regroupement des cours et le regroupement des intervalles de temps. Dans un premier temps, les groupes de cours doivent être définis, puis les cours sont sélectionnés en groupes de matières par les étudiants correspondants et ces groupes de matières doivent être programmés à des plages horaires différentes. Le crénaux de temps sont également en groupes de quatre heures et divisés en deux crénaux de deux heures. Les cours alloués aux groupes horaires mentionnés sont définis comme un cluster, c'est-à-dire qu'un groupe de cours est programmé sur un intervalle de temps donné (17).

Aubin et Ferland (1989) ont mis au point une procédure plus générale pour traiter des problèmes d'horaire à grande échelle (18). Ils ont séparé le problème de l'horaire en deux sous-problèmes qui sont appelés sous-problème d'horaire et sous-problème de regroupement. Ils ont proposé une approche heuristique pour résoudre ce problème par l'application simultanée de sous-problèmes d'horaire et de regroupement, jusqu'à ce qu'une solution finale soit trouvée et améliorée autant que possible.

1.2.2 Les méthodes métaheuristiques

1.2.2.1 Définition

Une métaheuristique est un algorithme d'optimisation visant à résoudre des problèmes d'optimisation difficile (souvent issus des domaines de la recherche opérationnelle, de l'ingénierie ou de l'intelligence artificielle) pour lesquels on ne connaît pas de méthode classique plus efficace (19).

Les métaheuristiques sont généralement des algorithmes stochastiques itératifs, qui progressent vers un optimum global, c'est-à-dire l'extremum global d'une fonction, par échantillonnage d'une fonction objectif. Elles se comportent comme des algorithmes de recherche, tentant d'apprendre les caractéristiques d'un problème afin d'en trouver une approximation de la meilleure solution (d'une manière proche des algorithmes d'approximation).

Il existe un grand nombre de métaheuristiques différentes, allant de la simple recherche locale à des algorithmes complexes de recherche globale.Ces méthodes utilisent cependant un haut niveau d'abstraction, leur permettant d'être adaptées à une large gamme de problèmes différents.

Les approches métaheuristiques comprennent deux méthodes principales, l'une basée sur la population et l'autre à solution unique (20).

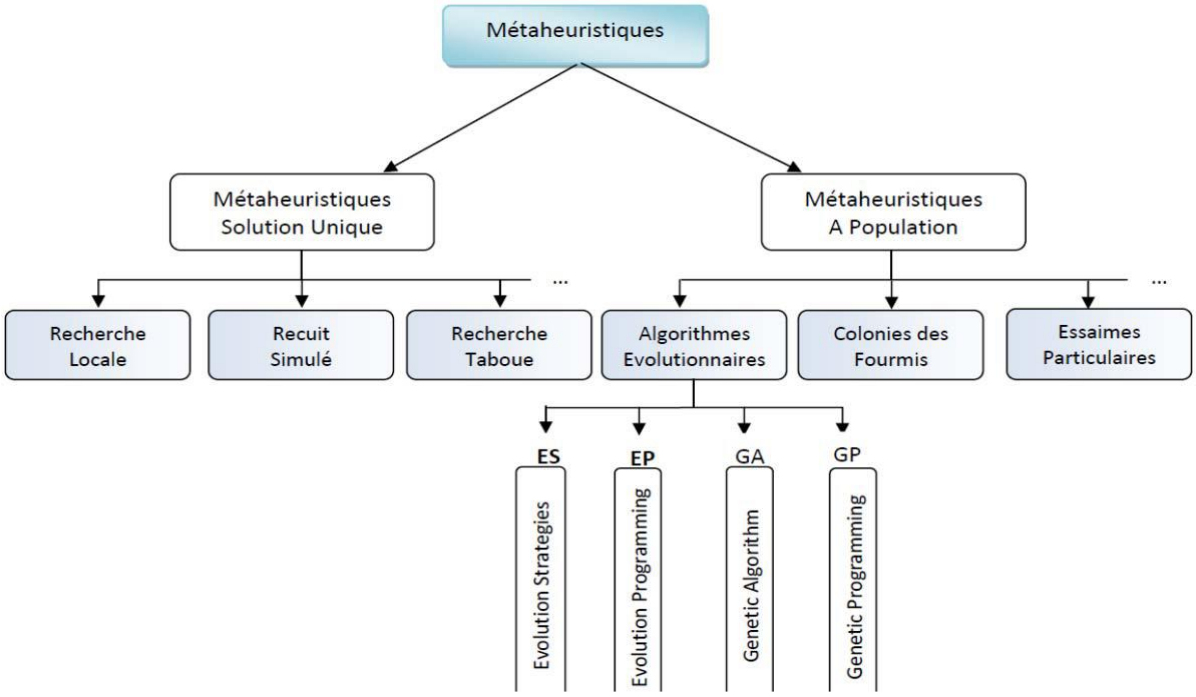

Figure 2 : Classes des métaheuristiques.

1.2.2.2 Approches métaheuristiques basées sur la population

Dans la méthode basée sur la population, nous avons d'abord un certain nombre de personnes ou de solutions initiales où cet ensemble de personnes est appelé population initiale.

A chaque itération des approches métaheuristiques basées sur la population, un mécanisme de sélection est utilisé pour sélectionner la ou les meilleures solutions de la population actuelle, puis, en fonction de la méthode métaheuristique, certains changements sont appliqués sur les

solutions sélectionnées afin que l'amélioration soit obtenue dans la solution, maintenant ces solutions améliorées remplacent les anciennes solutions.Cette procédure se poursuit jusqu'à ce qu'une solution souhaitable soit trouvée (21).

Cette méthode comprend, entre autres, les algorithmes suivants : algorithmes évolutifs et génétiques, algorithmes de colonies de fourmis, algorithme mimétique, algorithme de recherche d'harmonie, optimisation partielle des essaims et optimisation des colonies d'abeilles artificielles.

1.2.2.2.1Algorithmes évolutifs et génétiques

Les algorithmes évolutifs (EA) sont basés sur un modèle informatique s'inspirent du mécanisme d'évolution naturel. Les EA agissent sur une population de solutions possibles et comprennent trois étapes:

1) La sélection.

2) La régénération.

3) Le remplacement.

Dans la phase de sélectionles personnes ayant une meilleure forme physique sont sélectionnées pour être les parents de la prochaine génération.En phase de régénération deux opérateurs de croisement et de mutation sont réalisés sur les parents qui ont été sélectionnés dans la première phase et dans laphase de remplacement les personnes de la population d'origine (initiale) sont remplacées par les personnes nouvellement créées (22).

1.2.2.2.2Algorithmes génétiques

Les algorithmes génétiques appartiennent à la famille des algorithmes évolutionnistes. Leur but est d'obtenir une solution approchée à un problème d'optimisation, lorsqu'il n'existe pas de méthode exacte (ou que la solution est inconnue) pour le résoudre en un temps raisonnable. Les algorithmes génétiques utilisent la notion de sélection naturelle et l'appliquent à une population de solutions potentielles au problème donné. La solution est approchée par « bonds » successifs, comme dans une procédure de séparation et évaluation (Branch-And-Bound), à ceci près que ce sont des formules qui sont recherchées et non plus directement des valeurs.

Les algorithmes génétiques qui sont un sous-ensemble d'EA sont basés sur les étapes suivantes :

1) Générer la population initiale.

2) Évaluer la population générée en utilisant la fonction d'évaluation.

3) Sélectionner certaines personnes comme parents à croiser en fonction des informations obtenues à partir des fonctions d'évaluation.

4) Appliquer l'opérateur de croisement pour produire des enfants.

5) Appliquer l'opérateur de mutation sur les enfants.

6) Sélectionner les parents et les enfants pour former la nouvelle population de la génération future.

7) Appliquer l'opérateur de mutation sur les enfants.

8) Si la condition de terminaison est satisfaite, l'algorithme s'arrête.Sinon, il passe à la deuxième étape et continue (23).

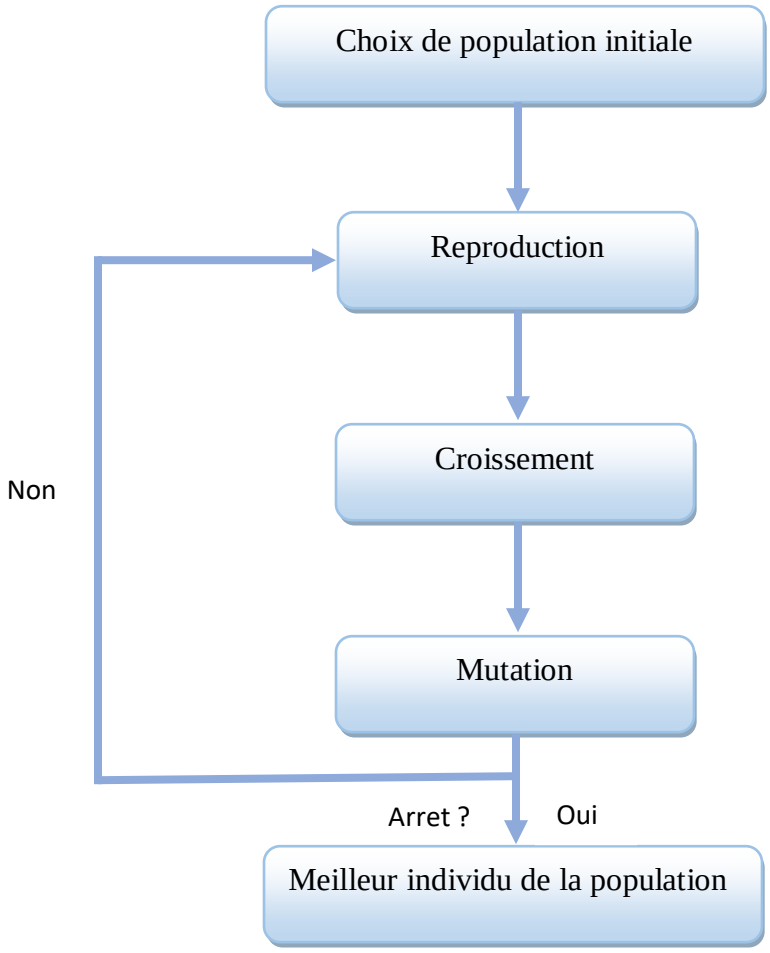

Figure 3 : Les étapes de fonctionnement d'un algorithme génétique.

1.2.2.2.3Algorithmes génétiques et le problème de l'emploi du temps

Khonggamnerd et Innet (2009) ont utilisé un algorithme génétique pour générer les horaires d'une université où le taux de croisement était de 70%. Cependant, aucune contrainte stricte n'a été violée dans l'établissement des horaires.Les contraintes utilisées entraînent une plus grande occupation des salles et une plus grande capacité des salles (24).

Cependant, Alsmadi, Abo-Hammour, Abu-Al-Nadi et Algsoon (2011) ont proposé un nouveau AG pour résoudre le problème UCTTP qui utilise une machine d'apprentissage (25).

Les résultats de cette technique consistent à minimiser le nombre de contraintes souples violées, à maximiser l'utilisation des salles disponibles et à réduire la charge de travail des enseignants.

1.2.2.3Algorithme d'optimisation des colonies de fourmis

1.2.2.3.1définition

Les algorithmes de colonies de fourmis (en anglais, Ant Colony Optimization, ou ACO) sont des algorithmes inspirés du comportement des fourmis, ou d'autres espèces formant un super-organisme, et qui constituent une famille de métaheuristiques d'optimisation (26).

Initialement proposé par Marco Dorigo et al. dans les années 1990, pour la recherche de chemins optimaux dans un graphe, le premier algorithme s'inspire du comportement des fourmis recherchant un chemin entre leur colonie et une source de nourriture. L'idée originale s'est depuis diversifiée pour résoudre une classe plus large de problèmes et plusieurs algorithmes ont vu le jour, s'inspirant de divers aspects du comportement des fourmis (27).

En anglais, le terme consacré à la principale classe d'algorithme est « AntColony Optimisation » (ACO). Les spécialistes réservent ce terme à un type particulier d'algorithmes. Il existe cependant plusieurs familles de méthodes s'inspirant du comportement des fourmis. En français, ces différentes approches sont regroupées sous les termes : « algorithmes de colonies de fourmis », « optimisation par colonies de fourmis », « fourmis artificielles » ou diverses combinaisons de ces variantes.

Cette méthode s'inspire du comportement des fourmis pour trouver un itinéraire entre le lieu de la fourmilière et la nourriture. Les fourmis se déplacent toujours au hasard pour trouver de la nourriture, puis elles placent une trace de phéromone.

D'autres fourmis trouvent cette route, la suivent et si elles atteignent la nourriture, elles rentrent chez elles et placent une piste en plus de la piste précédente.La phéromone s'évapore

légèrement là où les résultats de la route deviendraient moins attrayants pour la fourmi suivante ; par conséquent, la recherche aléatoire est limitée à cet aliment.

Le but de cette approche est le déplacement des fourmis artificielles pour trouver l'itinéraire le plus court .

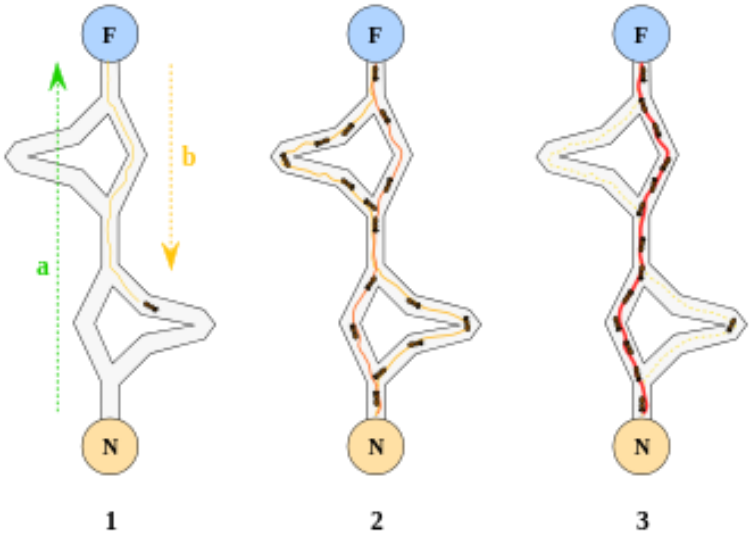

Figure 4 : Processus de l'algorithme d'optimisation des colonies de fourmis.

1.2.2.3.2Colonies de fourmis et emploi du temps

L'utilisation du système de colonies de fourmis Max – Min pour générer l'horaire des cours universitaires par Socha et al. (2002) a conduit à la construction d'un chemin optimal où chaque chemin pourrait générer un graphique constructif pour attribuer des cours à des intervalles de temps affectés par la quantité de phéromone dans une plage horaire (28).

Cependant, l'application de l'algorithme d'optimisation des colonies de fourmis par Mayer, Nothegger, Chwatal et Raidl (2008) pour le problème UCTTP a été post-inscrite selon le jeu de données ITC-2007 où les fourmis attribuent des événements aux salles et aux intervalles de temps en fonction de deux types de phéromone Tsij et Tyij (29).

Cet algorithme a de bonnes performances et conduit à de bons résultats avec un temps d'exécution élevé.

L'application d'un système hybride de colonies de fourmis a été proposée par Ayob et Jaradat (2009) pour résoudre le problème UCTTP (30).

Ici, deux types de système de fourmis hybrides, y compris une combinaison de recuit simulé (SA) avec colonie de fourmis (AC) et de recherche tabou (TS) avec (AC) ont été présentés.

Un certain nombre de fourmis effectuent l'attribution complète des cours aux plages horaires en fonction d'une liste prédéfinie.

La sélection des probabilités de créneaux horaires par les fourmis pour allouer des cours a été effectuée en utilisant des informations heuristiques et des informations d'un mécanisme de coordination indirecte parmi les agents et les activités au sein d'un environnement.

1.2.2.4Algorithmes mémétiques

1.2.2.4.1Définition

Les algorithmes mémétiques appartiennent à la famille des algorithmes évolutionnistes. Leur but est d'obtenir une solution approchée à un problème d'optimisation, lorsqu'il n'existe pas de méthode de résolution exacte en un temps raisonnable. Les algorithmes mémétiques sont nés d'une hybridation entre les algorithmes génétiques et les algorithmes de recherche locale. Ils utilisent le même processus de résolution que les algorithmes génétiques mais utilisent un opérateur de recherche locale après celui de mutation. L'intérêt de cette classe d'algorithmes est l'apport de la diversification de la partie génétique accompagnée par l'intensification de la recherche locale. On peut classer les algorithmes mémétiques dans les métaheuristiques (31).

Le principe général des algorithmes mémétiques est semblable à celui des algorithmes génétiques à la différence près qu'un opérateur de recherche locale est ajouté après celui de mutation. Ce dernier permet d'ajouter une intensification à la diversification apportée par le principe de l'algorithme génétique (31). Un algorithme génétique a besoin pour son fonctionnement d'une population initiale. Celle-ci peut-être construite de manière aléatoire ou de façon gloutonne. L'algorithme itère différentes étapes ensuite jusqu'à ce qu'un critère d'arrêt propre à chaque problème/utilisateur est atteint :

- ✓ sélection de deux parents (individus) dans la population.
- ✓ application d'un opérateur de croisement entre les deux parents afin de générer un nouvel enfant (individu).
- ✓ application d'un opérateur de recherche locale à l'enfant.
- ✓ mise à jour de la population avec l'enfant (selon la stratégie garder les meilleures solutions, un échantillon large, etc.).

1.2.2.4.1 Algorithme mémétique et emploi de temps

L'algorithme mémétique a été proposé par Jat et Shengxiang (2008) pour résoudre le problème UCTTP grâce à une combinaison de méthode de recherche locale dans des algorithmes génétiques (32).

Une recherche locale a été exécutée sur les événements et une autre sur les intervalles de temps.

1.2.3 Approches métaheuristiques basées sur une solution unique

Cette méthode est également l'une des méthodes métaheuristiques qui utilise une solution unique pour analyser le problème au lieu d'utiliser une population initiale pour résoudre un problème d'optimisation, de sorte qu'au début une seule solution soit sélectionnée en fonction de certains critères et jusqu'à la fin du processus de résolution, cette solution unique est manipulée et déplacée jusqu'à ce que l'amélioration finale de la solution finale soit obtenue. L'étape de terminaison se produit lorsque la condition et le critère finaux sont satisfaits (33).

1.2.3.1 Recherche Tabou

1.2.3.1.1 Définition

La recherche tabou est une métaheuristique d'optimisation présentée par Fred W. Glover en 1986. On trouve souvent l'appellation recherche avec tabous en français. Cette méthode est une métaheuristique itérative qualifiée de recherche locale au sens large (34).

L'idée de la recherche tabou consiste, à partir d'une position donnée, à en explorer le voisinage et à choisir la position dans ce voisinage qui minimise la fonction objectif. Il est essentiel de noter que cette opération peut conduire à augmenter la valeur de la fonction (dans un problème de minimisation) : c'est le cas lorsque tous les points du voisinage ont une valeur plus élevée. C'est à partir de ce mécanisme que l'on sort d'un minimum local (35).

Le risque cependant est qu'à l'étape suivante, on retombe dans le minimum local auquel on vient d'échapper. C'est pourquoi il faut que l'heuristique ait de la mémoire : le mécanisme consiste à interdire (d'où le nom de tabou) de revenir sur les dernières positions explorées. Les positions déjà explorées sont conservées dans une file FIFO (appelée souvent liste tabou) d'une taille donnée, qui est un paramètre ajustable de l'heuristique. Cette file doit conserver des positions complètes, ce qui dans certains types de problèmes, peut nécessiter l'archivage d'une grande quantité d'informations. Cette difficulté peut être contournée en ne gardant en mémoire que les mouvements précédents, associés à la valeur de la fonction à minimiser (36).

1.2.3.1.2 Recherche tabou et emploi de temps

L'algorithme de recherche Tabou a été appliqué par Alvarez, Crespo et Tamarit (2002) pour la première fois pour affecter des étudiants aux groupes de cours afin de générer des emplois du temps de haute qualité et d'équilibrer le nombre d'étudiants qui se sont inscrits dans chaque groupe et de bons résultats ont été obtenus (37).

Le processus d'allocation de la méthode d'Alvarez et al (2002) comporte deux phases :

1) la première phase, pour générer un ensemble de solutions pour un étudiant.

2) la seconde phase, une combinaison d'un ensemble de solutions et l'application de TS avec des stratégies locales pour obtenir des emplois du temps de haute qualité sans considérer la (les) pire (s) solution (s) pour chaque étudiant.

Aladag, Hocaoglu et Basaran (2009) ont présenté l'influence des structures voisines sur un algorithme TS pour résoudre le problème UCTTP où l'influence des mouvements simples et swap sont testés sur des opérations TS basées sur des structures voisines (38).

Aussi, deux nouvelles structures voisines ont été proposées en utilisant des mouvements simples et swap.

Ici, quatre structures voisines appliquées et la comparaison des résultats obtenus à partir de ces structures a été spécifiée.

L'algorithme TS comprend l'utilisation de stratégies avancées et de composants communs comme la liste Tabou, diverses mémoires, des structures voisines, etc. L'un des principaux facteurs est l'influence de l'efficacité de l'algorithme basé sur les structures voisines définies qui dépend de la nature des problèmes.

Ici, deux structures voisines basées sur différents types de mouvements comme simples, swap et deux autres structures voisines ont été utilisées qui ont été appliquées par Aladag et Hocaoglu (2007) et Alvarez et al (2002), respectivement.

Le but est de combiner les différents effets dans l'algorithme TS appliqué.

En raison des résultats obtenus, de multiples comparaisons ont été présentées statistiquement parmi toute la structure voisine.

1.2.3.2 Recuit simulé

1.2.3.2.1 Définition

Le recuit simulé (SA) (en anglais Simulated Annealing) est une méthode empirique (métaheuristique) inspirée d'un processus utilisé en métallurgie. On alterne dans cette dernière des cycles de refroidissement lent et de réchauffage (recuit) qui ont pour effet de minimiser l'énergie du matériau. Cette méthode est transposée en optimisation pour trouver les extrema d'une fonction (39).

Elle a été mise au point par trois chercheurs de la société IBM, S.Kirkpatrick, C.D.Gelattet M.P.Vecchien1983,et indépendamment par V.Černyen1985.

La méthode vient du constat que le refroidissement naturel de certains métaux ne permet pas aux atomes de se placer dans la configuration la plus solide. La configuration la plus stable est atteinte en maîtrisant le refroidissement et en le ralentissant par un apport de chaleur externe, ou bien par une isolation (40).

1.2.3.2.2 Recuit simulé et emploi du temps

Pour résoudre le problème UCTTP, la combinaison de la chaîne voisine de Kempe dans l'algorithme de recuit simulé a été présentée par Tuga, Berretta et Mendes (2007) (41).

Dans cette méthode, les contraintes dures sont reformulées par relaxation; donc ces contraintes sont créées sous la forme d'une contrainte souple assouplie. Cependant, le problème de relaxation est analysé en deux étapes:

1) créer une solution réalisable basée sur un graphe heuristique.

2) un algorithme de recuit simulé a été utilisé pour minimiser les violations des contraintes souples (dans la deuxième phase, une heuristique basée sur la chaîne voisine de Kempe a été utilisée).

Cependant, l'approche de recuit simulé est présentée par Aycan et Ayav (2008) pour résoudre le problème UCTTP (42).

Ils ont comparé l'efficacité de différents algorithmes de recherche voisins basés sur une recherche simple, une recherche d'échange, une recherche simple swap et ont calculé le coût d'exécution pour chaque méthode.

La plus grande satisfaction de la planification des horaires est obtenue par une combinaison de trois algorithmes mentionnés.

1.2.3.3 Recherche locale

1.2.3.3.1 Définition

En algorithmique, la recherche locale est une méthode générale utilisée pour résoudre des problèmes d'optimisation, c'est-à-dire des problèmes où l'on cherche la meilleure solution dans un ensemble de solutions candidates. La recherche locale consiste à passer d'une solution à une autre dans l'espace des solutions candidates (l'espace de recherche) jusqu'à ce qu'une solution considérée comme optimale soit trouvée ou que le temps imparti soit dépassé (43).

Un algorithme de recherche locale part d'une solution candidate et la déplace de façon itérative vers une solution voisine. Cette méthode est applicable seulement si une notion de voisinage est définie sur l'espace de recherche. Par exemple, le voisinage d'un arbre recouvrant est un autre arbre recouvrant qui diffère seulement par un noeud. Pour le problème SAT3, les voisins d'une bonne affectation sont habituellement les affectations qui diffèrent seulement par la valeur d'une variable. Le même problème peut avoir plusieurs définitions différentes de voisinage, l'optimisation locale avec des voisinages qui limitent les changements à k composantes de la solution est souvent appelée k-optimale (44).

Habituellement, chaque solution candidate a plus d'une solution voisine ; le choix de celle vers la quelle se déplacer est pris en utilisant seulement l'information sur les solutions voisines de la solution courante, d'où le terme de recherche locale. Quand le choix de la solution voisine est fait uniquement en prenant celle qui maximise le critère, la métaheuristique prend le nom de hillclimbing(escaladedecolline).

Le critère d'arrêt de la recherche locale peut être une limite en durée. Une autre possibilité est de s'arrêter quand la meilleure solution trouvée par l'algorithme n'a pas été améliorée depuis un nombre donné d'itérations. Les algorithmes de recherche locale sont des algorithmes sous-optimaux puisque la recherche peut s'arrêter lorsque la meilleure solution trouvée par l'algorithme n'est pas la meilleure de l'espace de recherche. Cette situation peut se produire même si l'arrêt est provoqué par l'impossibilité d'améliorer la solution courante car la solution optimale peut se trouver loin du voisinage des solutions parcourues par l'algorithme.

1.2.3.3.3 Recherche locale et emploi du temps

Joudaki, Imani et Mazhari (2010) ont utilisé un algorithme de recherche locale et une méthode MA pour résoudre le problème UCTTP (45). Ces chercheurs ont présenté une méthode basée sur une combinaison améliorée de SA et MA. Dans leurs recherches, l'algorithme SA est appliqué comme une routine de recherche locale qui augmente la capacité d'extraction de MA.

Cependant, la modification de l'opérateur de croisement dans MA et la création de la population initiale par une méthode heuristique ont été effectuées dans cet algorithme. L'opérateur d'optimisation est effectué en optimisant les chromosomes générés et en minimisant le nombre de violations des contraintes et il est ajouté en tant que nouvel opérateur à MA. Les facteurs de succès de MA sont d'utiliser la capacité heuristique des algorithmes évolutifs et de combiner cet avantage avec la capacité d'extraction des procédures de recherche locales. Shengxiang et Jat (2011) ont utilisé une stratégie de recherche locale et guidée dans le cadre du processus GA pour résoudre le problème UCTTP (46).

Ici, la stratégie de recherche guidée utilise une structure de données pour créer des enfants, où cette structure stocke les informations extraites des bonnes personnes des générations précédentes.

La séparation de recherche locale est une technique d'extraction qui a la capacité d'améliorer l'efficacité de recherche des GA.

Les résultats de consolidation de cette recherche locale en GA sont satisfaisants.

L'objectif est de maximiser les allocations et de minimiser la violation des contraintes souples. GSGA (GuidedSearchGeneticAlgorithm) comprend une stratégie de recherche guidée et une technique de recherche locale.

Cependant, dans la GSGA, une technique de recherche locale est appliquée pour améliorer la qualité des personnes en recherchant dans trois structures voisines.

Ici, le but est d'étudier l'efficacité des stratégies GSGA avec LS pour le problème UCTTP et un cadre uniforme de combinaison de stratégies GA et LS standard a été utilisé et l'utilisation de LS dans GSGA aboutit au développement d'approches GSGA pour résoudre le problème UCTTP appelé EGSGA.

1.2.3.4 Autre méthode de résolution basé sur une solution unique

L'algorithme de recherche de voisinage variable (VNS) a été présenté par Abdullah, Burke et Mc Colloum (2005) pour résoudre le problème UCTTP qui proposait un VNS de base puis pour utiliser un critère d'acceptation exponentiel de Monte Carlo par chaque solution (47).

L'idée principale est d'appliquer un critère d'acceptation de Monte Carlo pour l'amélioration des explorations par l'acceptation de la meilleure solution avec la probabilité donnée afin de trouver le nombre de voisins promis.

Encore une fois, Abdullah, Burke et McColloum, 2007a, 2007b ont présenté une amélioration itérative randomisée avec un algorithme composite voisin (RIICN) pour améliorer son algorithme précédemment présenté qui est en fait la combinaison de VNS et de recherche locale (47).

La liste Tabou a été appliquée à la pénalité des structures voisines inefficaces et non promises après un nombre donné d'itérations (48).

1.3 Conclusion

Nous pouvons conclure que le problème de la planification des emplois du temps à l'université a fait l'objet d'études de nombreux chercheurs, à travers lesquels ils ont utilisé de nombreuses méthodes pour tenter de résoudre ce problème, et nous nous sommes concentrés dans ce premier chapitre sur les différentes expériences dans lesquelles des méthodes métaheuristiques ont été utilisées.

Dans le chapitre suivant nous nous intéressons aux algorithmes génétiques utilisés pour résoudre le problème de l'emploi du temps.

Chapitre 2 : Les algorithmes génétiques pourla résolution du problème de l'emploi du temps

2.1 Inroduction

Les algorithmes génétiques tentent de simuler le processus d'évolution naturelle suivant le modèle darwinien dans un environnement donné. Ils utilisent un vocabulaire similaire à celui de la génétique naturelle. Cependant, les processus naturels auxquels ils font référence sont beaucoup plus complexes. On parlera ainsi d'individu dans une population. L'individu est représenté par un chromosome constitué de gènes qui contiennent les caractères héréditaires de l'individu. Les principes de sélection, de croisement, de mutation s'inspirent des processus naturels de même nom.

Pour un problème d'optimisation donné, un individu représente un point de l'espace d'états, une solution potentielle. On lui associe la valeur du critère à optimiser, son adaptation. On génère ensuite de façon itérative (des populations d'individus sur lesquelles on applique des processus de sélection, de croisement et de mutation. La sélection a pour but de favoriser les meilleurs éléments de la population pour le critère considéré (les mieux adaptés), le croisement et la mutation assurent l'exploration de l'espace d'états. Le problème de l'amélioration de la planification de l'emploi du temps dans les universités réside dans le respect des contraintes, et il est naturellement divisé en deux phases: trouver des solutions acceptables puis trouver la solution de coût optimale entre les deux. Selon la méthode utilisée, cette division est plus ou moins nette dans la résolution. L'utilisation de l'algorithme génétique convient à l'exploration rapide et approfondie d'une vaste zone de recherche et est capable de fournir de nombreuses solutions (49). Dans une situation où l'ensemble des solutions acceptables est complexe, l'acceptabilité peut être rendue intrinsèque à la représentation choisie ou incorporée dans la génération de chromosomes (mutation, hybridation) ou dans une fonction qui peut être améliorée. Dans ce chapitre nous nous intéressons aux algorithmes génétiques utilisés pour résoudre le problème de l'emploi du temps. Dans la première partie, nous décrirons les principes de l'algorithme génétique puis nous présentons quelques algorithmes utilisés pour résoudre le problème de l'emploidu temps.

2.2 Principe de l'algorithme génétique

2.2.1 Codage des variables

L'un des facteurs les plus importants pour les algorithmes génétiques, si ce n'est le plus important, est la façon dont sont codées les solutions (ce que l'on a nommé ici les chromosomes), c'est-à-dire les structures de données qui coderont les gènes. ily a différentes techniques de codages (20).

Le processus de codage peut être effectué par utilisation des : bits, nombres, arbres, tableaux, listes ou tous autres objets. La littérature définit deux types de codage : binaire et réel.

2.2.1.1 Codage binaire

La procédure normale pour coder un algorithme génétique ayant plusieurs paramètres est de coder chaque paramètre comme une séquence de bits.

Les séquences sont ensuite placées l'une après l'autre pour former une grande séquence, c'est le chromosome.

Le chromosomereprésente le vecteur des paramètres.

Chaque séquence du vecteur total représente un gène.

2.2.1.2 Codage réel

La première technique de codage utilisée dans les algorithmes génétiques est le codage binaire mais,certainement, le codage binaire pose des problèmes, et le codage réel est plus précis (50).

Ce type de codage présente certains avantages par apport au codage binaire :

- ✓ Le codage réel est robuste pour les problèmes considérés mais plus difficile que le codage binaire.
- ✓ Ce codage nécessite une adaptation des opérateurs de croisement et mutation.

2.2.2 La population initiale

Il existe plusieurs mécanismes pour générer une population initiale, le choix de l'initialisation se fera par apport aux connaissances que l'utilisateur a sur le problème.

Généralement une initialisation aléatoire est la plus uniforme possible afin de favoriser une exploration de l'espace de recherche maximum, sera la plus adaptée. Lorsqu'on crée une population initiale, les valeurs des paramètres (les allèles des gènes) sont créées aléatoirement

avec une distribution uniforme sur l'intervalle spécifié. Par ailleurs, cette étape présente un problème principal qui est celui de choix de la taille de la population. La taille de la population initiale est également laissée à l'appréciation du programmeur. Il n'est souvent pas nécessaire d'utiliser des populations démesurées.

2.2.3 Adaptation

Le principe de la fonction d'adaptation (fitness) est d'associer une valeur à chaque individu. Cette valeur a pour but de choisir l'individu le mieux adapté par apport à son environnement. Nous allons donc attribuer un indice de qualité à chacun de ces individus.

La fonction d'adaptation des individus est laissée au programmeur en fonction du problème à optimiser ou à résoudre. Cette fonction est plus facile à formuler lorsque il ya peu de paramètres.Par contre, lorsqu'il ya beaucoup de paramètres ou lorsqu'ils sont corrélés, la fonction d'adaptation est plus difficile à définir. Dans ce cas la fonction devient une somme pondérée de plusieurs fonctions.

2.2.4 Sélection

Le principe de la sélection est d'enrichir la population en croisant des individus. Cette phase s'effectue par prendre des morceaux de solutions de certains individus et d'autres morceaux d'autres individus pour créer des nouveaux individus qui, on l'espère, seront des solutions meilleures aux problème traité. Il est tout à fait possible de choisir des individus au hasard et de les mélanger aléatoirement pour créer de nouveaux individus.

On trouve essentiellement quatre types de méthodes de sélection différentes :

- ✓ La méthode de la roulette.
- ✓ La méthode "élitiste".
- ✓ La sélection par tournois.
- ✓ La sélection par rang.

2.2.5 Croisement

Dans la méthode de croissement on prend en entrée un couple d'individus parents P_1et P_2 renvoie un couple d'individus enfants C_1et C_2obtenus en choisissant aléatoirement un point de croisement (ou éventuellement plusieurs points de croisement pour éviter certains effets de bord

du codage) dans les chromosomes et en recopiant dans le fils C_1les gènes de P_1jusqu'au point de croisement puis en complétant avec les gènes de P2. On effectue l'opération symétrique pour C2.

La littérature définit plusieurs opérateurs de croisement. Ils différent selon le type de codage adapté et la nature du problème traité.

Dans le codage binaire il ya cinq types de croissement :

1) Croisement en 1-point,

2) Croisement en 2-points,

3) Croisement en n-points,

4) Croisement uniforme,

5) Croisement avec trois parents.

Dans le codage réel, il ya trois types de croissement :

1) L'opérateur de Croisement PMX (PartiallyMappedCrossover).

2) L'opérateur de Croisement CX (Cycle Crossover).

3) L'opérateur de Croisement OX (Order Crossover).

2.2.6 Mutation

L'opérateur de mutation classique prend en entrée un individu (I) sélectionné pour la mutation et renvoie un individu mutant (I') obtenu par transformation locale de l'un des gènes de (I). Par exemple, si le chromosome d'un individu est codé avec une chaîne de bits, on peut le muter en complémentant l'un de ses gènes/bits.

Comme la méthode de croissement, la mutation diffère selon le type de codage adapté.

Dans un codage binaire, lorsqu'il y a mutation, les bits sont changés de 0 à 1 ou de1 à 0.Lorsqu'un bit subit une mutation, on peut le percevoir comme une perturbation du paramètre réel.

Dans la mutation en codage réel, les opérateurs de mutation les plus utilisés sont les suivants :

1) Mutation par inversion.

2) Mutation par insertion.

3) Mutation par déplacement.

4) Mutation par permutation.

2.3 Une nouvelle technique d'algorithme génétique pour résoudre les problèmes de l'emploi du temps des cours à l'université

Cette approche a été appliquée par Alsmadi, Abo-Hammour, Abu-Al-Nadi et Al-gsoon (2011), à travers laquelle ils ont suivi une méthode basée sur l'algorithme génétique (25). Nous expliquerons ces étapes dans ce qui suit.

2.3.1 Population initiale

Ils ont généré plusieurs individus qui sont libres de la violation de contraintes dure. Chaque individu se compose d'un certain nombre d'entrées (Ne) où chaque entrée comprend un certain nombre de paramètres qui sont nécessaires pour former un calendrier approprié. Aucun enseignant ne peut donner deux conférences ou plus en même temps.

Les principales contraintes dures considérées sont :

✓ Deux conférences ou plus ne peuvent pas avoir lieu dans la même salle de cours à la même période.

✓ Le nombre maximum de périodes par semaine est de 15 périodes. Ce nombre ne peut pas être dépassé.

✓ Pour un groupe spécifique, chaque cours doit être attribué à un seul instructeur qui enseigne habituellement le cours.

✓ Les séances de théorie de type cours nécessitent 3 heures sont séparées au cours de la semaine, mais les classes de type «travaux pratiquesau laboratoire» nécessitent 3 heures continues en une seule journée.

✓ Les séances de type «travaux pratiques au laboratoire» sont attribuées à un certain laboratoire, pas dans une salle de cours.

2.3.2 Évaluation de Fitness

L'équation (1) de Fitness utilisée dans cette approche est :

$$F = \sum_{i=1}^{Ne} pd\left(1 - \frac{nvd(i)}{nvdmax}\right) + ps\left(1 - \frac{nvs(i)}{nvsmax}\right) (1)$$

Où Pd est le poids des contraintes dures, nvd est le nombre des violations de contrainte dur, ndv max est le nombre maximum de violations possibles de contraintes dures, Ps est le poids des contraintes souples, nvs est le nombre de violations de contraintes souples, nvs max est le nombre maximum de violations possibles de contrainte souples.

Les contraintes souples utilisées dans l'approche proposée sont :

- ✓ Contraintes temporelles souples (TSC) :

  Les enseignants peuvent préférer des moments spécifiques pour leurs cours.

- ✓ Contraintes souples de salle (SSC) :

  Les enseignants peuvent choisir des salles spécifiques pour leurs cours.

- ✓ Contrainte souple de niveau (NSC) :

  Des cours de même niveau peuvent être donnés en même temps ou à des plages horaires différentes.

- ✓ Pré requis de contrainte souple (PSC) :

  Les cours et leurs prés requis sont programmés pour que les étudiants puissent s'inscrire facilement.

- ✓ Contrainte souple de surcharge de l'instructeur (ISSC):

  Cette contrainte permet d'obtenir une distribution de cours adaptée à tous les instructeurs afin d'éviter les surcharges.

2.3.3 Sélection

Ils ont utilisé le type de sélection basé sur le rang où un nombre prescrit d'individus de parents est choisi avec la plus haute aptitude selon le rapport basé sur le rang(rbr). Le croisement est effectué en choisissant les parents au hasard dans cette sous-population, qui est de taille Rrb * Np où Np est le nombre d'individus dans la population initiale. Les 10% d'individus les plus adaptés de la population initiale seront sélectionnés pour entrer dans le pool de croisement.

2.3.4 Croisement

Dans leur recherche, ils sélectionnent au hasard les individus du pool de croisement, puis ils définissent un nombre aléatoire. Si la valeur du nombre aléatoire est inférieure à une certaine probabilité de croisement, alors ils font des croisements, sinon ils passent sans aucun changement.

2.3.5 Mutation

La nouvelle progéniture, après croisement, est choisie au hasard pour effectuer la mutation. Ce processus est régi par une certaine probabilité, Pm.

2.3.6 Remplacement

Le schéma de remplacement général est utilisé dans cette approche où le nombre de meilleurs parents réussis (N_e) dépend du facteur de remplacement (R_f) et du nombre d'individus de la population (Npop) tels que :

$$N_e=(1-R_f)N_{pop}$$

D'autre part, le nombre de descendants élites réussis (Ne) dépend du facteur de remplacement donnant No = Rf * Npop. Par conséquent, la taille totale de la population (Ptotal) est donnée par No plus Ne.

2.3.7 Résiliation

L'AG prend fin lorsqu'un certain critère de convergence est satisfait. Dans l'approche de planification proposée, l'AG prendra fin lorsque l'aptitude devient égale ou supérieure à 0,99 ou lorsque le nombre maximal de générations est atteint.

2.3.8 Simulations et résultats

Le tableau suivant présente les informations générales utilisées par l'AG :

Tableau 1 : Données générales utilisées par l'AG

No.	Opérateur	quantité/Type
1	Nombre d'individus	1200
2	Probabilité de croisement	0.7
3	Probabilité de mutation	0.1
4	Nombre de générations	500
5	Mécanisme de sélection	Sélection basée sur le rang
6	Type de croisement	Croisement à point unique

Il est important de mentionner que les taux de croisement et de mutation ont été sélectionnés en fonction de l'adéquation des résultats obtenus par les différentes simulations.

Le tableau suivant présente les résultats de la comparaison où l'on peut voir que la méthode GA fournit de meilleurs résultats.

Tableau 2 : Comparaison des résultats préparés à la main et du GA.

Type de violations	**Préparé à la main**	**GA préparé**
Violations des contraintes de temps	0	0
Violations de contrainte de salles	0	0
Surcharge des instructeurs	100	4
Cours de même niveau / même heure	180	120
Violations des conditions préalables	256	250
Sections hors facultés	**7**	0

Le tableau suivant montre le nombre de violations (Nv) dans les contraintes souples par rapport à différents poids.

Tableau 3 : Le nombre de violations (Nv) dans les contraintes souples par rapport à différents poids.

p_s	TSC	SSC	NSC	ISSC	PSC
	N_v	N_v	N_v	N_v	$\boldsymbol{N_v}$
0	480	536	354	5	333
0.1	474	558	355	5	332
0.2	484	554	354	5	336
0.3	493	560	346	5	330
0.4	489	550	350	5	334
0.5	490	546	351	5	340
0.6	486	552	343	5	331
0.7	497	542	340	5	331
0.8	489	554	343	5	334
0.9	481	558	347	5	336
1	482	550	342	5	333

On voit également que le nombre de violations dans chaque contrainte est presque le même par rapport au changement de son poids. Par exemple, le nombre maximal de violations dans la contrainte souple temporelle (TSC), qui est de 497, pour un poids de 0.7, n'est pas très différent du nombre minimal de violations, qui est de 474, pour un poids de 0.1. En fait, cela indique que

la sensibilité (Sv) de cette approche aux changements de poids est très faible. Par conséquent, le grand avantage est qu'il n'y a pas besoin d'une expérience utilisateur. Pour vérifier le Sv exact, la sensibilité peut être donnée par l'équation (2) suivante :

$$\text{Sv} = \frac{Nvmax - Nvmin}{Nvmax} * 100\% \qquad (2)$$

Où nvmax est le nombre maximum de violations de contraintes et nvmin est le nombre minimum de violations de contraintes. Sv pour la contrainte souple de temps est obtenue comme ((497 – 474)/497) * 100% = 4,63%. Il est important de mentionner que la sensibilité aux contraintes dures (temps et espace) est de 0% pour le poids de toute contrainte forte. En effet, l'approche proposée est réalisée avec les avantages suivants (par rapport aux autres travaux récemment publiés) :

1. Aucun individu de la population initiale n'inclut de violation de contraintes dures (51).

2. La disponibilité des salles est considérée comme l'une des fortes contraintes pour que tous les cours soient dispensés dans les salles du département spécifié, contrairement aux travaux proposés par Dammak et al (52)et Aladag et al. (53).

3. Dans le processus de préparation de l'emploi du temps, pas besoin de modification de fonction pour corriger le planning tel qu'il a été réalisé dans le travail de Burke et Newall (50).

2.4 Approche de coloration trans-génétique pour un problème d'horaire

Cette approche a été appliquée par Mina G. Asham, Moataz M. Soliman, et Rabie A. Ramadan (2011), à travers laquelle ils proposent d'utiliser à la fois GC et GA pour résoudre les problèmes d'emplois du temps des cours et des examens (16).

2.4.1 Coloration de graphes et algorithmes génétiques

La coloration de graphes (GC) et les algorithmes génétiques (GA) sont deux méthoddes qui se sont avérées efficaces dans la production de l'emploi du temps des cours. Dans cette section, ils explorent la version modifiée des approches de coloration de graphe et d'algorithmes génétiques utilisées pour l'emploi du temps des cours données dans (54)pour s'adapter aux exigences du planning.

2.4.1.1 Approche de la coloration des graphes

Le problème de coloration de graphe est un problème NP complet classique dans lequel on a un graphe G = (V, E), où V est l'ensemble des sommets du graphe, et E est l'ensemble des

arêtes reliant les sommets du graphe. Le degré d'un sommet est une valeur attribuée à chaque sommet qui représente le nombre d'arêtes qui lui sont connectées. Alors que deux sommets X et Y sont dits triplets si et seulement s'il existe un sommet Z tel que :

- ✓ X est connectée à Z.
- ✓ Y est connecté à Z.
- ✓ X n'est pas connecté à Y.

L'approche a tendance à diviser V en le nombre minimum de groupes, de sorte que :

- ✓ Chaque groupe est coloré avec une couleur distincte.
- ✓ Tous les éléments d'un même groupe sont colorés de la même couleur.
- ✓ Les deux sommets connectés à une arête doivent avoir des couleurs différentes.

Le problème d'automatisation de l'emploi du temps des cours / classes est réduit à un problème de coloration des graphes. Cette réduction est effectuée en représentant chaque classe avec un sommet dans le graphe, et il y a un bord entre deux classes quelconques si et seulement si ces deux classes ont un élève ou un enseignant en commun. Comme le montre la figure 5, l'approche se compose de trois phases qui sont : la coloration des sommets, l'attribution des salles et la planification des classes.

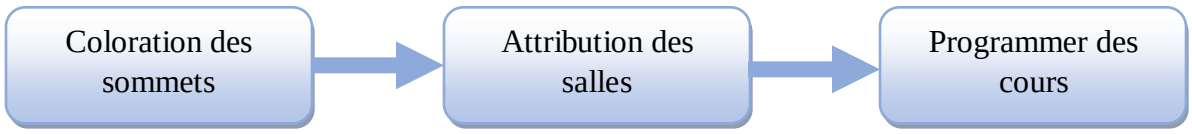

Figure 5 : Phases de coloration du graphe

Dans la phase 1, la coloration des sommets du graphe se fait de manière à ce qu'aucun sommet adjacent n'ait la même couleur. L'algorithme de la figure 6 consiste en une boucle principale qui continue à s'exécuter jusqu'à ce qu'il n'y ait plus de triplets dans le graphe G. Cette boucle principale est l'ensemble d'instructions de 2 à 7.

À l'étape 2, le sommet avec le nombre maximum d'arêtes connectées à lui est sélectionné et nommé comme max, puis l'ensemble des triplets de ce sommet est récupéré pour obtenir le triple de degré maximum de max et le nommez maxTriple. Ces deux sommets sélectionnés sont fusionnés en supprimant maxTriple et en connectant max à tous les sommets connectés à

maxTriple. Enfin, nous devons mettre à jour le degré de max et vérifier la condition de fin de la boucle principale.

1. Let ***G*** = the set of all vertices in the graph.
2. Set ***max*** = the vertex with the maximum degree.
3. Set ***maxTriples*** = the set of all triples of max.
4. Set ***maxTriple*** = the vertex with the maximum degree in ***maxTriples***.
5. Merge ***max*** and ***maxTriple*** by deleting ***maxTriple*** from ***G*** and connecting ***max*** with all vertices connected to ***maxTriple***.
6. Update the degree of ***max***.
7. Go to step 2 until no more triples exist in ***G***.
8. Color each group of merged vertices with the same color.

Figure 6 : Coloration du graphe - phase 1.

1. Let ***Rooms*** = the set of all rooms available.
2. Let ***Groups*** = the set of all colored groups.
3. For each group ***grp*** in ***Groups***:
    a. Copy ***Rooms*** to ***temp***.
    b. Sort ***grp*** in ascending order of capacity.
    c. For each class ***cls*** in ***grp***:
        i. Set the room of ***cls*** to the smallest room ***r*** in which ***cls*** fits.
        ii. Remove ***cls*** from ***grp***.
        iii. Remove ***r*** from ***temp***.
    d. Remove ***grp*** from ***Groups***.

Figure 7 : Coloration du graphe - phase 2.

Dans la phase 2, affectation des salles donnée dans la figure 7, appliquez l'algorithme d'attribution des salles pour attribuer des salles à tous les cours, en attribuant toutes les salles disponibles à chaque groupe de cours colorés avec la même couleur une salle à la fois, jusqu'à ce que tous les cours aient leurs salles attribuées.

A cette phase, il ya un ensemble de groupes, où chaque groupe se compose d'un ensemble de classes non conflictuelles, et chaque classe est attribuée à une salle dans laquelle elle s'intègre le mieux.

En d'autres termes, nous avons satisfait à toutes les contraintes dures du problème. La technique de coloration de graphe n'étant utile que dans la satisfaction des contraintes dures. Cela se fait par un algorithme simple illustré à la figure 8.

```
1. Let Groups = the set of colored groups.
2.For each group grp in Groups:
    a. For each day d in the week:
      i.  For each slot s in the day:
          1. Loop through all students and teachers to
             check all soft constraints if grp is assigned
             to slot s in day d.
          2. If no constraints violated, assign grp to slot
             s in day d.
    b. If all slots in all days violate constraints, assign
       grp to the first empty slot s in day d.
```

Figure 8 : Coloration du graphe - phase 3.

2.4.2 Approche des algorithmes génétiques

Les algorithmes génétiques(GA) sont une méthode de résolution de problèmes d'optimisationbasée sur la recherche locale. Dans un GA, la solution est générée en combinant des solutionsparentes plutôt qu'en modifiant une seule solution. L'AG reçoit un ensemble de solutionsaléatoires à utiliser pour générer de nouvelles solutions. En outre, il produit la meilleuresolution disponible lorsque la condition de fin est remplie. La figure 9 montrel'organigramme des algorithmes génétiques.

Le processus commence par la génération de la population initiale; cela se fait en générant des chromosomes aléatoires à utiliser dans les générations ultérieures. L'algorithme garde une trace de certains des meilleurs chromosomes ayant la meilleure fonction de remise en forme afin de ne pas être supprimés. Si la meilleure aptitude répond aux critères d'arrêt de l'AG, elle prend fin; sinon, il passe par les processus de croisement et de mutation. En raison du caractère aléatoire du

croisement et de la mutation, certains chromosomes peuvent enfreindre les règles du cours; par exemple, un cours peut exister plus d'une fois ou même ne pas exister du tout dans le chromosome (le nombre de fois où le cours est programmé n'est pas égal à 1). Par conséquent, un processus de réparation chromosomique est ajouté à l'AG pour résoudre ces problèmes. Le pseudo code de l'algorithme génétique est illustré à la figure 10.

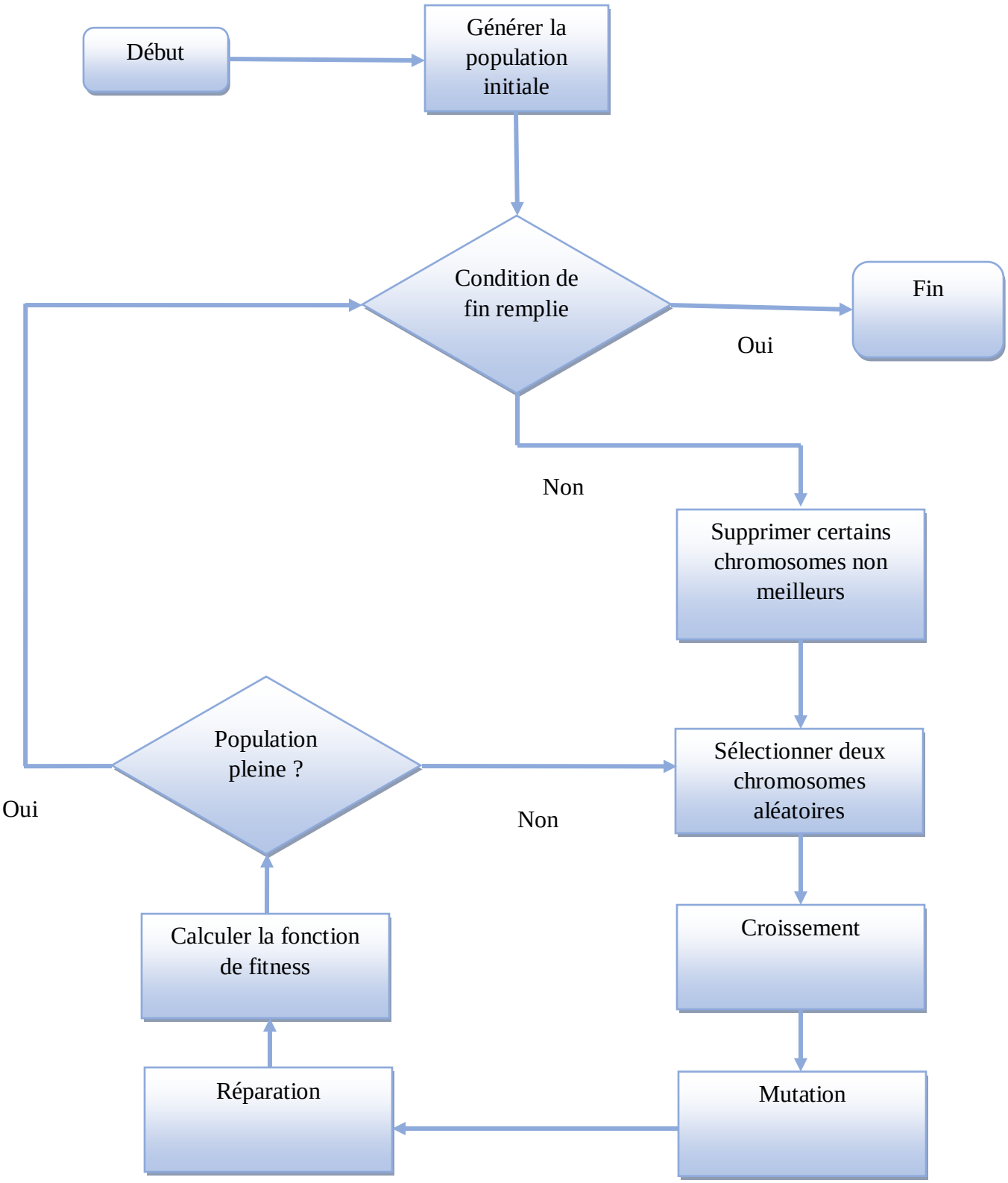

Figure 9 : Concept d'algorithme génétique.

Pour que l'AG fonctionne correctement, trois décisions importantes doivent être prises avec soin. La première décision est la structure du chromosome et la seconde est la fonction de fitness / évaluation tandis que la troisième est le critère d'arrêt. Dans cet approche, la structure du chromosome est définie comme l'emploi du temps d'une classe comme illustré à la figure 11. En d'autres termes, un chromosome représente un empoi du temps de classe / examen valide et efficace. En même temps, puisque l'AG est utilisé pour résoudre l'ensemble du problème, il doit donc satisfaire toutes les contraintes dures, et autant que possible les contraintes souples. Par conséquent, la fonction de fitness, ou fonction de coût, est sélectionnée pour être une fonction composée qui se compose de deux valeurs, une***Fitness dure*** pour le nombre de contraintes dures violées et ***une Fitness souple*** pour le nombre de contraintes souples violées. Ces deux valeurs sont additionnées pour donner la fonction de fitness. Le troisième facteur important est la condition d'arrêt, qui est (hardFitness == 0 &&softFitness<max) où max est la valeur maximale autorisée pour le nombre de contraintes souples à violer. Cette valeur est choisie expérimentalement. D'autres paramètres importants doivent être choisis tels que; le nombre d'itérations, le type de croisement et la probabilité, le pourcentage de mutation, la taille de la population, le nombre de meilleurs chromosomes par itération à conserver et le nombre de nouveaux d'enfants générés par itération.

1. *Insert random chromosomes into the population until it's full.*
2. *While the ending condition is not met, do*
    a. *Remove some of the chromosomes available.*
    b. *Select two chromosomes by random.*
    c. *Cross them over to get a new offspring.*
    d. *Mutate the new offspring.*
    e. *Repair the new offspring.*
    f. *Evaluate the fitness function of the new offspring and add it to the population.*
    g. *Go to "b" until the population is full.*
    h. *Check the resetting condition, if met; replace all non-best chromosomes in the population with new randomly generated ones.*

Figure 10 :pseudo-code d'algorithme génétique.

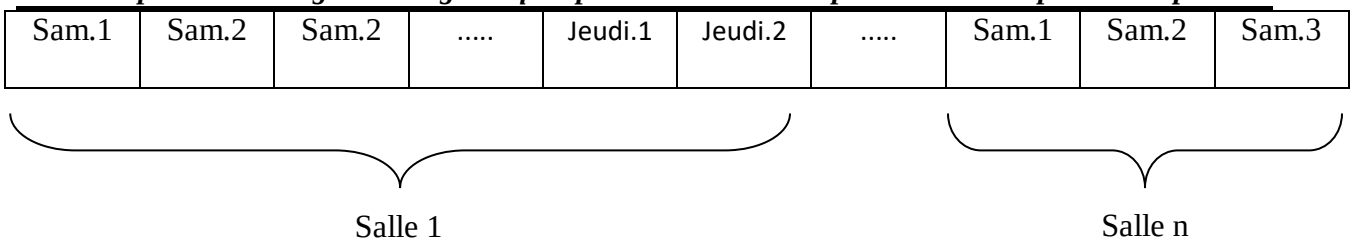

Figure 11 : structure du chromosome de l'algorithme génétique.

2.4.3 Coloration génétique

Dans cette section, nous présentons les travaux qui proposent d'utiliser à la fois GC et GA pour résoudre les problèmes d'emploi du temps des cours et des examens. Sur la base de cette expérience d'utilisation de l'AG et du GC dans la planification du temps, ils pourraient résumer certains des inconvénients des deux algorithmes comme suit :

1) GC utilise un algorithme de force brute pour satisfaire les contraintes souples, ce qui n'est pas une bonne idée, en particulier avec des problèmes de grande taille.
2) GC affecte la classe / le cours au premier emplacement vide si tous les emplacements violent une ou plusieurs des contraintes souples.
3) En GA, la longueur du chromosome augmente avec l'augmentation de la taille du problème. Par exemple, dans notre cas, la longueur du chromosome pourrait être de 30 * n (6 jours * 5 créneaux), où n est le nombre de salles. Supposons que n = 100, par conséquent, la longueur du chromosome sera de 3000 gènes, ce qui est très grand. Certes, cela affecte le traitement des chromosomes et le temps de fonctionnement global de l'AG.
4) GA doit attendre que la valeur Fitness dure converge vers zéro, ce qui peut prendre beaucoup de temps.

Pour résoudre ces problèmes, ces travaux proposent d'utiliser GC pour satisfaire les contraintes dures et GA pour satisfaire les contraintes souples. Ils estiment qu'une telle combinaison améliore l'efficacité de la (des) solution (s) de programmation et réduira le temps de fonctionnement dans la plupart des cas. Par conséquent, aucune force brute n'est requise pour la satisfaction des contraintes souples. En outre, la coloration génétique attribue la classe à l'emplacement qui minimise le nombre de contraintes souples violées. De plus, la longueur du chromosome sera réduite à une valeur constante (30 * 6 jours * 5 créneaux). Néanmoins, la coloration génétique ne vérifie même pas la valeur de Fitness dure car elle est sûrement zéro.

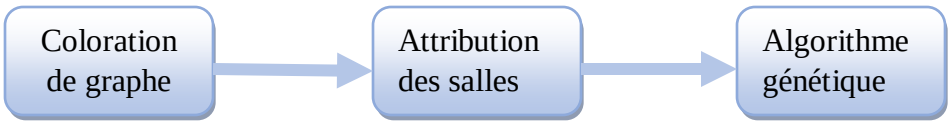

Figure 12 : Phases de coloration génétique.

De même, l'algorithme de coloration génétique se compose de trois phases comme le montre la figure 12. Le problème est d'abord réduit en un problème de coloration de graphe et résolu en conséquence. Ceci est similaire à la première phase de la solution de coloration de graphique expliquée dans la section 2.4.1.1. La deuxième phase est l'affectation de salles qui est également similaire à la deuxième phase de l'approche de coloration de graphe expliquée dans la section 2.4.1.1. Le pseudo-code pour les deux phases est représenté sur les figures 6 et 7 respectivement. Ici, c'est le rôle de l'AG de travailler sur les contraintes souples. La sortie de la phase 2 est un ensemble de groupes colorés. L'objectif de l'algorithme génétique est de trouver le meilleur moyen d'attribuer ces groupes colorés aux créneaux horaires en minimisant le nombre de contraintes souples violées. Le pseudocode de la phase 3, GA, est donné sur la figure 13. Dans cette phase, le même processus GA, expliqué précédemment, aura lieu. Le processus comprend le croisement, la mutation, la procédure de réparation et l'évaluation de la fonction de fitness. Cependant, le processus GA diffère cette fois de celui indiqué précédemment en raison du changement de certains paramètres.

```
1. Insert random chromosomes into the population until it's full.
2. While the ending condition is not met, do
        a. Remove some of the chromosomes available.
        b. Select two chromosomes by random.
        c. Cross them over to get a new offspring.
        d. Mutate the new offspring.
        e. Repair the new offspring.
        f. Evaluate the fitness function of the new offspring and add it to
           the population.
        g. Go to "b" until the population is full.
        h. Check the resetting condition, if met; replace all non-best
           chromosomes in the population with new randomly generated ones.
```

Figure 13 : Coloration génétique phase 3

Sam.1	Sam.2	Sam.3	Sam.4	.....	Jeudi.1	Jeudi.2	Jeudi.3	Jeudi.4

Figure 14 : Structure du chromosome de coloration génétique.

La nouvelle structure du chromosome est réduite d'une manière que le nombre de jours multiplié par le nombre d'emplacements par jour comme le montre la figure 14. Concernent la fonctionne de fitness, puisque l'algorithme génétique sera utilisé pour satisfaire uniquement les contraintes souples, il n'est pas nécessaire d'avoir un composé des fonctions. Par conséquent, la fonction de fitness sera uniquement le nombre de contraintes souples violées. De plus, le critère d'arrêt est modifié pour être basé uniquement sur la condition de Fitness souple <max, où max est la valeur maximale autorisée pour le nombre de contraintes souples à violer.

2.4.4 Simulations et résultats

Dans les résultats de cette approche, il ya une comparaison entre les performances des algorithmes hybrides de coloration génétique et graphique par rapport à l'AG et au GC en tant que solutions autonomes aux problèmes de l'emploi du temps. Ils ont mis en œuvre les trois algorithmes à cet effet et ont conçu certains des cas de test pour mesurer leurs performances. Toutes les expériences réalisées dans cette approche ont été implémentées sur un processeur Intel Quad Core 2,83 GHz, avec 12 Mo de cache, et Windows 7 32 bits avec 4 Go de RAM. Cette implémentation a utilisé C # dot Net sur dot Net framework 4 sous Visual Studio 2010. De plus, pour avoir des comparaisons équitables entre les algorithmes proposés, le test de l'algorithme génétique est fait dans de nombreux cas de test avant de trouver des meilleurs résultats tels que :

- ✓ Pourcentage de mutation égal à 1%.
- ✓ Nombre maximum d'itérations égal à 1000.
- ✓ Nombre de chromosomes dans la population est égal à 30.

2.4.4.1 Cas de test 1(au niveau du département)

Ce cas de test considère un problème de petite taille. Ce périmètre ne prend en compte que 250 étudiants, 20 professeurs, 70 cours et 20 salles. En fait, c'est la taille réelle de leur département informatique. En termes de temps d'exécution moyen, la figure 15 montre que l'algorithme de coloration de graphe prend le moins de temps d'exécution tandis que l'algorithme génétique prend près de 46 secondes pour atteindre une solution appropriée qui est la pire parmi les trois algorithmes. Dans le même temps, le temps de coloration génétique n'est pas considéré comme mauvais par rapport à l'algorithme génétique mais il est équivalent à l'algorithme de coloration du graphe. Cependant, la figure 16 montre un autre point de vue en termes de

performances des algorithmes. Par exemple, comme on peut le voir, cela ne signifie pas qu'en raison du fait que l'algorithme de coloration de graphe prend le moins de temps, il produit le meilleur emploi du temps en termes de performances. En fait, l'approche de coloration génétique donne les meilleures performances tandis que l'algorithme de coloration de graphe viole la plupart des contraintes souples. De plus, la coloration génétique est quelque chose d'intermédiaire en termes de performances par rapport aux autres algorithmes, tandis que son temps d'exécution est très petit par rapport au temps d'exécution de l'algorithme génétique.

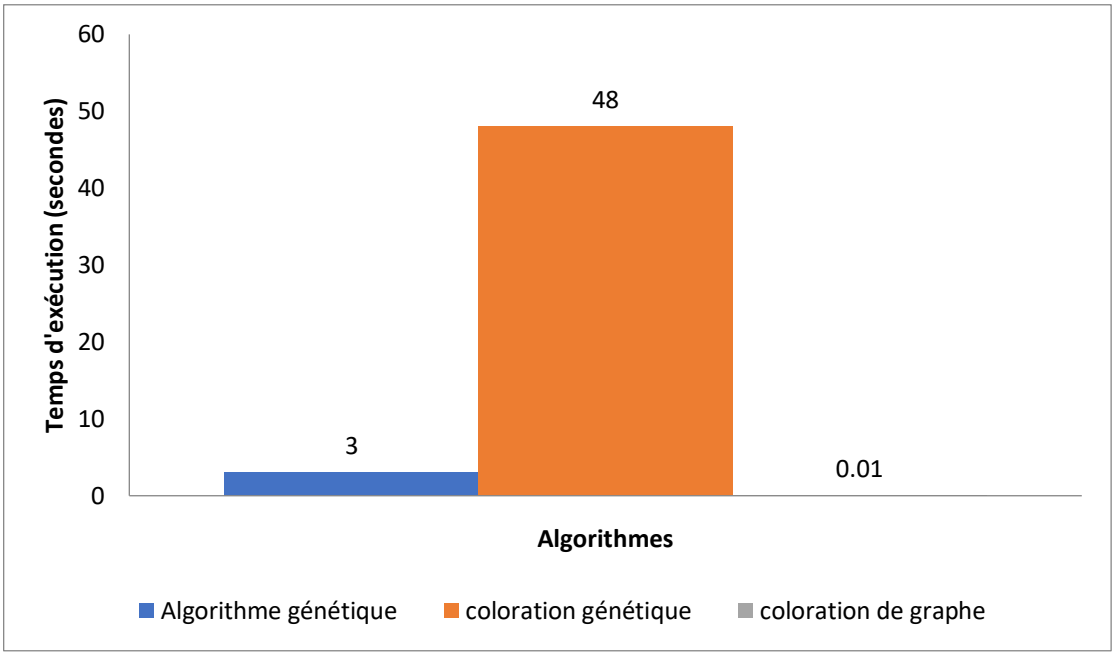

Figure 15 :Durée d'exécution des trois algorithmes au niveau du département.

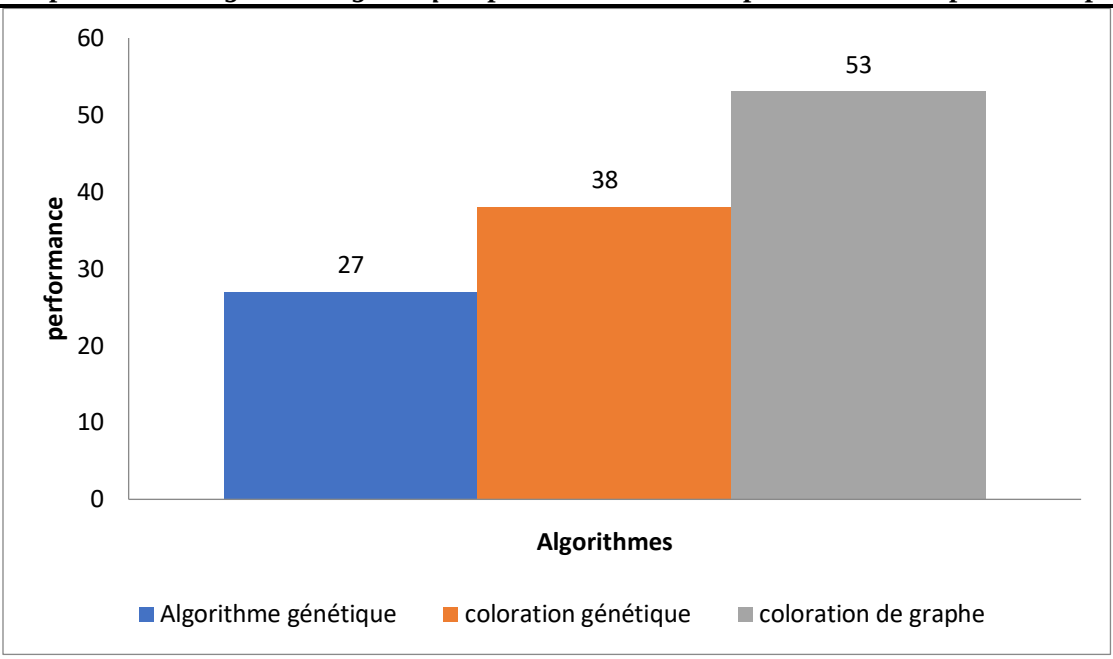

Figure 16 : Comparaison de la fonction de fitness des trois algorithmes au niveau du dépprtement.

2.4.4.2 Cas de test 2 (au niveau de la faculté)

Dans ce cas, le test des performances des trois algorithmes est fait avec des problèmes de taille moyenne. La portée du corps professoral utilisée dans cette section est d'environ 500 étudiants, 40 professeurs, 40 aides à l'enseignement, 140 cours et 40 salles. En termes de temps d'exécution, les résultats sont illustrés à la figure 17 et ils sont similaires aux résultats au niveau du département où l'algorithme génétique est l'algorithme le plus chronophage. La coloration graphique est toujours meilleure que la coloration génétique près de la moitié du temps. Cependant, la coloration des graphes et la coloration génétique sont bien meilleures que l'algorithme génétique de près de 14 fois. D'autre part, en termes de fitness des algorithmes, la coloration génétique, comme le montre la figure 18, se situe en quelque sorte entre la coloration génétique et la coloration de graphes. C'est donc un compromis entre le temps et la fitness.

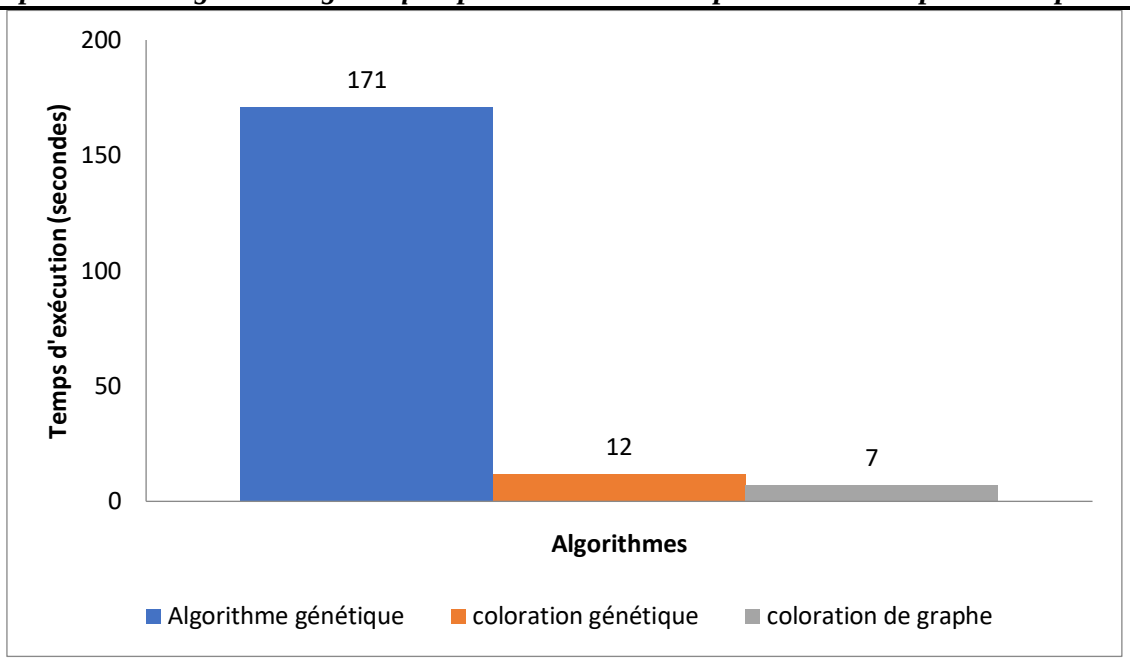

Figure 17 : Durée d'exécution des trois algorithmes en termes de portée de la faculté.

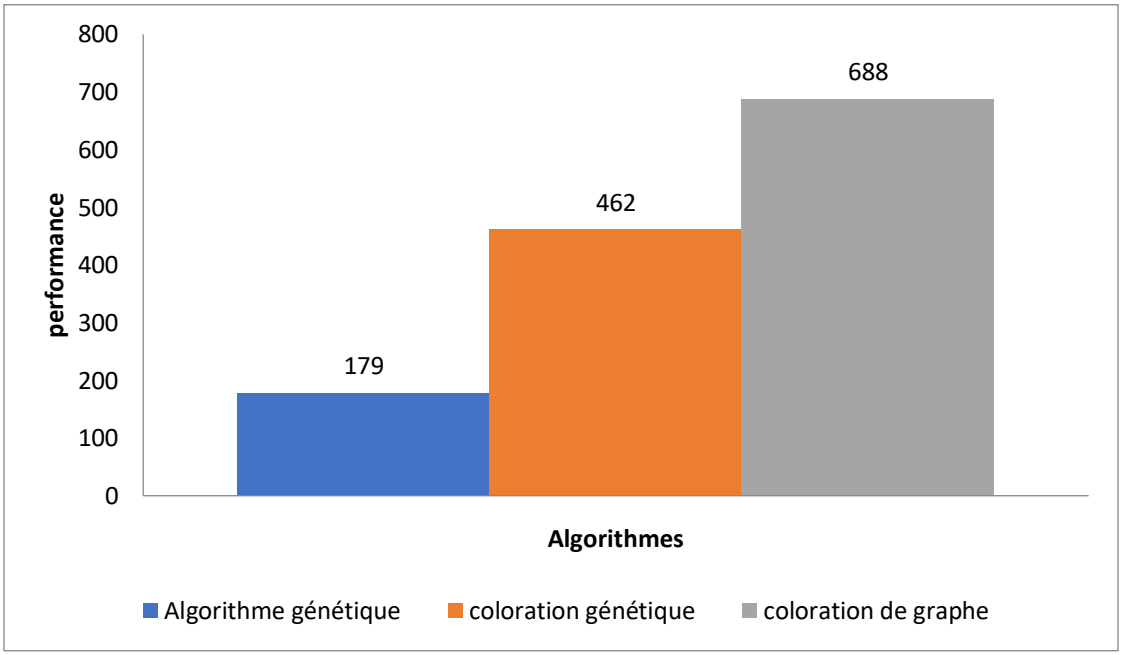

Figure 18 : Comparaison de fitness entre les trois algorithmes en termes de portée de lafaculté

2.4.4.3 Cas de test 3 (au niveau de l'université)

Un autre cas de test est examiné dans cette section pour une portée au niveau de l'université dans laquelle des problèmes avec 750 étudiants, 60 professeurs, 60 aides à l'enseignement, 210 cours et 60 salles. En termes de temps d'exécution, voir la figure 19, il semble que le temps de fonctionnement soit doublé par rapport aux problèmes au niveau de la faculté. Par exemple, l'algorithme génétique prend près de 7 heures pour produire une solution.

Cependant, l'algorithme génétique s'adapte toujours à la fois aux algorithmes de coloration de graphes et de coloration génétique en termes de performances, comme le montre la figure 20. Dans le même temps, la coloration génétique est bien meilleure que l'algorithme de coloration de graphesue.

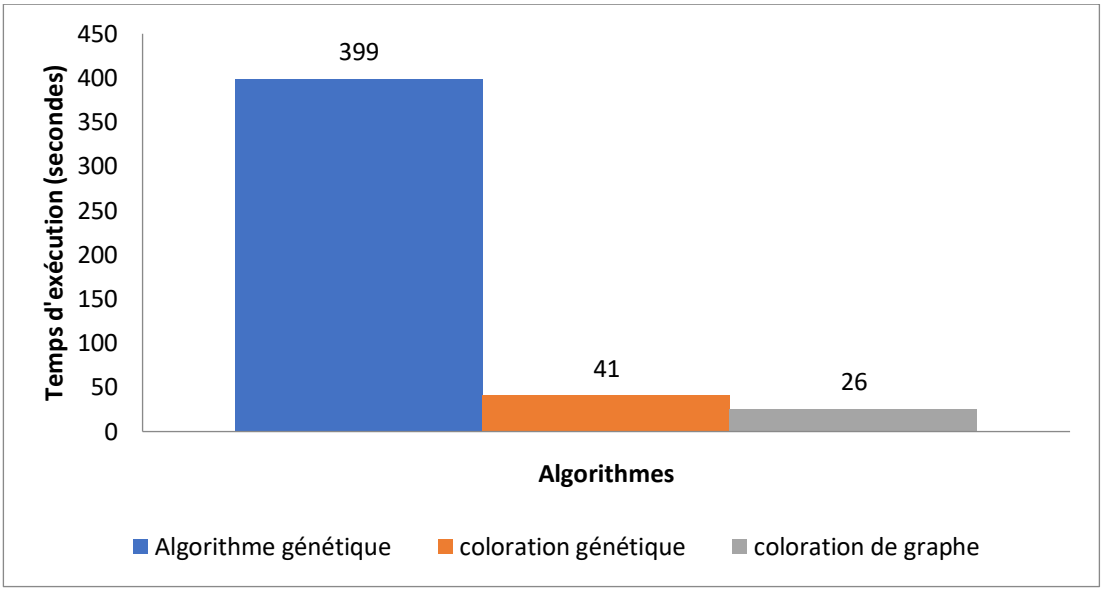

Figure 19 : Durée de fonctionnement des trois algorithmes en termes de portée universitaire.

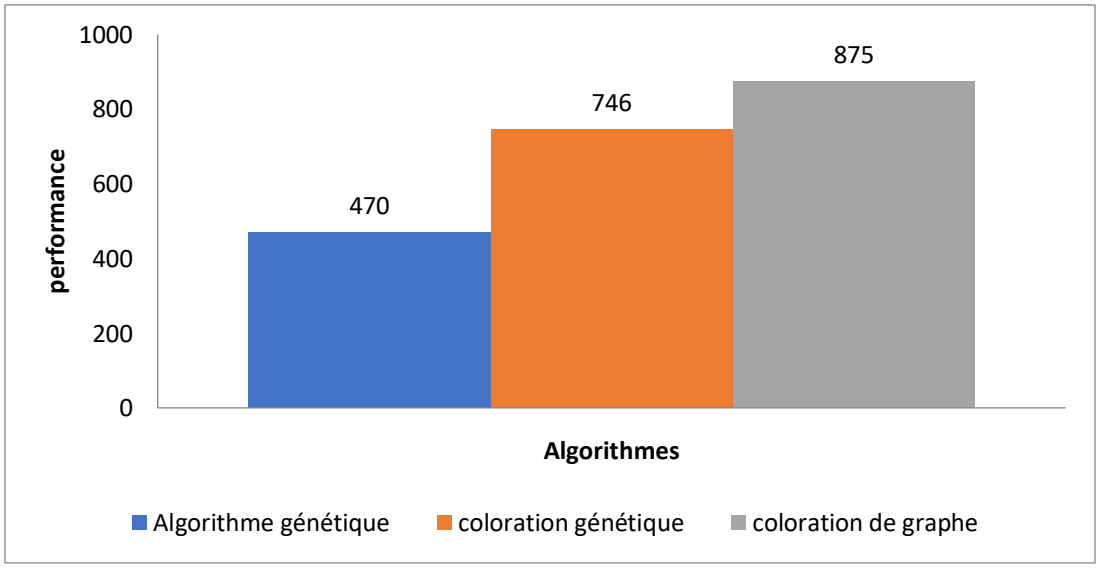

Figure 20 : Comparaison de fitness entre les trois algorithmes en termes de portée de la faculté.

2.5 Sur l'amélioration de l'efficacité de la planification automatique des emplois du temps universitaires avec un algorithme génétique appliqué

Khonggamnerd&Innet, 2009 proposent un modèle d'algorithme génétique pour améliorer l'efficacité de l'agencement automatique des emplois du temps universitaires. Le modèle est développé à partir des travaux précédents présentés dans (24). Des contraintes plus strictes sur la limitation de l'espace sont prises en compte.

2.5.1 La conception élémentaire et la description de l'algorithme

Les chromosomes horaires universitaires sont construits en un groupe d'éléments (E). Selon cette approche, les éléments comprennent trois types de données : événement (Ev), enseignant (En) et groupe (G). Un ensemble de E représentera comme éléments constitutifs des chromosomes. Ainsi, E = {Ev, En, G} représentera les éléments du chromosome. Chacun de Ev, En et G a un sous-ensemble.

2.5.2 La fonction de fitness

La fonction de fitness pour résoudre les problèmes spécifiés dans cette étude est définie par L'équation (3) suivante :

$$f(gen) = \sum_{i=1}^{4} \mathrm{w}_{\mathrm{i}}^{\mathrm{hard}} \times \mathrm{cost}_{\mathrm{i}}^{\mathrm{hard}} + \sum_{i=1}^{4} \mathrm{w}_{\mathrm{i}}^{\mathrm{soft}} \times \mathrm{cost}_{\mathrm{i}}^{\mathrm{soft}} \quad (3)$$

Cette fonction est une somme pondérée entre les poids de contraintes dures ($\mathrm{w}_{\mathrm{i}}^{\mathrm{hard}}$) Multipliés par le cout de ces contraintes ($\mathrm{cost}_{\mathrm{i}}^{\mathrm{hard}}$) et les poids de contraintes souples ($\mathrm{w}_{\mathrm{i}}^{\mathrm{soft}}$) multipliés par le cout des contrainte souples ($\mathrm{cost}_{\mathrm{i}}^{\mathrm{soft}}$).

2.5.3 Opérateur génétique

2.5.3.1 Croisement

Dans cette étape un échange des éléments de deux chromosomes parents est fait.

2.5.3.2 Mutation

Le processus de mutation de chromosomes a été effectué en modifiant les éléments du chromosome pour créer un nouveau chromosome.

2.5.3.3 Sélection

La sélection est la méthode pour choisir le chromosome requis dans le pool de croisement. Les chromosomes sélectionnés seront les chromosomes parents de la prochaine génération.

2.5.4 Simulation et résultats

Les spécifications de ce problème sont présentées dans le Tableau 4.

Tableau 4 : Spécifications du problème

N°	Description	Quantité
1	Nombre de groupes d'étudiants	28
2	Nombre de salles	4
3	Nombre de laboratoires	2
4	Nombre de jours d'enseignement dans une semaine	5
5	Nombre de périodes dans une journée	6
6	Nombre d'événements	55
7	Nombre d'enseignants	23

Le tableau 5 montre la valeur utilisée du poids pour chaque contrainte pour obtenir les résultats simulés.

Tableau 5 : La valeur du poids pour chaque contrainte.

poids	La valeur
w_i^{hard} (i = 1, 2, 3 ou 4)	10000
w_i^{soft} (i = 1, 2, 3 ou 4)	1

Un effet de la probabilité de croisement sur la valeur de fitness a été étudié.Les valeurs de fitness ont été déterminées lorsqu'il était stable.Le résultat de cette étude a été présenté dans le tableau 6.

Tableau 6 :Effet de la probabilité de croisement sur la valeur de fitness.

Probabilité de croisement	Nombre de générations	Valeur de fitness
0.00	438	110012
1.00	163	110012
0.30	316	50016
0.40	368	90020
0.50	494	60012
0.60	442	70014
0.65	477	40017
0.70	445	34
0.75	478	30026

D'après les résultats de ce tableau on peut voir que la valeur de fitness minimale se produit lorsque la probabilité de croisement est comprise entre 0.65 et 0.75 et la valeur minimale de fitness se produit lorsque la probabilité de croisement est égale 0.70.

La figure 21 présente l'évolution de la valeur de fitness lors de la variation de la probabilité de croisement à 0.65, 0.70 et 0.75.

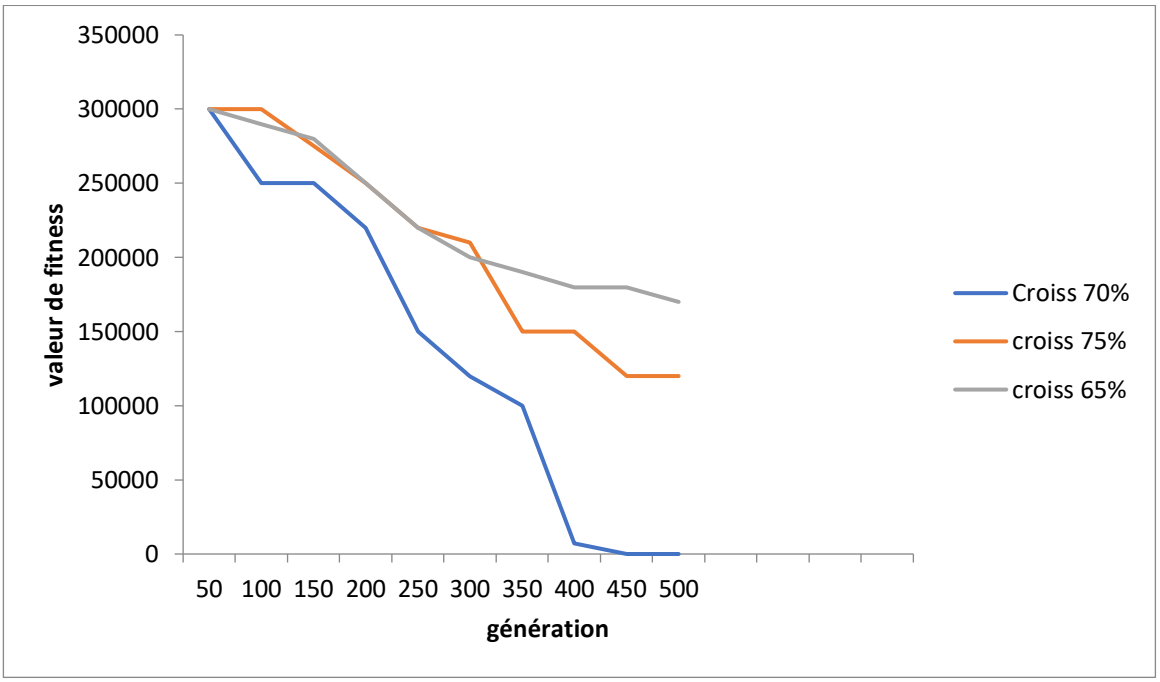

Figure 21 : Evolution de la valeur de fitness en fonction la variation de la probabilité de croisement à 0.65, 0.70 et 0.75.

Nous notons que lorsque le nombre de générations augmente, la valeur de fitness diminue.De plus, le graphique montre que la probabilité de croisement de 0.70 fournit le meilleur résultat dans ce problème spécifique. Par conséquent, la simulation a été répétée avec une probabilité de croisement de 0.70 pour faire la moyenne de la valeur de fitness.Le résultat est illustré dans le tableau 7.

Tableau 7 : Nombre moyen de contrainte survenue à 0.70 taux de croisement.

N°	génération	Valeur de fitness	Contraintes dures moyennes	Contraintes souples moyennes
1	445	34		
2	438	28		
3	441	26		
4	458	21		

5	462	19	0	**25.1**
6	468	17		
7	444	22		
8	459	29		
9	455	28		
10	442	27		

2.6 Conclusion

A travers les résultats présentés, nous concluons que les algorithmes génétiques sont les plus adaptés pour résoudre le problème d'ordonnancement de type emploi du temps d'un département à l'université. Nous choisirons l'algorithme proposé par Khonggamnerd&Innet, 2009 car il correspond à notre besoin.

Dans le chapitre suivant nous décrirons l'algorithme choisi en détails.

.

Chapitre 3 :Description de l'algorithme choisi

3.1 Introduction

Après avoir passer en revue, en chapitre 2, quelques algorithmes génétique pour résoudre le problème de l'emploi du temps, nous avons sélectionné l'un de ces algorithmes que nous allons implémenter. cet algorithme est proposé par Khonggamnerd&Innet, 2009.

Dans ce chapitre on commence par présenter le problème puis on décrira les étapes de l'algorithme choisi.

3.2 Présentation du problème

Le problème de l'emploi du temps est un problème NP-difficile et pour y trouver des solutions, il faut définir des contraintes dures et souples (55), dans notre cas on a défini ses contraintes comme suit :

3.2.1 Contraintes dures

- ✓ Un enseignant ne peut pas enseigner des matières différentes au cours de la même période d'une journée.
- ✓ Une salle ne peut être occupée par différents cours dans la même période d'une journée.
- ✓ Un étudiant ne doit pas étudier plus d'une matière dans la même période de la journée.
- ✓ Un groupe est sa section ne sont pas autorisés à étudier une matière au même temps.

3.2.2 Contraintes souples

- ✓ Il ne devrait pas y avoir plus de deux périodes libres entre les séances de cours voisines pendant la journée d'étude.
- ✓ La période 3 ou 4 devrait être libre pour l'heure du déjeuner chaque jour.
- ✓ Il ne devrait pas y avoir plus de 4 périodes d'études continues par jour.
- ✓ Il ne devrait pas y avoir plus de 4 périodes continues occupées pour les enseignants par jour.

3.3 Description de l'algorithme génétique

3.3.1 Développement de l'algorithme génétique

Les étapes de l'algorithme génétique développé sont :

1) Création d'une solution initiale (population initiale) présentée par des chromosomes.
2) Définir la fonction de fitness et calcul de la valeur de fitness de chaque chromosome.
3) Sélection de deux chromosomes parents P1 et P2 qui possèdent la plus grande valeur de fitness.
4) Faire un croisement des parents choisis pour créer une nouvelle génération (enfant).
5) Faire une mutation des gènes dans le chromosome enfant créé qui possède la meilleure valeur de fitness.
6) Continue le processus jusqu'à ce que la valeur de fitness soit satisfaite ou que l'utilisateur mette fin au processus.

3.3.2 La présentation du chromosome de l'AG

Les chromosomes appliqués par l'AG sont construit en un groupe d'éléments (E) (56), Les éléments comprennent trois types de données, qui sont enseignant (L), matière (M) et Salle (S). Chaque élément peut être présenté sous la forme : E= {L, M, S}.Un ensemble de E représentera comme éléments constitutifs des chromosomes.Ainsi, E = {E1,…, En} représentera comme éléments d'entrée de l'emploi de temps tandis que chacun de E (1-n) comprendra un ensemble conséquent de L, M et S.Chacun d'eux a un sous-ensemble de chacun. Il peut être représenté par L = {L1, L2, L3,…, Ln_1}, qui représentera les enseignants de ce semestre.De la même manière qu'un ensemble de matières et un ensemble de salles dans ce semestre seront comme : M = {M1, M2, M3,…, Mn_2} et S = {S1, S2, S3,…, Sn_3}, respectivement.

Un chromosome sera enchaîné suivant les séquences de périodes qui donnent la longueur de ce chromosome comme indiqué dans l'équation (4).

$$lChrom = G * P(n) \qquad (4)$$

Tels que :

lChrom :La longueur ou le nombre de bits de chaque chromosome,

G :Nombre de Groupe et de section d'étudiants.

P(n) : Nombre de périodes dans l'emploi du temps.

groupes	Groupe1				Groupe2				
indices	0	1	....	29	30	31	....	....	59
périodes	1	2	....	30	31	32	....	....	60
Chrom	E1	E3		E10	E15	E11			E17

Figure 22 : Présentation du chromosome

3.4 La fonction de fitness

La fonction fitness est utilisée pour mesurer la pertinence de l'emploi du temps et choisir le meilleur. La Fonction de fitness est construite à partir des contraintes de l'emploi du temps. La fonction de fitness que nous avons utiliséeest donnée par l'équation (5).

$$\frac{1}{\sum \text{contrainte } dure + \sum \text{contrainte } souple + 1} (5)$$

3.5 Opérateurs génétiques

Une application AG est un processus pour améliorer et développer les chromosomes des meilleures solutions en sélectionnant le chromosome qui a la meilleure valeur fitness.Ensuite, un tel chromosome subi le processus génétique, qui comprend deux opérateurs génétiques : le croisement et la mutation.Le processus de croisement commence par deux chromosomes initiaux appelés parents et produit deux enfants. Les données de L, M et S ont été intégrées au niveau de l'élément (l'évenement) de l'emploi du temps. Le processus de croisement et de mutation dans l'emploi du temps sera exécuté à l'intérieur de chaque chaîne de chromosomes, occupée par des groupes d'étudiants. Il peut être montré dans la figure suivante.

groupes	Groupe1				Groupe2				
indices	0	1	....	29	30	31	....	....	59
périodes	1	2	....	30	31	32	....	....	60
Chrom1	E1	E3		E10	E15	E11			E17
	Opérateur génétique				Opérateur génétique				
Chrom2		E7	E7		E2		E11		

Figure 23 : Groupement de chromosomes pour le processus génétique

3.5.1 Croisement

Le processus de croisement est un échange d'éléments de deux parents chromosomes. Notre méthode de croisement proposée estexpliquée comme suit :

1. Dans le premier chromosome (parent1),choisir une position aléatoirement pour exécuter le croisement.
2. On prend les éléments sélectionnés dans le premier chromosome et les mettre directement dans le nouveau chromosome (enfant 1).
3. Dans une liste, on sauvegarde les indices du premier chromosome qui ne sont pas sélectionné.
4. On examine les indices de la liste avec les indices du deuxième chromosome (parent2). Si l'élément de l'indice dans la liste est différent par rapport al'élément de l'indice dans le deuxième chromosome on l'ajoute dans le nouveau chromosome et on le supprime de la liste. Une fois la liste est vide on s'arrête. La figure 24 montre un exemple du processus de croisement.

indices	0	1	2	3	...	29	30	31	32	33	34	...	n
périodes	1	2	3	4	...	30	31	32	33	34	35		n
Parent1	E1	E3		E8		E10	E15	E11	E5		E4		En

indices	0	1	2	3	...	29	30	31	32	33	34	...	n
Enfant1	E1					E10		E11					

Figure 24 : Sélection des éléments aléatoirement.

Liste = {1, 2, 3, 30, 32, 33, 34}

indices	0	1	2	3	...	29	30	31	32	33	34	...	n
périodes	1	2	3	4	...	30	31	32	33	34	35		n
Parent2	E1	E3	E6			E10	E14	E11		E15			En

indices	0	1	2	3	...	29	30	31	32	33	34	...	n
Enfant1	E1	E1	E3	E6		E10	E14	E11		E15			

Figure 25 : Les éléments sélectionnés à partir du deuxième chromosome.

3.5.2 Mutation

Le processus de mutation a été faiten modifiant les éléments au hasard dans le chromosome enfant. Cela peut êtreexpliqué en détails comme suit :

1. On sélectionne deux positions P1 et P2 aléatoirement dans le nouveau chromosome pour exécute le processus de mutation.
2. On fait un changement d'élément de deux positions sélectionnées.
   La figur26 montre le processus de mutation.

p1 p2

indices	0	1	2	3	...	29	30	31	32	33	34	...	n
Enfant1	E1	E1	E3	E6		E10	E14	E11		E15			

indices	0	1	2	3	...	29	30	31	32	33	34	...	n
Enfant1	E1	E1	E15	E6		E10	E14	E11		E3			

Figure 26 : Exemple du processus de mutation.

3.5.3 Sélection

La sélection est la méthode pour choisirles chromosome souhaités dans le pool de croisement. Les chromosomes sélectionnés seront les chromosomes parents de la prochaine génération. Deux chromosomes qui ont la meilleure (ou la plus grande) valeur de fitness sont choisis.

3.6 Conclusion

Dans ce chapitre, nous avons présenté les concepts de base de l'algorithme génétique proposé et nous avons expliqué les différents processus de cet algorithme que nous allons implémenté.

Chapitre 4 :Implémentation et résultats

4.1 Introduction

Après avoir présenté les étapes de l'algorithme génétique proposé, nous avons fini par la phase d'implémentation de notre application.

Dans ce chapitre, nous allons d'abord présenter l'environnement de développement.

Ensuite, nous décrierons les principales interfaces graphiques de l'application réalisée et nous finirons par une expérimentation et analyse des résultats obtenus.

4.2 Environnement de développement

Pour développer notre application, nous avons utilisé le langage de programmation Java car c'est un langage orienté objet parmi les langages les plus utilisés dans le domaine de l'intelligence artificielle et plus adapté pour l'implémentation des algorithmes génétiques.Ce langage permet aussi la création d'interfaces graphiques grâce à sa bibliothèque JavaFX. C'est aussi parce que c'est un langage dont le code exécutable est portable. L'environnement de développement et d'exécution est le JDK 1.8.

Nous avons utilisé NetBeans 8.0 comme éditeur du code source.

4.2.1 Le langage java

Java est un langage de programmation orienté objet inspiré du langage C++, qui a été créé par Sun Microsystems en 1995 (57).Il permet de créer des programmes qui peuvent être exécutés dans un navigateur web, développer des applications côté serveur, combiner des applications ou des services basés sur le langage Java et écrire des applications efficaces pour les téléphones portables. Java est disponible dans tous les systèmes d'exploitation tels que Windows, Mac, Linux et dans les téléphones portables sous Android. Cette multiplicité de support est avantageuse, car cela permet aux développeurs de créer un programme et de le faire fonctionner sur plusieurs plateformes sans devoir recréer un nouveau programme (58).

4.2.2 La plate-forme Java FX

JavaFX est créée pour fournir aux développeurs Java une nouvelle plate-forme graphique légère et haute performance.JavaFX permet aux développeurs d'intégrer des graphiques vectoriels, des animations, du son et des vidéos récupérés sur Internet dans une application riche,

dense et interactive,aussi elle étend la technologie Java grâce à l'utilisation de n'importe quelle bibliothèque Java au sein d'une application JavaFX et améliore l'efficacité du travail entre concepteurs et développeurs, les premiers pouvant utiliser les outils de leur choix tout en collaborant avec les seconds.JavaFX peut être utilisée pour créer des interfaces utilisateur graphiques pour n'importe quelle plate-forme (par exemple, bureau, web, mobile, etc.) (59).

Il a été annoncé pour la première fois par Sun lors de JavaONE 2007. La version 1.0 a été publiée en décembre 2008, ciblée sur la plate-forme de bureau, tandis que la version 1.1 a été publiée récemment pour les appareils mobiles (60).

4.2.3 Java Dévelopment Kit(JDK)

Le Java Development Kit (JDK) est l'un des trois principaux packages technologiques utilisés dans la programmation Java, avec la JVM (Java Virtual Machine) et le JRE (Java RuntimeEnvironment) (61).

C'est un progiciel contenant une variété d'outils et d'utilitaires permettant de développer, d'empaqueter, de contrôler et de déployer des applications conçues pour toute plate-forme Java standard, notamment Java Platform Standard Edition (Java SE), Plate-forme Java Micro Edition (JavaME) ; et Java Platform Enterprise Edition (Java EE) (62).

Les principaux outils du JDK sont :

- ✓ java : lechargeurd'application Java.
- ✓ javac : le compilateur, qui convertit le code source enfichiercontenant lebytecode Java.
- ✓ javadoc : le générateur de documentation, qui génère automatiquement de la documentation à partir des commentaires du code source.
- ✓ jar : l'archiveur, qui met sous forme d'un paquetage unique l'ensemble desfichiers classen unfichier JAR.
- ✓ jdb : le débogueur.

4.2.4 NetBeans

NetBeans IDE est un environnement de développement intégré (IDE) gratuit et open source qui permet de développer des applications de bureau, mobiles et Web. L'IDE prend en charge le développement d'applications dans divers langages, notamment Java, HTML5, PHP et C++. L'IDE fournit une prise en charge intégrée pendant tout le cycle de développement, de la création

du projet au débogage, au profilage et au déploiement. L'IDE fonctionne sous Windows, Linux, Mac OS X et d'autres systèmes UNIX (63).

4.3 Les principales interfaces graphiques de l'application

4.3.1 La fenêtre principale

La fenêtre principale de notre application est illustrée dans la figure 27et qui contient une barre d'outils permettent à l'utilisateur d'accéder aux fonctionnalités principales de l'application.

Figure 27 :la fenêtre principale.

4.3.1.1 Barre d'outils

La barre d'outils de notre application est illustrée dans la figure 28. Elle est composée de six boutons (raccourcis) présentés comme suit :

Figure 28 :La barre d'outils de l'application « EPDT informatique ».

- ✓ Le premier bouton ( ) permet à l'utilisateur de créer un nouveau EPDT (Emploi De Temps).
- ✓ Le deuxième bouton ( )permet à l'utilisateur de charger un EPDT (fichier) existant et qui a été sauvegardé auparavant.
- ✓ Le troisième bouton ( )permet à l'utilisateur de sauvegarderl'EPDT en cours d'utilisation.

- ✓ Le quatrième bouton ( ) permet à l'utilisateur d'éditer les paramètres de l'EPDT.
- ✓ Le cinquième bouton ( ) permet d'arrêter l'exécution (La recherche d'un meilleur emploi du temps) à un moment donné, avec la possibilité de la reprendre plus tard.
- ✓ Le dernier ( ) bouton permet de démarrer l'exécution.

4.3.2 La fenêtre « Données de base »

Cette fenêtre (**figure 29**) sert à saisir les données de base de l'EPDT : La liste des matières, des enseignants (Professeurs), des groupes, des sections, des salles et affecter les enseignants aux groupes (et sections).

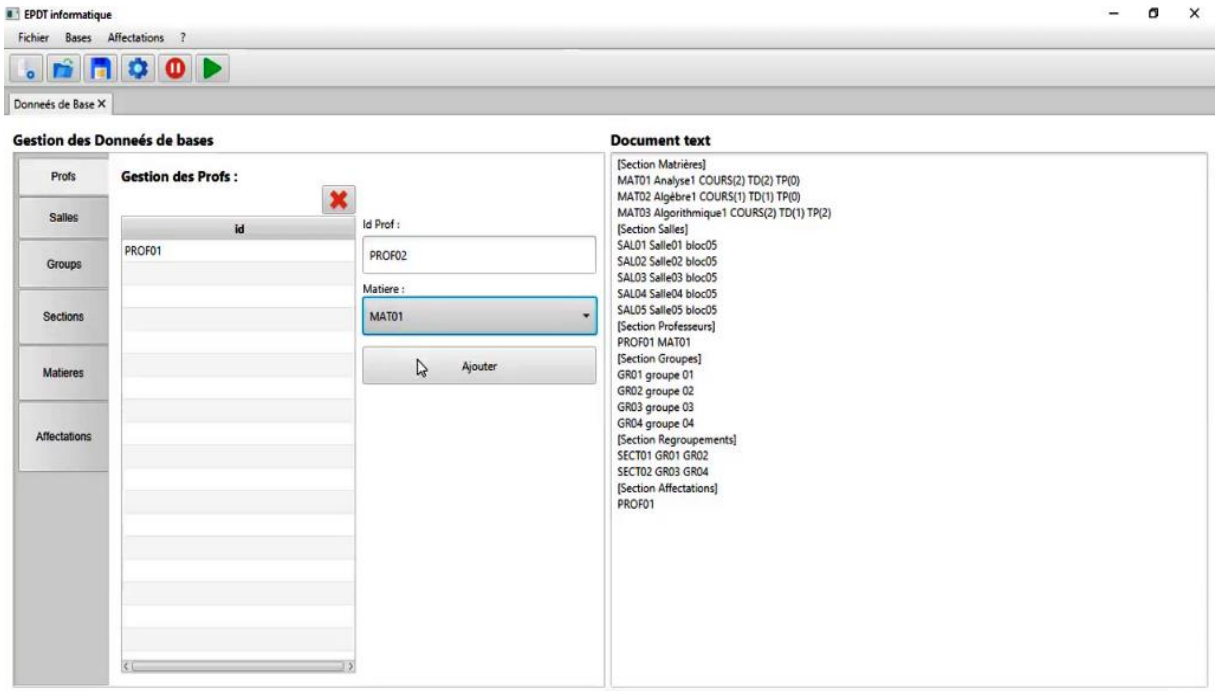

Figure 29 : Fenêtre « Données de base ».

Dans cette fenêtre l'utilisateur peut ajouter, supprimer, ou modifier les données de base de l'EPDT. Les données sont sauvegardées dans un fichier texte.

4.3.3 Paramètres de l'algorithme génétique

Cette fenêtre(**Figure 30**) sert au paramétrage de l'algorithme génétique. Elle permet également à l'utilisateur de définir les paramètres de l'algorithme génétique pour l'exécution.

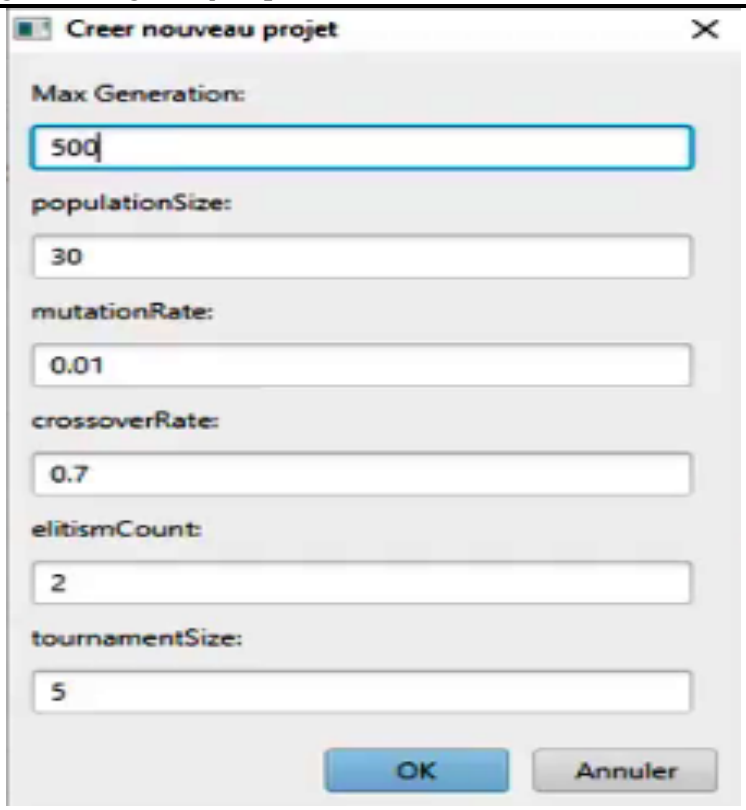

Figure 30 :Fenêtre des paramètres.

- ✓ **Maxgeneration** : Le nombre maximum de générations de l'algorithme génétique.
- ✓ **populationSize** : La taille de la population.
- ✓ **mutationRate :**Le taux (probabilité) de mutation : désigne la quantité de changement de l'information génétique au cours du temps.
- ✓ **crossoverRate** : La probabilité à utiliser pour le croisement afin de créer un nouveau chromosome (Enfant).
- ✓ **elitismCount :**Le nombre d'individus choisis pour la prochaine génération.
- ✓ **tournamentSize :** La taille de sélection par tournoi.

4.3.4 La fenêtre d'exécution

Cette fenêtre (**Figure 31**) permet l'affichage de l'emploi du temps après l'exécution de l'algorithme.L'utilisateur peut afficher l'EPDT d'un groupe, d'une salle ou d'un enseignant.

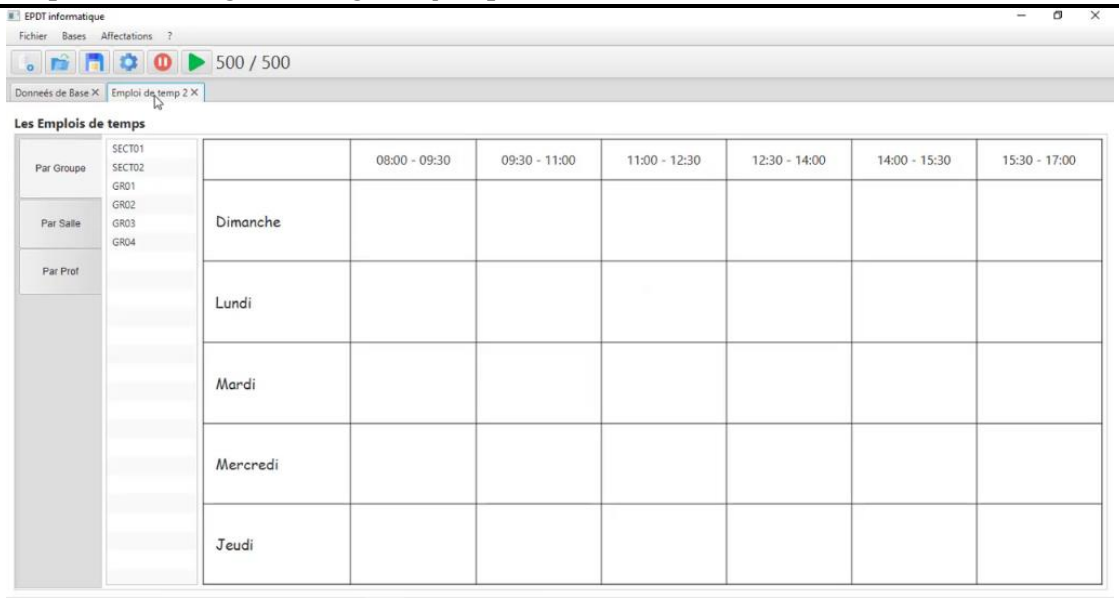

Figure 31 : Fenêtres d'exécution.

4.3.5 Structures de données

Coté programmation, nous avons utilisé les structures de données suivantes :

- ✓ Une classe AlgorithmeGenetic pour implémente les opérations génétiques.
- ✓ Les classes de mise en œuvre des structures de données.
- ✓ La classe principale.

4.3.5.1 La classe AlgorithmeGenetic

Cette classe est le code Java de notre algorithme génétique détaillé dans le chapitre 3, elle contient notamment les opérateurs génétiques.

```
public class AlgorithmGenetic {

    private int populationSize;
    private double mutationRate;
    private double crossoverRate;
    private int elitismCount;
    protected int tournamentSize;

    public AlgorithmGenetic() {...2 lines }

    public AlgorithmGenetic(int populationSize, double mutationRate, double crossove
    public Population initPopulation(DataSet ds) {...5 lines }

    public double calcFitness(Individual individual, DataSet ds) {...11 lines }
    public void evalPopulation(Population population, DataSet ds) {...7 lines }
    public Population mutatePopulation(Population population) {...25 lines }
    public Individual crossover2Individual(Individual parent1,Individual parent2,int
    public boolean containe(ArrayList<Integer> parent1Genes,int id){...6 lines }
    public Population crossover(Population population) {...27 lines }
    public Individual selectParent(Population population) {...12 lines }
    public int RandomNumberInRange(int min, int max) {...7 lines }
}
```

Figure 32 : La classe AlgorithmeGenetic.

4.3.5.2 Classes de structure de donnée

Les classes de mise en œuvre contiennent les données et les méthodes de manipulations.

- ✓ **La classe Dataset** : Cette classe contient des méthodes qui calculent le nombre de violations des contraintes dures et souples (**Figure 33**).
- ✓ **La classe Individual :** Cette classe contient la présentation de chromosomes (**Figure 34**).
- ✓ **La classe population :**Une classe décrivant une population donnée d'individus (**Figure 35**).

```
public class DataSet {

    private ArrayList<Groupe> listGroupe = new ArrayList<Groupe>();
    private ArrayList<Matrière> listMatrière = new ArrayList<Matrière>();
    private ArrayList<Prof> listProf = new ArrayList<Prof>();
    private ArrayList<ProfMatiere> listProfMatiere = new ArrayList<ProfMatiere>();
    private ArrayList<Salle> listSalle = new ArrayList<Salle>();
    private ArrayList<Section> listSection = new ArrayList<Section>();
    private ArrayList<Evenment> EVEN = new ArrayList<Evenment>();

    public DataSet() {...2 lines }

    public DataSet(DataSet cloneable) {...9 lines }

    public void Clone(Individual individual) {...15 lines }

    public int calcAffrontements() {...75 lines }

    public ArrayList<Groupe> getListGroupe() {...3 lines }

    public ArrayList<Matrière> getListMatrière() {...3 lines }

    public ArrayList<Prof> getListProf() {...3 lines }

    public ArrayList<ProfMatiere> getListProfMatiere() {...3 lines }

    public ArrayList<Salle> getListSalle() {...3 lines }

    public ArrayList<Section> getListSection() {...3 lines }
```

Figure 33 : La classe dataset.

```
public class Population {
    private Individual population[];
    private double populationFitness = -1;
    public Population(int populationSize) {...4 lines }

    public Population(int populationSize, DataSet dataSet) {...11 lines }

    public Individual[] getPopulation() {...3 lines }

    public double getPopulationFitness() {...3 lines }

    public void setPopulation(Individual[] population) {...3 lines }

    public void setPopulationFitness(double populationFitness) {...3 lines }

    /** Find fittest individual in the population ...6 lines */
        public Individual getFittest(int offset) {...18 lines }
        public Individual setIndividual(int offset, Individual individual) {...
        public Individual getIndividual(int offset) {...3 lines }
        public Individual[] getIndividuals() {...3 lines }
        public void shuffle() {...9 lines }
}
```

Figure 34 : La classe population.

```
public class Individual {

    private Evenment[] chromosome;
    private double fitness = -1;

    public Individual(DataSet dataSet) {...32 lines }

    public void InsertFakeData(){...10 lines }
    public void ClearFakeData(){...7 lines }
    public Individual(int length) {...6 lines }



    public int RandomNumberInRange(int min, int max) {...7 lines }

    public Evenment[] getChromosome() {...3 lines }

    public double getFitness() {...3 lines }

    public void setChromosome(Evenment[] chromosome) {...3 lines }

    public void setFitness(double fitness) {...3 lines }

    public Evenment getGene(int offset) {...3 lines }

    public void setGene(int offset, Evenment gene) {...3 lines }

    public void printIndividual(){...17 lines }
}
```

Figure 35 : La classe Individual.

4.3.5.3 La classe principale.

Cette classe appelée Controller qui contient le constructeur et une méthode Run qui reçoit en entrée le fichier ***Dataset.txt*** et exécute l'algorithme génétique dans un thread et envoi le résultat. La figure 36 donne un extrait du code de cette classe.

```
T = new Thread() {
   public void run() {

      AlgorithmGenetic GA = new AlgorithmGenetic(populationSize, mutationRate, crossoverRate, elitismCount,

      Population population = GA.initPopulation(ds);

      GA.evalPopulation(population, ds);

      int generation = 1;
      int maxGenerations = MaxGeneration;

      while (generation <= maxGenerations && population.getFittest(0).getFitness() != 1.0) {

          System.out.println("G" + generation + " Best fitness: " + population.getFittest(0).getFitness());

          population = GA.crossover(population);

          population = GA.mutatePopulation(population);

          changeJLabel(countGeneration, generation + " / " + maxGenerations);
          System.out.println("");

          generation++;

      }
      Individual i=population.getFittest(0);
      ds.Clone(i);
```

Figure 36 : Extrait de code de la classe principale.

4.3.6 Expérimentations et résultats

Dans cette partie nous allons discuter les résultats obtenus avec notre applicationEPDT. Les données de base utilisées dans cette étude sont présentées dans le tableau suivant :

Tableau 8 : Liste des données de base.

Donnée de base	quantité
Enseignant	14
Groupe	4
Section	2
Matière	3
Salle, amphi, laboratoire	11

Dans ces résultats, on voit l'efficacité de l'algorithme génétique pour la résolution d'un problème d'EPDT.

4.3.6.1 Évolution de la valeur de fitness

Nous avons étudié l'évolution de la valeur de fitness d'une génération à l'autre avec les paramètres

suivant :**maxgeneration=1000,popolationSize=60,mutationRate=0.01,crossoverRate=0.7, elitismCount=2,tournamentSize=5.**

Nous avons remarqué que la valeur de fitness augmente au fur et à mesure que l'on avance dans les générations. La figure suivante(**Figure 37**) montre l'évolution de la valeur de fitness.

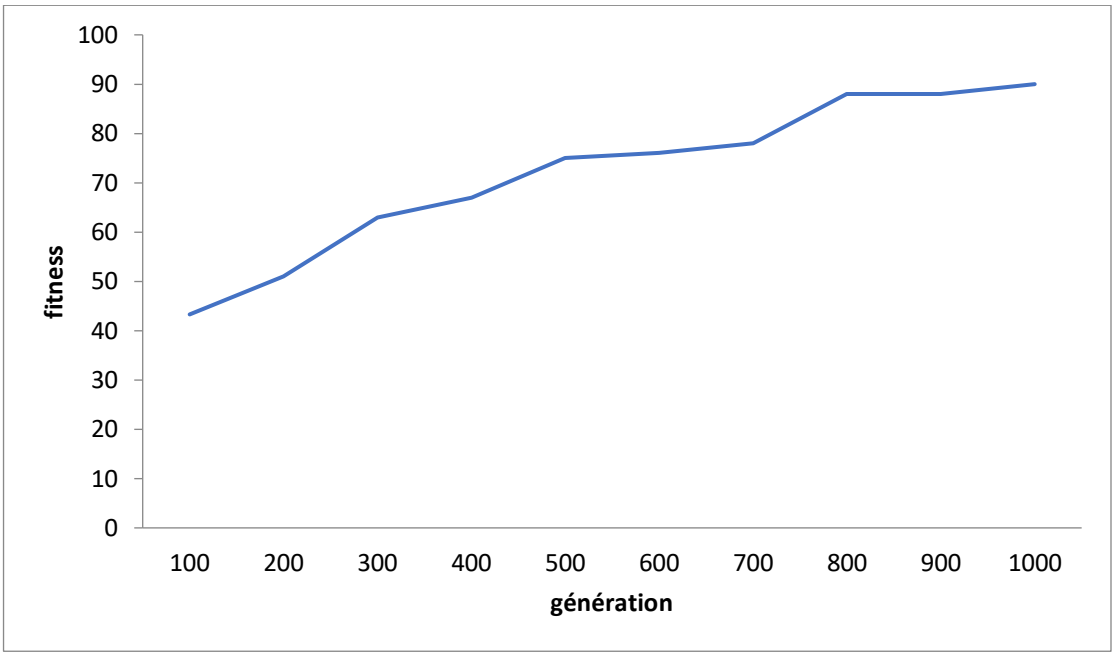

Figure 37 : Evolution de la valeur de fitness.

4.3.6.2 Pourcentage de contraintes non respectées

La **figure 38** suivante montre le pourcentage de contraintes souples non respectées pour chaque génération. Nous avons enregistré le pourcentage de contraintes non respectées avec une augmentation du nombre de générations (les paramètres sont les mêmes que dans l'étude précédente).

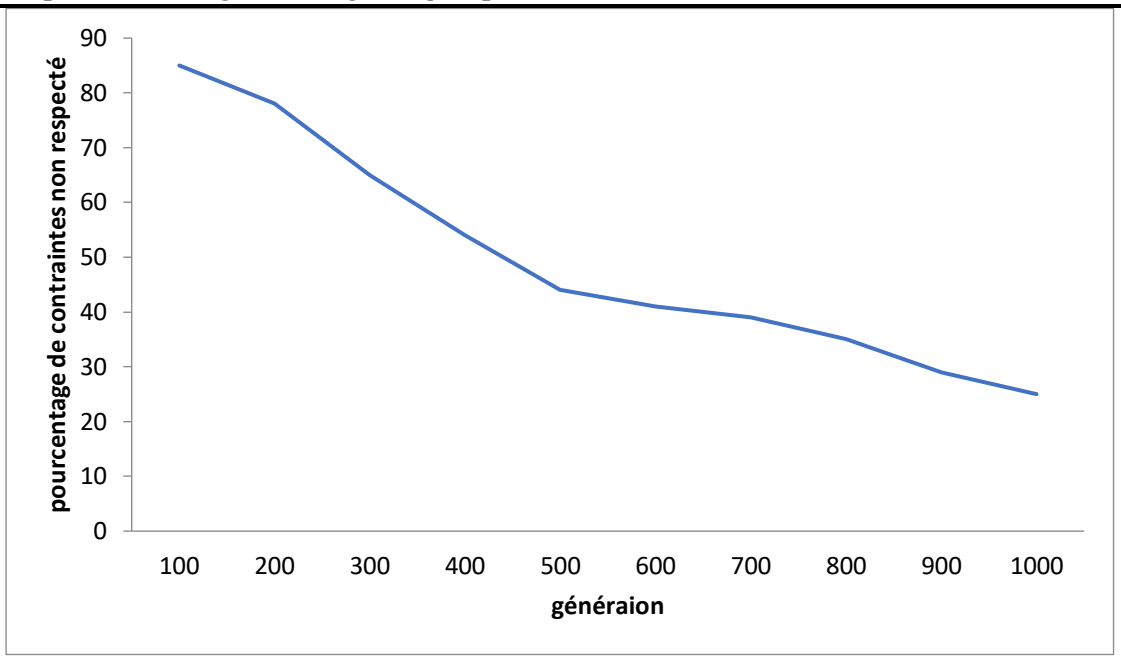

Figure 38 :Pourcentage de contraintes non respectées.

A partir de la **figure 38**, on remarque une diminution du pourcentage de contraintes non respectées au fur et à mesure que le nombre de générations augmente.

4.3.6.3 Résultats via l'application

Les **figures 39, 40,41** représentent les résultats après l'exécution de l'application EPDT.

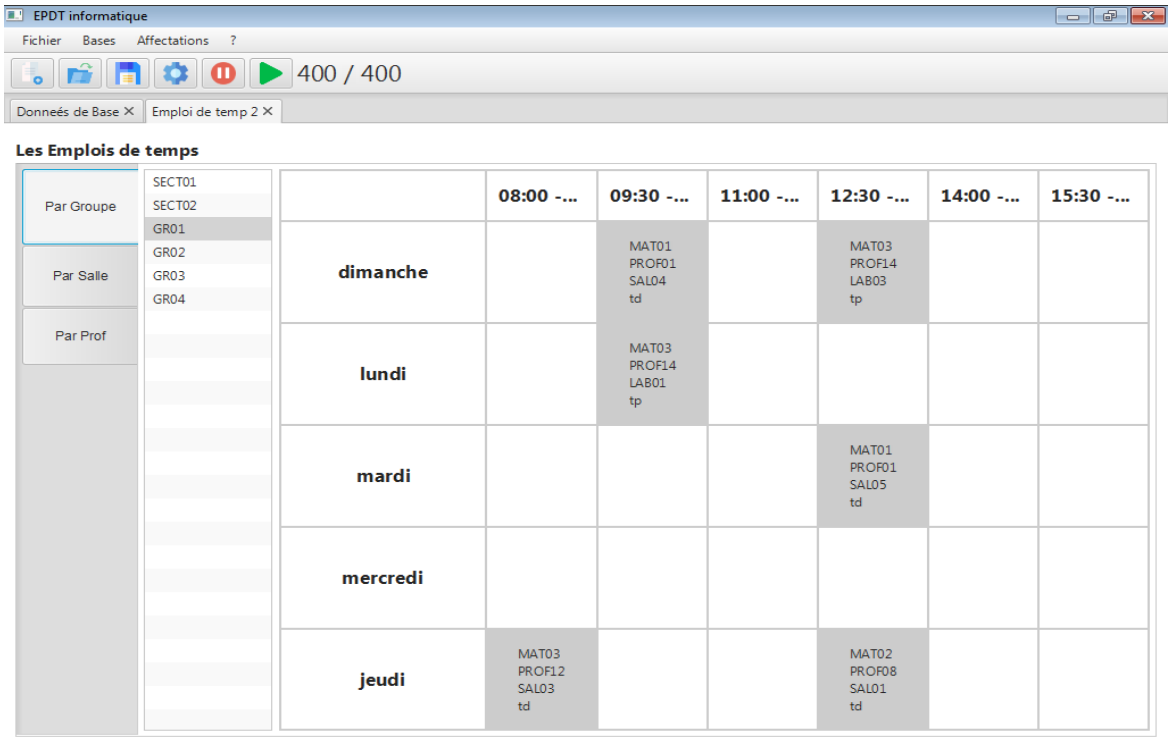

Figure 39 : Fenêtre d'affichage par groupe.

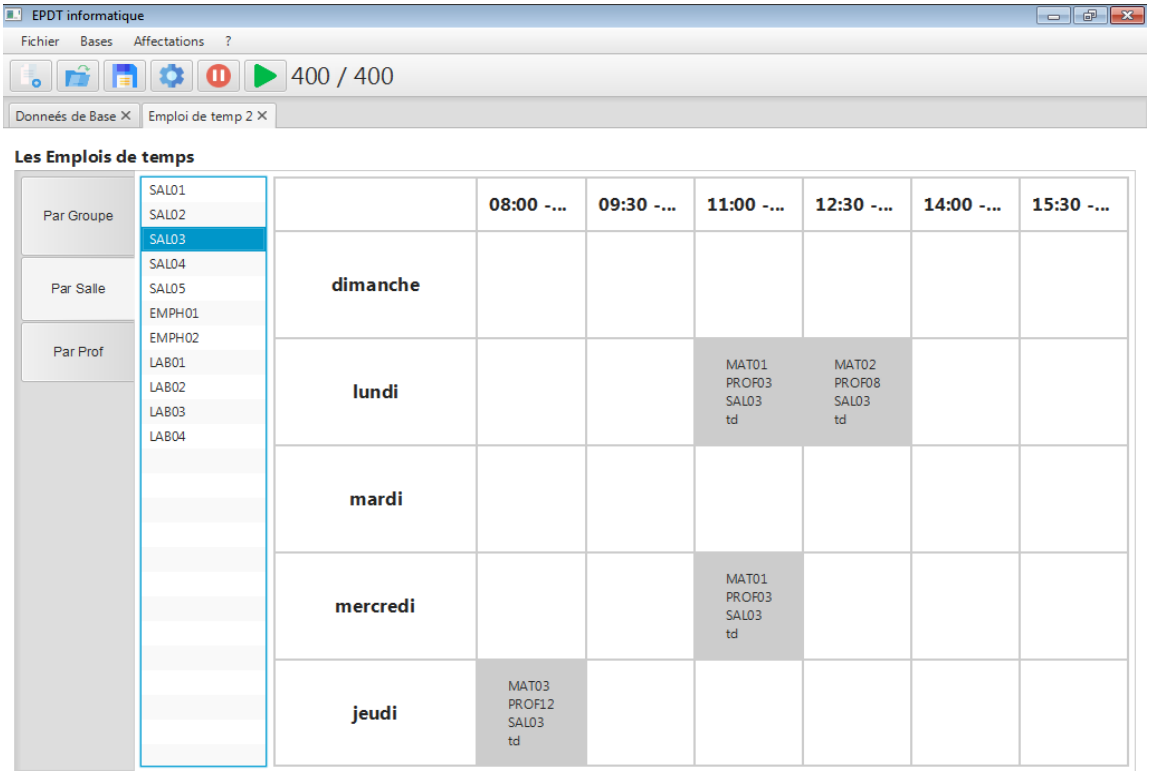

Figure 40 : Fenêtre d'affichage par salle.

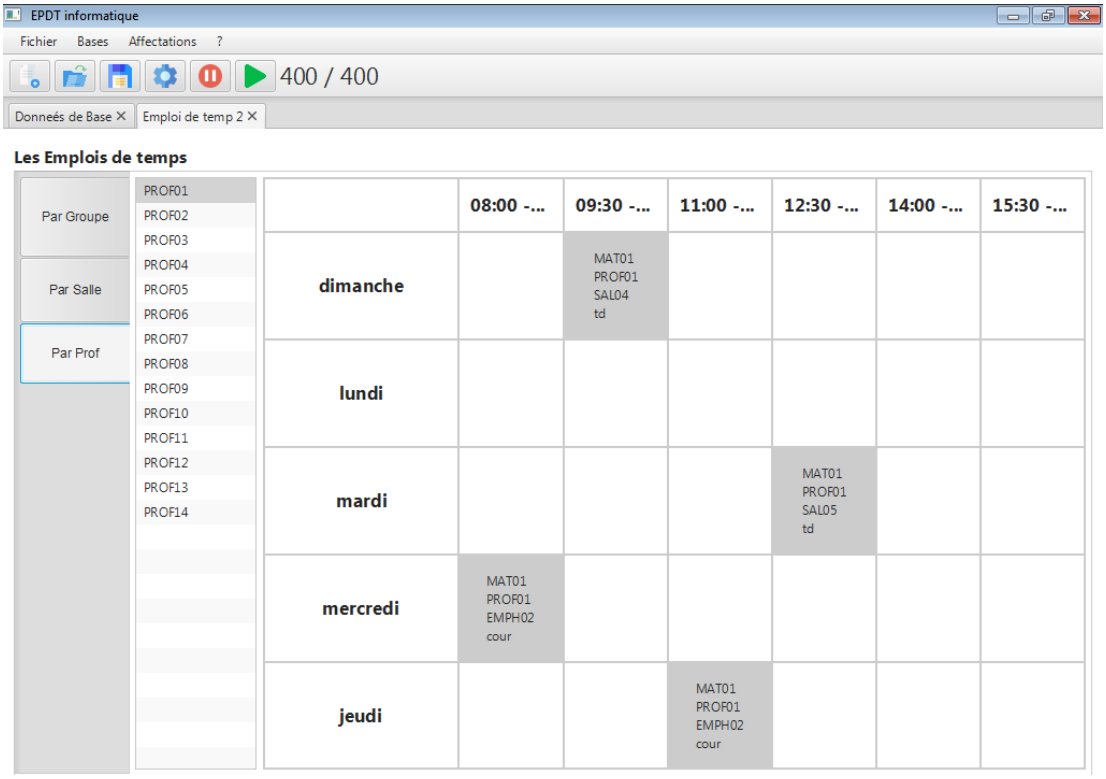

Figure 41 : Fenêtre d'affichage par enseignant.

400/400 : est le numéro de génération en cours. C'est-à-dire, l'algorithme génétique s'est arrêté après après 400 générations définies dans les paramètres.

4.4 Conclusion

Dans le cadre de ce travail, nous avons implémenté un algorithme à base de population (algorithme génétique) pour résoudre le problème de l'emploi du temps, afin d'améliorer sa qualité et atteindre la satisfaction des contraintes.

Les résultats obtenus montrent que les algorithmes génétiques sont efficaces pour résoudre le problème de l'emploi du temps.

Conclusion générale

Dans ce travail, nous nous sommes intéressés au problème de la génération automatique d'un emploi du temps au niveau d'un département à l'universités, car résoudre le problème à la main prend beaucoup de temps et ne donne pas forcément de bons résultats et conduit à l'émergence de nouveaux problèmes dans l'avenir, notamment avec l'augmentation du nombre de groupes.

Afin de trouver des solutions au problème de l'emploi du temps, des chercheurs ont proposé de nombreuses idées et méthodes. L'utilisation d'heuristiques à base de populations est l'une des meilleures solutions suggérées.

Le but des algorithmes à base d'heuristiques est d'obtenir une solution approchée à un problème d'optimisation, lorsqu'il n'existe pas de méthode exacte pour le résoudre en un temps raisonnable.

Les algorithmes génétiques utilisent la notion de sélection naturelle et l'appliquent à une population de solutions potentielles au problème donné. La solution est approchée par des améliorations successives, comme dans une procédure de séparation et évaluation (Branch & Bound), à ceci près que ce sont des formules qui sont recherchées et non plus directement des valeurs.

Notre objectif dans ce mémoire est de trouver une solution au problème de l'emploi de temps basée sur le principe des algorithmes génétiques.

Avec notre application EPDT nous avons testé l'efficacité d'un algorithme génétique pour résoudre le problème de l'emploi de temps.

Les résultats montrent que les algorithmes génétiques sont efficaces dans le problème d'emploi du temps car ils sont capables d'améliorer les résultats pour approcher la solution optimale.

Bibliographie

1. **Feizi-Derakhshi, Babaei, &Heidarzadeh,.** *A survey of approaches for university course timetabling problem.* Antalya : In Proceedings of 8th international symposium on intelligent and manufacturing systems (IMS 2012), 2012. pp. 307–321.

2. **Lewis.** *Metaheuristics for university course timetabling.* 2006.

3. **A. Colorni, M. Dorigo, and V. Maniezzo.** *Genetic algorithms - a new approach to thetimetable problem.* 1990.

4. **Carter, M. W.** *A survey of practical applications of examination timetabling algorithms.* 1986. pp. 193–202.

5. **Chan, G. M. White and P. W.** *Towards the construction of optimal examination timetables.* s.l. : INFOR 17, 1979. pp. 219–229.

6. **Abramson, D.** *Constructing school timetables using simulated annealing: sequential andparallel algorithms.* s.l. : Management Science, 1991. pp. 98–113.

7. **Hertz, A.** *Tabu search for large scale timetabling problems.* s.l. : European journal of operationsresearch, 1991. pp. 39–47.

8. Recherche opérationnelle. *Wikipedia.* [En ligne] https://fr.wikipedia.org/wiki/Recherche_op%C3%A9rationnelle.

9. Coloration de graphe. *wikipidia.* [En ligne] https://fr.wikipedia.org/wiki/Coloration_de_graphe.

10. **Gotlib, C. C.** *The construction of class-teacher timetables. Proceedings of IFIP Congress.* 1963. pp. 73–77.

11. *An upper bound for the chromatic number of a graph and its application to timetabling problems.* **Welsh, D. J. A., & Powell, M. B.** 1967, The Computer Journal, pp. 85–86. 10.

12. *An Introduction to TimeTabling.* **De Werra, D.** 19, European Journal of Operational Research, pp. 151–162.

13. *Split vertices in vertex colouring and their application in developing a solution to the faculty timetable problem.* **Selim, S. M.** 31(1), The Computer Journal, pp. 76–82.

14. *Bipartite graph edge coloring approach to course timetabling.* **Hafizah, A. R., & Zaidah, I.** s.l. : IEEE, 2010, pp. 229–234.

15. *Graph coloring for class scheduling.* **Dandashi, A., & Al-Mouhamed, M.** Koura, Lebanon : University of Balama,Department of Computer Science , 2010.

16. *Trans genetic coloring approach for timetabling problem.* **Asham, G. M., Soliman, M. M., & Ramadan, A. R.** s.l. : Artificial intelligence techniques novel approaches & practical applications IJCA, pp. 17–25.

17. *Theory and Methodology Implementation of a university course and examination timetabling system.* **Dimopoulou, M., & Miliotis, P.** 130, 2001, European Journal of Operational Research, pp. 202–213.

18. *A large scale timetabling problem.* **Aubin, J., & Ferland, J. A.** 16, s.l. : Computers and Operations Research, 1989, pp. 67–77.

19. Métaheuristique. *wikipédia.* [En ligne] https://fr.wikipedia.org/wiki/M%C3%A9taheuristique.

20. **Dr, LEMOUARI.** *Introduction aux Métaheuristiques.* 2014.

21. **Christian Blum, Andrea Roli.** *Metaheuristics in combinatorial optimization:Overview and conceptual comparison.* s.l. : ACM Computing Surveys, 2003. pp. 268-308. Vol. 35. 3.

22. **Johann Dréo, Alain Petrowski, Éric Taillard, Patrick Siarry.** *Métaheuristiques pour l'optimisation difficile.* Éd. Eyrolles, Paris : Broché, septembre 2003. p. 356.

23. Algorithme génétique. *wikipédia.* [En ligne] https://fr.wikipedia.org/wiki/Algorithme_g%C3%A9n%C3%A9tique.

24. **Khonggamnerd, P., & Innet, S.** *On improvement of effectiveness in automatic university timetabling arrangement with applied genetic algorithm.* s.l. : IEEE, 2009.

25. **Alsmadi, O. MK., Abo-Hammour, Z. S., Abu-Al-Nadi, D. I., & Algsoon, A.** *A novel genetic algorithm technique for solving university course timetabling problems.* s.l. : IEEE, 2011.

26. Algorithme de colonies de fourmis. *wikipédia.* [En ligne] https://fr.wikipedia.org/wiki/Algorithme_de_colonies_de_fourmis.

27. *Distributed Optimization by Ant Colonies.* **A. Colorni, M. Dorigo et V. Maniezzo.** Paris, France : actes de la première conférence européenne sur la vie artificielle, 1991, Elsevier Publishing, pp. 134-142.

28. *A max-min ant system for the university course timetabling problem.* **Socha, K., Knowles, J., & Samples, M.** Springer-Verlag : Lecturer notes in computer science, 2002, In Proceedings of the 3rd international workshop on ant algorithms (ANTS 2002), Vol. 2463, pp. 1–13.

29. *Solving the post enrolment course timetabling problem by ant colony optimization.* **Mayer, A., Nothegger, C., Chwatal, A., & Raidl, G.** In Proceedings of the 7th international conference on the practice and theory of automated timetabling.

30. *Hybrid ant colony systems for course timetabling.* **Ayob, M., & Jaradat, G.** Selangor, Malaysia : s.n., 2009, In IEEE 2nd conference on data mining and optimization 27–28 October, pp. 120–126.

31. Algorithme mémétique. *wikipédia.* [En ligne] https://fr.wikipedia.org/wiki/Algorithme_m%C3%A9m%C3%A9tique.

32. **Shengxiang, Y., & Jat, N. S.** *Genetic algorithms with guided and local search strategies for university course timetabling.* s.l. : IEEE Transactions on Systems, 2011.

33. **J. Dréo, J.-P. Aumasson, W. Tfaili, P. Siarry.** *Adaptive Learning Search.* s.l. : a new tool to help comprehending metaheuristics, to appear in the International Journal on Artificial Intelligence Tools.

34. Recherche tabou. *wikipédia.* [En ligne] https://fr.wikipedia.org/wiki/Recherche_tabou.

35. **Glover, F. et Laguna, M.** *Tabu search.* Dordrecht : Kluwer Academic Publishers, 1997.

36. **J., Dreo, et al.** *Métaheuristiques pour l'optimisation difficile ; recuit simulé, recherche tabou.* Eyrolles : s.n., 2003.

37. **Alvarez, R., Crespo, E., & Tamarit, J. M.** *Design and implementation of a course scheduling system using Tabu search.* s.l. : European Journal of Operational Research, 2002. pp. 512–523.

38. **Aladag, C. H., Hocaoglu, G. A., & Basaran, M.** *The effect of neighborhood structures on tabu search algorithm in solving course timetabling problem.* s.l. : Expert Systems with Application, 2009. pp. 12349–12356.

39. Recuit simulé. *wikipédia.* [En ligne] https://fr.wikipedia.org/wiki/Recuit_simul%C3%A9.

40. **Aarts, E. H. L., and v. Laarhoven.** *Statistical cooling: A general approach to combinatorial optimization problems, Philips J. Res.* 1985. pp. 193-226. Vol. 4.

41. **Tuga, M., Berretta, R., & Mendes, A.** *A hybrid simulated annealing with kempe chain neighborhood for the university timetabling problem.* s.l. : In 6th IEEE/ACIS international conference on computer and information science (ICIS 2007), 2007.

42. **Aycan, E., & Ayav, T.** *Solving the course scheduling problem using simulated annealing.* s.l. : IEEE, 2008.

43. Recherche locale (optimisation). *wikipédia.* [En ligne] https://fr.wikipedia.org/wiki/Recherche_locale_(optimisation).

44. **Hoos, H.H. and Stutzle, T.** *Stochastic Local Search: Foundations and Applications.* s.l. : Morgan Kaufmann, 2005.

45. **Joudaki, M., Imani, M., & Mazhari, N.** *Using improved Memetic algorithm and local search to solve University Course Timetabling Problem (UCTTP).* Doroud, Iran : s.n., 2010.

46. **Jat, N. S. & Shengxiang, Y.** *A memetic algorithm for the university course timetabling problem.* s.l. : In IEEE 20th IEEE international conference on tools with artificial intelligence, 2008. pp. 427–433.

47. **Asmuni, H., Burke, E. K., & Garibaldi, J. M.** *Fuzzy multiple heuristic ordering for course timetabling.* London, UK : s.n., 2005. pp. 302–309.

48. **Asmuni, H.** *Fuzzy methodologies for automated university timetabling solution.* s.l. : Ph.D. Thesis, School of Computer Science University of Nottingham, 2008.

49. **Aarts, E.H.L. & van Laarhoven, P.J.M.** *Statistical cooling : A general Approach to combinatorial optimization problems.* s.l. : Phillips Journal of Research, 1985. pp. 193-226.

50. **E. Burke, J.P. Newall,.** *Solving examination timetabling problems through adaption of heuristic ordering.* s.l. : Annals of operations research, 2004. pp. 107-134.

51. **AL-Milli, N.R.** *Hybrid Genetic Algorithms with great deluge for course timetabling.* s.l. : IJCSNS International Journal of computer science and network security, 2010. pp. 283-287. Vol. 10.

52. **C.H. Aladag, G. Hocaolglu, M.A. Basaran.** *The effect of neighborhood structures on tabu search algorithm in solving course timetabling problem.* s.l. : Expert systems with applications, 2009. pp. 12349-12356. Vol. 36.

53. **A. Dammark, A. Elloumi, H. Kamoun.** *Course Timetabling at Tunisian university.J. Syst. ScisystEng.* 2008. pp. 334-352. Vol. 17.

54. **B. Paechter, M. G. Norman, and H. Luchian,.** *Extensions to a memetic timetabling system.* s.l. : in E.K. Burke and P.M. Ross, eds., Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling, Lawrence Erlbaum Associates, 1995.

55. **P. Srinin, P. Rojanavasu, E. Pin-nguen.** *The Automatic Timetabling Problem by using Genetic.* s.l. : Joint Conference on ComputerScience and Software Engineering(JCSSE2005), 2005.

56. **K. Paitoonwattanakij, K. Vongvipaporn,.** *GA For Scheduling School Timetable,J.Natl.Res.Council Thailand.* 2000.

57. logiciel java. *sokeo.* [En ligne] https://sokeo.fr/logiciel-java-definition/.

58. java. *greelane.* [En ligne] https://www.greelane.com/fr/science.

59. what is javafx. *greelan.* [En ligne] https://www.greelane.com/fr/science-technologie math%c3%a9matiques/linformatique/what-is-javafx-2034192/.

60. what does javafx mean you . *dzone.* [En ligne] https://dzone.com/articles/what-does-javafx-mean-you..

61. what is jdk . *infoworld.* [En ligne] https://www.infoworld.com/article/3296360/what-is-the-jdk-introduction-to-the-javadevelopment-kit.html..

62. JDK 16.0.1 Release Notes. *oracle.* [En ligne] https://www.oracle.com/java/technologies/javase/16-0-1-relnotes.html.

63. NetBeans. *wikipedia.* [En ligne] https://fr.wikipedia.org/wiki/NetBeans..