République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique

Université 20 Août 1955 Skikda
Faculté des Sciences de l'Ingénieur

Département d'Informatique

Mémoire Présenté pour l'obtention du diplôme de
Magister en Informatique

Option : Systèmes d'Information et de Connaissances

Titre du Mémoire

Conception et implémentation d'un algorithme d'optimisation pour la Génération automatique d'emploi du temps Universitaire

Présenté par :
Rédha MILI

Devant le jury :

Président :	Dr KHOLLADI Mohamed- Khireddine	Université de Constantine
Rapporteur :	M.C. LAMROUS Sid	UTBM France
Co-Encadreur	Pr . SERIDI Hamid	Université de Guelma
Examinateurs :	M.C. SERIDI Hassina	Université d'Annaba
	M.C. CHIKHI Salim	Université de Constantine

Soutenu le 15 juillet 2010

Remerciements

Je tiens à remercier tous les membres du groupe chronos , Marie-Ange MANIER ,Sid LAMROUS, Hakim MABED, Oumaya BAALA, Claude Renaud et HERVE .MANIER de m'avoir si bien accueilli dans le groupe durant ce stage.

Je tiens à remercier et à exprimer ma profonde gratitude envers tous les membres du laboratoire SET de l'université de technologie Belfort Montbéliard,tous les membres de l'équipe d'optimisation, professeurs, maître de conférences et doctorants pour leur qualités humaines .Nul doute qu'il fut pour moi une importante source de stimulation intellectuelle et que j'ai beaucoup appris avec eux.

Je considère que c'est une chance d'avoir travaillé avec eux.

Je tiens à remercier tous ceux qui ont fait pour que ce travail se réalise, mes responsables en France et ceux d'Algérie chacun de part son nom : SERIDI abdel hamid , Mouloud BELACHIA .

Sans oublier Messieurs Khiredinne KHOLLADI et Smaine MAAZOUZI

Pour terminer un spécial remerciement pour Sid AHMED LAMROUS, sans qui se travaille et ce stage n'auraient pas eu lieu.

Résumé

La génération automatique d'emploi du temps universitaire peut être décrite comme l'allocation des ressources (Créneaux Horaires, Salles, …) aux événements (Séances de cours, TD et TP), tout en essayant de satisfaire un ensemble de contraintes. L'approche proposée est la résolution du problème en utilisant un algorithme génétique pour la création de l'emploi du temps, couplé avec une méthode de recherche locale pour la répartition des étudiants entre les groupes. L'expérimentation de l'algorithme sur les données réelles de l'UTBM inscrit une bonne perspective de recherche.

Mots Clés : *Optimisation combinatoire, Méta-heuristique, Algorithmes génétiques, Emploi du temps.*

Summary

Automatic generation of university Timetabling can be described as the allocation of resources (time slots, rooms ...) to events (lesson, TD and TP), while trying to meet a set of constraints.
The proposed approach is solving the problem using a genetic algorithm to create the Timetabling, coupled with a local search method for the distribution of students between the groups. Testing the algorithm on actual data of the registered UTBM good research perspective.

***Keywords**: combinatorial optimization, meta-heuristics, genetic algorithms, Timetabling*

ملخص

تعتبر جدولة الأحداث داخل أي هيئة من أهم تطبيقات البرمجة في الإعلام الآلي ، خاصة إذا كانت هذه الهيئة تتميز بكثرة الأحداث و تنوعها مثل الجامعة .

ويتمحور هذا البحث على معالجة جدولة مواعيد الجامعة المختلفة، و هذا باعتبار هذه العملية كعملية تخصيص للموارد (الوقت، الغرف...) للأحداث و هي الدروس مع التقييد بمجموعة من القيود.

و قد تم استخدام خوارزمية جينية لتهيئة جدولة المواعيد، مع البحث عن أحسن وسيلة لتوزيع الطلاب على المجموعات.

الكلمات المفتاحية

حوارزمية جينية – الاحداث – تخصيص الموارد

Introduction

La construction d'un emploi du temps est un travail long et difficile à mettre en place et qui doit être répétée à intervalle régulier. C'est pour cette raison que l'automatisation d'emplois du temps, est une tâche d'une grande importance car elle permet de gagner beaucoup d'heures de travail pour les employés, les institutions et les entreprises, et de fournir des solutions optimales avec satisfaction de contrainte en quelques minutes, ce qui peut accroître la productivité, la qualité de l'éducation, qualité des services et enfin, la qualité de vie.

La difficulté de cette tache fastidieuse et répétitive, qui fait généralement intervenir de nombreux éléments d'information, est liée à la nature des contraintes qu'il s'agit de satisfaire. A celle-ci s'ajoutent des caractéristiques bien particulières de l'institution ou l'organisation concernée.

Parce qu'il y a plusieurs modèles du problème, des contraintes qui changent, et des caractéristiques particulières pour chaque institution ou organisation, il devient de plus en plus difficile de trouver une solution générale pour les problèmes d'emploi du temps, et c'est pour cela que ce domaine a besoin de plus en plus de recherche.

Dans le domaine de l'éducation, le problème connu sous le nom d'emploi du temps consiste à organiser des rencontres entre des enseignants et des groupes d'étudiants en définissant un lieu et une heure de rencontre tout en satisfaisant un ensemble de contraintes. Il existe différentes versions de ce dernier mais la plupart peuvent être classifié en trois grandes catégories qui sont : emploi du temps pour école, emploi du temps pour examen et emploi du temps universitaire.

Dans ce mémoire nous allons nous intéresser au problème de génération automatique d'emploi du temps universitaire et plus précisément à celui de l'université de technologie de Belfort-Montbéliard (UTBM). Ce travaille s'inscrirt dans le cadre du projet Chronos qui a pour but de construire une application de génération

automatique d'emploi du temps pour l'UTBM basé sur les algorithmes d'optimisation. La problématique de génération d'emploi du temps à l'UTBM représente un vrai challenge semestriel. Le mode d'enseignement basé sur l'inscription individuelle des étudiants aux UVs reste la principale difficulté.

L'existence d'un horaire respectant à la lettre toutes les contraintes n'est pas garantie dans le cas général. En réalité, il s'agit de trouver un horaire aussi bon que possible dans un temps de calcul raisonnable tout en tolérant la présence de conflits, le nombre de ces conflits doit être minimisé pour que l'horaire soit jugé satisfaisant par les différents parties concernées (direction de l'université, Intervenant, Etudiants, etc.)

Le premier chapitre de ce mémoire, nous parle du domaine de l'optimisation combinatoire et des différentes méthodes utilisées pour résoudre ce genre de problème. Le deuxième chapitre est un état de l'art sur les emplois du temps, nous introduisons le problème d'optimisation dans le domaine des emplois du temps et nous discutons les variantes du problème. Enfin, nous présentons des techniques et approches trouvées dans la littérature. Le troisième chapitre présentera les algorithmes génétiques en générale et l'algorithme qu'on a développé couplé à une méthode de recherche locale pour la résolution de notre problème d'emploi du temps. Une analyse des résultats serra donnés et puis nous finirons avec une conclusion générale.

Nous remarquons au lecteur que dans la suite de ce mémoire, nous ne ferons aucune différence entre les termes, emploi du temps, calendrier et horaire qui dans la littérature référent au même problème dans la plus part des cas.

Chapitre 1

L'Optimisation Combinatoire

Le présent chapitre présente un état de l'art sur l'optimisation combinatoire et décrit un ensemble de méthodes de la littérature

Dans un premier temps nous donnerons quelques définitions dans le but de comprendre la problématique générale que nous voulons résoudre dans ce mémoire (§.1) (§.2).

Par la suite, nous présenterons en détail les méthodes les plus utilisées et les plus connues dans le domaine de l'optimisation (§.3) et nous nous intéresserons plus particulièrement aux méta-heuristiques (§.4).

Pour finir, nous proposerons quelques éléments d'analyse des méta-heuristiques et nous donnerons quelques indications pratiques pour faciliter le choix d'une méta-heuristique (§.5) tout en sachant qu'il n'existe pas de comparaison générale et fiable des différentes méta-heuristiques dans la littérature

Introduction

La résolution d'un problème d'optimisation consiste à explorer un espace de recherche afin de maximiser (ou minimiser) une fonction donnée. Les complexités (en taille ou en structure) relatives de l'espace de recherche et de la fonction à maximiser conduisent à utiliser des méthodes de résolutions radicalement différentes. En première approximation, on peut dire qu'une méthode déterministe est adaptée à un espace de recherche petit et complexe et qu'un espace de recherche grand nécessite plutôt une méthode de recherche stochastique (recuit simulé, algorithme génétique, ...).

1. L'optimisation combinatoire

L'optimisation combinatoire est un domaine des mathématiques discrètes qui a attiré l'attention de nombreux chercheurs ces 40 dernières années. Le nombre élevé de publications dans la littérature spécialisé témoigne de cet intérêt toujours croissant qui est du aux nombreuses applications pratiques pouvant être formulées sous la forme d'un problème d'optimisation combinatoire.
On peut trouver de nombreuse définitions des problèmes d'optimisation combinatoire, celle que nous présentons dans ce mémoire est celle de Collette et Siarry [8] qui propose la définition suivante :

Définition 1.1 *Problème d'optimisation*

Un problème d'optimisation se définit comme la recherche du minimum ou du maximum d'une fonction donnée. Mathématiquement, dans le cas d'une minimisation, un problème d'optimisation se présentera sous la forme suivante :

Minimiser $f(s)$ (fonction à optimiser)

Avec $f(s) \leq 0$ (m contraintes d'inégalités) (1.1)

Et $h(s) = 0$ (p contraintes d'égalité)

En d'autres termes, résoudre un problème d'optimisation $P(S, f)$ revient à déterminer une solution $s^* \in S$ minimisant ou maximisant la fonction f avec S l'ensemble des solutions ou l'espace de recherche et $f: S \to Y$ une application ou une fonction d'évaluation qui à chaque configuration s associe une valeur $f(s) \in Y$.

Il est possible de passer d'un problème de maximisation à un problème de minimisation [18] grâce à la propriété suivante :

$$\max_{s \in S} f(s) \cong -\min_{s \in S}(-f(s)) \quad (1.2)$$

Généralement, une solution $s \in S$ est un vecteur d'un espace à N dimensions.

Définition 1.2 *problème d'optimisation continue*

Dans le cas des variables réelles, on a : $S \subseteq \mathbb{R}^N$. On parle alors de problème d'optimisation en variable continue.

Un problème d'optimisation continue (PO) peut être formulé de la façon suivante :

$$(PO) \min_{s \in \mathbb{R}^N} f(s) \quad (1.3)$$

Définition 1.3 *Problème d'optimisation combinatoire*

Un problème d'optimisation combinatoire est un problème d'optimisation dans le quel l'espace de recherche S est dénombrable.

Un problème d'optimisation combinatoire (POC) peut être formulé ainsi :

$$(POC) min_{s \in \mathbb{Z}^N} f(S) \quad (1.4)$$

Où une solution s est un vecteur composé de N valeur entière, soit $S \subseteq \mathbb{Z}^N$

La principale différence entre un problème d'optimisation continue et un problème d'optimisation combinatoire repose sur l'utilisation de variables discrètes. Dans les deux catégories de problèmes, une solution $s \in S$ est une instanciation des variables $x_i \in X$, où i est l'indice de la variable dans [1, N], et X est le vecteur de dimension N correspondant à la solution, et $f(s)$ est son évaluation. Résoudre ces problèmes revient à trouver une solution optimale appelée aussi optimum global.

Définition 1.4 *Optimum Global*

Une solution est un optimum global à un problème d'optimisation s'il n'existe pas d'autres solutions de meilleure qualité. La solution $s^* \in S$ est un optimum global ssi :

$$\forall s \in S \begin{cases} f(s^*) \leq f(s) & \text{dans le cas de minimisation} \\ f(s^*) \geq f(s) & \text{dans le cas de maximisation} \end{cases} \quad (1.5)$$

La figure 1 schématise la courbe d'une fonction d'évaluation en faisant apparaitre l'optimum global dans le cas d'un problème de minimisation.

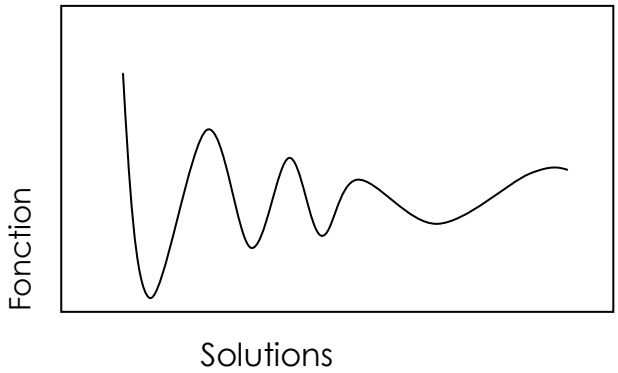


Figure 1 Courbe d'une fonction d'évaluation

2. Théorie de la *Complexité*

Un problème est dit *polynomial* s'il existe un algorithme permettant de trouver une solution optimale pour toutes ses instances en un temps polynomial par rapport à la taille de l'instance [20]. Un tel algorithme est dit *efficace* pour le problème en question.

C'est notamment le cas de certains problèmes de plus court chemin dans un graphe

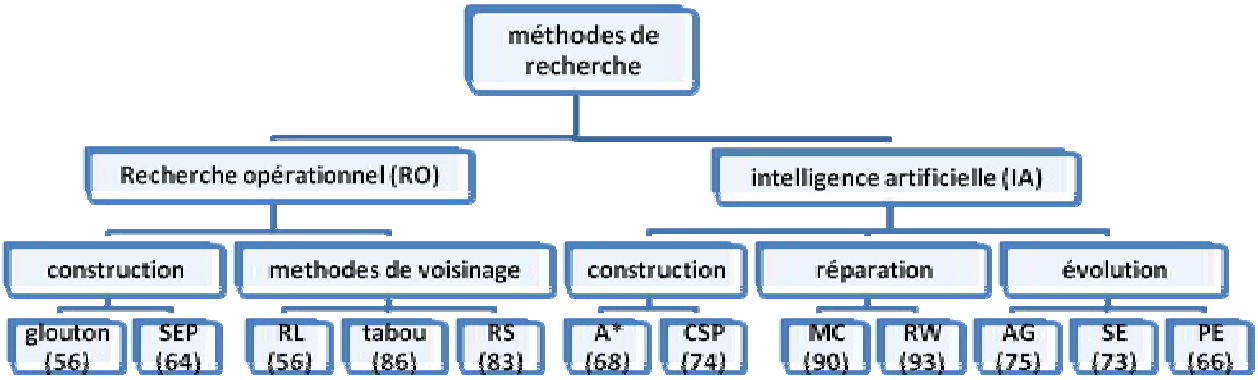
valué, du recouvrement d'un graphe valué par un arbre de poids minimum, des problèmes classiques de flots. Cependant, pour la majorité des problèmes d'optimisation combinatoire, aucun algorithme polynomial n'est connu actuellement.
La difficulté intrinsèque de ces problèmes est bien caractérisée par la théorie de la NP-complétude [20]. De nombreux problèmes d'optimisation combinatoire (la plupart de ceux qui sont vraiment intéressants dans les applications !) ont été prouvés NP-difficiles.
Cette difficulté n'est pas seulement théorique et se confirme hélas dans la pratique. Il arrive que des algorithmes exacts de complexité exponentielle se comportent efficacement face à de très grosses instances - pour certains problèmes et certaines classes d'instances. Mais c'est très souvent l'inverse qui se produit ; pour de nombreux problèmes, les meilleures méthodes exactes peuvent être mises en échec par des instances de taille modeste, parfois à partir de quelques dizaines de variables seulement. Par exemple, on ne connaît aucune méthode exacte qui soit capable de colorier de façon optimale un graphe aléatoire de densité 1/2 lorsque le nombre de sommets dépasse 90 [12]. Pour traiter les grosses instances de ce type de problèmes, on se contente de solutions approchées obtenues avec une méthode heuristique.

3. Introduction aux Méthodes de résolution

Il existe un grand nombre de problèmes d'optimisation. Les problèmes les plus classiques de programmation linéaire utilisent une fonction objective ainsi que des contraintes linéaire. Un problème d'optimisation exprimé sous la forme d'un programme linéaire cherche à maximiser une fonction de N variables x_i en respectant m contraintes linéaires.
Ce type de problème est résolvable de manière exacte via différentes méthodes comme la méthode du simplexe par exemple pour les problèmes en variable continue. La principale caractéristique d'une méthode exacte est la garantie de trouver un optimum global. Beaucoup d'études ont été mené afin de modéliser sous la forme d'un problème de programmation linéaire les problèmes d'optimisation combinatoire en général [8], Certains problèmes combinatoires ont pu être modélisés sous forme linéaire cependant la plupart d'entre eux ne le sont pas.

3.1. Un panorama

Un très grand nombre de méthodes de résolution existent en RO et en IA pour l'optimisation combinatoire. La figure 2 met en parallèle les méthodes représentatives développées en RO et en IA, avec à titre indicatif la date approximative d'apparition de chaque méthode.



Légende :
SEP : séparation & évaluation progressive
RL : recherche locale
Tabou : recherche tabou
RS : recuit simulé

CSP: méthodes complètes CSP
MC : heuristique min-conflit
RW : heuristique random walk
AG : algorithmes génétiques
SE : stratégies d'évolution
PE : programmation évolutive

Figure 2 Classement des méthodes de résolution

D'une manière très générale, les méthodes de résolution suivent quatre approches différentes pour la recherche d'une solution : l'approche de construction, l'approche de relaxation, l'approche de voisinage et l'approche d'évolution.

Ces méthodes font partie de deux groupes de nature différente. Le premier groupe comprend les méthodes exactes d'arborescence qui garantissent la complétude de la résolution : c'est le cas de SEP, A* et CSP. Le temps de calcul nécessaire d'une telle méthode augmente en général exponentiellement avec la taille du problème à résoudre (dans le pire des cas). Pour améliorer l'efficacité de la recherche, on utilise des techniques variées pour calculer des bornes permettant d'élaguer le plus tôt possible des branches conduisant à un échec. Parmi ces techniques, on peut citer

les différentes relaxations [12] : la relaxation de base en programmation linéaire, la relaxation lagrangienne, la relaxation agrégée (*surragote relaxation*) et la décomposition lagrangienne. De plus, on emploie des heuristiques pour guider les choix de variables et de valeurs durant l'exploration de l'arborescence.

Le second groupe comprend les méthodes approchées dont le but est de trouver une solution de bonne qualité en un temps de calcul raisonnable sans garantir l'optimalité de la solution obtenue. Les méthodes approchées sont fondées principalement sur diverses heuristiques, souvent spécifiques à un type de problème [12].

Les métaheuristiques constituent une autre partie importante des méthodes approchées et ouvrent des voies très intéressantes en matière de conception de méthodes heuristiques pour l'optimisation combinatoire.

Dans la suite nous nous intéresserons essentiellement aux métaheuristiques : les méthodes de voisinage et les algorithmes évolutifs. Nous présentons également les possibilités de combiner différentes méthodes pour créer des méthodes hybrides.

Nous commençons dans la section suivante par une brève introduction de l'approche de construction afin de mettre en contraste le principe d'augmentation successive de cette approche avec les principes de réparation et d'évolution.

3.2. Approche de construction

L'approche de construction est probablement la plus ancienne et occupe traditionnellement une place très importante en optimisation combinatoire et en intelligence artificielle [12]. Une méthode de construction construit pas à pas une solution de la forme $s = (<V1,v1> <V2,v2>...<Vn,vn>)$. Partant d'une solution partielle initialement vide $s = ()$, elle cherche à étendre à chaque étape la solution partielle $s = (<V1,v1>...<Vi-1,vi-1>)$ $(i \in n)$ de l'étape précédente. Pour cela, elle détermine la prochaine variable Vi, choisit une valeur vi dans Di et ajoute $<Vi,vi>$ dans s pour obtenir une nouvelle solution partielle $s = (<V1,v1>...<Vi-1,vi-1> <Vi,vi>)$. Ce processus se répète jusqu'à ce que l'on obtienne une solution complète.

Durant la recherche d'une solution, une méthode de construction fait intervenir des heuristiques pour effectuer chacun des deux choix : le choix de la variable suivante et le choix de la valeur pour la variable. Les méthodes de cette classe diffèrent entre elles selon les heuristiques utilisées. En général, les heuristiques portent plus

souvent sur le choix de variables que sur le choix de valeurs car les informations disponibles concernant le premier choix semblent souvent plus riches.
La performance de ces méthodes dépend largement de la pertinence des heuristiques employées, *i.e.*, de leur capacité d'exploiter les connaissances du problème[12].
Un premier type de méthodes de construction est représenté par les *méthodes gloutonnes*. Une méthode gloutonne consiste à fixer à chaque étape la valeur d'une variable sans remettre en cause les choix effectués précédemment [18]. Par exemple, une heuristique gloutonne bien connue pour la coloration introduite par Brélaz est la suivante [12]: pour choisir le noeud suivant à colorier, prendre celui dont les nœuds adjacents sont déjà coloriés avec le plus grand nombre de couleurs différentes et lui assigner la couleur autorisée de plus petit rang possible. Les méthodes gloutonnes sont généralement rapides, mais fournissent le plus souvent des solutions de qualité médiocre [12].

Un deuxième type de méthode de construction est représenté par les méthodes avec *retour arrière*. Une méthode de retour arrière avec une stratégie de recherche en profondeur d'abord consiste à fixer à chaque étape la valeur d'une variable.
Aussitôt qu'un échec est détecté, un retour arrière est effectué, *i.e.*, une ou plusieurs instanciations déjà effectuées sont annulées et de nouvelles valeurs recherchées [12]. Par exemple, un algorithme typique avec retour arrière pour la résolution d'un problème de satisfaction de contraintes cherche à prolonger à chaque étape l'assignation courante de manière consistante. En cas d'échec, un retour arrière est effectué sur la dernière variable instanciée possédant encore des valeurs non essayées. Les méthodes avec retour arrière sont en général complètes et de complexité exponentielle [12]. Pour réduire le nombre de retour arrière (et le temps de recherche), on utilise des techniques de filtrage afin d'anticiper le plus tôt possible les échecs.

Un troisième type de méthode de construction concerne de nombreux algorithmes basés sur le principe de séparation et évaluation progressive [12]. Un exemple typique est l'algorithme A* avec une stratégie « meilleur d'abord » pour la recherche d'un plus court chemin dans un graphe valué. Cet algorithme débute avec un nœud

initial et prolonge ensuite parmi l'ensemble de chemins partiels développés le chemin dont la longueur est la plus faible, en utilisant une estimation de la longueur restante pour calculer la borne inférieure.

Lorsqu'un chemin complet est trouvé, les chemins partiels dont la longueur est supérieure à celle du chemin complet sont éliminés. De manière générale, on utilise des techniques de relaxations pour obtenir des bornes aussi serrées que possible.

3.3. Recherche locale

La recherche locale, appelée aussi la descente ou l'amélioration itérative, représente une classe de méthodes heuristiques très anciennes [8].
Traditionnellement, la recherche locale constitue une arme redoutable pour attaquer des problèmes réputés très difficiles tels que le voyageur de commerce et la partition de graphes [12]. Contrairement à l'approche de construction, la recherche locale manipule des configurations complètes durant la recherche.
Une méthode de recherche locale est un processus itératif fondé sur deux éléments essentiels : un voisinage et une procédure exploitant le voisinage. Plus précisément, elle consiste à :

1) débuter avec une configuration quelconque s de X.
2) choisir un voisin s' de s tel que $f(s') < f(s)$ et remplacer s par s'.
3) répéter jusqu'à ce que pour tout voisin s' de s, $f(s') > f(s)$.

Cette procédure fait intervenir à chaque itération le choix d'un voisin qui améliore la configuration courante. Plusieurs possibilités peuvent être envisagées pour effectuer ce choix. Il est possible d'énumérer les voisins jusqu'à ce qu'on en découvre un qui améliore strictement (première amélioration). On peut également rechercher le meilleur voisin (meilleure amélioration). Cette dernière solution peut sembler plus coûteuse, mais le voisin découvert sera en général de meilleure qualité.
De plus, l'utilisation d'une structure de données appropriée peut souvent permettre de trouver directement ce meilleur voisin.
Comme l'espace des solutions X est fini, cette procédure de descente s'arrête toujours, et la dernière configuration trouvée ne possède pas de voisin strictement

meilleur qu'elle-même. Autrement dit, la recherche locale retourne toujours un optimum local.

L'avantage principal de cette méthode réside dans sa grande simplicité et sa rapidité. Mais les solutions produites sont souvent de qualité médiocre et de coût très supérieur au coût optimal [12]. Pour remédier à ce problème, la solution la plus simple est la *méthode de relance aléatoire* qui consiste à générer une nouvelle configuration de départ de façon aléatoire et à recommencer une descente. On remarque cependant que cette solution ne tire aucun profit des optima locaux déjà découverts. Une autre solution consiste à accepter des voisins de même performance que la configuration courante. Cette approche permet à la recherche de se déplacer sur les plateaux, mais n'est pas suffisante pour ressortir de tous les optima locaux. D'autres raffinements plus élaborés sont également possibles, par exemple : l'introduction de voisinages variables et les techniques de réduction ou d'élargissement [12].
La recherche locale est à la base de métaheuristiques comme la méthode tabou et des méthodes hybrides.

4. Métaheuristiques

Maintenant, nous allons présenter trois grandes classes de métaheuristiques, à savoir les méthodes de voisinage, les algorithmes évolutifs, et les méthodes hybrides.

4.1. Méthodes de voisinage

Les méthodes de voisinage sont fondées sur la notion de voisinage. Nous allons donc introduire d'abord cette notion fondamentale ainsi que quelques notions associées citées par J. K. Hao et al [12].

Définition 1.5 Soit X l'ensemble des configurations admissibles d'un problème, on appelle *voisinage* toute application $N : X \rightarrow 2^X$. On appelle *mécanisme d'exploration* du voisinage toute procédure qui précise comment la recherche passe d'une

configuration $s \in X$ à une configuration $s' \in N(s)$. Une configuration s est un *optimum (minimum) local* par rapport au voisinage N si $f(s) \leq f(s')$ pour toute configuration $s' \in N(s)$.

Une méthode typique de voisinage débute avec une configuration initiale, et réalise ensuite un processus itératif qui consiste à remplacer la configuration courante par l'un de ses voisins en tenant compte de la fonction de coût. Ce processus s'arrête et retourne la meilleure configuration trouvée quand la condition d'arrêt est réalisée. Cette condition d'arrêt concerne généralement une limite pour le nombre d'itérations ou un objectif à réaliser. Un des avantages des méthodes de voisinage réside précisément dans la possibilité de contrôler le temps de calcul : la qualité de la solution trouvée tend à s'améliorer progressivement au cours du temps et l'utilisateur est libre d'arrêter l'exécution au moment qu'il aura choisi.
Les méthodes de voisinage diffèrent essentiellement entre elles par le voisinage utilisé et la stratégie de parcours de ce voisinage. La recherche locale est un exemple simple de cette classe de méthodes.
Un voisinage peut être représenté à l'aide d'un graphe orienté : les nœuds sont les configurations et il existe un arc entre deux nœuds s et s' si s' est voisin de s. Plus précisément, on a la définition suivante :

Définition 1.6 Soit N un voisinage et X l'ensemble des configurations admissibles d'une instance d'un problème. Le graphe (orienté) de l'espace de solutions S induit par N est défini par $G_N = (X, E)$ tel que pour tout $s, s' \in X$, $\langle s, s' \rangle \in E$ si et seulement si $s' \in N(s)$.

A chaque arc $\langle si, sj \rangle \in E$ du graphe, on peut également associer une valeur $c_{ij} = f(s_j) - f(s_i)$ qui correspond à la variation de coût entre s_i et s_j.
Avec cette notion de graphe, une méthode de voisinage peut être vue comme un processus qui parcourt un chemin du graphe. A chaque nœud, le mécanisme de parcours choisit l'arc à parcourir essentiellement en fonction des valeurs des arcs partant du nœud courant. L'efficacité de la méthode dépend donc de deux choses : la structure du graphe déterminée par le voisinage et la façon de parcourir le graphe déterminée par le mécanisme de parcours du voisinage.

Les méthodes de voisinage, telles que nous venons de les présenter, constituent une approche définie de manière extrêmement générale. Dans la suite, nous présentons un ensemble de métaheuristiques fondées sur la notion de voisinage, notamment le recuit simulé, les méthodes d'acceptation avec seuil, le bruitage, la recherche tabou et GRASP.

4.1.1. *Le recuit simulé (Simulated Annealing)*

La méthode de recuit simulé [6] s'inspire du processus de recuit physique. Ce processus utilisé en métallurgie pour améliorer la qualité d'un solide cherche un état d'énergie minimale qui correspond à une structure stable du solide.
En partant d'une haute température à laquelle le solide est devenu liquide, la phase de refroidissement conduit la matière liquide à retrouver sa forme solide par une diminution progressive de la température. Chaque température est maintenue jusqu'à ce que la matière trouve un équilibre thermodynamique. Quand la température tend vers zéro, seules les transitions d'un état à un état d'énergie plus faible sont possibles.
Dans les années 50 pour simuler l'évolution d'un tel processus de recuit physique on utilisa une méthode stochastique pour générer une suite d'états successifs du système en partant d'un état initial donné. Tout nouvel état est obtenu en faisant subir un déplacement (une perturbation) aléatoire à un atome quelconque. Soit ΔE la différence d'énergie occasionnée par une telle perturbation. Le nouvel état est accepté si l'énergie du système diminue ($\Delta E \leq 0$).
Sinon, il est accepté avec une probabilité définie par : $p(\Delta E,T) = \exp(-\Delta E /(C_b \times T))$ où T est la température du système et C_b une constante physique connue sous le nom de *constante de Boltzmann*.
A chaque étape, l'acceptation ou non d'un nouvel état dont l'énergie est supérieure à celle de l'état courant est déterminée de manière probabiliste : un réel $0 \leq q < 1$ est tiré aléatoirement et ensuite comparé avec $p(\Delta E,T)$. Si $q \leq p(\Delta E,T)$, alors le nouvel état est accepté pour remplacer l'état courant, sinon, l'état courant est maintenu.
Après un grand nombre de perturbations, un tel processus fait évoluer le système vers un état d'équilibre thermodynamique selon la *distribution de Boltzmann* qui est

définie par la probabilité de se trouver dans un état d'énergie E :
Pr(E) = c(T) x exp(- E/(Cb x T)) où c(T) est un facteur de normalisation.

Le recuit simulé peut être vu comme une version étendue de la méthode de descente. Le processus du recuit simulé répète une procédure itérative qui cherche des configurations de coût plus faible tout en acceptant de manière contrôlée des configurations qui dégradent la fonction de coût. A chaque nouvelle itération, un voisin $s' \in N(s)$ de la configuration courante s est généré de manière aléatoire. Selon les cas, ce voisin sera soit retenu pour remplacer celle-ci, soit rejeté. Si ce voisin est de performance supérieure ou égale à celle de la configuration courante, *i.e.*, $f(s') \leq f(s)$, il est systématiquement retenu. Dans le cas contraire, *s'est* accepté avec une probabilité $p(\Delta f, T)$ qui dépend de deux facteurs : d'une part l'importance de la dégradation Δf = f(s') - f(s*)* (les dégradations plus faibles sont plus facilement acceptées), d'autre part un paramètre de contrôle T, la température (une température élevée correspond à une probabilité plus grande d'accepter des dégradations).
La température est contrôlée par une fonction décroissante qui définit un schéma de refroidissement. Les deux paramètres de la méthode définissent la longueur des paliers et la fonction permettant de calculer la suite décroissante des températures.

En pratique, l'algorithme s'arrête et retourne la meilleure configuration trouvée lorsque aucune configuration voisine n'a été acceptée pendant un certain nombre d'itérations à une température ou lorsque la température atteint la valeur zéro.
La performance du recuit simulé dépend largement du schéma de refroidissement utilisé. De nombreux schémas théoriques et pratiques ont été proposés. De manière générale, les schémas de refroidissement connus peuvent être classés en trois catégories [12]:

- **réduction par paliers** : chaque température est maintenue égale pendant un certain nombre d'itérations, et décroît ainsi par paliers.
- **réduction continue**: la température est modifiée à chaque itération.
- **réduction non-monotone**: la température décroît à chaque itération avec des augmentations occasionnelles.

Le recuit simulé constitue, parmi les méthodes de voisinage, l'une des plus anciennes et des plus populaires. Il a acquis son succès essentiellement grâce à des résultats pratiques obtenus sur de nombreux problèmes NP-difficiles [12].

4.1.2. Les algorithmes d'acceptation avec seuil (Threshold algorithms)

Ces algorithmes sont des variantes du recuit simulé: la différence se situe au niveau de l'acceptation de dégradation à chaque étape. Dans un algorithme d'acceptation avec seuil, une telle décision est prise de manière déterministe. A chaque itération *k*, l'acceptation d'un voisin $s' \in N(s)$ se base uniquement sur une fonction auxiliaire r(s',s) et un seuil T_K : s' est accepté si $r(s',s) < T_k$. La fonction r(s',s) et le seuil T_k peuvent être définis de nombreuses manières. Dans le cas le plus simple, la fonction r(s',s) est définie par $\Delta f = f(s') - f(s)$. Le paramètre de seuil est défini de manière analogue à la température T du recuit simulé : il est initié à une valeur élevée puis décroît progressivement à chaque fois qu'un nombre prédéterminé d'itérations est effectué. Les seuils ainsi générés correspondent à une suite de valeurs positives décroissantes $T_1 \geq T_2 \geq \ldots \geq T_{k-1} \geq T_K \geq 0$ et $T_K \rightarrow 0$. L'idée est de diminuer petit à petit la chance d'accepter des configurations qui dégradent la fonction de coût.

Quand T_K tend vers 0, l'algorithme réalise une recherche de descente aléatoire.
Ces algorithmes ont été utilisés pour résoudre des problèmes d'optimisation et ont obtenu des résultats intéressants [12]. La difficulté essentielle de cette approche se situe au niveau de la détermination des seuils pour une application donnée.

4.1.3. Méthode du bruitage

La méthode de bruitage [12] a été introduite sur des problèmes dont la donnée comporte un ensemble de nombres réels, par exemple, le voyageur de commerce (arêtes valuées par des réels) et le problème d'affectation quadratique (matrices de réels). Cette méthode fait appel à une notion de bruitage de la donnée qui est définie de la façon suivante : la donnée bruitée est produite à partir de la donnée initiale en ajoutant à chacun des réels une composante calculée comme le produit de trois éléments :

a) une fonction aléatoire à valeurs sur l'intervalle [0,1],
b) un paramètre permettant de contrôler le niveau du bruit,
c) le plus grand des réels concernés, afin de normaliser le niveau du bruit par rapport à la donnée.

L'exécution de l'heuristique comporte plusieurs étapes. Chaque étape consiste à calculer un bruitage de la donnée, puis à effectuer une descente prenant en compte la fonction de coût calculée à partir de la donnée bruitée. Le niveau du bruit est décrémenté au début de chaque nouvelle étape, et la descente s'effectue à partir de la configuration résultant de l'étape précédente. Il existe deux variantes pour le bruitage :

1) chaque descente sur la donnée bruitée est suivie par une descente effectuée sur la donnée non bruitée. L'objectif consiste à mieux tenir compte de la donnée réelle puisqu'un véritable optimum local est alors atteint.
2) la configuration courante est régulièrement remplacée par la meilleure configuration obtenue depuis le début.

Cette méthode a été appliquée avec succès au problème du voyageur de commerce et a obtenu de très bons résultats [12].

4.1.4. GRASP

La méthode GRASP (pour *Greedy Randomized Adaptive Search Procedure*) est une procédure itérative. A proprement parler, GRASP est une méthode hybride car elle cherche à combiner les avantages des heuristiques gloutonnes, de la recherche aléatoire et des méthodes de voisinage. Un algorithme GRASP répète un processus composé de deux étapes : la construction d'une solution suivie par une descente pour améliorer la solution construite.

Durant l'étape de construction, une solution est itérativement construite : chaque itération ajoute un élément dans la solution partielle courante. Pour déterminer l'élément qui sera ajouté, on utilise une liste des meilleurs candidats obtenus avec

une fonction gloutonne et on prend *au hasard* un élément dans cette liste. La liste des meilleurs candidats est dynamiquement mise à jour après chaque itération de construction. Cette étape de construction continue jusqu'à ce qu'une solution complète soit obtenue.

A partir de cette solution, une descente est appliquée pour améliorer la solution. Une procédure GRASP répète ces deux étapes et retourne à la fin la meilleure solution trouvée. Les deux paramètres de cette méthode sont donc la longueur de la liste de candidats et le nombre d'itérations autorisées.

4.1.5. La recherche tabou

La méthode tabou a été développée par Glover et indépendamment par Hansen [12]. Cette méthode fait appel à un ensemble de règles et de mécanismes généraux pour guider la recherche de manière intelligente au travers de l'espace des solutions.
A l'inverse du recuit simulé qui génère de manière aléatoire une seule solution voisine $s' \in N(s)$ à chaque itération, tabou examine un échantillonnage de solutions de $N(s)$ et retient la meilleure s' même si s'*est* plus mauvaise que s. La recherche tabou ne s'arrête donc pas au premier optimum trouvé. Cependant, cette stratégie peut entraîner des cycles, par exemple un cycle de longueur 2 : $s \rightarrow s' \rightarrow s \rightarrow s'$... Pour empêcher ce type de cycle, on mémorise les k dernières configurations visitées dans une mémoire à court terme et on interdit tout mouvement qui conduit à une de ces configurations. Cette mémoire est appelée la *liste tabou*, une des composantes essentielles de cette méthode. Elle permet d'éviter tous les cycles de longueur inférieure ou égale à k. La valeur de k dépend du problème à résoudre et peut éventuellement évoluer au cours de la recherche.

La mémorisation de configurations entières serait trop coûteuse en temps de calcul et en place mémoire et ne serait sans doute pas la plus efficace. En fait, la liste tabou mémorise des *caractéristiques* des configurations au lieu de configurations complètes. Plus précisément, lorsqu'un mouvement vient d'être effectué, c'est généralement la caractéristique perdue par la configuration courante qui devient tabou. Par exemple, avec la structure vectorielle de configuration définie

précédemment pour les problèmes d'affectation sous contraintes, un type de caractéristique très simple est le couple <*V,v*>, *V* et *v* étant respectivement une variable et une valeur possible pour la variable. Supposons qu'un mouvement qui affecte la valeur *v* à la variable *V* en remplacement de la valeur *v0* soit effectué, la caractéristique <*V,v0*> est alors mémorisée et interdite pour les *k* itérations suivantes. Autrement dit, la variable *V* qui vient d'abandonner la valeur *v0* ne pourra pas reprendre cette valeur pendant cette durée. Le résultat est que non seulement la configuration courante ne pourra pas réapparaître lors des *k* prochaines itérations, mais que de nombreuses autres configurations seront également interdites.

Lorsque les listes tabou font intervenir des caractéristiques de modifications, les interdictions qu'elles engendrent peuvent s'avérer trop fortes et restreindre l'ensemble des solutions admises à chaque itération d'une manière jugée trop brutale. Un mécanisme particulier, appelé *l'aspiration*, est mis en place afin de pallier cet inconvénient. Ce mécanisme permet de lever le statut tabou d'une configuration, sans pour autant introduire un risque de cycles dans le processus de recherche. La fonction d'aspiration peut être définie de plusieurs manières.
La fonction la plus simple consiste à révoquer le statut tabou d'un mouvement si ce dernier permet d'atteindre une solution de qualité supérieure à celle de la meilleure solution trouvée jusqu'alors.

Il existe d'autres techniques intéressantes pour améliorer la puissance de la méthode tabou, en particulier, *l'intensification* et *la diversification*. Toutes les deux se basent sur l'utilisation d'une mémoire à long terme et se différencient selon la façon d'exploiter les informations de cette mémoire.

L'intensification se fonde sur l'idée d'apprentissage de propriétés favorables : les propriétés communes souvent rencontrées dans les meilleures configurations visitées sont mémorisées au cours de la recherche, puis favorisées pendant la période d'intensification. Une autre manière d'appliquer l'intensification consiste à mémoriser une liste de solutions de bonne qualité et à retourner vers une des ces solutions.

La diversification a un objectif inverse de l'intensification : elle cherche à diriger la recherche vers des zones inexplorées. Sa mise en œuvre consiste souvent à modifier temporairement la fonction de coût pour favoriser des mouvements n'ayant pas été effectués ou à pénaliser les mouvements ayant été souvent répétés.
L'intensification et la diversification jouent donc un rôle complémentaire. Comme pour la plupart des méthodes heuristiques, il n'existe pas de résultats théoriques garantissant la convergence d'une procédure tabou vers un optimum global. La raison principale de cet état de fait tient à la nature même de la méthode. Celle-ci étant hautement adaptative et modulable, son analyse par les outils mathématiques est rendue plus difficile [12]. Des résultats pratiques très intéressants ont été obtenus pour de nombreux problèmes d'optimisation combinatoire, et en particulier, pour des problèmes d'affectation sous contraintes.
Cette méthode se caractérise par sa stratégie agressive de recherche (choix d'un des meilleurs mouvements à chaque itération) combinée avec différentes possibilités permettant de s'adapter au problème et d'intégrer des connaissances spécifiques. Cela demande naturellement un effort particulier d'adaptation de la méthode.

4.2. Algorithmes évolutifs

Le terme « algorithmes évolutifs » englobe une autre classe assez large de métaheuristiques. Ces algorithmes sont basés sur le principe du processus d'évolution naturelle [10]. Les algorithmes évolutifs doivent leur nom à l'analogie avec les mécanismes d'évolution des espèces vivantes.
Un algorithme évolutif typique est composé de trois éléments essentiels :

1) une *population* constituée de plusieurs individus représentant des solutions potentielles (configurations) du problème donné.
2) un *mécanisme d'évaluation* de l'adaptation de chaque individu de la population à l'égard de son environnement extérieur.
3) un *mécanisme d'évolution* composé d'opérateurs permettant d'éliminer certains individus et de produire de nouveaux individus à partir des individus sélectionnés.

Du point de vue opérationnel, un algorithme évolutif typique débute avec une population initiale souvent générée aléatoirement et répète ensuite un cycle

d'évolution composé de 3 étapes séquentielles :

1) mesurer l'adaptation (la qualité) de chaque individu de la population par le mécanisme d'évaluation.
2) sélectionner une partie des individus.
3) produire de nouveaux individus par des recombinaisons d'individus sélectionnés.

Ce processus se termine quand la condition d'arrêt est vérifiée, par exemple, quand un nombre maximum de cycles (générations) ou un nombre maximum d'évaluations est atteint. Selon l'analogie de l'évolution naturelle, la qualité des individus de la population devrait tendre à s'améliorer au fur et à mesure du processus.

Parmi les composantes d'un algorithme évolutif, l'individu et la fonction d'adaptation correspondent respectivement à la notion de configuration et à la fonction d'évaluation dans les méthodes de voisinage. La notion de mécanisme d'évolution est proche de celle du mécanisme de parcours du voisinage ; en particulier, on peut

interpréter une itération d'une méthode de voisinage comme un mécanisme d'évolution portant sur une population réduite à un seul individu.

Cependant, les opérateurs sont sensiblement différents. En effet, un algorithme évolutif comporte un ensemble d'opérateurs tels que la sélection, la mutation et éventuellement le croisement.

La *sélection* a pour objectif de choisir les individus qui vont pouvoir survivre et/ou se reproduire pour transmettre leurs caractéristiques à la génération suivante.
La sélection se base généralement sur le principe de conservation des individus les mieux adaptés et d'élimination des moins adaptés. Le *croisement* ou recombinaison cherche à combiner les caractéristiques des individus parents pour créer des individus enfants avec de nouvelles potentialités dans la génération future. La *mutation* effectue de légères modifications de certains individus.
Les algorithmes évolutifs ont été appliqués à de très nombreux problèmes complexes

et difficiles, y compris des problèmes d'optimisation (continue ou combinatoire). D'une manière générale, on peut distinguer trois grandes familles d'algorithmes évolutifs [12] : les algorithmes génétiques, la programmation évolutive et les stratégies d'évolution. Ces méthodes se différencient par leur manière de représenter les données et par leur façon de faire évoluer la population d'une génération à l'autre.

4.2.1. Algorithmes génétiques

Les algorithmes génétiques classiques introduits par Holland s'appuient fortement sur un *codage universel* sous forme de chaînes 0/1 de longueur fixe et un ensemble d'opérateurs *génétiques* : la mutation, l'inversion et le croisement [10]. Un individu sous ce codage, appelé un chromosome, représente une configuration du problème. Les opérateurs « génétiques » sont définis de manière à opérer aléatoirement sur un ou deux individus sans aucune connaissance sur le problème.

Le *croisement* permet de produire deux nouveaux individus (enfants) à partir de deux individus (parents). Par exemple, le croisement bi-points consiste à choisir aléatoirement deux points de croisement et à échanger les segments des deux parents déterminés par ces deux points. Le croisement réalise donc uniquement des recombinaisons de valeurs (gènes) existantes entre deux parents et ne permet pas d'introduire de nouvelles valeurs dans les individus enfants. Pour cela, on applique la mutation. L'opérateur de *mutation* consiste à changer aléatoirement la valeur de certaines variables dans un individu. Dans les algorithmes génétiques la mutation est considérée comme un opérateur secondaire par rapport au croisement. Un cycle d'évolution complet d'un algorithme génétique est formé par l'application des opérateurs de sélection, croisement et mutation sur une population d'individus.

Bien que les algorithmes génétiques soient considérés aujourd'hui comme une méthode d'optimisation, l'objectif initial consistait à concevoir des systèmes d'apprentissage généraux, robustes et adaptatifs, applicables à une large classe de problèmes. En particulier, Holland s'est intéressé à l'élaboration d'une technique de programmation permettant l'évolution des programmes informatiques par reproduction, croisement et mutation. Cette motivation est à l'origine de l'universalité

des algorithmes génétiques : ni le codage ni les opérateurs génétiques ne demandent des connaissances spécifiques du problème. Selon ce principe, pour un problème donné, il suffit de trouver une transformation des paramètres du problème en chaînes 0/1 et d'appliquer les opérateurs génétiques sur une population de solutions potentielles ainsi codées. C'est grâce à cette universalité que Holland a pu réaliser des analyses théoriques sur les algorithmes génétiques conduisant à la théorie des schémas et à la caractérisation du rôle et de l'importance du croisement.

L'universalité d'un tel algorithme pose évidemment des problèmes d'efficacité en pratique. En effet, en tant que méthode d'optimisation, un algorithme génétique classique se base uniquement sur des opérateurs « aveugles » et est donc rarement en mesure de produire des résultats comparables à ceux d'une méthode de voisinage[12].

Une technique pour remédier à ce problème consiste à *spécialiser* l'algorithme génétique au problème donnée. Plus précisément, à la place des opérateurs aléatoires, la mutation et le croisement sont adaptés en se basant sur des connaissances spécifiques du problème. De cette manière, la recherche est mieux guidée et donc plus efficace. Il est désormais établi que pour être efficace en optimisation, il est indispensable d'intégrer des connaissances du problème [12].Une autre voie intéressante pour améliorer l'efficacité des algorithmes génétiques consiste à combiner le cadre génétique avec d'autres méthodes de résolution. Ce point sera développé ultérieurement dans la section sur les méthodes hybrides.
Les algorithmes génétiques spécialisés ou hybrides ont été appliqués à de nombreuses applications dans des domaines très variés. Quant aux algorithmes génétiques purs, leurs résultats en optimisation combinatoire sont en général faibles [12].

4.2.2. Programmation évolutive

La programmation évolutive s'appuie sur un codage approprié du problème à résoudre et sur les opérations de mutation adaptées au codage [12]. Le codage d'un tel algorithme dépend du problème à résoudre. Par exemple, pour un problème

d'optimisation dans le domaine des réels, les individus d'une population seraient des vecteurs de réels. De manière similaire, une permutation serait utilisée pour représenter un tour dans le problème du voyageur de commerce. A partir d'un codage donné pour un problème, la mutation ou opérateur d'évolution spécifique sera définie. Par exemple, dans le cas du problème du voyageur de commerce, la mutation basée sur le codage de permutation pourrait être l'opération d'inversion : prendre deux villes dans le tour et inverser l'ordre du segment défini par les deux villes. Ainsi l'analogie est forte avec les méthodes de voisinage : une mutation correspond à un mouvement dans un algorithme de voisinage.
Un cycle d'évolution typique pour la programmation évolutive est le suivant : chaque configuration de la population courante est copiée dans une nouvelle population.

Les configurations sont ensuite mutées, conduisant à de nouvelles configurations.
L'ensemble des configurations entre ensuite dans une étape de compétition pour survivre dans la génération suivante.
La programmation évolutive a été initialement introduite pour simuler l'intelligence qui est définie sur l'hypothèse suivante : la caractéristique principale de l'intelligence est la capacité d'adaptation comportementale d'un organisme à son environnement [12]. Selon un modèle très simpliste, cette tâche de simulation revient à prédire une séquence de symboles appartenant à un alphabet fini à partir des séquences déjà observées. Dans ce but, des automates d'états finis sont choisis pour représenter les individus d'une population. Ainsi, à chaque automate de la population est donnée une série de symboles pris dans une séquence déjà observée et la sortie de l'automate est mesurée par rapport au résultat déjà connu. Cette mesure constitue l'adaptation de l'individu. L'étape suivante consiste à créer, pour chaque individu, un individu enfant par une mutation aléatoire de l'individu parent. La mutation est assurée par une des 5 opérations suivantes : changer le symbole d'une sortie, changer un état de transition, ajouter ou supprimer un état et changer l'état initial.

4.2.3. Stratégies d'évolution

Les stratégies d'évolution sont conçues dès le départ pour résoudre des problèmes d'optimisation continus [12]. Dans un algorithme SE, les individus sont des points

(vecteurs de réels). Comme la programmation évolutive, les SE n'utilisent que la mutation et la sélection.
L'algorithme le plus simple, noté (1+1)-ES, manipule un seul individu. A chaque génération (itération), l'algorithme génère par mutation un individu enfant à partir de l'individu parent et sélectionne l'un ou l'autre pour le conserver dans la population (selon l'adaptation de chaque individu). Le processus s'arrête quand la condition d'arrêt est vérifiée, définie souvent par le nombre d'itérations, le temps de calcul réalisé ou l'écart entre deux individus de deux itérations successives.

La *mutation* dans un tel algorithme est aléatoirement appliquée à tous les composants de l'individu pour produire un enfant. Cette mutation fonctionne selon les principes suivants : un enfant ressemble à ses parents, et plus (moins) un changement est important, moins (plus) la fréquence de changement est élevée.
Cet algorithme (1+1)-ES se généralise en un algorithme (m+l)-ES qui signifie que m parents génèrent l enfants à chaque génération et qu'une sélection ramène ensuite la population de m+l individus à m individus. De plus, la recombinaison a été également introduite dans ces algorithmes [12].

4.3. Méthodes hybrides et autres

Le mode d'hybridation qui semble le plus fécond concerne la combinaison entre les méthodes de voisinage et l'approche d'évolution [12]. L'idée essentielle de cette hybridation consiste à exploiter pleinement la puissance de recherche de méthodes de voisinage et de recombinaison des algorithmes évolutifs sur une population de solutions. Un tel algorithme utilise une ou plusieurs méthodes de voisinage sur les individus de la population pendant un certain nombre d'itérations ou jusqu'à la découverte d'un ensemble d'optima locaux et invoque ensuite un mécanisme de recombinaison pour créer de nouveaux individus. Comme pour les algorithmes génétiques spécialisés, la recombinaison doit impérativement être adaptée au problème traité.
Cette approche a permis de produire d'excellents voire les meilleurs résultats sur des *benchmarks* réputés de problèmes de référence [12].
Les algorithmes hybrides sont sans doute parmi les méthodes les plus puissantes.

Malheureusement, les temps de calcul nécessaires peuvent devenir prohibitifs à cause du nombre d'individus manipulés dans la population. Une voie pour résoudre ce problème est la parallélisation de ces algorithmes sur des machines parallèles ou sur des systèmes distribués.

Nous venons d'évoquer l'hybridation d'un algorithme évolutionniste avec des méthodes de voisinage. Il est également possible d'hybrider d'autres types d'approches, par exemple une heuristique gloutonne et une méthode de voisinage ou un algorithme évolutif.

5. Analyse des métaheuristiques

Dans cette section, nous proposons quelques éléments d'analyse des métaheuristiques,. Enfin, nous donnons quelques indications pratiques pour faciliter le choix d'une métaheuristique.

5.1. Exploitation et exploration

Les inventeurs des algorithmes génétiques ont introduit les notions de l'exploitation et de l'exploration [12]. L'exploitation insiste sur la capacité d'examiner en profondeur par une méthode des zones de recherche particulières alors que l'exploration met en avant la capacité de découvrir des zones de recherche prometteuses. Ces deux notions complémentations concernent au fait l'ensemble des méthodes de recherche. Il est donc pertinent d'analyser l'ensemble des métaheuristiques en fonction de ces deux notions.

Les méthodes de voisinage entendent exploiter les bonnes propriétés de la fonction de voisinage pour découvrir rapidement des configurations de bonne qualité.

Elles reposent sur l'hypothèse que les zones les plus prometteuses sont situées à proximité des configurations (déjà visitées) qui sont les plus performantes. Le principe d'exploitation consiste à examiner en priorité ces zones.

Dans les algorithmes génétiques, la sélection a pour effet de concentrer la recherche autour des configurations de meilleure performance. Dans les méthodes de voisinage, l'exploitation est principalement assurée par la préférence accordée à une configuration de bonne performance dans la procédure de réparation.

Plusieurs méthodes introduisent des mécanismes spécifiques supplémentaires d'exploitation : c'est le cas de l'intensification dans la méthode tabou.

Cependant, l'application systématique du seul principe d'exploitation ne permet pas une recherche efficace [12]. En effet, l'exploitation conduit à confiner la recherche dans une zone limitée qui finit par s'épuiser. Le cas de l'amélioration itérative rapidement piégée dans un optimum local illustre cruellement ce phénomène. Une autre illustration souvent évoquée est fournie par le problème de convergence prématurée des algorithmes génétiques : du fait de la sélection, la population finit par n'être constituée que d'individus tous similaires. L'une des préoccupations majeures dans les algorithmes génétiques consiste d'ailleurs à préserver le plus longtemps possible une diversité suffisante dans la population. Face à ce type de difficulté, la solution consiste à diriger la poursuite de la recherche vers de nouvelles zones, *i.e.*, à recourir à l'exploration.

En plus de la relance, les heuristiques emploient principalement deux autres stratégies dans le but d'explorer : la première consiste à perturber aléatoirement et la seconde à caractériser les régions visitées pour pouvoir ensuite s'en éloigner.

La première stratégie qui est aussi la plus simple consiste à introduire des perturbations aléatoires : c'est le cas pour les mutations aléatoires dans les algorithmes génétiques ainsi que pour la génération aléatoire d'un voisin dans le recuit simulé. Dans les deux cas, la configuration courante est altérée de manière aléatoire et un mécanisme d'acceptation est appliqué *a posteriori*. Ces deux méthodes admettent donc en permanence des perturbations aléatoires. Dans la méthode GSAT, le mécanisme de base repose au contraire sur une règle beaucoup plus déterministe (dite agressive) consistant à choisir systématiquement le meilleur voisin. Une manière efficace d'améliorer GSAT de base est l'option « *random walk* » qui consiste à alterner les mouvements de base et des mouvements totalement aléatoires.

La deuxième stratégie pour explorer consiste à mémoriser au cours de la recherche

des caractéristiques des régions visitées et à introduire un mécanisme permettant de s'éloigner de ces zones. C'est ce que fait la méthode tabou avec la liste tabou (court terme) et avec la diversification (long terme). La recherche locale guidée reprend cette même idée. Citons également la pondération qui utilise une variante de cette idée. Nous pouvons remarquer que cette seconde stratégie nécessite l'utilisation d'une mémoire.

Notons enfin que la recombinaison est parfois présentée comme une manière supplémentaire de contribuer à l'exploration.

Nous venons de voir qu'il existe différents modes d'exploration. Les métaheuristiques se différencient également selon la manière dont elles font varier l'intensité de l'exploration au cours de la recherche. Le recuit simulé présente une manière spécifique de procéder : le niveau des perturbations aléatoires (responsables de l'exploration), lié au paramètre de température, est initialement fixé à un niveau élevé et abaissé progressivement au cours de la recherche. Le même principe de perturbations aléatoires décroissantes se retrouve dans les algorithmes d'acceptation avec seuil et, sous une autre forme, dans la méthode de bruitage. La méthode tabou illustre une manière très différente de doser l'exploration : un mécanisme particulier d'exploration (la diversification) succède régulièrement à de longues phases de recherche.

La recombinaison constitue un autre principe général qui complète l'exploitation et l'exploration. Elle consiste à construire de nouvelles configurations en combinant la structure de deux ou plusieurs bonnes configurations déjà trouvées. Cette idée issue des algorithmes génétiques (le croisement) a été introduite dans des méthodes de voisinage, par exemple dans GSAT et dans tabou (« *relinking paths* »). Des travaux récents ont montré qu'une recombinaison appropriée de solutions peut améliorer sensiblement l'efficacité de la recherche. Rappelons également que ce principe est aussi la base principale des algorithmes hybrides qui allient une méthode de voisinage et la recombinaison.

5.2. Méthodes générales et méthodes spécifiques

Contrairement aux algorithmes traditionnels dédiés à un problème spécifique, les métaheuristiques constituent des mécanismes très généraux qui peuvent être adaptés pour traiter de nombreux problèmes différents. Pour expliquer l'efficacité d'un algorithme spécifique, on invoque souvent le fait qu'il utilise des connaissances spécifiques du problème. Pour expliquer l'efficacité d'une métaheuristique, deux types d'arguments opposés sont généralement avancés.

Selon certains, des mécanismes généraux suffisamment puissants ont par eux mêmes la faculté de mener efficacement la recherche sans disposer d'information spécifique du problème considéré (contexte de boîte noire) : c'était notamment le point de vue classique porté sur les algorithmes génétiques [10].
Malheureusement, de même qu'il ne peut pas exister de stratégie avantageuse dans un jeu de hasard comme la roulette, il existe également des limitations théoriques fondamentales qui ruinent les espoirs d'une méthode aveugle dans le cas le plus général.

Selon le point de vue opposé, la puissance d'une métaheuristique est d'abord liée à son aptitude à intégrer des connaissances spécifiques du problème [12]. La connaissance du problème la plus fondamentale réside dans le codage du problème et dans le choix de la fonction de voisinage. Plusieurs auteurs insistent sur l'importance du codage du problème.

Les métaheuristiques tentent également d'améliorer leur efficacité en incorporant des connaissances supplémentaires dans leurs opérateurs. Plus les opérateurs d'une méthode utilisent des connaissances spécifiques, plus la méthode dispose de moyens potentiels pour conduire efficacement la recherche. En contrepartie, l'intégration de ces connaissances spécifiques (en supposant que ces connaissances soient disponibles) nécessite un effort pour spécialiser ou adapter la méthode. La méthode tabou vise à incorporer le plus possible de connaissances du problème pour atteindre le maximum d'efficacité : l'utilisateur doit notamment définir de façon pertinente le type de caractéristique figurant dans la liste tabou. De leur coté, les

algorithmes génétiques se sont éloignés du modèle standard pour intégrer également des connaissances du problème : codage non binaire, opérateurs spécifiques. Au contraire, le recuit simulé est parfois présenté comme une méthode facile à adapter à un problème et qui ne tente pas d'exploiter de connaissances spécifiques.

En général, une méthode offrant des possibilités d'intégrer des connaissances du problème a plus de chance de produire de bons résultats, mais demande un effort d'adaptation et de spécialisation. Au contraire, une méthode très générale qui prétend n'intégrer aucune connaissance propre ne peut pas être compétitive.

5.3. Quelle métaheuristique utiliser ?

Le premier problème pratique qui se pose à un utilisateur confronté à une application concrète est d'effectuer un choix parmi les différentes métaheuristiques disponibles. Ce choix est d'autant plus difficile qu'il n'existe pas de comparaison générale et fiable des différentes métaheuristiques. Cependant, il est possible de caractériser les métaheuristiques selon quelques critères généraux, ce qui pourrait faciliter ce choix.

Dans l'analyse réalisée par J. K. Hao [12] une synthèse décrite par le tableau ci-dessous met en relation cinq critères avec cinq métaheuristiques parmi les plus représentatives. Il doit être clair que le choix d'une métaheuristique appropriée ne constitue qu'une condition nécessaire. La qualité des solutions trouvées par une méthode peut être très variable selon l'implémentation réalisée.

	AI	RS	tabou	AG	AH
Simplicité	0	-	-	-	- -
Facilité d'adaptation	0	0	-	-	-
Connaissance	0	0	+	+	+
Qualité	0	+	+ +	+	++ (+++)
Rapidité	0	-	-	- -	- -

Tableau 1. *Comparaison générale des principales métaheuristiques*

Dans ce tableau, les cinq métaheuristiques comparées sont les suivantes :

- AI : amélioration itérative (descente) avec relance,
- RS : recuit simulé,
- tabou : méthode tabou,
- AG : algorithme génétique adapté,
- AH : algorithme hybride.

Les critères de comparaisons retenus sont les suivants :

- simplicité de la métaheuristique, *i.e.* simplicité de la méthode elle-même,
- facilité d'adaptation au problème,
- possibilité d'intégrer des connaissances spécifiques du problème,
- qualité des meilleures solutions trouvées,
- rapidité, *i.e.*, le temps de calcul nécessaire pour trouver une telle solution (sur une machine séquentielle).

La méthode d'amélioration itérative est utilisée comme point de référence pour l'ensemble des méthodes : les signes -, 0, + indiquent des performances respectivement inférieures, égales, supérieures à celles obtenues par l'amélioration itérative.

Le critère de *qualité* utilisé dans le tableau correspond à la meilleure qualité qu'il est possible d'obtenir par une exécution prolongée. Le critère de *rapidité* représente le temps de calcul typiquement nécessaire pour obtenir une telle solution.

5.4. Bilan

Les métaheuristiques présentent de nombreux avantages mais aussi un certain nombre de limitations. Nous résumons ci-dessous les différentes caractéristiques de ces méthodes citées dans [12]:

- généralité et application possible à une large classe de problèmes,
- efficacité pour de nombreux problèmes,
- possibilité de compromis entre qualité des solutions et temps de calcul,

- existence de preuves de convergence asymptotique pour certaines méthodes,
- optimum non garanti,
- nécessité d'adaptation de la méthode au problème traité,
- nécessité de réglage des paramètres,
- difficulté de prévoir la performance.

Tout d'abord, rappelons qu'il s'agit d'heuristiques dont l'objectif n'est pas le même que celui des méthodes exactes. En particulier, ces méthodes ne garantissent pas de trouver une solution optimale ni de prouver l'optimalité de la solution trouvée – sauf si une borne inférieure de la fonction de coût est connue et qu'une solution trouvée atteint cette borne.
Dans ces conditions, méthodes complètes et métaheuristiques sont plutôt complémentaires que véritablement concurrentes. Ces deux approches peuvent d'ailleurs également coopérer de différentes manières : l'une des plus courantes consiste à utiliser une méthode de voisinage dans une méthode complète afin de déterminer une borne ou de fixer la valeur d'une variable.

6. Conclusion

Dans ce chapitre nous avons dans un premier temps défini des notions de base sur les problèmes d'optimisation combinatoire et pour finir les différentes méthodes utilisées pour les résoudre.
Nous avons constaté que les métaheuristiques constituent une classe de méthodes approchées adaptables à un très grand nombre de problèmes. Elles ont révélé leur grande efficacité pour fournir des solutions approchées de bonne qualité pour un grand nombre de problèmes d'optimisation classiques et d'applications réelles de grande taille. C'est pourquoi l'étude de ces méthodes est actuellement en plein développement.
Afin de résoudre des instances de taille et de difficulté croissantes, il faut mettre au point des méthodes toujours plus puissantes. Pour atteindre cet objectif, au moins deux voies privilégiées se développent : l'hybridation de méthodes et la parallélisation.

Chapitre 2
Etat de l'Art

Le présent chapitre traite le problème de la génération automatique d'emploi du temps. Nous présenterons les différentes formulations du problème trouvées dans la littérature.

Nous étudierons la faisabilité, l'optimalité et la complexité du problème (§.2).

La problématique de la génération automatique d'emploi du temps sera présentée comme un problème de recherche dans un premier temps (§.3) et comme un problème d'optimisation dans un second (§.4).

Nous verrons ensuite les différentes variantes du problème d'emploi du temps trouvées dans la littérature (§.5) et Pour finir nous étudierons les différentes méthodes et techniques trouvées pour le résoudre. Parmi l'ensemble des méthodes et techniques nous nous intéresserons plus particulièrement aux algorithmes génétiques.

Introduction

De nombreux auteurs croient que le problème des emplois du temps ne peut être complètement automatisé. La raison en est double: d'une part, il ya des raisons qui font qu'un emploi du temps soit meilleur qu'un autre qui ne peuvent facilement être exprimées dans un système automatique. D'autre part, l'espace de recherche est généralement énorme, une intervention humaine peut aider vers la recherche prometteuses que le système par lui-même peut-être pas en mesure de trouver.

La littérature contient un grand nombre de variantes du problème calendrier. Ces variantes diffèrent les unes des autres par le type d'institution concernée (université ou haute école) et par le type des contraintes. Les annotations bibliographiques trouvées dans les articles étudiés du problème de calendrier dressent la liste de nombreux documents qui ont apparu avant 1980. Schaerf [17] dans une étude donne un aperçu des différentes formulations du problème de calendrier et les classifie en trois grandes classes: calendrier pour école, calendrier pour examen et calendrier pour cours universitaires. Bien entendu, cette classification est brut, et il ya beaucoup de problèmes de calendrier dans le monde réel qui n'appartient pas ces trois catégories.

1. Différents problèmes et formulations

Le problème du calendrier scolaire de base est également connu sous le nom du modèle classe-enseignant (class-teacher model). Une classe est un groupe d'étudiants qui on le même ensemble de cours et généralement demeurent ensemble tout au long de la semaine. Le problème le plus simple consiste à l'assignation de séances (leçons) à des périodes de telles manière qu'aucun enseignant ou classe ne soit impliqué dans plus d'une séance à la fois.

La problématique majeure dans cette version simplifier du problème de calendrier scolaire est quand les enseignants ou les classes ont plus de séances que de créneaux disponibles.

D'un autre coté, on a les calendriers universitaires qui peuvent être classifié en deux grandes catégories. Calendrier pour examen et calendrier pour cours. La différence principale entre les deux types de calendrier est qu'on peut organiser plusieurs examens dans une même salle. Schaerf [17] donne une formulation par programmation linéaire au problème du calendrier pour examen et décrit des variantes alternatives.

Le problème du calendrier universitaire consiste en la planification d'un ensemble de cours dans un certain nombre de salles et de périodes. La principale différence entre le problème de calendrier universitaire et celui de l'école est que l'université peut avoir des cours qui regroupent des étudiants appartenant à des groupes différents. Si deux cours ont des étudiants en communes, y a conflit, et ils ne peuvent pas se tenir à la même période. En outre, le problème à l'université est que la disponibilité d'une salle (et sa taille) joue un rôle important, alors que dans l'école ils sont souvent négligés parce que, dans la plupart des cas, nous pouvons supposer que chaque classe a sa propre salle.

2. Faisabilité, Optimalité et Complexité

Dans certains cas, le problème d'emploi du temps consiste à trouver le calendrier qui satisfait toutes les contraintes. Dans ce cas, le problème est formulé comme un *problème de recherche*.

Dans d'autres cas, le problème est formulé tel un *problème d'optimisation.* Ce qui est requis est un emploi du temps qui satisfait les contraintes dures et essaye de maximiser (ou minimiser) une fonction objectif qui intègre les contraintes. Comme on le verra plus loin, dans quelques approches, la formulation du problème d'optimisation est juste une application d'une technique d'optimisation au problème de recherche. Dans ce cas, ce qui est maximisée est appelé distance de faisabilité et même quand le problème est un vrai problème d'optimisation, la distance de faisabilité peut être intégré dans la fonction objectif [17]. Ceci est généralement fait pour facilité la recherche de la meilleure solution.

Dans les deux cas (Recherche et optimisation), nous définissons le problème sous-jacent qui est le problème de décider s'il y a une solution dans le cas d'un problème

de recherche, et le problème de décider s'il y a une solution avec une valeur donnée pour la fonction objectif concernant le problème d'optimisation. Quand on mentionne la complexité du problème, nous referons à la complexité du problème de décision sous-jacent.

Le problème sous-jacent est NP-complet dans la plus part des variantes [1][4][13][15]. Et une solution exacte est possible seulement dans les petit cas (e.g moins de 10 cours par semaine) [17]. Dans le monde réel le nombre de cours avoisine dans la plus part des cas 100 cours. Il s'ensuit que seules les méthodes approchées peuvent être utilisé pour résoudre ce type de problème et cela sans garantie d'obtenir *l'optimum*.

3. Le problème d'emploi du temps vu comme un problème de recherche

Plusieurs formulations peuvent être trouvées dans la littérature, celle proposée ici est issu de l'étude réalisée par sharef [17].

Il y a q cours K_1, …, K_q. pour chaque cours K_i, m séances sont organisées et R programmes son crées S_1, …, S_R où S_j est le programme pour les cours qui regroupe les mêmes étudiants. Ce qui veut dire que les cours programmés dans S_i doivent être programmé à des périodes différentes.

Le nombre de période est P, et l_k est le nombre de séance qui peut être planifié à la période K (c.à.d. le nombre de salle disponible à la période K). La formulation du problème est la suivante :

Chercher y_{ik} $\quad (i = 1..q; k = 1..p)$

$$\sum_{k=1}^{p} y_{ik} = k_i \quad (i = 1..q) \qquad (2.1)$$

$$\sum_{i=1}^{q} y_{ik} \leq l_k \quad (k = 1..p) \qquad (2.2)$$

$$\sum_{i \in S_l} y_{ik} \leq 1 \quad (l = 1..r; k = 1..p) \qquad (2.3)$$

$$y_{ik} = 0 \; ou \; 1 \quad (i = 1..q; k = 1..p) \qquad (2.4)$$

Où $y_{ik} = 1$ si la séance du cours K_i est planifiée à la période k, et $y_{ik} = 0$ sinon.

La contrainte (1) impose qui chaque cours ai un nombre de séance correct.

La contrainte (2) vérifie pour chaque période, que le nombre de séance programmé

ne dépasse pas le nombre de salle.
La contrainte (3) évite que les séances qui sont en conflit ne soient organisées au même moment.

4. Le problème d'emploi du temps vu comme un problème d'optimisation

Sharef [17] explique dans son étude que le même problème de recherche vu dans la section précédente peut être transformé en un problème d'optimisation en modifiant la fonction objectif de la façon suivante :

$$\max \sum_{i=1}^{q} \sum_{k=1}^{p} d_{ik} y_{ik} \qquad (2.5)$$

Où d_{ik} est la désirabilité de voir la séance du cours K_i planifie à la période k.

On peut passer d'un problème de recherche à un problème d'optimisation ou vis vers ça on choisissant d'intégré les contraintes dans la fonction objectif ou pas. Aussi, on peut intégrer une partie des contraintes et optimiser une autre partie.
Dans la littérature, les contraintes d'un problème d'optimisation sont le plus souvent divisées en deux ensembles, les contraintes dures et les contraintes relâchées.
Les contraintes dures sont celles qui doivent absolument être satisfaites par la méthode utilisée. Pour cette ensemble de contraintes le problème ressemble un problème de recherche.
Les contraintes relâchées sont celles qui de préférence doivent être respectées. Pour ce deuxième ensemble de contrainte, le problème est un problème d'optimisation.

5. Variantes du problème d'emploi du temps universitaire

Maintenant nous allons essayer de voir brièvement quelques variantes du problème d'emploi du temps. Sharef dans sont études [17] décrit un ensemble de variante qui diffère les institutions les une des autres :

a) **L'indisponibilité** : En plus du problème de base, une contrainte qui concerne la disponibilité des enseignants peut venir s'ajouter au problème de base. Certains intervenants peuvent ne pas être disponibles durant des journées

bien spécifiques au cour d'un semestre d'études et que l'emploi du temps doit prendre en charge.

b) **Les sections** : Dans certaines universités, certains cours se répètent plus d'une fois au cours de la semaine. En particulier, ces cours impliquant un grand nombre d'étudiants et appartenant à plusieurs programmes sont divisés en plusieurs sections. La création de différentes sections d'un cours peut contribuer à réduire le nombre de conflits dans un calendrier [15]. Par exemple, supposons que le programme des études S1 comporte les cours K1 et K2 et le programme des études S2 comporte K1 et K3. Supposons également qu'une séance de K2 a lieu au moment p et une séance de K3 au moment q. Dans ce cas, la séance du cours K1 ne peut avoir lieu ni au temps p, ni au moment q. Toutefois, si K1 est donné pour deux sections, alors la séance d'une section peut avoir lieu au moment p, et la séance de l'autre, au moment q.

c) **Répartition des étudiants** : Une autre variante du problème traite le regroupement des étudiants comme faisant partie du problème d'emploi du temps. Le problème est résolu grâce à une procédure itérative qui essaye durant chaque itération de résoudre les deux problèmes. Dans un premier temps l'emploi du temps et dans un second les regroupements des étudiants.

d) **Durée des séances** : Une variante encore moins connu est celle ou la durée de la séance de même type est variable.

e) **Affectation des salles** : Concernant l'affectation des salles, deux choix sont envisageable : soit les groupes d'étudiant ne change jamais de salle durant l'année ou aucune salle n'est affecté par défaut.

Nous venons de voir ici les Cinque principales caractéristiques trouvées dans la littérature qui différent un calendrier universitaire d'une institution à une autre.

6. Différentes approches et techniques pour la résolution du problème

Un certain nombre de technique et approche ont été proposés dans la littérature pour résoudre le problème d'emplois du temps dans le domaine de l'éducation en général et l'emploi du temps universitaire en particulier.

La plus part des première technique étaient basées sur la simulation de l'être humain pour la construction d'un emploi du temps [17]. L'emploi du temps était construit pas à pas, on ajoute leçon âpres leçon à l'emploi du temps partiel en essayant de satisfaire l'ensemble de toutes les contraintes à chaque fois.
Par la suite, un ensemble d'autres méthodes ont été appliquées dans le domaine de la recherche pour résoudre le problème. Entre programmation linéaire en nombre entier, Réseaux de flux, Le problème a été même réduit à un simple problème de coloriage de graphe.

Plus récemment, des approches fondées sur les techniques de recherche utilisées également en Intelligence Artificielle [14] apparu dans la littérature, nous avons recuit simulé, Abramson [2] propose un algorithme utilisant la technique du recuit simulé (SA) appliqué avec un algorithme séquentielle et parallèle. Cependant, cet algorithme n'est pas en mesure d'échapper à un minimum local une fois que la température est devenue trop faible, recherche tabou [15], algorithmes génétiques [13], des techniques de satisfaction de contraintes et programmation logique [21].

7. Algorithmes Génétiques appliqués aux emplois du temps universitaires

Comme nous venons de le voir ci-dessus, plusieurs formulations du problème d'emploi du temps universitaire ont été proposées et appliqué.
Les approches récentes trouvées dans la littérature pour résoudre le problème des emplois du temps utilisent les algorithmes évolutionnaires [1], [3], [5], [7], [9] et en particulier les algorithmes génétiques comme une puissante méthode pour résoudre le problème.
Différentes adaptations des algorithmes génétiques ont été appliquées pour la résolution du problème d'emploi du temps. Elles différent les unes des autres par :

1. Le type de codage utilisé pour les individus de la population.
2. Le choix de la fonction d'évaluation.
3. L'operateur de sélection.

4. Les operateurs de croisement et de mutation spécifiques à la problématique et au codage défini.

5. La stratégie d'insertion des nouveaux individus générés grâce aux operateurs de croisement et de mutation.
6. La sélection des paramètres pour l'exécution de l'AG (Taille population, taux de mutation, taux de croisement, etc…)

Néanmoins, le critère jugé pertinent pour différencier un algorithme génétique d'un autre pour la résolution du problème emploi du temps reste le codage.
Les algorithmes génétiques peuvent être divisés en trois grands groupes sur la base de la représentation de leur gène direct, indirect et structuré. Une méthode directe encode directement tous les événements attributs (classe, cours, maître de conférences, salle, jour, etc), tandis que une méthode indirecte juste encode certaines parties de la solution [9].

La phase de codage du gêne dans l'élaboration d'un algorithme génétique est un facteur très important pour la réussite final [7], le codage affecte la performance de l'algorithme, vitesse et qualité des résultats. Karova [13] explique brièvement les trois types de codage qui sont :

7.1. Le codage direct

C'est le codage le plus simple, il ne tient compte d'aucune contrainte lors de la création de l'individu. L'évaluation et le respect des contraintes sont laissés entièrement à la fonction fitness. Pour cette raison, la représentation du chromosome ne nécessite aucun prétraitement.
Adamidis et Arapakis [3] proposent une GA directe qui utilise une matrice NxM pour représenter le calendrier. Chaque élément de cette matrice est un gène. Un alphabet de caractères est utilisé pour représenter les attributs des événements. Un avantage de cette méthode est d'utiliser un algorithme de filtrage pour générer des solutions réalisables.
Erben et Keppler [10] recommandent une méthode directe pour le problème de calendrier de cours hebdomadaires. La méthode proposée utilise la technique de codage classique qui représente le chromosome comme une chaîne de caractères. Dans cette approche, le calendrier a été traduit en faits PROLOG définie comme une

cartographie d'un élément de cinq vecteur: classe, enseignant, enseignement, chambre.

Sigl et al. [21] utilisent un cube 3D correspondant aux salles, jours et périodes pour modéliser le calendrier. Dans cette méthode, tous les gènes d'un individu représentent un groupe d'étudiant et chaque individu représente un calendrier. L'algorithme commence par un calendrier non admissible, et tente d'obtenir un qui est admissible. Sigl et al. améliorent les performances de l'algorithme utilisant la sélection par le tournoi qui permet d'accepter une grande taille de la population sans ralentir l'algorithme.

L'avantage de cette représentation est que le décryptage du chromosome est directe est ne nécessite aucune procédure particulière.

L'inconvénient majeur est que la fonction fitness doit prendre en compte toutes les contraintes dures et relâchées ce qui nous donne une fonction fitness très compliqué et une convergence plus lente pour notre AG vu le nombre élevé de contrainte.

7.2. Le codage indirect

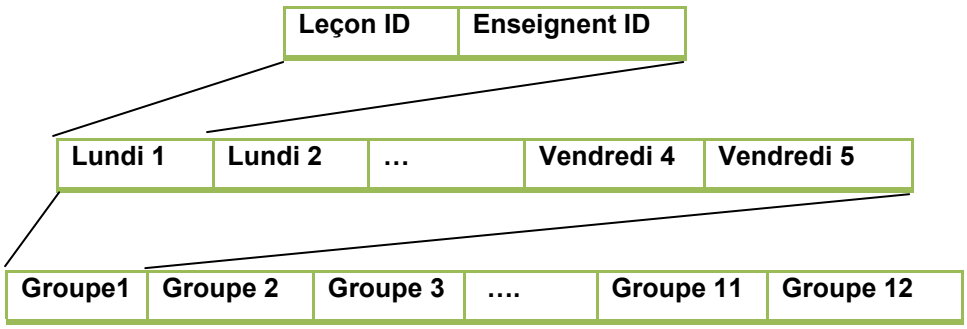
Le codage direct n'est pas toujours possible. Quand l'espace de recherche est fortement contraint, le besoin de réduire cette espace est plus que nécessaire. C'est pour cette raison qu'on essaye de satisfaire un ensemble de contrainte dans la génération de la solution initiale.

Dans [1] [22] un codage indirect pour la résolution du problème d'emploi du temps a été utilisé, la solution est représentée telle une chaîne de caractères. Deux configurations sont possible, soit les indices du vecteur représentent les événements à planifier et le contenu est le créneau au quel est programmé l'événement ou l'inverse, les indices du vecteur représentent tout les créneaux possible est leur contenu représente les événements à planifier.

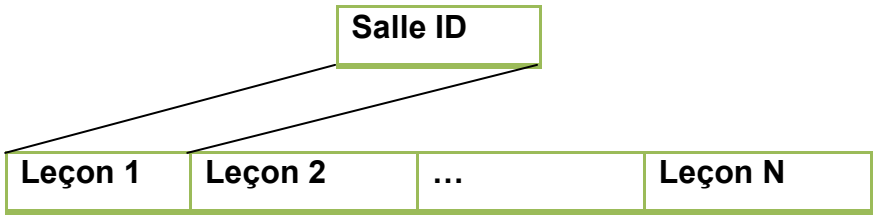
La plus part du temps, une phase de structuration est nécessaire avant la génération de la solution initiale.

7.3. Le codage structuré Le codage structuré est le codage le moins utilisé, plusieurs chromosomes sont utilisés pour coder plusieurs niveaux de la structure.

Ueda et al, [23] ont opté pour un codage structuré pour la génération de la solution. Un premier codage est crée pour optimiser le problème à un premier niveau puis la solution trouvé est utilisée dans le codage de la solution globale à un deuxième niveau (figure 3).



Génotype pour la planification de groupe



Génotype pour l'allocation de salle

Figure 3 Le génotype de l'AG

8. Conclusion

Parmi les trois codages présentés ci-dessus, le codage direct reste le plus utilisé au vu des articles traités.

Dans ce chapitre nous avons eu un aperçu général des différentes méthodes trouvé dans la littérature qui ont été utilisées pour la résolution du problème d'emplois du temps et en particulier les différentes adaptations des algorithmes génétiques au problème. Dans le chapitre suivant nous verrons une adaptation d'un algorithme génétique pour la résolution de notre problématique qui est la génération de l'emploi du temps de l'université de technologie de Belfort Montbéliard.

Chapitre 3

Génération Automatique d'Emploi du temps Universitaire

Le présent chapitre traite du problème d'organisation d'emplois du temps universitaires et plus précisément celui de l'UTBM (Université de Technologie Belfort-Montbéliard) dont la particularité du type d'enseignement rend plus complexe l'optimisation. En effet, l'UTBM adopte un enseignement dit « à la carte », dans la mesure où chaque étudiant à la possibilité de choisir ses unités de valeur. Nous traiterons cette problématique en suivant le fil conducteur suivant : d'abord la caractérisation des données (§.1), puis la formulation des contraintes (§.2), et enfin, la modélisation du problème (§.3).

Nous détaillerons l'élaboration de l'algorithme génétique que nous proposons pour la génération de l'emploi du temps (§.4). Aussi, nous verrons l'adaptation d'une méthode de recherche locale pour la génération des groupes d'étudiant (§.5). Les deux algorithmes seront couplés dans le but d'obtenir la meilleure solution possible.

La fin du chapitre sera consacrée à l'expérimentation de l'algorithme sur les données réelles de l'UTBM, nous présenterons les résultats numériques obtenus par l'algorithme et leurs représentations graphiques (§.7).

Introduction

La confection d'emploi du temps est une tache fastidieuse et répétitive qui fait généralement intervenir de nombreux éléments d'information. La difficulté du problème qui en résulte est liée à la nature des contraintes qu'il s'agit de satisfaire. A celle-ci s'ajoute la crainte que quelques incompatibilités ne se manifestent après la publication de l'emploi du temps définitif. Les techniques manuelles utilisées actuellement par la plupart des responsables horaires sont portées à disparaitre. Ces derniers sont prêts à faire appel à des outils informatiques qui les soulageraient dans leur besogne en leur fournissant un horaire complet, ou tout au moins un squelette substantiel qui ne nécessiterait que quelques retouches minimes.

Le problème de confection de l'emploi du temps de l'université de Belfort Montbéliard serra traité dans ce chapitre où la problématique vient du fait que l'université propose un enseignement à la carte qui engendre une conflictualité supplémentaire, qui est les conflits entre groupes d'étudiants. Nous proposerons une adaptation d'un algorithme génétique pour résoudre le problème de l'emploi du temps couplée avec une méthode de recherche locale (heuristique) pour la répartition des étudiants entre les groupes.

1. Présentation des données

Les modèles mathématiques proposés dans la littérature sont souvent trop restrictif pour prendre en compte toutes les contraintes qui interviennent dans le processus de construction d'un emploi du temps. L'algorithme d'optimisation qu'il s'agit de concevoir doit être suffisamment flexible pour tenir en compte des particularités propres de l'établissement et des différents changement qui peuvent surgir par la suite.

Les éléments d'information ci-dessous sont pris en considération lors de l'établissement d'un emploi du temps de cours semestriel :

(a) Un ensemble de périodes $P= \{p_1, \ldots, p_{np}\}$ pendant lesquelles différentes leçons peuvent être placées.

(b) Un ensemble d'intervenants PR= {pr_1, …, pr_{npr}}

(c) Un ensemble de Groupes GR= {gr_1, …, gr_{ngr}}. Chaque groupe regroupe un ensemble d'étudiants qui ne possède pas forcement le même programme d'études.

(d) Un ensemble de filières F= {f_1, …, f_{nf}}

(e) Un ensemble de salles S= {s_1, …, s_{ns}}. Il y a plusieurs types de salle, salle de cours, de Travaux dirigés (TD), Travaux pratiques (TP) et salles de Sport (gymnase).

(f) Un ensemble de sites ST= {st_1, …, st_{nst}} distant les uns par rapport aux autres. Deux sites sont distants si les intervenants et les étudiants n'ont pas suffisamment de temps pour se déplacer de l'un à l'autre durant une pause.

(g) Un ensemble d'unités de Valeur UV= {uv_1, …, uv_{nuv}}, chaque cours uv_i est caractérisé par les informations suivantes :

- Un ensemble d'intervenants,
- Un ensemble de groupes,
- Un ensemble de filières,
- Le nombre de séance de cours durant un semestre,
- Le nombre de séance de TD durant un semestre,
- Le nombre de séance de TP durant un semestre,
- Le site sur lequel le cours doit être donné,
- L'état d'espacement (nombre de séance de cours à effectuer avant un TD ou un TP, …).

  On note UP= {up_1, …, up_{nup}} la liste des séance qui résulte de la donnée de l'ensemble d'UV={uv_1, …, uv_{nuv}}

2. Formulation des contraintes

Un horaire peut être considéré comme satisfaisant s'il convient aux professeurs, aux

élèves et s'il respecte la disponibilité des salles. Le souci d'assurer un certain confort aux professeurs et aux élèves rend particulièrement difficile la confection d'un emploi du temps. Il est bien difficile de recenser toutes les exigences susceptibles

d'intervenir dans un problème d'emplois du temps. Certaines découlent directement des données décrites dans la section précédente alors que d'autres viennent se superposer selon le cas. Nous mentionnons ci dessous les contraintes les plus fréquentes et celle dont nous allons tenir compte par la suite.

En clair, le problème de la confection d'un emploi du temps universitaire consiste à donner une heure de début j $\in$ P à chaque leçon $up_i \in \{up_1, \ldots, up_{nup}\}$ tout en satisfaisant les contraintes suivantes :

(1) **Non chevauchement des Intervenants** : un intervenant ne peut pas enseigner plus d'une leçon à la fois.

(2) **Contraintes de site pour Intervenants :** L'enseignant ne change pas de site au cours d'une demi-journée.

(3) **Contraintes de pause pour Intervenants :** L'enseignant a au moins une pause de 45 minutes aux heures de midi.

(4) **Disponibilité des Intervenants :** L'emploi du temps essaie de respecter la journée de disponibilité des enseignants. Aucune différence n'est faite entre les intervenants extérieurs et le personnel de l'université.

(5) **Non chevauchements des étudiants** : Un étudiant ne peut suivre qu'une seule unité pédagogique à la fois.

(6) **Contraintes de pause Étudiants :** L'étudiant a au moins une pause de 45 minutes aux heures de midi.

(7) **Contraintes de site Étudiants** L'étudiant ne change pas de site au cours d'une demi-journée.

(8) **Contraintes d'affectation d'étudiants** : Un étudiant est affecté à un seul groupe de cours, TD ou TP par UV.

(9) **Contraintes sur les horaires des UPs :** On ne peut commencer une unité pédagogique si on ne peut la finir dans les horaires prédéfinis. Par exemple, on ne peut commencer un cours de deux heures à 19h.

(10) **Contraintes de positionnement des UPs:** Toutes les unités pédagogiques doivent être positionnées.

(11) **Contraintes d'affectation de salles** : Dans une salle, il ne peut y avoir qu'une seule unité pédagogique programmée par créneau horaire

(12) **Contraintes de choix de salles :** les affectations des salles doivent respecter les demandes des responsables d'UV.

(13) **Contraintes de capacité de salle :** Une unité pédagogique ne peut être planifiée dans une salle ne permettant pas de recevoir l'ensemble des étudiants lié à cette unité.

(14) **Contraintes jours non ouvrés :** aucun enseignement ne peut avoir lieu durant une journée non ouvré.

L'existence d'un horaire respectant à la lettre toutes les contraintes n'est pas garantie dans le cas général. En réalité, il s'agit de trouver un horaire aussi bon que possible dans un temps de calcul raisonnable tout en tolérant la présence de conflits ; le nombre de ces conflits doit être minimisé pour que l'horaire soit jugé satisfaisant par les différents parties concernées (direction de l'université, Intervenant, Etudiants, etc.)

3. Modélisation du problème

Soit C = $C_d \cup C_r$ une partition de l'ensemble des contrainte (1) à (14) décrites dans la section précédente. Une contrainte est dites Dure ou Relâchée selon qu'elle appartient à l'ensemble C_d ou à son complémentaire C_r.

- Un emploi du temps est acceptable si toutes les contraintes énoncées dans la section précédente sont satisfaite.
- Un emploi du temps est admissible si toutes les contraintes dures sont satisfaites.

Selon la définition précédente, un emploi du temps est acceptable s'il est admissible et s'il satisfait en plus les contraintes relâchées.

La modélisation de la confection d'un emploi du temps en termes de problème d'optimisation combinatoire découle des définitions précédentes. L'espace des solutions X contient tous les emplois du temps admissible H alors que la fonction objectif $f(H)$ mesure le degré de violation des contraintes relâchées dans l'emploi du

temps H ∈ X. L'objectif est de trouver un emploi du temps H* violant un minimum de contraintes relâchées. Les Algorithmes Génétiques seront utilisés dans le but de trouver un emploi du temps H^* associé à une valeur $f(H^*)$ aussi faible que possible.

Dans l'espace X, la recherche d'un emploi du temps acceptable dépend fortement de la manière dont sont définis les ensembles C_d et C_r. le cas C_d= Ø donne lieu à un espace X de taille énorme et à une fonction f difficile à optimiser en raison de sa complexité. Le cas opposé (C_r= Ø) n'est pas envisageable car l'espace X qui en résulte serait trop restreint, voir vide dans certain cas.
Il s'agit de trouver un compromis entre la taille de X et de la complexité de la fonction objectif f.

La partition retenue (voir tableau 2) peut paraitre surprenante si l'on songe à l'importance des contraintes (1) et (5). Introduire ces deux contraintes dans la liste des contraintes dures, pénaliserai fortement la génération de solution admissible. Aussi intéressent, les contraintes (9), (10), (12), (13) devrait être relâchées mais nous avons observé que dans notre cas de figure que la façon de gérer ces contraintes n'influence pas vraiment la qualité des solutions produites. Les contraintes (8), (10),(12), (14) ont finalement été placées dans la classe C_r pour simplifier la fonction objectif et être sur d'obtenir un emplois du temps final dans lequel elles sont satisfaites.

Contraintes Dures	Contraintes Relâchées
a. Contraintes de pause pour Intervenants (2)	i. Non chevauchement des Intervenants (1)
b. Contraintes de pause Étudiants (6)	ii. Contraintes de site pour Intervenants (3)
c. Contraintes d'affectation d'étudiants (8)	iii. Disponibilité des Intervenants (4)
d. Contraintes sur les horaires des UPs (9)	iv. Non chevauchements des étudiants (5)
e. Contraintes de positionnement des UPs (10)	v. Contraintes de site Étudiants (7)
f. Contraintes d'affectation de salles (11)	
g. Contraintes de choix de salles (12)	
h. Contraintes de capacité de salle (13)	
i. Contraintes jours non ouvrés (14)	

.Tableau 2 *Bipartition des contraintes*

4. Adaptation des algorithmes génétiques

Les algorithmes génétiques tentent de simuler le processus d'´evolution 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 composé d'un chromosome lui-même constitue de gènes qui

contiennent les caractères héréditaires de l'individu. Les principes de *sélection*, de *croisement* et 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. On lui associe la valeur du critère à optimiser, sa Fitness. On génère en suite 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, le croisement et la mutation assurent une exploration efficace de l'espace d'états.

On commence par générer une population d'individus. Pour passer d'une génération K à la génération K+1, les trois opérations suivantes sont répétées pour tous les éléments de la population K.

Des couples de parents P1 et P2 sont sélectionnées en fonction de leur fitness. L'operateur de croisement leur est appliqué avec une probabilité P_c (généralement autour de 0.6) et génère des couples d'enfants C1 et C2. D'autres éléments P sont sélectionnés en fonction de leur adaptation, l'operateur de mutation leur est appliqué avec la probabilité P_m (P_m est généralement inferieur à P_c) et génère des individus mutés P'. La Fitness des enfants (C1, C2) et des individus mutés est ensuite avalée avant insertion dans la nouvelle population.

Plusieurs critères d'arrêt de l'algorithme sont possibles :

- le nombre de générations peut être fixé à priori.
- temps constant.
- l'algorithme peut être arrêté lorsque la population n'évolue plus suffisamment rapidement.

Pour utiliser un algorithme génétique sur un problème d'optimisation on doit donc disposer d'un principe de codage des individus, d'un mécanisme de génération de la population initiale et d'operateurs permettant de diversifier la population au cours des générations et d'explorer l'espace de recherche.

4.1. Génération d'un emploi du temps initial (Solution)

Elle est usuellement aléatoire (diverses stratégies sont d'ailleurs envisageables pour échantillonner correctement un espace de recherche complexe ou de grande dimensionnalité). C'est là que l'on peut injecter la ou les solutions initiales au problème que l'on a pu obtenir par ailleurs (par exemple à l'aide d'autres techniques de résolution). Si l'initialisation a théoriquement peu d'importance (on converge en limite toujours vers l'optimum global), on constate expérimentalement que cette étape influence énormément la variance des résultats (et aussi la rapidité d'obtention de ceux-ci !). Il est ainsi souvent très avantageux d'injecter le maximum de connaissances a priori sur le problème par le biais de l'initialisation : placer dans la population initiale des individus que l'on sait proches par différents aspects de l'optimum recherché ne peut qu'accélérer la convergence de l'algorithme.
Dans notre cas, une méthode de type constructif est utilisée pour établir un emploi du temps initial. Tout d'abord deux structures de donnée sont crées (Voir Tableau 1 et 2)

La première structure trie les données salles, semaines, jours et heures. Elle est représentée par le tableau suivant :

ID_Creneau	Salle	Semaine	Jour	Heure
1	**E107**	**1**	**1**	**1**
2	**E107**	**1**	**1**	**2**
3	**E107**	**1**	**1**	**3**
4	**E107**	**1**	**1**	**4**
….				
N	**T305**	**16**	**6**	**4**

Tableau 3 *Les Créneaux*

Ce tableau est construit de façon à ne pas contenir les jours fériés (respect de la contrainte (14)) et ne pas avoir des créneaux doubles. Chaque créneau est unique, indexé par sont ID_Creneau et correspond à un début de séance possible.

La deuxième structure qui elle, trie les données UV, type UV, numéro de Groupe et numéro de séance est décrite par le tableau suivant :

ID_Evenement	UV	Type	N°Groupe	N°Séance
1	**AG51**	**C**	**1**	**1**
2	**AG51**	**C**	**1**	**2**
3	**AG51**	**C**	**2**	**1**
4	**AG51**	**C**	**2**	**2**
…..	**…..**			
M	**VI50**	**TP**	**4**	**5**

Tableau 4 *Les événements*

La taille du tableau 1 définie le nombre de séance à planifier durant deux semaines de cours, chaque ligne correspond à une séance.

Ayant d'un coté tout les événements à planifier et d'un autre, tout les débuts de créneaux possible, Un individu est alors constitué d'un tableau d'entiers qui contient exactement M cases (une case par ligne de Tableau2). Tel que les indexes du tableau représentent les événements et leurs contenu les créneaux aux quels ils sont affectés (voir tableau).

1	2	3	4	5	6	7	….	M-1	M
N-5	12	N	32	1	235	54	….	6	543

Tableau 5 *Individu*

Lors de la construction de la population initiale, nous nous efforçons de satisfaire le maximum de contraintes relâchées. Le but est d'accéléré la convergence et de réduire la complexité de la fonction objectif.

4.2. Operateur de sélection

La méthode utilisée est la sélection par la roulette ; elle consiste à associer à un chromosome i de la population une portion proportionnelle égale à :

$$\frac{1/f_i}{\sum_{j \in Population} 1/f_i} \qquad (1)$$

Où f_i est la valeur de la fonction objectif de l'individu i.

Ainsi les individus qui ont les valeurs faibles de la fonction d'évaluation peuvent avoir une chance d'être choisis lors des opérations de croisement et de mutation.

4.3. Operateur de croisement

L'operateur de croisement qu'on a développé pour notre type de codage manipule deux parents choisis au hasard parmi les individus sélectionnés pour le croisement. Le processus passe deux étapes (voir figure) qui sont :

Etape 1 : on effectue un croisement uniforme entre les deux parents choisi avec une probabilité de croisement P_{c1} pour les parents et une probabilité de croisement P_{c2} pour les gênes.

Le choix de ce type de croisement s'explique par sa capacité de contrôler le nombre de gêne modifiés sur un individu lors d'un croisement. En donnant un taux de croisement entre gêne très petit, on autorisera des petit changement sur les individu et vis vers ça, Si le taux de mutation entre gêne est grand, alors le changement lors de chaque croisement de parant serra grand.

Etape 2 : dans le cas où on obtient des enfants avec des créneaux doubles, une procédure de correction est lancée. Elle consiste à changer la valeur d'un des deux gènes choisi aléatoirement par un créneau non utilisé par l'individu en question.

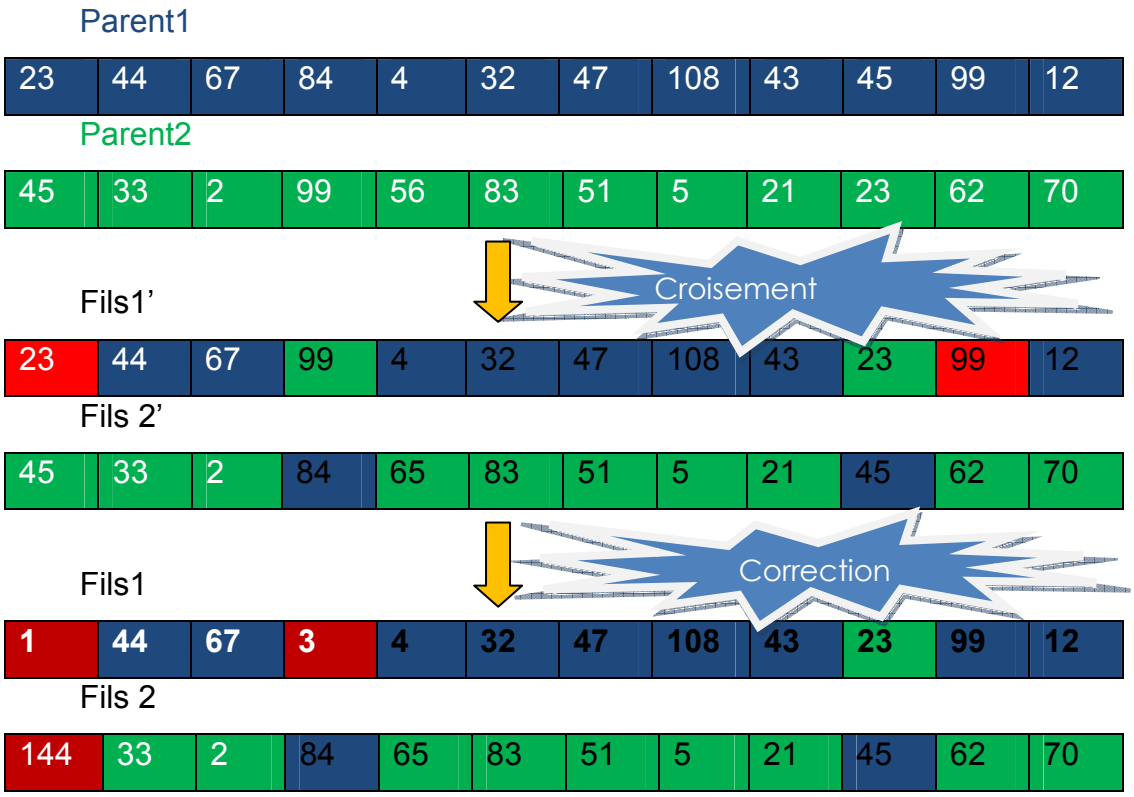


Figure 4 *Operateur de croisement*

Dans la littérature il arrive de trouver l'operateur de correction définie autant qu'un operateur à part entière. Pour rester conforme au model classique des algorithmes génétiques définie par goldberg, nous avons jugé opportun de définir la correction comme faisant partie de l'operateur de croisement.

4.4. Operateur de mutation

Les operateurs de mutation qu'on a développé évitent d'établir des populations uniformes incapables d'évoluer. Ils permettent de modifier les valeurs des gènes de chromosomes. Nous définirons les deux operateurs de mutation qu'on utilise. Bien sur, différemment des operateurs de croisement, ceux de la mutation n'ont besoin que d'un seul chromosome.

- **Mutation par inversion:**
  Elle consiste à choisir deux gènes de même type aléatoirement et à échanger leur contenu (Créneaux) avec une probabilité de mutation P_{m1}.

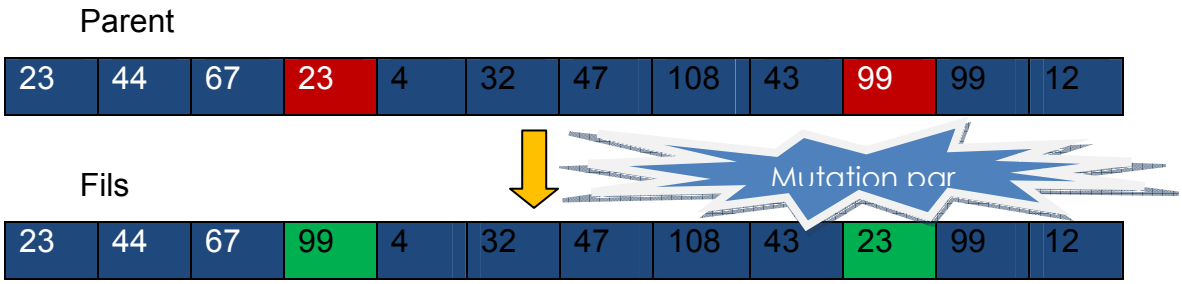


Figure 5 *Operateur de croisement 1*

- **Mutation par changement de créneau :**
  Elle consiste à choisir un gêné aléatoirement et à le remplacé par un créneau non utilisé par l'individu avec une probabilité de mutation P_{m2}.

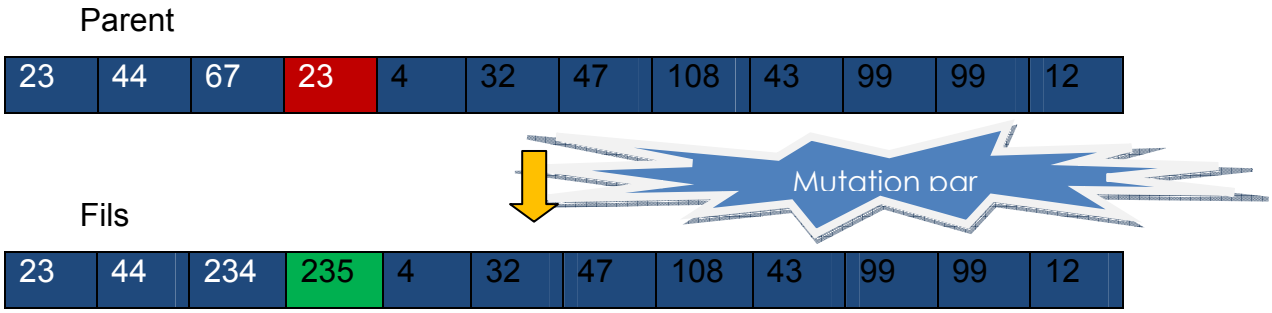


Figure 6 *Operateur de mutation 2*

4.5. Méthode d'insertion

Nous avons utilisé, la méthode d'insertion élitisme qui consiste à recopier le meilleur chromosome de l'ancienne population dans la nouvelle. Celle-ci est complétée par les solutions résultantes d'operateurs de croisement et de mutation en veillant à ce que la taille de la population reste fixe de génération en génération.

4.6. Formulation de la fonction d'évaluation

La qualité d'un emploi du temps est estimée par agrégation des différentes fonctions

partielles. Une pondération est utilisée pour spécifier le niveau d'importance de chacune des contraintes. La meilleure solution trouvée pour le problème est alors estimée en regard à la valeur de sa fonction objectif. Une valeur de coût minimal signifie une solution de bonne qualité.

La fonction objectif f : X →N mesurant la distance qui sépare un emploi du temps admissible H d'un horaire acceptable est de la forme :

$$f(H) = \sum_{r \in C_r} P_r * f_r(H) \qquad (3.1)$$

Où $f_r(H)$ est le degré de violation de la contrainte $r \in c_r$ dans l'emploi du temps H

et P_r est un poids positif permettant de différencier les contraintes relâchées selon leur importance relative.

Les notations ci dessous se rapportent à un emploi du temps admissible H. Elles sont utiles pour la présentation des cinq composantes $f_r(H)$ qui apparaissent dans la définition de la fonction objectif.

UP_i : Ensemble des séances données à la périod

e / $\in \{1,2,3,4,\dots,n\}$

UP_j : Ensemble des séances qui concerne l'UV$_i$.

$Inter\_Chev(x,y)$=1 si l'intervenant enseigne deux leçon simultanément, sinon $Inter\_Chev(x,y)$ = 0.

$Groupe\_Chev(x,y)$=1 si le groupe suit deux leçon simultanément, sinon $Groupe\_Chev(x,y)$ = 0.

$Enseigne(i,k)$ =1 si l'intervenant i est programmé à la période k, sinon $Enseigne(i,k) = 0$. $Disponible(i,k)$ =1 si l'intervenant i est disponible à la période k, sinon $Disponible(i,k)$= 0.

$Change\_site(j) = 1$ si un intervenant ou un étudiant change de site au cours d'une demi-journée, sinon $Change\_site(j) = 0$

- $f_1(H)$ calcule le nombre de fois où un intervenant enseigne deux leçons simultanément

$$f_1(H) = \sum_{i=1}^{n} \sum_{(x,y)\in UP_i} Inter\_Chev(x,y) \qquad (3.2)$$

- $f_2(H)$ calcule le nombre de fois où un groupe suit deux leçons simultanément

$$f_2(H) = \sum_{i=1}^{m} \sum_{(x,y)\in UP_j} groupe\_Chev(x,y) \qquad (3.3)$$

- $f_3(H)$ calcule le nombre de violation de la disponibilité des enseignants

$$f_2(H) = \sum_{i=1}^{n} \sum_{k=1}^{m} Enseigne(i,k)\ Disponible(i,k) \qquad (3.4)$$

- $f_4(H)$ calcule le nombre de fois où un intervenant change de site au cours de la même demi journée

$$f_2(H) = \sum_{i=1}^{n} Change\_site(i) \qquad (3.5)$$

- $f_5(H)$ calcule le nombre de fois où un étudiant change de site au cours de la même demi journée

$$f_2(H) = \sum_{i=1}^{m} Change\_site(i) \qquad (3.6)$$

5. Répartition des étudiants entre les Groupes

Comme nous l'avons expliqué au début de notre mémoire, nous traitons un problème d'emploi du temps pour une université où l'enseignement est à la carte. Cela rend la tache encore plus difficile dans le sens où ce mode de choix des UVs permet d'avoir une complexité et une conflictualité supplémentaire lors de la génération des groupes d'étudiants qui influe directement sur l'espace de recherche de notre algorithme génétique.

5.1. Espace de recherche

L'espace de recherche de l'algorithme génétique est divisé en trois sous ensembles (voir figure 7) qui sont :

- Espace des solutions mauvaises : c'est l'ensemble des solutions qui violent les contraintes dures.
- Espace des solutions admissibles : c'est l'ensemble des solutions qui respectent toutes les contraintes dures.
- Espace des solutions acceptables : c'est l'ensemble des solutions qui respectent toutes les contraintes (dures et relâchées)

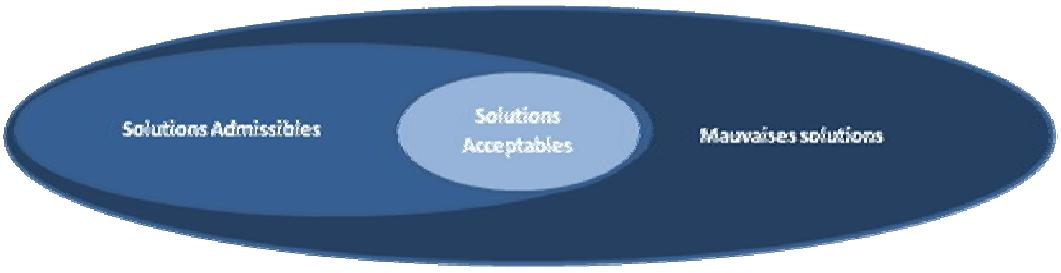


Figure 7 *Espace de recherche de l'AG*

L'espace de recherche que nous venons de décrire est variable en fonction d'un paramètre très important qui est la répartition des étudiants entre les groupes.

Plus le nombre de groupe qui ont des étudiants en commun est grand, plus l'espace de recherche des solutions acceptables de l'AG qui construit l'emploi du temps est réduit, voir vide. Et vis vers ça, plus le nombre des groupe d'étudiants qui sont en conflit est petit, plus l'espace de recherche des solutions acceptable est grand et aussi, plus on a de possibilité de tomber sur un optimum global. C'est pour cette raison qu'on a d'adapté une méthode de recherche locale pour améliorer la répartition des étudiants entre les groupes.

5.2. Adaptation d'une méthode de recherche locale pour la génération de groupe d'étudiants

Dans cette partie nous allons mettre en pratique le principe de la méthode de descente (méthodes d'amélioration itérative) pour résoudre le sous problème lié à la répartition des étudions.

Le principe des méthodes de descente consiste, à partir d'une solution de départ X_0, à engendrer une suite (finie) de solution X_i de telle sorte que X_{i+1} soit meilleure que X_i : $f(X_{i+1}) \prec f(X_i)$ pour tout i. parmi les paramètres qui doivent être précisés pour mettre cette méthode en œuvre figure la façon d'engendrer la nouvelle configuration courante. Pour cela, il est habituel d'utiliser la notion de *voisinage d'une solution*.

On définit généralement le voisinage d'une solution à l'aide d'une transformation élémentaire (ou locale) [7]. On appelle transformation toute opération permettant de changer une solution $X \in S$ en une autre solution X' de S. Une transformation sera considéré comme élémentaire (ou locale) si elle ne modifie que faiblement la structure de la solution à laquelle on l'applique [7].

5.2.1. Génération de la solution initiale

Dans un premier temps, l'affectation des étudiants au groupe est faite d'une façon aléatoire tout en essayant de satisfaire deux contraintes qui sont :

- Un étudiant n'est affecté qu'à un seul groupe par type d'UV

- Les groupes d'un même type d'UV (cours, TD ou tp de l'UV_i) doivent être presque de même taille.

Une fois cette phase terminer, grâce à des operateurs bien spécifiques, ont essaye de trouver le voisinage de la solution initial qui améliore la répartition.

5.2.2. Voisinage

La notion de voisinage dépend donc de la transformation locale considérée. Ainsi, pour la transformation élémentaire évoqué plus haut, le voisinage d'une chaîne X de n bits est l'ensemble des n chaines de n bits possédant exactement n-1 bits en commun avec X. une autre transformation pourrait induire un voisinage de cardinal différent : par exemple, celle définie par le changement simultané de deux bits conduirait à un cas de voisinage possèdent n(n-1)/2 éléments …
Dans notre, nous avons opté pour deux voisinage :

- une permutation entre deux étudiants de deux groupe de même type d'UV (voir Fig 8)
- Changer le groupe à un seul étudiant, dans ce deuxième cas de figure, il faut juste vérifier qu'on ne dépasse pas la taille maximale autorisée pour un groupe.

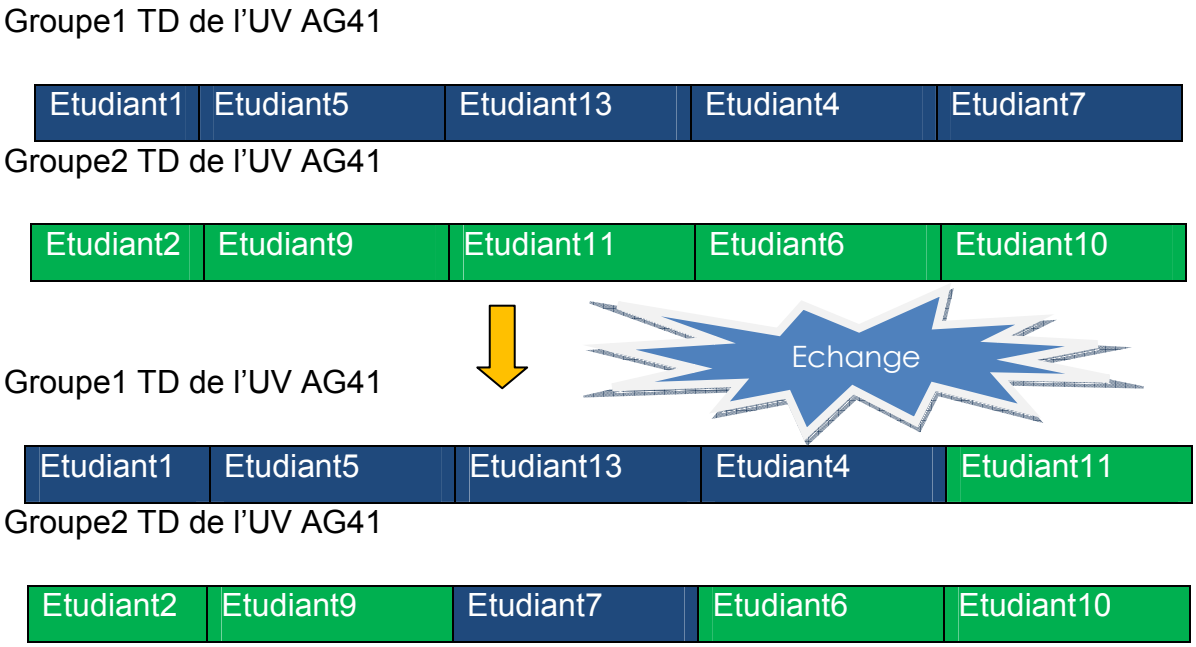


Figure 8 *permutation de deux étudiants*

5.2.3. Fonction d'évaluation

Soit Gr_i: est l'ensemble des groupe. $i \in [1,..,n]$

Le tableau 6 représente la matrice des conflits intergroupes qui nous permettra de calculer la fitness de la fonction objectif.

	Gr_1	Gr_2	Gr_3	Gr_4	Gr_5	…	Gr_n
Gr_1	1	0	1	0	1	0	1
Gr_2	0	1	1	1	0	1	0
Gr_3	1	1	1	0	0	1	1
Gr_4	0	1	0	1	0	0	0
Gr_5	1	0	0	0	1	1	1
…	0	1	1	0	1	1	1

Gr_n	1	0	1	0	1	1	1

Tableau 6 *matrice des conflits*

Les notations ci dessous se rapportent à une répartition des étudiants entre les groupes d'UV. Elles sont utiles pour la présentation de la fonction objectif.

Conflits (x,y) = 1 si un même étudiant appartient au groupe x et au groupe y, sinon Conflits (x,y) = 0.

$$f(x) = \sum_{x,y \in Gr} conflit\ (x,y) \qquad (3.7)$$

6. L'algorithme d'optimisation global

L'algorithme d'optimisation et de génération d'emploi du temps est la combinaison des deux algorithmes décrit précédemment. Chacun d'eux joue un rôle bien spécifique pour l'amélioration de la solution finale.

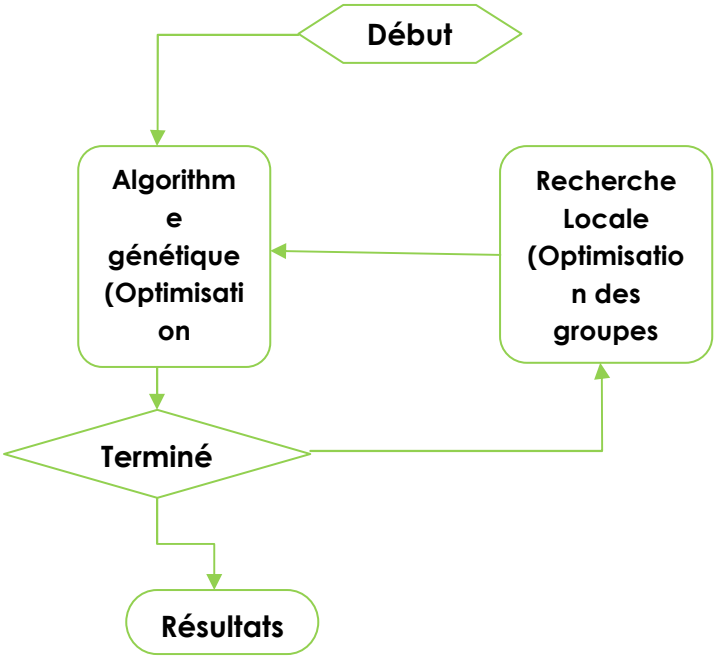


Figure 9 *schéma fonctionnel de l'algorithme*

7. Résultats

L'approche décrite dans la section précédente a été développée avec C++ sous Windows avec une machine qui a un processeur Intel core 2 Duo et 2 Go de RAM. Nous présentons ici les meilleurs résultats obtenus parmi toutes les exécutions réalisées. Notre algorithme génétique à générer 30000 générations durant les 5 jours d'exécution pour obtenir les résultats. Généralement, un emploi du temps avec un grand nombre de données prend des mois pour être confectionné à la main.

Plusieurs expériences ont été réalisé avec les donnée réelles du semestre universitaire automne 2006 pour ajusté les paramètres de l'algorithme génétique.

Les poids de la fonction objectif ont été fixés à égalité lors des tests effectués. P_1=1, P_2=1, P_3=1 qui sont respectivement les poids des contraintes (1) (4) (5).

Dans un premier temps, nous avons examiné le comportement de notre AG sans la méthode d'optimisation qui repartie les étudiants.

7.1. Données utilisé

L'algorithme mis en place a été testé sur les données réelles de l'université de technologie de Belfort Montbéliard.

Nombre d'UV	Nombre de Groupes	Nombre de sites	Nombre de départements	Nombre de filières	Nombre d'étudiants	Nombre de salles
437	1200	3	6	18	1200	128

Tableau 7 *données*

7.2. Paramètres de l'AG

Les paramètres de l'algorithme génétique qui ont permis d'avoir les meilleurs résultats sont décrits dans le tableau ci-dessous :

Probabilité de croisement entre individus	Probabilité de croisement entre gênés	Probabilité de mutation 1	Probabilité de mutation 1	Nombre d'individu	Nombre de génération
0.8	0.05	0.03	0.03	10	30000

Tableau 8 *Paramètres de l'AG*

7.3. Résultats

Les figures 1 et 2 illustre la convergence de l'algorithme génétique sous l'influence de la méthode de recherche locale.

	AG couplé avec la RL pour la répartition des étudiants		AG sans la RL pour la répartition des étudiants	
	meilleure	moyenne	meilleure	moyenne
Nombre conflits enseignants	134	140	187	196
Nombre Conflits indisponibilité	5	6	18	18
Nombre conflits étudiants	753	781	1449	1508
Fitness globale	892	928	1654	1723

Tableau 9 *comparatif entre l'AG couplé et*

NON couplé avec la recherche locale

7.4. Représentation des résultats

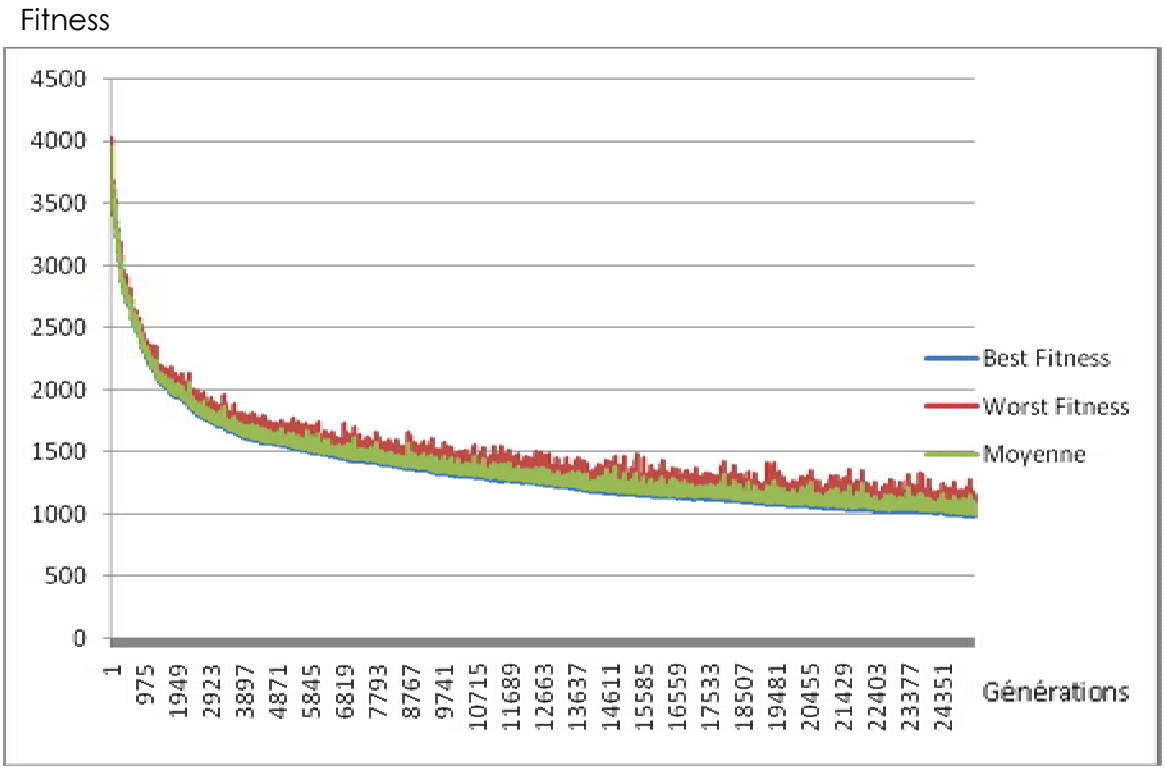
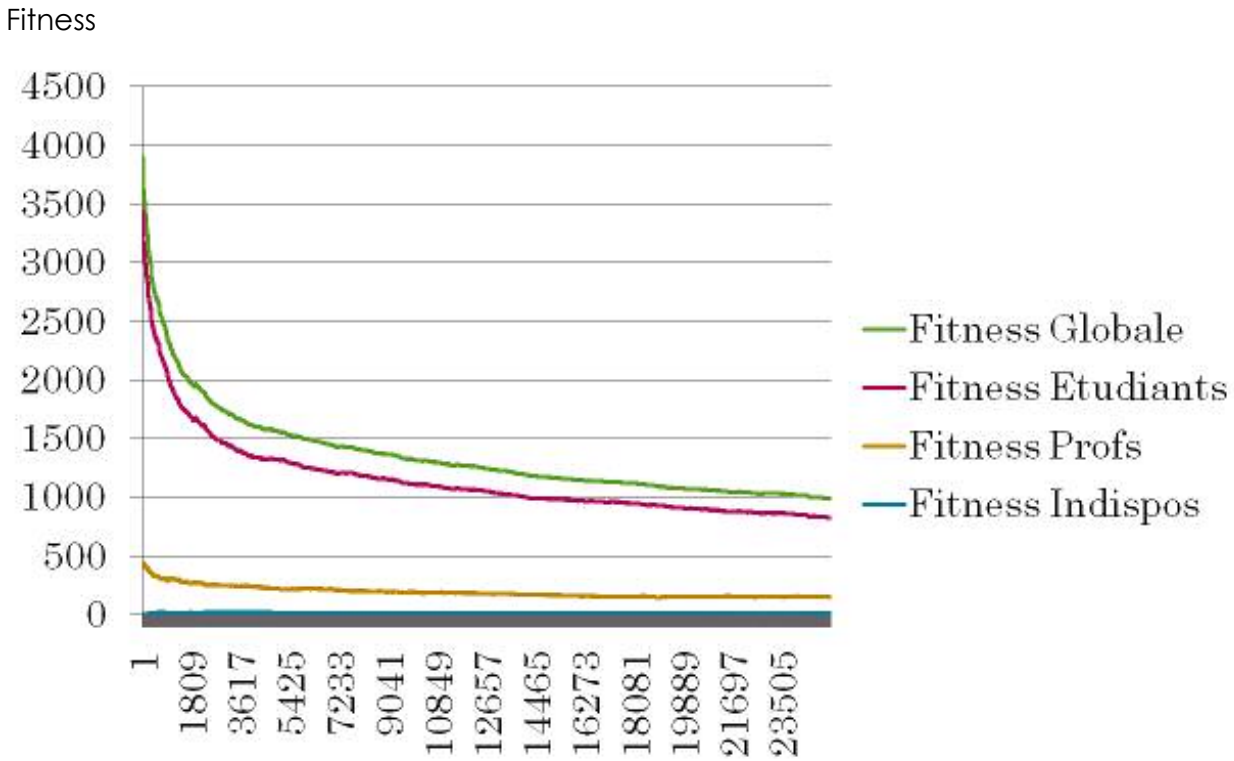


Figure 10 *Convergence de l'algorithme génétique sous l'influence de la recherche locale (1)*



Générati

Figure 11 *Convergence de l'algorithme génétique sous l'influence de la recherche locale (2)*

Signalons pour conclure que les méthodes décrites dans ce chapitre ont été proposées pour résoudre le problème d'emploi du temps de l'UTBM en intégrant la répartition des étudiants dans la problématique globale de confection de l'emploi du temps. Sachant qu'a l'UTBM, la répartition des étudiants se fait indépendamment de la confection de l'emploi du temps.

Il est rare d'aboutir à un emploi du temps qui fasse l'unanimité auprès de toutes les personnes concernées. Nous avons remarqué qu'il est relativement difficile d'établir un horaire qui garantisse à la fois un programme compact pour les intervenants et une répartition équitable pour les étudiants.

Conclusion Générale

Les problèmes liés aux emplois du temps sont à la fois intéressants, difficiles, et surtout différents d'une institution à une autre.

L'enseignement à la carte qui caractérise notre problématique rend les nombreux travaux trouvés dans la littérature non adéquat pour résoudre le problème.
Dans ce mémoire nous avons conçu un algorithme génétique couplé avec méthode de recherche locale pour la génération automatique de l'emploi du temps de l'UTBM.
Il est délicat de conclure une étude de ce type, les résultats obtenus sont très encourageants et nous prouvent encore une fois que les algorithmes génétiques sont aptes à résoudre des problèmes très complexes (NP Complet) comme les emplois du temps.
Même s'il est illusoire de vouloir résumer tous les objectifs réels au travers d'une simple fonction, ce résultat témoigne de la qualité des méthodes développées.

À notre connaissance, il n'existe pas de problèmes standards dans le domaine de la génération automatique d'emploi du temps. Cette situation rend particulièrement difficile la comparaison des nombreuses méthodes de construction d'horaire dans la littérature.

Bibliographie

[1] S.Abdullah, E.K Burke, B.McCollum, “A hybrid evolutionary approach to the university course timetabling problem” in: IEEE Congress on Evolutionary Computation, 2007. CEC 2007, 25-28 Sept, 1764-1768, 2007

[2] D. Abramson, “Constructing School Timetables Using Simulated Annealing: Sequential and Parallel Algorithms”, *Management Science*, INFORMS, Maryland, USA, 1991, vol. 37, no. 1, pp. 98-113.

[3] P.Adamidis, P.Arapakis, “Evolutionary Algorithms in Lecture Timetabling”, in: Proceedings of the 1999 Congress on Evolutionary Computation, 1999. CEC 99., Washington, 1151 Vol. 2, 1999

[4] V.Bhatt, R.Sahajpal, “Lecture timetabling using hybrid genetic algorithms”, in: Proceedings of International Conference on Intelligent Sensing and Information Processing (2004), 29- 34 , 2004.

[5] E.K. Burke, D.G. Elliman, and R.F. Weare, “A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems”, *Proc. 6th Int. Conf. on Genetic Algorithms*, Pittsburg, Morgan Kaufmann, 1995, pp. 605-610

[6] I.Charon, A.Germa, O.Hudry “Méthodes d’optimisation combinatoire” , Masson, Paris, 1996

[7] C. K. Chan, H. B. Gooi, M. H. Lim, “Co-evolutionary algorithm approach to a university timetable system”, in: Proceedings of the Evolutionary Computation on 2002. CEC '02 , IEEE Computer Society Washington, Pages 1946-1951 , 2002

[8] Y.Collette and P.Siarry. Optimisation Multiobjectif. Eyrolles, 2002.

[9] D.Danciu, “Evolutionary timetabling using biased genetic operators”, in: Proceedings of the 25th International Conference on Information Technology Interfaces, 2003. ITI 2003. 16-19 June 2003, 477- 482, 2003

[10] W. Erben and J. Keppler, “A Genetic Algorithm Solving a Weekly Course-Timetabling Problem”, *Selected papers from the First International Conference on Practice and Theory of Automated Timetabling*, Edinburgh, U.K. pp. 283-295, 1995.

[12] D.E.Goldberg “Algorithmes Génétiques : Exploration, optimisation et apprentissage automatique ”, Addison-Wesley France, 1989

[13] J. K. Hao, P. Galinier et M. Habib, 'Méthaheuristiques pour l’Optimisation. Combinatoire et l’Affectation Sous Contraintes', Revue d’Intelligence, vol. 13, n°2, pp. 283-324, 1999

[13] M.KAROVA "Solving Timetabling Problems Using Genetic Algorithms". *27th international Spring seminar on Electronics Technology*, 2004.

[14] V. Kumar. "Algorithms for constraint-satisfaction problems: A survey". *AI Magazine*, 13(1):32-44, 1992.

[15]L.F.Lai; N.L.Hsueh; L.T.Huang; T.C.Chen, “An Artificial Intelligence Approach to Course Timetabling”, 18th IEEE International Conference Tools with Artificial Intelligence, Page(s):389 – 396, 2006

[16] M. Rahoual and R. Saad, “Solving Timetabling Problems by Hybridizing Genetic Algorithms and Tabu Search”, *Proc. 6th Int. Conf. on the Practice and Theory of Automated Timetabling*, Brno, Czech Republic, 2006, pp. 467-472.

[17] A. Schaerf, “A Survey of Automatic Timetabling”, *Artificial Intelligence Review*, Springer Netherlands, vol. 13, no. 2, pp. 87-127, 1999.

[18] A. Schaerf. “Local search techniques for large high school timetabling problems”. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 29(4):368--377, 1999.

[19] A. Schaerf and L.D. Gaspero, “Local Search Techniques for Educational Timetabling Problems”, *Proc. 6th International Symposium on Operations Research in Slovenia*, Preddvor, Slovenia, pp. 13-23, 2001.

[20] M.Sakarovitch, “Programmation Discrète”, Hermann 1984.

[21] B. Sigl, M. Golub, and V. Mornar, “Solving Timetable Scheduling Problem by Using Genetic Algorithms”, *Proc. 25th IEEE Int. Conf. on Information Technology Interfaces*, Cavtat/Dubrovnik, Croatia, pp. 519-524, 2003.

[22] N.D.Thanh, “Solving Timetabling Problem Using Genetic Algorithms and Heuristic Algorithms”, International Conference on software Engineering, IEEE, 2007

[23] H.Ueda, D.Ouchi, K.Takahashi and T.Miyahara, “A Co-Evolving Timeslot/Room Assignment Genetic Algorithm Technique for University Timetabling ”, Practice and Theory of Automated Timetabling III, 48-63, 2001

[24] L.Zhang, S.Lau, “Constructing university timetable using constraint satisfaction programming approach”, Computational Intelligence for Modelling, Control and Automation, page(s): 55- 60, 2005