République Algérienne Démocratique Et Populaire
Ministère de L'Enseignement Supérieur et de la Recherche Scientifique

Université 8 mai 1945 de Guelma

Faculté Des Mathématiques, Informatique Et Sciences de la Matière
Département d'informatique

THÈSE DE DOCTORAT Troisième Cycle LMD

Option : Informatique

Présentée par
CHERAITIA Meryem

THÈME

Planification d'examens

Soutenue publiquement le : 08/01/2018 devant le jury composé de :

Pr. Hamid SERIDI	Prof	U. 8 Mai 1945, Guelma	Président
Pr. Salim HADDADI	Prof	U. 8 Mai 1945, Guelma	Directeur de thèse
Pr. Meziane AIDER	Prof	USTHB	Examinateur
Pr. Rachid BOUDOUR	Prof	U. Badji mokhtar, Annaba	Examinateur

Année 2018

Je dédie ce modeste travail :

À tous ceux que j'aime. . .

Meryem...

Remerciements

Avant tout, Je remercie Allah qui nous a donné vie et santé pour le parachèvement de ce modeste travail.

Je tiens à remercier après de tout cœur, mon directeur de thèse Monsieur ***Salim HADDADI****, Professeur à l'université 8 Mai 1945 à Guelma pour m'avoir permis de travailler à leur côté, merci pour l'encadrement de mon travail et pour son encouragement, ainsi que son soutien tout au long de la thèse. Je le remercie pour tout son aide, son enthousiasme et sa patience ont beaucoup facilité et agrémenté mon travail. Il a été toujours disponible pour répondre aux questions que je lui posais. Ses remarques m'ont permis de faire progresser ce travail et soutenu tout au long de ces années.*

Je tiens aussi à remercier tout particulièrement Monsieur ***Abdellah SALHI,*** *professeur à l'Université de Essex UK, pour son merveille accueil, Il m'a fait bénéficier de ses compétences et de sa rigueur, et m'a encouragé à de nombreuses reprises ainsi pour notre travail collaboratif concernant le problème de planification des examens qui a abouti à une très belle publication.*

Je tiens également à remercier Monsieur ***Hamid SERIDI****, professeur à l'Université de Guelma, pour l'honneur qu'il m'a fait, en acceptant la présidence de ce jury, son aide, ses inestimables conseils et ses orientations précises.*

Je tiens également à remercier tous les membres du jury : ***Pr. Rachid BOUDOUR et Pr. Meziane AIDER,*** *d'avoir bien voulu examiner mes travaux de thèse et d'avoir participé au jury de cette thèse.*

Je tiens à remercier ma mère, ma sœur, mes frères, et toute la famille pour leurs encouragements et soutiens pendant toute la période de la thèse en leurs souhaitant une belle vie.

Enfin, je tiens à remercier mes amis et mes collègues, en leurs espérant une bonne continuation et une bonne chance.

... Un grand merci à tous !

ملخص

انشاء الجداول الزمنية للامتحانات هو مشكل من مشاكل التحسين التوافقي، المعروف بصعوبته من وجهة النظر النظرية وكذلك العملية. اقتراح الجدول الأمثل هو مهمة معقدة للغاية للتنفيذ ويتطلب وقتا طويلا مبالغا فيه. لهذا السبب، نختار بدلا من ذلك حل تقريبي يمكن الحصول عليه في غضون فترة زمنية معقولة. يتكون النموذج الأكثر انتشارا في هذا المجال، بعد انشاء حل أولي (أو عائلة من الحلول الأولية)، في سلسلة من التحسينات المحلية المتعاقبة.

والهدف من هذه الرسالة مزدوج. 1) استحداث طرق قادرة على إنتاج حلول ذات جودة عالية، وتنافسية مع الطرق المقترحة سابقا. 2) التأكد من أن هذه الطرق عامة بما يكفي لتطبيقها بشكل أعم على المشاكل والسياقات الأخرى.

تظهر ثلاثة مساهمات:

- الأولى هي الأدلة العليا حيث يتم استخدام محاكاة الصلب كخوارزمية رفيعة المستوى تقود البحث على مستوى منخفض لتحسين نشر الامتحانات المتضاربة.
- وتتألف المساهمة الثانية من خوارزمية هجينة جديدة تجمع بين خوارزمية الانتشار النباتي والبحث المحلي.
- تقدم المساهمة الثالثة نموذجا جديدا لجداول مواعيد الامتحانات وطريقة جديدة لحلها استنادا إلى إعادة تكرير طريقة من مرحلتين.

النتائج التي تم الحصول عليها على مجموعة البيانات من جامعة تورونتو تبين أن طرقنا الثلاثة توفر نتائج ممتازة وتنافسية مع الطرق المقترحة سابقا.

الكلمات المفتاحية الجدول الزمني للامتحانات، الأدلة العليا، محاكاة الصلب، خوارزمية انتشار النبات، البحث المحلي المكرر، الجوار ذو الحجم الأسي، المسار الهاميلتوني ذو الطول الأدنى، تحقيق أقصى قدر من ارقام 1 المتتابعة.

Résumé

La planification d'examens est un problème d'optimisation combinatoire bien connu pour être difficile du point de vue théorique et pratique. Proposer un planning optimal est une tâche extrêmement complexe à mettre en œuvre et nécessite un temps exagérément long. Pour cette raison on opte plutôt pour une solution approchée mais qui peut être obtenue en un temps raisonnable. Le paradigme le plus répandu dans ce domaine consiste, après avoir généré une solution initiale (ou une famille de solutions initiales), en une séquence d'améliorations locales successives.

L'objectif de cette thèse est double. 1) Développer des méthodes capables de produire des solutions de qualité, compétitives avec l'état de l'art. 2) Faire en sorte que ces méthodes soient suffisamment génériques pour être appliquées plus généralement à d'autres problèmes et à d'autres contextes.

Trois contributions ressortent :

- La première est une méta-heuristique où le recuit simulé est utilisé comme algorithme de haut niveau pilotant une recherche locale de bas niveau pour améliorer la diffusion des examens en conflit.
- La deuxième contribution consiste en un nouvel algorithme hybride combinant l'algorithme de propagation de plantes (PPA) et la recherche locale (LS).
- La troisième contribution introduit un nouveau modèle de planification d'examens ainsi qu'une nouvelle méthode pour le résoudre fondée sur la réitération d'une méthode en deux phases.

Les résultats obtenus sur les données standards de l'Université de Toronto montrent que nos trois approches fournissent d'excellents résultats et s'avèrent compétitives avec l'état de l'art.

Mot-clés Planification d'examens, méta-heuristique, recuit simulé, algorithme de propagation de plantes, recherche locale itérée, voisinage de taille exponentielle, chemin hamiltonien de longueur minimum, maximisation du nombre de blocs de 1 consécutifs.

Abstract

Exam timetabling is a combinatorial optimization problem, well-known to be hard from theoretical as well as practical point of view. Constructing an optimal planning is an extremely complex task which requires a prohibitive amount of computing time. For this reason, we usually opt for an approximate solution that is obtained within a reasonable allowed time. The most widespread paradigm in this domain consists, after generating an initial solution (or a family of initial solutions), of a sequence of successive local improvements.

The objective of this thesis is twofold: 1) Developing methods capable of producing quality solutions, competitive with the state of art. 2) Ensure that these methods are generic enough to be applied to more general problems and contexts.

Three contributions emerge:

- The first is a meta-heuristic where the simulated annealing is used as a high-level algorithm driving a low-level local search to improve the widespread of the conflicting exams.
- The second contribution consists of a new hybrid algorithm combining the plant propagation algorithm and local search.
- The third contribution introduces a new realistic model for exam timetabling and a new method to solve it, based on iterating a two-phase method.

The results obtained on benchmark datasets from the University of Toronto show that our three approaches provide excellent results and are competitive with the state of the art.

Key words Exam scheduling, meta-heuristics, simulated Annealing, plant propagation, iterated local search, exponential size neighborhood, minimum length hamiltonian path, consecutive block maximization

Table des matières

Table des figures

Liste des tableaux

Acronymes

UESP	Uncapacitated Exam Scheduling Problem
SA	Simulated Annealing
LS	Local Search
ILS	Iterated Local Search
MLHP	Minimum Length Hamiltonian path
CBM	Consecutive Block Maximisation
VLSN	Very Large Scale Neighborhood

Introduction

D'un côté, la mise au point d'un emploi du temps est une tâche répétitive et fastidieuse. Elle est omniprésente puisqu'elle affecte presque toutes les grandes institutions telles que les établissements d'enseignement pour le planning des cours [54] et la planification des examens comme nous le verrons dans cette thèse, les opérations militaires [58], les hôpitaux pour la rotation des infirmières [70] ou la gestion des soins [53], les transports publics [33], les pompiers [7], la navigation maritime [11], et ainsi de suite. D'un autre côté, c'est une tâche très difficile, car elle nécessite l'introduction de nombreuses contraintes et d'un espace de solution qui, bien que fini, croît de manière exponentielle avec la taille du problème.

La production d'un emploi du temps peut être considérée à la fois comme un problème d'affectation, puisqu'elle consiste à affecter des ressources (salles, étudiants, infirmières, policiers, pompiers ...), et comme un problème d'ordonnancement puisqu'il s'agit aussi de programmer les ressources dans le temps. La production d'emploi du temps est un problème difficile, aussi bien que sur le plan théorique que calculatoire.

Le problème de production d'emploi du temps dans les établissements d'enseignement s'est montré d'un intérêt considérable au cours des deux dernières décennies. En raison du nombre croissant d'institutions et d'étudiants, la planification dans ce domaine est devenue plus exigeante. Ces établissements se distinguent les uns des autres par différents environnements, différentes stratégies, différentes réglementations. Cela implique de prendre en compte de nombreuses et diverses contraintes. Les chercheurs ont proposé, dans le temps, plusieurs paradigmes pour faire face à la difficulté croissante de ces problèmes.

Parmi les problèmes de planification dans les établissements d'enseignement, le problème de planification d'examens est l'un des plus étudiés. Il concerne l'attribution d'un ensemble prédéfini d'examens à un nombre fixe de périodes afin de satisfaire un ensemble de contraintes. Ces dernières peuvent varier considérablement d'un établissement à l'autre (voir [67, 85] pour les différents types de planification d'examens et les contraintes les plus courantes). Certaines contraintes sont nécessaires et doivent être pleinement satisfaites. De telles contraintes sont qualifiées de dures. Elles ont trait aux limites physiques du monde réel. Un surveillant, par exemple, ne peut pas être affecté à deux salles distinctes en même temps. Une solution satisfaisant les contraintes dures s'appelle un planning réalisable. D'autres contraintes, profondément liées aux problèmes de planification, sont les contraintes molles. Celles-ci sont seulement souhaitables et peuvent être contournées. Un exemple est celui des vœux des enseignants. Les contraintes molles ne peuvent pas être complètement satisfaites en raison de leur nature conflictuelle [85]. Habituellement, le but du problème est de minimiser leur violation. La qualité d'un planning dépend cependant de la mesure avec laquelle les contraintes molles sont satisfaites.

Dans cette thèse, nous traitons le problème de planification d'examens sans contraintes de capacité. C'est un problème simple dans sa formulation car il ne comporte que deux contraintes, une dure et une molle. Notons, toutefois que le faible nombre de contraintes n'implique pas qu'un problème soit facile à résoudre. Dans ce contexte, les données benchmark de l'Université de Toronto constituent, comme pour la plupart des chercheurs, le pool d'instances utilisées pour tester nos méthodes, et les comparer avec l'état de l'art.

La thèse comprend six chapitres, organisés de la façon suivante. Le chapitre 1 présente une introduction à l'optimisation combinatoire et quelques notions basiques liées à la théorie de complexité. Le chapitre 2 décrit le modèle mathématique du problème de planification d'examens étudié dans cette thèse et propose une étude bibliographique. Le chapitre 3 constitue un condensé de l'arsenal des méthodes utilisées. Le chapitre 4 suggère une heuristique pour la résolution du

problème de planification d'examens, basée sur le recuit simulé et la recherche locale. Cette méthode a été publiée [34]. Le chapitre 5 propose une méta-heuristique consistant en une hybridation de l'algorithme de propagation de plantes et de la recherche locale. Cette partie a été également publiée [35]. Le chapitre 6 présente un nouveau modèle de planification d'examens avec une approche de résolution qui consiste à itérer la recherche locale dans des voisinages de taille exponentielle. Le contenu de ce chapitre a été soumis pour publication.

Avertissement Puisque certains concepts n'ont pas d'équivalent convaincant en français, ils seront désignés par leur nom original. On peut citer, par exemple, Kempe chain, stratégie token-ring, règle roulette-wheel.

Chapitre 1

Optimisation combinatoire et classes de complexité

1.1 Introduction

L'optimisation représente l'acte de rendre optimal, c'est-à-dire de résoudre les problèmes qui consistent en la minimisation (ou maximisation) d'une fonction par rapport à un ensemble de choix. L'optimisation combinatoire, aussi appelée optimisation discrète, concerne les problèmes où l'espace des solutions est discret. De nombreux problèmes peuvent se modéliser comme des problèmes d'optimisation combinatoire. Dans ce chapitre, nous décrivons le domaine de l'optimisation combinatoire, ainsi que ses concepts de base. En outre, nous présentons certaines méthodes parmi les plus représentatives, développées pour résoudre les problèmes d'optimisation combinatoire et on termine par une introduction informelle à la théorie de la complexité.

1.2 Optimisation combinatoire

1.2.1 Définition

Dans sa forme la plus générale, un problème d'optimisation consiste à trouver les meilleures valeurs d'une ou plusieurs variables suivant une fonction objectif donnée tout en respectant un ensemble de contraintes [75]. La difficulté du problème dépend de la nature des variables, discrètes ou continues, et de la nature des contraintes. Dans le cas de variables discrètes, on parle alors de problèmes d'optimisation combinatoire [74].

Un problème d'optimisation combinatoire $\Phi = (S, f)$ est défini comme suit :

— Un ensemble de variables $X = x_1, \ldots, x_n$

— Chaque variable x_i est associée à un domaine D_i, c'est-à-dire $x_i \in D_i$

— Un ensemble de contraintes reliant les variables

— Une fonction objectif pour minimiser (ou pour maximiser) : $f : D_1 \times \cdots \times D_n \to \mathbb{R}$

L'ensemble S s'appelle espace de recherche. Une solution possible pour le problème Φ est un élément $s \in S$ tel que $s = \{v_1, \ldots, v_n \mid v_i \in D_i\}$ et toutes les contraintes sont satisfaites. La résolution d'un problème d'optimisation combinatoire avec une fonction objectif à minimiser consiste à trouver une solution s^* telle que $\forall s \in S, f(s^*) \leqslant f(s)$. En principe, chercher la meilleure solution dans un ensemble discret ou fini est un problème facile à résoudre : il suffit d'énumérer et comparer les qualités de toutes les solutions possibles. Cependant, en pratique, cette énumération peut prendre beaucoup trop de temps si S est de grande taille. Or, le temps de recherche d'une solution optimale est un facteur déterminant. Pour cette raison, des techniques efficaces deviennent une nécessité pour résoudre Φ [12].

1.2.2 Problèmes d'optimisation combinatoire

L'optimisation combinatoire est un domaine scientifique pluridisciplinaire qui occupe une place très importante dans trois grands domaines scientifiques : la recherche opérationnelle, les mathématiques discrètes et l'informatique. De nombreuses applications pratiques peuvent être formulées sous la forme d'un problème d'optimisation combinatoire. Voici quelques exemples :

— Problème du sac-à-dos : compte tenu d'un ensemble d'objets, chacun avec un poids et une utilité, il s'agit de déterminer quels objets choisir afin que le poids total soit inférieur ou égal à une limite donnée et que l'utilité globale soit aussi grande que possible. Il tire son nom du problème rencontré par quelqu'un qui est contraint par un sac à dos de taille fixe et doit le remplir avec les objets les plus utiles. Ce problème est plus fréquemment rencontré, entre autres, pour le chargement des conteneurs, les avions ou bateaux de fret ou pour gérer la mémoire d'un microprocesseur [55].

— Problème du voyageur de commerce : compte tenu d'une liste de villes et des distances entre chaque paire de villes, un voyageur de commerce doit visiter toutes les villes, chacune une et une seule fois, tout en minimisant la distance totale parcourue. Le problème de tournées de véhicules généralise ce problème en posant la question suivante : quel est l'ensemble optimal d'itinéraires pour une flotte de véhicules afin de fournir un ensemble de clients [51] ?

— Problème de planification d'horaires : l'objectif est de trouver des périodes de temps appropriées pour un certain nombre de tâches nécessitant des ressources limitées. Selon la nature du problème, les contraintes du problème peuvent varier et il peut y avoir de nombreux objectifs différents. Les problèmes de planification peuvent survenir dans de nombreux contextes différents. Parmi la vaste famille de ces problèmes, on trouve celui de planification d'examens dans les établissements d'enseignement et qui représente notre thème de recherche dans cette thèse [65].

1.2.3 Méthodes de résolution

Au fur et à mesure que la taille de l'espace de recherche croît, le coût des algorithmes peut augmenter de façon exponentielle, rendant impossible la recherche d'une solution. D'ailleurs, la plupart des problèmes d'optimisation combinatoires n'ont pas à ce jour de solution efficace valable pour toutes les données. Une autre façon de s'attaquer à ces problèmes est de trouver une solution sous-optimale mais dans un délai raisonnable. Dans certains cas, on peut même trouver la solution optimale au problème. Étant donnée l'importance de ces problèmes, un grand nombre de méthodes de résolution ont été développées. De telles techniques peuvent être divisées en deux groupes principaux :

— **Méthodes exactes** Elles représentent une méthodologie de recherche classique qui évoque des procédures mathématiques. Habituellement, des formulations mathématiques sont incorporées afin de représenter l'objectif et les contraintes. De tels algorithmes résolvent toujours un problème d'optimisation à l'optimalité. Des exemples de ces approches comprennent la programmation en nombres entiers, les méthodes de coupes, les méthodes par séparation et évaluation (Branch-and-Bound) et des combinaisons de ces dernières (Branch-and-Cut, Branch-and-cut-and-Price).

Généralement ces méthodes consistent à l'énumération de toutes les solutions possibles de l'espace de recherche. Pour une énumération efficace, telles méthodes disposent de techniques de détection des échecs et d'heuristiques pour l'orientation des différents choix. De telles approches visent à trouver une solution optimale pour un problème particulier. En outre, le modèle mathématique de ces approches doit être soigneusement développé et traité [88].

— **Méthodes approchées** Elles constituent une tendance très intéressante pour traiter les problèmes d'optimisation combinatoire de grande taille. Elles sont indispensables pour les problèmes NP-complets puisque qu'elles consistent à explorer une sous-partie de l'espace de recherche afin de trouver une solution sous-optimale dans un temps raisonnable. Dans ce cas,

ces méthodes ne garantissent pas l'optimalité de la solution trouvée. L'objectif d'une méthode approchée est de se rapprocher le plus possible de la valeur optimale dans un temps raisonnable qui est au maximum le temps polynomial. Typiquement ce type de méthodes, appelée aussi heuristiques sont utilisées afin de générer des solutions initiale pour les méthodes exactes (par exemple Branch-and-Bound). Ainsi, elles sont utiles pour les problèmes en temps réel ou ayant des données de grande taille.

En effet, ces méthodes ont prouvé leur efficacité dans le domaine de l'optimisation combinatoire et particulièrement avec l'apparition des méta-heuristiques, à partir des années 1980, qui représentent des méthodes plus puissantes et générales adaptables et applicables à une large classe de problèmes.

Les méta-heuristiques deviennent une partie importante de l'optimisation moderne. Un large éventail d'algorithmes ont émergé au cours des deux dernières décennies, telles que l'optimisation par essaim de particules deviennent de plus en plus populaires.[50]

1.3 Classes de complexité

Vers le début du XX^{e} siècle, sur la base de travaux en logique formelle, plusieurs mathématiciens commencèrent à s'intéresser aux propriétés mathématiques des algorithmes. Alain Turing [100] a introduit la définition formelle d'un algorithme déduite de la notion de langage formel pour une machine abstraite, appelée machine de Turing.

Comme définition simple : Un algorithme est une suite finie d'instructions permettant de résoudre un problème donné partant d'un état initial et d'une entrée initiale (peut-être vide), les instructions décrivent un calcul qui, lorsqu'il est exécuté, passe par un nombre fini d'états successifs bien définis, produisant finalement une "sortie". Dans l'optimisation combinatoire, un algorithme permet

la recherche d'une solution optimale dans un ensemble discret . Lorsque l'algorithme ne garantit pas la recherche d'une solution optimale, mais une solution aussi proche que possible de la solution optimale, en parle alors des heuristiques.

La performance des algorithmes peut être mesurée par différentes méthodes,citons par exemple :

- Le temps consacré à trouver la solution ;
- L'espace mémoire requis pour l'exécuter ;
- La qualité de la solution ;
- La capacité de l'algorithme à s'adapter aux changements d'entrée du problème.

La théorie de la complexité a pour but d'évaluer les algorithmes en termes d'"efficacité" et de discriminer les problèmes en ceux qui sont "faciles" et ceux qui sont "durs" selon qu'ils peuvent être résolus par des algorithmes efficaces ou non [13].

Un problème important qui se pose lorsque l'on considère les problèmes d'optimisation combinatoire est la classification des problèmes difficiles.

Pour introduire la notion des classes de complexité de problèmes, nous présentons la définition des problèmes de décision : Un problème de décision est un problème dont la réponse à la question posée est "oui"ou "non".

Pour distinguer les problèmes faciles et difficiles, on considère la classe de problèmes qui peuvent être résolus par une machine de Turing déterministe dans un temps polynomial et des problèmes qui peuvent être résolus par une machine de Turing non déterministe en temps polynomial.

Nous ne voulons pas formaliser la caractérisation des classes P et NP via les machines de Turing et préfèrent décrire les caractéristiques et notions liées à ces classes à un niveau plus intuitif [72]. Nous ne voulons pas entrer dans les détails et renvoyer le lecteur à un livre sur la théorie de la complexité [102] pour une lecture plus approfondie..

1.3.1 Classe P (polynomial time)

Une définition simple pour caractériser les problèmes qui appartiennent à P est :

Definition 1.1. Un problème est dans P si cela peut être résolu par un algorithme en temps polynomial.

Les problèmes dans P peuvent donc être résolus efficacement par un algorithme de complexité polynomiale. Des exemples de problèmes appartenant à cette classe sont le problème de test de primalité et la programmation linéaire.

1.3.2 Classe NP (non-deterministic polynomial time)

Une classe qui est intuitivement associée à des problèmes difficiles s'appelle NP.

Definition 1.2. Un problème de décision est dans NP si la vérification de la validité de la solution peut être effectuée dans un temps polynomial.

Les problèmes dans NP peuvent être résolus par un algorithme de complexité polynomiale pour une machine non déterministe il est donc pas nécessaire qu'une solution soit calculable en temps polynomial. Il est seulement nécessaire que nous puissions vérifier la solution du problème en temps polynomial. Par conséquent, $P \subseteq NP$ détient (légèrement abuser de la notation en limitant P aux problèmes de décision), et il est largement admis que $P \neq NP$.

1.3.3 Classe des problèmes NP-complets

Certains problèmes de NP apparaissent plus difficiles à résoudre pas un algorithme polynomial.

Definition 1.3. Un problème est NP-complet s'il est aussi difficile à résoudre que n'importe quel autre problème de NP.

On peut dire, les problèmes les plus difficiles de la classe NP représentent la classe des problèmes NP-complets.

1.3.4 Classe des problèmes NP-difficiles

De nombreux problèmes d'optimisation et de décision, y compris le problème de couverture de sommets, sont au moins aussi difficiles que n'importe quel problème dans NP. De tels problèmes sont appelés NP-dur. En effet, pour qu'un problème soit NP-complet, il faut qu'il soit dans la classe NP, i.e., que l'examen de chaque cas puisse être réalisé efficacement, par une procédure polynomiale. Si on enlève cette contrainte d'appartenance à la classe NP, on obtient la classe plus générale des problèmes NP-difficiles

Definition 1.4. Un problème s'appelle NP-difficile s'il est au moins aussi difficile que n'importe quel problème dans NP.

Chapitre 2

Problème de planification d'examens : Position du problème et état de l'art

2.1 Introduction

La mise au point d'un planning d'examens constitue, au niveau des institutions académiques, une tâche administrative ardue et fastidieuse car souvent répétée à chaque fin de session de cours. La solution manuelle nécessite généralement une quantité importante de temps, parfois plusieurs jours, voire des semaines de travail. Par ailleurs, la solution obtenue est insatisfaisante et comporte des erreurs en raison du grand nombre de données (étudiants, examens, contraintes et préférences). Ces défis ont motivé les efforts de recherche à exploiter la puissance de l'ordinateur pour construire ces plannings et aussi d'élaborer des approches applicables pour de nombreux scénarios.

2.2 Position du problème

Le problème d'élaboration de planning d'examens (UESP) peut être défini comme celui d'affecter un ensemble d'examens, chacun prévu pour un certain groupe d'étudiants, à un nombre limité de périodes, tout en respectant un ensemble prédéfini de contraintes, qui diffèrent d'un problème a un autre, suivant la spécificité de l'établissement en question et les caractéristiques attendues de l'emploi du temps recherché.

Les contraintes sont souvent classées en deux catégories, dures et molles. Les contraintes dures doivent être impérativement satisfaites. Le planning qui satisfait toutes les contraintes dures est appelé planning réalisable. Les contraintes molles expriment souvent des vœux. Elles peuvent être violées, mais, il est souhaitable de les satisfaire autant que possible. Les contraintes molles varient (et entrent parfois en conflit les unes avec les autres) d'un établissement à l'autre en termes de type et importance. Le degré de satisfaction de ces contraintes définit la qualité globale d'un planning donné [85]. Souvent, puisqu'il est impossible de satisfaire toutes les contraintes molles, l'objectif ultime consiste à "minimiser"leur violation. Autrement dit, il s'agit de trouver une solution qui satisfait toutes les contraintes dures et qui minimise la violation des contraintes molles.

Dans la pratique, l'importance des contraintes molles diffère d'un établissement à l'autre. Un bon planning pour un établissement peut être inacceptable pour un autre. Par exemple, certaines universités n'ont pas assez de ressources (salles) pour accueillir les examens alors que pour certaines autres cette question n'est pas préoccupante.

A cause des variantes de mesure et de contraintes qui diffèrent d'un établissement à l'autre, plusieurs bases de données du monde réel, modèles et formulations ont été présentées par divers chercheurs. Les données benchmark de l'Université de Toronto sont les plus largement utilisées dans la littérature pour évaluer la performance des différentes méthodes proposées. En 1996, Carter et al. [31] ont introduit un ensemble de treize instances du monde réel de diverses universités à

travers le monde (trois écoles et cinq universités canadiennes, une américaine, une du Royaume-Uni et une université du Moyen-Orient).

Ces instances ne prennent pas en compte la capacité des salles et périodes, les écarts entre les jours et les week-ends consécutifs. Ces ensembles de données ont été largement utilisés comme bancs d'essai, en introduisant des différentes dimensions des problèmes et des caractéristiques qui ont eu lieu dans ces universités de 1983 à 1993. Ces ensembles de données sont accessibles au public et peuvent être téléchargées gratuitement à partir de ftp ://ftp.mie.utoronto.ca/pub/carter/testprob/. Les paramètres associés à chaque problème sont présentés dans le tableau 2.1 [85]. La première colonne donne un nom à l'instance. Les deuxième et troisième colonnes présentent respectivement le nombre d'examens et d'étudiants. Le quatrième fournit le nombre de périodes autorisées, et la cinquième la densité (en pourcentage) de la matrice des conflits.

TABLE 2.1: Caractéristiques des instances

	Nombre d'examens	Nombre d'étudiants	Nombre de périodes	Densité
CAR91	682	16,925	35	0.13
CAR92	543	18,419	32	0.14
EAR83	189	1,125	24	0.27
HEC92	81	2,823	18	0.42
KFU93	461	5,349	20	0.06
LSE91	381	2,726	18	0.06
RYE93	486	11,483	23	0.07
STA83	139	611	13	0.14
TRE92	261	4,360	23	0.18
UTA92	622	21,266	35	1.13
UTE92	184	2,750	10	0.08
YOR83	181	941	21	0.29

Les informations proposées dans chaque instance sont le nombre total d'étudiants inscrits pour chaque examen et l'ensemble des examens choisis par chaque étudiant. La densité des conflits pour chaque instance de problème est calculée en divisant la somme des examens en conflit (ayant au moins un étudiant en commun) par le nombre total d'élément dans la matrice de conflit.

Supposons que m étudiants doivent subir n examens. Soit $(c_{ij})_{n\times n}$ la matrice des conflits, où c_{ij} est le nombre d'étudiants prenant les deux examens i et j. Soit

θ le nombre de périodes autorisées. Nous considérons un problème de satisfaction de contraintes, UESP, où une seule contrainte dure et une contrainte molle sont prises en compte. La contrainte dure stipule que les examens avec des étudiants en commun doivent être affectés à différentes périodes tandis que la contrainte molle exige la dispersion des examens au sein des périodes données de manière à minimiser le nombre des étudiants qui prennent des examens à proximité.

Soit $t_k \in \{1, \cdots, \theta\}$ l'entier qui spécifie la période associée à l'examen k. Considérons le graphe $G = (X, E)$ où X est un ensemble de n sommets, qui correspondent aux examens, et E est un ensemble d'arêtes de telle sorte qu'il y a une arête reliant les sommets i et j si $c_{ij} \neq 0$.

La θ-coloration du graphe G consiste à affecter des couleurs aux sommets de telle sorte que chaque sommet est coloré et que deux sommets adjacents sont de couleurs différentes. La θ-coloration du graphe G donne donc un planning d'examen réalisable (en assignant chaque examen coloré avec "couleur"$t \in \{1, \cdots, \theta\}$ à une période t). L'espace de solution de UESP est l'ensemble de toutes les θ-colorations du graphe G, qui est fini même s'il peut augmenter de façon exponentielle avec la taille du graphe. Le but de UESP est de choisir une θ-coloration qui satisfasse la contrainte molle. Une façon astucieuse de traiter la question a été proposée dans [31], et a été, par la suite, largement adoptée : trouver une θ-coloration du graphe G de façon à minimiser

$$\frac{1}{m} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} \times prox\left(t_i, t_j\right) \tag{2.1}$$

où

$$prox\left(t_i, t_j\right) = \begin{cases} 2^5/2^{|t_j - t_i|} & \text{if } 1 \leq |t_j - t_i| \leq 5 \\ 0 & \text{sinon} \end{cases} \tag{2.2}$$

avec

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} \times Y\left(t_i, t_j\right) = 0 \tag{2.3}$$

où

$$Y(t_i, t_j) = \begin{cases} 1 & \text{if } t_i = t_j \\ 0 & \text{sinon} \end{cases} \tag{2.4}$$

La fonction objectif 2.1 représente en quelque sorte la violation de la contrainte molle. Le cout de proximité $prox(t_i, t_j)$ évalue la qualité du planning en pénalisant la proximité des deux examens en conflit. Si les examens en conflit i et j sont affectés à des périodes adjacentes t_i et t_j (telle que $|t_i - t_j| = 1$) la pénalité est la plus élevé, 16. S'il y a une (resp. deux, trois, quatre) journée(s) libre(s) entre les deux périodes t_i et t_j ($|t_i - t_j| = 2$, resp 3, 4, 5), alors la pénélité est de 8, resp. 4, 2, 1. Dans tous les autres cas, la pénalité $prox(t_i, t_j)$ est 0. L'équation 2.3 représente la contrainte dure : aucun étudiant ne peut passer plus d'un examen à la fois.

UESP apparaît clairement comme un problème d'optimisation combinatoire. Il est difficile puisque même le problème de décision de savoir s'il existe un planning réalisable est le problème bien connu de la θ-coloration qui est NP-difficile. Étant donné que UESP est difficile, une approche pratique et raisonnable est d'employer une heuristique capable de trouver des plannings sous-optimaux en un temps de calcul acceptable.

2.3 Complexité

UESP est modélisé ici comme un problème d'optimisation combinatoire. Le taille du problème dans ce modèle augmente avec le nombre d'étudiants, de périodes, d'examens. Dans [36], Cooper et Kingston ont montré que UESP est NP-dur.

2.4 Etat de l'art

Le problème de planification d'examens est l'un des problèmes d'ordonnancement les plus étudiés. Ce problème a été étudié depuis 1960 et a fait l'objet de nombreuses études publiées dans la littérature. Certains des plus intéressants sont dus à [19, 20, 66, 76, 85, 91, 92].

Dans ce qui suit, sans être exhaustifs, nous discutons les différentes approches publiées dans la littérature pour résoudre UESP. Il faut cependant mentionner que le temps de calcul de la solution n'est quasiment jamais mentionné.

Une grande variété d'approches, traitant de tous les paradigmes de la recherche opérationnelle, ont été proposées. Les premiers travaux se concentrent sur des approches exactes. Puisque le temps de calcul nécessaire pour trouver une solution exacte augmente de façon exponentielle avec la taille de l'instance, jusqu'à nos jours peu d'effort y a été consenti [68, 103].

Contrairement aux méthodes exactes, les méthodes approximatives ne fournissent pas nécessairement une solution optimale, mais seulement une "bonne"solution en un temps de calcul aussi faible que possible [81]. La majorité des nouvelles approches approximatives proposées pour l'ensemble de données de l'université de Toronto peuvent être classées en trois catégories.

2.4.1 Approches basées sur la coloration de graphe

UESP sans contraintes molles peut être représenté comme un modèle de coloration de graphe. Les sommets du graphe représentent les examens, les couleurs représentent les périodes et les arêtes entre deux sommets indiquent le conflit (au moins un étudiant est inscrit dans les deux examens correspondants) entre les examens qui ne doivent pas être programmés en même temps. Les poids sur les arêtes représentent le nombre des étudiants communs inscrit dans les deux examens. L'objectif est de trouver une coloration sans que deux sommets adjacents n'aient la même couleur. Les premières approches pour résoudre le problème de coloration

de graphes ont porté sur les heuristiques. Le principe général de ces heuristiques est d'estimer la difficulté, et affecter les examens un par un, en commençant par le plus difficile.

Un certain nombre d'heuristiques ont été développées pour résoudre UESP : le plus grand degré (Largest degree), le degré de saturation (Saturation degree), le degré le plus pondéré (Largest weighted degree), le plus grand enrôlement (Largest enrolement) et le degré de couleur (Color degree) [25]. Malkawi et al. [62] ont utilisé la technique de coloration des nœuds de graphe pour UESP. L'algorithme commence par la création du graphe en utilisant la matrice de poids qui donne pour chaque nœud le poids des arêtes reliant ce nœud. Ensuite, les couleurs ont été attribuées aux nœuds du graphe à partir de trois heuristiques.

Des recherches récentes comprennent l'utilisation de la technique de classement adaptatif combinée avec les heuristiques de coloration de graphe où une heuristique est utilisée pour mesurer la difficulté d'affecter un examen particulier. Les examens difficiles sont affectés d'abord et les examens les plus faciles sont prévus vers la fin du processus. Abdul Rahman et al. [87] ont proposé une stratégie de classement adaptatif basé sur l'optimisation de la roue qui grince (Squeaky wheel optimization). Le plus grand degré et le degré de saturation ont été utilisés comme heuristiques de classement intégrant des stratégies de brassage. Une solution est construite élément par élément en utilisant un ordre de priorité initial des éléments à chaque étape. L'affectation d'un examen sélectionné à une période a été effectuée par une composante stochastique. La difficulté des examens non affectés augmente à chaque itération, si l'examen ne peut pas être affecté.

Asmuni et al. [14] ont utilisé trois heuristiques d'ordonnancement basées sur la coloration des graphes. La logique floue a été utilisée pour mesurer la difficulté de planification des examens afin de guider la construction des plannings faisables et de haute qualité. Saber et al. [89] ont utilisé une nouvelle hyper-heuristique constructive de coloration de graphe pour résoudre UESP. Dans ce travail, quatre hybridations d'heuristiques ont été utilisées pour classer les examens. Elles produisent quatre listes triées : plus grand degré, degré de saturation, plus grand

degré de couleur et plus grand enrôlement. Pour chaque liste, l'indice de difficulté d'affecter le premier examen est calculé en tenant compte de l'ordre dans toutes les listes pour obtenir une évaluation combinée de sa difficulté. L'examen le plus difficile à être affecté est prévu d'abord dans une période en utilisant la technique de roulette (Roulette wheel selection). Burke et al. [26] ont introduit une nouvelle stratégie consistant à utiliser des combinaisons linéaires des heuristiques primitives pour les graphes pondérés. Les poids des combinaisons d'heuristiques définissent les rôles spécifiques de chaque heuristique simple qui contribue au processus d'ordonnancement des sommets.

Abdul-Rahman et al. [1] ont introduit une décomposition et une stratégie de classement adaptative pour construire le planning. Les examens ont été divisés en ensembles difficiles et faciles avec un ensemble supplémentaire situé entre les deux appelé ensemble limite. Au départ, tous les examens sont supposés faciles. Ceux qui ne peuvent pas être placés dans une période du planning en raison de certains conflits avec d'autres examens déjà affectés, sont inclus dans l'ensemble difficile à traiter par la suite. Une stratégie de brassage a été utilisée pour changer l'ordre donné des examens. Le plus grand degré et le degré de saturation ont été utilisés pour réorganiser les examens dans les ensembles difficiles et faciles séparément. L'approche était similaire à celle proposée par Qu et Burke [82] avec quelques différences. Cette approche commence par un classement initial d'examens à l'aide du degré de saturation. La taille de l'ensemble difficile qui est précédé est augmentée quand on obtient une solution faisable ou une solution améliorée. En outre, aucune heuristique n'a été utilisée pour réorganiser les examens dans les sous-ensembles.

2.4.2 Méta-heuristiques

Une méta-heuristique est un algorithme de haut niveau conçu pour conduire une simple heuristique de bas niveau pour trouver de bonnes solutions à un problème d'optimisation, en échantillonnant l'ensemble de solutions qui est trop grand pour

être complètement considéré. Une méta-heuristique commence soit par une solution initiale (basée sur la recherche locale) ou un ensemble initial de solutions (basé sur la population), suivi d'une procédure d'amélioration. Un aperçu récent est donné dans [71].

Avec la naissance des méta-heuristiques, un grand pas est fait vers la résolution des problèmes d'optimisation combinatoire. Un grand nombre d'approches basées sur les méta-heuristiques ont été prises en considération. Abdullah et al. [2] ont présenté une approche hybride qui commence avec la génération d'une population aléatoire. Le mécanisme électromagnétique «Electromagnetic-like mechanism (EM)» a été appliqué pour calculer la force pour chaque solution. Puis l'algorithme du grand déluge (Great Deluge) utilise la valeur de la force pour calculer le taux de décroissance.

Etemadi et charkari [43] ont développé une méthode de planification basée sur la classification et la recherche taboue. Tout d'abord, les examens ont été regroupés en grappes appropriées qui sont inférieures ou égales au nombre de périodes, puis, pour chaque classe obtenue, une période appropriée est attribuée en utilisant la recherche taboue (Tabu search). Enfin, la structure des groupes est modifiée de manière que la contrainte molle satisfaite, atteint un maximum.

Pais et Amaral [73] mettent en œuvre un système expert de décision : système d'interférence à base des règles floues dans la tâche de réglage des paramètres de version simple de la recherche taboue (plus précisément pour régler la "tabu tenure") sur la base de deux concepts, fréquence et inactivité. Ces concepts sont liés respectivement au nombre de fois qu'un mouvement est introduit dans la liste taboue et la dernière fois que le déplacement a été tenté et a été empêché. La solution initiale a été obtenue par l'heuristique de degré de saturation. Ensuite, de nouvelles solutions ont été obtenues en utilisant deux structures de voisinages. La durée de l'état tabou pour chaque mouvement est déterminée individuellement en utilisant le système d'interférence à base des règles floues.

Al-Betar et al. [6] ont étudié différents cas qui combinent les principales composantes de la recherche harmonieuse (Harmony search) afin d'enquêter sur leur efficacité sur la qualité des solutions produites pour UESP. Trois principaux groupes de critères ont été pris en compte : mémoire, considération aléatoire et ajustement de hauteur. Chacun de ces critères a été associé à la recombinaison, le hasard et les structures de voisinage, respectivement. Abdullah et al. [3] ont hybridé la liste taboue et l'algorithme mimétique (Memetic algorithm). La population initiale a été générée en utilisant l'heuristique de degré de saturation. Ensuite, les opérateurs de croisement et de mutation ont été appliqués pour produire des améliorations significatives dans la qualité de la solution. La liste taboue est utilisée pour pénaliser les solutions inefficaces.

Alzaqebah et Abdallah [8] ont hybridé un algorithme de colonie d'abeilles artificielles (CAA) modifié avec un algorithme de recherche locale. L'heuristique de plus grand degré de coloration de graphes a été utilisée pour générer des solutions initiales. Pour améliorer les performances de l'algorithme, une stratégie de rupture de sélection et une stratégie auto-adaptative pour sélectionner les structures de voisinage sont ajoutés. Au lieu de l'utilisation d'un algorithme de recherche locale, Alzaqebah et Abdallah (2014) [9] ont employé le recuit simulé (Simulated Annealing) afin d'atteindre un équilibre entre les processus d'exploration et d'exploitation. Les résultats expérimentaux montrent que CAA hybridé avec la stratégie de sélection disruptive surpasse l'algorithme CAA seul.

La colonie d'abeilles a été utilisée aussi par Alzaqebah et Abdallah [10] avec trois principales modifications y compris la stratégie de sélection, le mécanisme d'auto-adaptatif, et les algorithmes de recherche locale. Quatre stratégies de sélection ont été utilisées dans le but d'augmenté la diversité de la population. Ils ont incorporé un mécanisme d'auto-adaptatif pour la sélection du voisinage pendant le processus de recherche. Pour plus d'amélioration de la recherche locale, deux algorithmes, à savoir, le recuit simulé et l'escalade de colline (Hill climbing) avec l'acceptation tardive "Late acceptance hill climbing", ont été utilisés.

Eley [42] a présenté deux approches pour UESP. La première est le système ANTCOL. Il se compose de n cycles, dans chacun de ces cycles chacune des m fourmis construit une solution réalisable en utilisant l'heuristique de degré de saturation et les pistes de phéromones. Ces examens assignés sont ensuite évalués selon la fonction objectif donnée et l'expérience accumulée au cours du cycle est utilisée pour mettre à jour les traces de phéromones. La seconde approche est MMAS qui diffèrent d'ANTCOL dans la façon dont ils utilisent l'information existante. Les sentiers de phéromones ne sont mis à jour que par la fourmi qui a généré la meilleure solution dans un cycle. L'escalade de colline a été utilisée comme la recherche locale pour améliorer les solutions trouvées par les fourmis.

Burke et al. [23] ont utilisé l'algorithme génétique (Genetic Algorithm) pour sélectionner intelligemment les voisinages appropriés au sein de la méthode de recherche de voisinage variable. Chaque chromosome représente une suite de neuf voisinages qui seront appliqués à la solution actuelle. Le processus de recherche commence par le premier voisinage et quand la solution n'est pas acceptée selon une approche de descente (steepest descent), le voisinage suivant est appliqué. A chaque itération de l'algorithme génétique, 70% de la population sont modifiées de manière aléatoire en utilisant des opérateurs de croisement et de mutation. Kolonias et al. [57] ont présenté une hybridation de l'algorithme génétique et la Composante de la descente la plus profonde (steepest descent component). Tout d'abord, les solutions initiales sont créées aléatoirement sur CPU (Central Processing Unit), puis, en utilisant la plateforme GPU (Graphics Processing Unit), l'algorithme génétique est appliqué. L'évaluation des solutions ait lieu et une nouvelle génération est créée. Ensuite, un algorithme de la composante de descente la plus profonde est utilisé pour améliorer encore la qualité des solutions.

Turabieh et Abdullah [99] ont proposé un algorithme hybride d'essaim de poissons (fish swarm) en simulant le mouvement du poisson lors de la recherche de la nourriture à l'intérieur de l'eau. Le mouvement du poisson pour la création d'une nouvelle solution est déterminé sur la base d'un algorithme de recherche simplex Nelder-Mead. Les solutions sont acceptées ou rejetées selon l'algorithme du grand déluge ou la descente la plus profonde. Ahandani et al. [4] Cette recherche

examine l'utilisation de l'optimisation discrète par l'essaim de particules (particle swarm optimization). Les solutions initiales de cette étude sont produites par les heuristiques de coloration de graphe. Une combinaison de mutation, opérateur de recombinaison spécialiste et les heuristiques de coloration de graphe sont utilisés pour mettre à jour la position des particules. Une nouvelle méthode de recherche locale, appelée "two staged hill climbing", est proposée et est utilisée pour l'hybridation. Trois structures pour l'algorithme de l'optimisation discrète par l'essaim de particules et trois stratégies pour l'hybridation ont été proposées.

2.4.3 Hyper-heuristiques

La nouvelle tendance est aux hyper-heuristiques qui sont des méthodes plus générales et plus puissantes. Une hyper-heuristique est considérée comme une heuristique qui choisit des heuristiques [22]. Plusieurs applications d'hyper-heuristique pour UESP ont été rapportées, Un survey presque exhaustif est proposé par Pillay (2014) [79].

Dans la methode proposée par Burke et al. [22] , la recherche taboue a été utilisée aussi comme une hyper-heuristique pour rechercher les meilleures séquences d'heuristiques qui produisent la meilleure qualité de planning. La liste d'heuristiques initiales est composée seulement de l'heuristique de degré de saturation. A chaque étape de la recherche taboue, une séquence d'heuristiques est utilisée pour générer le planning. La qualité de planning généré est utilisée pour guider cette étape de la recherche taboue.

Pillay et Banzhaf. [77] ont utilisé la programmation génétique pour générer une hyper-heuristique d'heuristiques de bas niveau. Chaque heuristique est utilisée pour attribuer un nombre précis d'examens. Les opérateurs de croisement et de mutation sont employés pour évoluer la population de séquences d'heuristiques.

Qu et Burke [83] ont utilisé une hyper-heuristique à base de graphes (Graph Based Hyper-heuristic) pour générer itérativement les séquences d'heuristiques. L'heuristique de degré de saturation avec l'heuristique de degré le plus pondéré

sont hybridées de manière adaptative aux différentes étapes de construction de la solution. Les auteurs ont également étudié et analysé plusieurs différentes techniques de recherche de haut niveau.

Qu et al. [84] présentent une approche heuristique de construction automatisée qui construit des solutions étape par étape en utilisant l'hybridation du plus grand degré avec le degré de saturation adaptative. Les séquences d'heuristiques sont obtenues en utilisant une méthodologie d'hyper-heuristique basée sur un graphe aléatoire itératif. Burke et al. [24] ont présenté une approche qui peut être considérée comme hybridation dynamique de différentes heuristiques de coloration de graphe pour construire des solutions de différentes qualités.

Burke et al. [27] ont présenté une approche hyper-heuristique qui hybride des heuristiques de bas niveau en utilisant "Kempe chain"et la permutation des périodes en deux étapes. Dans la première étape, des séquences aléatoires d'heuristiques sont générées et analysées automatiquement. Les heuristiques répétées dans les meilleures séquences sont fixes tandis que le reste est maintenu vide. Dans la deuxième étape, les séquences sont générées par l'assignation aléatoire d'heuristiques aux positions vides dans une tentative de trouver la meilleure séquence d'heuristique.

Qu et al. [86] ont présenté une approche hyper-heuristique hybride basée sur des algorithmes à estimation de distribution. Au cours de l'évolution, l'estimation de la distribution des gènes fournit naturellement un aperçu de la façon dont les heuristiques de bas niveau sont hybridées. Cela permet d'analyser l'efficacité des heuristiques de bas niveau et aussi de concevoir des mécanismes de recherche plus intelligents.

2.5 Conclusion

Dans ce chapitre nous avons présenté le problème de planification d'examens. Le modèle le plus utilisé du problème a été détaillé. Nous avons présenté également

un certain nombre de techniques qui ont été appliquées au cours des deux dernières décennies à ce problème. Les heuristiques de coloration de graphe ont dominé la littérature dans les premières années. Au cours des dernières années, des techniques plus avancées telles que les méta-heuristiques et hyper-heuristiques sont devenues plus populaires. En outre, de nouvelles idées ont été développées ces dernières années pour améliorer la qualité de la solution par l'introduction de l'hybridation de différentes approches.

Chapitre 3

Méthodes utilisées

3.1 Recuit simulé

Le recuit est le processus physique consistant à chauffer un solide jusqu'à son point de fusion, puis à le refroidir. Kirkpatrick et al. [56] ont découvert l'analogie entre ce processus de recuit et l'optimisation combinatoire, en notant la correspondance entre respectivement les états du système, les changements d'état, la température et l'énergie du premier, les solutions réalisables, les déplacements, le paramètre de contrôle et la fonction objectif de ce dernier. Ils ont ensuite proposé un algorithme d'optimisation combinatoire, qu'ils ont appelé "Recuit Simulé", qui émule le processus de recuit. Depuis son introduction, SA a connu un grand nombre d'applications réussies dans un large éventail de domaines [95]. Cette section rappelle quelques faits basiques. Pour plus de détails, on se reportera aux excellents didacticiels [40, 69].

Considérons un problème d'optimisation combinatoire avec S comme espace de solution et une fonction objectif $f : S \longrightarrow \mathbb{R}$. Tout élément $s \in S$ est une solution. Le voisinage d'une solution donnée s est l'ensemble $N(s)$ de toutes les solutions qui peuvent être obtenues à partir de s par «de petits changements» (la définition précise d'un voisinage est donc spécifique au problème). Étant donné

$s \in S$, tout $s' \in N(s)$ est appelé voisin de s, et le passage de s à s' est appelée transition.

Puisque SA est une version intelligente de la fameuse recherche locale (LS), on peut mieux le comprendre si l'on commence par décrire cette dernière. Dans LS, nous commençons par une solution initiale $s_{initiale} \in S$ qui est temporairement considérée comme la meilleure solution actuelle ($s_{meilleure} = s_{initiale}$). A chaque itération, la meilleure solution actuelle $s_{meilleure}$ est examinée pour l'amélioration en ne considérant que les solutions qui se trouvent dans son voisinage. S'il existe $s \in N(s_{meilleure})$ tel que $f(s) < f(s_{meilleure})$ alors nous avons obtenu une amélioration. Dans ce cas, la solution s devient la meilleure courante ($s_{meilleure} = s$), et la méthode réitère. Sinon, $f(s) \geq f(s_{meilleure})$ pour tout $s \in N(s_{meilleure})$, et LS finit. Notons que l'algorithme qui vient d'être décrit est une recherche locale gloutonne, et que des variantes «non-gloutonnes» existent. La stratégie consistant à accepter uniquement les transitions d'amélioration présente un sérieux inconvénient : la recherche se retrouve piégée dans une solution optimale locale (due à la nature myope de la recherche).

SA, avec beaucoup d'autres méta-heuristiques, a le même principe directeur, sauf que l'on peut échapper à la région d'attraction, en permettant à la recherche d'accepter des transitions non améliorantes. Cependant, la recherche doit être effectuée avec soin afin d'assurer la convergence vers un optimum global. Dans le cas de SA, la recherche est contrôlée d'une manière inspirée par le processus de recuit. L'algorithme basique est présenté en Algorithme 3.1.

En ligne 6, $nb(k)$ est le nombre de transitions à la température T_k. En ligne 12, α est la fonction de réduction ou le taux de refroidissement. A chaque itération, l'algorithme procède de la façon suivante : un voisin de la meilleure solution actuelle est choisi au hasard ; si elle est meilleure, la transition est acceptée ; autrement (ligne 10) on accepte la transition avec une probabilité $\exp(-\delta/T_k)$. Comme on peut le voir, la probabilité d'accepter une transition non améliorante dépend de deux facteurs, la température T_k et le niveau d'augmentation δ dans la fonction objectif f. Pour ces raisons, à la même température, les fortes augmentations sont

rejetées plus fréquemment que les petites. En outre, la probabilité d'accepter des transitions non améliorantes diminue régulièrement.

A partir de concepts en mécanique et des chaînes de Markov, la théorie établit que SA converge vers une solution optimale globale en un nombre exponentiel d'itérations pourvu que la configuration de voisinage satisfait la propriété d'accessibilité (on peut atteindre n'importe quelle solution à partir de n'importe quelle autre par une séquence finie de transitions). Les études empiriques, pour leur part, notent que les deux questions les plus critiques de toute implémentation de SA sont la configuration de voisinage et le schéma de refroidissement, qui est la stratégie utilisée pour contrôler l'algorithme.

Algorithme 3.1 Recuit simulé basique

Sélectionner $s_{initiale} \in S$
$s_{acceptée} \longleftarrow s_{initiale}$
$s_{meilleure} \longleftarrow s_{initiale}$
$k \longleftarrow 0$ // Compte le nombre de changements de température
Soit T_0 la température initiale
Tant que (critère d'arrêt non satisfait)
  Pour $it = 1, \cdots, nb(k)$
    Choisir $s \in N(s_{acceptée})$
    $\delta \longleftarrow f(s) - f(s_{acceptée})$
    Si ($\delta < 0$) $s_{acceptée} \longleftarrow s$
    sinon si (random $[0,1] < \exp(-\delta/T_k)$) $s_{acceptée} \longleftarrow s$
    si ($f(s_{meilleure}) > f(s_{acceptée})$) $s_{meilleure} \longleftarrow s_{acceptée}$
  Fin pour
  $T_{k+1} \longleftarrow \alpha(T_k)$
  $k \longleftarrow k + 1$
Fin tant que
Retourner $s_{meilleure}$

3.2 PPA : un aperçu

Les problèmes de recherche/optimisation dans le monde réel sont notoirement difficiles à résoudre. Or, il se trouve que des problèmes équivalents se rencontrent dans la nature. Par exemple, chercher de la nourriture est un problème de recherche, la propagation et la multiplication pour la pérennité et la survie sont des problèmes

d'optimisation. Ceux-ci sont bien résolus dans la nature par les êtres vivants au cours de leur évolution. Les chercheurs se sont aperçus que la nature est source d'inspiration. Beaucoup de paradigmes connus sont inspirés des processus de la vie, et PPA est l'un des plus récents.

Salhi et Fraga [90], entre autres, ont observé que les plantes, et en particulier le fraisier, montrent ce qui peut éventuellement être considéré comme un comportement intelligent. Ils ont remarqué que ce paradigme peut être exploité pour résoudre le problème d'optimisation combinatoire. Par la suite, ils ont suggéré un algorithme, PPA, qui émule la façon dont le fraisier se propage. PPA appartient à une famille d'algorithmes appelés «Plant intelligence based» [5] ou «Plant-inspired» [17].

3.2.1 Principe de base

Le principe de base de PPA est énoncé dans son pseudo-code en Algorithme 3.2. Toutes les plantes ont une stratégie de propagation sous-jacente. La plupart d'entre elles se propagent par les semences. Certaines, comme la fraise hybride commerciale, cependant, utilise les stolons. Rappelons que les hybrides ne se reproduisent pas par ensemencement de graines. Un stolon est une branche (rhizome), provenant de la plante mère, qui pousse sur le sol. Lorsqu'elle touche le sol, elle produit des racines qui donnent naissance à une autre plante. Certaines plantes utilisent leurs stolons pour explorer le voisinage où ils se trouvent afin de chercher de bons endroits pour se développer et se propager. En général, un bon endroit est celui qui est ensoleillé et a suffisamment de nutriments et d'humidité. Pour améliorer ses chances de survie, un fraisier met en œuvre cette stratégie de base :

— Si elle est dans un endroit fertile, il donne naissance à beaucoup de stolons courts ;

— si elle est dans un mauvais endroit, il génère peu de stolons longs.

La figure 3.1 illustre la méthode de propagation du fraisier en envoyant des courts et longs stolons :

FIGURE 3.1: La propagation du fraisier

Algorithme 3.2 PPA basique

1 Trier les plantes $p_1, ..., p_N$ en ordre croissant selon la fonction objectif
2 Répéter jusqu'à ce que le critère d'arrêt soit satisfait
3 $s \longleftarrow \lceil \%good \times N \rceil$
4 Pour $i = 1, ..., s$ plante p_i génère r stolons courts
5 Pour $i = s + 1, ..., N$ plante p_i génère R longs stolons
6 Trier la nouvelle population (plus large)
7 Retirer un pourcentage de plantes de la queue de la liste triée
8 //Soit N la taille de la population acceptée
9 Retourner p_1

La justification de cette stratégie est la suivante : lorsqu'une plante pousse dans un bon endroit, elle donne la chance à ses descendants de profiter pleinement des conditions favorables, en générant beaucoup d'entre eux dans son voisinage. Au contraire, quand la plante est dans un endroit moins bon, il est inutile de garder ses descendants trop près ; Elle envoie donc de longs stolons pour explorer plus loin, à la recherche de meilleures conditions. En outre, elle ne peut envoyer que peu de ces longs stolons car les ressources dont elle dispose sont limitées et la recherche de meilleures conditions n'a aucune garantie de réussite.

L'exploitation et l'exploration, et surtout l'équilibre entre les deux, sont des caractéristiques essentielles de toute heuristique efficace. L'exploitation et l'exploration se manifestent d'une manière contradictoire. Alors que l'exploitation

concentre la recherche autour d'un optimum local, l'exploration permet à la recherche d'échapper à la région d'attraction de l'optimum local. L'exploitation et l'exploration jouent un rôle central dans PPA. L'exploitation est mise en œuvre par la génération de stolons courts de plantes mères qui sont dans des endroits profitables, tandis que l'exploration est mise en œuvre par l'envoi de longs stolons des plantes mères vivant dans des endroits moins profitables.

Bien que PPA peut être démarré avec une seule plante, il est préférable de l'initialiser avec une population de plantes réparties de façon aléatoire dans l'espace de recherche (ligne 1 de l'algorithme 3.2). Il est nécessaire de trier (lignes 2 et 6) pour distinguer les s plantes bien situées et les $N - s$ plantes situées dans des endroits moins favorables, et pour rejeter les "mauvaises" plantes de la queue de la population triée (ligne 7). Notez qu'il y a pas de schéma de refroidissement pour contrôler l'algorithme. Seulement un pourcentage des plantes générées et de leurs parents est accepté à la ligne 7 de l'algorithme 3.2. Cette sélection a deux raisons. Tout d'abord, elle maintient la population avec une taille acceptable. Deuxièmement, comme dans la nature, elle donne la possibilité au meilleur de se reproduire. L'absence d'un schéma de refroidissement rend PPA plus robuste que des approches qui en sont basées comme le recuit simulé, «Great Deluge» et la méthode «Threshold Accepting».

PPA, comme la plupart des méta-heuristiques, nécessite une personnalisation à travers un ensemble de paramètres. Il existe des paramètres pour lesquels des décisions génériques doivent être prises : la taille N de la population initiale, la proportion $\%good$ de plantes considérées comme étant dans de bonnes conditions, le nombre r de stolons courts à générer à partir de parents situés en terrain favorable et le nombre R de longs stolons qui doivent être générés par des parents situés dans ses endroits moins favorables, $(r > R)$. D'autres paramètres nécessitent des décisions spécifiques au problème : espace de la solution, fonction objectif, la «distance» qu'un stolon doit parcourir, qu'il soit court ou long.

Tous les algorithmes inspirés de la nature souffrent de deux inconvénients sérieux : une théorie limitée pour les étayer et beaucoup de paramètres à estimer.

3.2.2 Application de PPA à l'optimisation combinatoire

Étant donné un ensemble fini X (espace de solution), tout $x \in X$ est appelé solution. La stratégie ci-dessus du fraisier peut être mise en œuvre dans PPA pour résoudre un problème d'optimisation combinatoire, c'est-à-dire rechercher la meilleure solution $x^* \in X$ selon la fonction objectif $f(x)$, $f(x^*) = \min_{x \in X} f(x)$.

Les plantes dans PPA correspondent aux solutions. Les changements d'état ou transitions correspondent au concept de génération de stolons. Le concept de qualité des plantes est clairement pris en compte par la fonction objectif, alors que la distance parcourue par un stolon depuis sa plante mère est captée par la notion de perturbation ou de transition. Un court trajet est obtenu à partir de son parent par une petite perturbation, alors qu'un long stolon est obtenu par un changement substantiel. La raison, en termes du problème d'optimisation combinatoire, est que de bonnes solutions peuvent être obtenues, soit à partir de bonnes solutions par de petites perturbations, soit à partir de mauvaises solutions mais par des changements substantiels.

Les stolons courts exploitent l'espace de la solution à proximité de leur parent en recherchant l'optimum local. Les stolons longs explorent l'espace de la solution afin d'échapper à l'optimum local. La stratégie du fraisier de survie dans son environnement peut donc être considérée comme celle de la recherche de points dans l'espace de solution qui donnent lieu à des meilleures valeurs de la fonction objectif.

3.3 LS, ILS, VLSN, "token-ring" : un aperçu

L'amélioration ou les heuristiques de recherche sont des outils puissants qui ont prouvé leur efficacité dans la résolution des problèmes d'optimisation combinatoire [52]. De nombreux auteurs ont utilisé avec succès ces méthodes dans leurs investigation du problème de planification d'examens [21, 23, 28, 30].

3.3.1 LS

Supposons que l'on se donne un ensemble fini S, dont les éléments sont appelés solutions, et on se demande de trouver une solution $s^* \in S$ minimisant une fonction donnée $f(s)$ définie sur S. La recherche locale (ou l'amélioration) est une méthode basique conçue pour résoudre un tel problème. La recherche commence par une solution initiale s_0, puis une meilleure, s_1, est trouvée. La recherche de s_1 est effectuée seulement dans le "voisinage "de s_0. Une fois que, s_1 est trouvée, la recherche est itérée en recherchant s_2 dans le voisinage de s_1, ... jusqu'à ce que s_k est trouvée telle qu'il n'y ait pas de meilleure solution dans son voisinage. $s^* = s_k$ est donc un minimum local de la fonction f.

Ce qui est important dans la recherche locale est la définition du concept plutôt vague de voisinage. L'ensemble des solutions considérées comme étant à proximité d'une solution donnée s, est appelé voisinage de s, et il est défini comme étant l'ensemble $N(s)$ de toutes les solutions qui peuvent être obtenues à partir de s par une petite "perturbation". Par conséquent, la définition précise de $N(s)$ dépend du problème. Au cours de la recherche locale, le passage d'une solution à l'un de ses voisins est désigné comme transition.

Contrairement aux techniques de recherche plus élaborées, la recherche locale est sans paramètre. Elle présente toutefois un sérieux inconvénient. En raison de sa nature myope, la recherche se retrouve piégée dans la région d'attraction de l'optimum local s^*.

3.3.2 ILS

Comme toute méta-heuristique, la recherche locale itérée (LSI) est un algorithme de haut niveau conçu pour conduire une heuristique de bas niveau pour trouver de bonnes solutions à un problème d'optimisation, en échantillonnant l'ensemble de solutions qui est trop grand pour être recherché complètement.

Le mécanisme de haut niveau de la LSI est la notion de "perturbation-itération". En général, l'heuristique de bas niveau employée est la recherche locale, et nous avons le schéma suivant :

a) Une solution initiale s est générée ;

b) La recherche locale est appliquée pour trouver un optimum local s^* dans $N(s)$;

c) s^* est perturbé pour obtenir une solution s ;

d) Les étapes b) et c) sont répétées jusqu'à ce que l'on rencontre une condition de terminaison.

Par conséquent, les principaux éléments constitutifs sont la recherche locale intégrée et la perturbation. L'excellent tutoriel [59] donne une description complète de la LSI. Il présente ses problèmes de mise en œuvre, souligne sa simplicité conceptuelle et rend compte de son efficacité. Par conséquent, il n'est pas surprenant que l'intérêt pour la LSI se développe de nos jours [32, 63, 101].

Puisque la LSI est sensible au choix de la recherche locale, Lourenço et al (2010) [59] recommandent de prendre soin de ce choix chaque fois que possible. Ils recommandent également un équilibre entre petites et grandes perturbations. En effet, si une perturbation est trop petite, on court le risque que la solution perturbée reste dans la région d'attraction de s^*. S'il est, au contraire, la perturbation trop grande, la solution obtenue sera presque aléatoire, et nous obtenons un "multi-start". Idéalement, la perturbation doit être "annulable" par l'heuristique de bas niveau.

3.3.3 VLSN

La recherche VLSN se réfère à la recherche où la taille du voisinage croît de façon exponentielle avec la taille de l'instance du problème, ou bien elle est si grande qu'il ne peut pas être recherché explicitement. Un bon tutoriel sur la recherche VLSN se trouve dans [80].

Une propriété importante des voisinages de taille exponentielle est qu'ils offrent la possibilité de trouver de meilleures solutions locales. Une contrepartie, cependant, est que cela prend beaucoup plus de temps de les explorer. Pour cette raison, la recherche VLSN est attrayante soit lorsque le voisinage peut être recherché efficacement, ou quand il ya une procédure pour rechercher seulement ses parties "prometteuses". Un fait notable est que la communauté des chercheurs en optimisation montre un intérêt croissant pour la recherche VLSN [39, 44, 46].

3.3.4 "Token-ring"

Une pratique courante consiste à combiner les voisinages afin d'accroître l'efficacité de la recherche. Nous pouvons imaginer plusieurs façons de combiner deux voisinages ou plus. Par exemple, étant donné une solution s et deux structures de voisinage N_1 et N_2, nous pouvons chercher dans $N_1 \cup N_2(s) = N_1(s) \cup N_2(s)$. Une autre idée est d'explorer $N_1(s)$, et quand un optimum local s' est trouvé, on cherche un nouvel optimum local, mais dans $N_2(s')$. Cette recherche est appelée token-ring [37], et est symbolisée par $N_1 \longrightarrow N_2$ [60].

Chapitre 4

Recuit simulé pour UESP

4.1 Introduction

Puisque UESP est NP-dur, une approche pratique et raisonnable consiste à utiliser une heuristique capable de trouver des plannings de bonne qualité en un temps de calcul acceptable. Dans ce chapitre nous proposons une heuristique basée sur le recuit simulé et la recherche locale. Notre heuristique peut être vue comme une méthode en deux phases : en utilisant une simple heuristique de coloration de graphe, la première phase commence par construire un planning faisable. Dans la deuxième phase, le recuit simulé est utilisé comme une heuristique itérative pour améliorer la qualité du planning (en minimisant la violation des contraintes molles, c'est-à-dire la dispersion des examens en conflit).

SA, en tant qu'une méthode de recherche locale, constitue une option intéressante pour résoudre les problèmes complexes d'optimisation. Son principal avantage est qu'il est potentiellement capable de fournir des solutions optimales ou très proches de l'optimum, mais son défaut le plus grave, cependant, est que c'est un processus qui prend beaucoup de temps. Malheureusement, nous ne connaissons que peu d'articles qui utilisent SA dans le cadre du problème de planification d'examens [21, 41, 67, 97, 98]), et un seul d'entre eux considère l'ensemble des données de l'Université de Toronto [67].

4.2 Notre contribution

L'efficacité de SA dépend fortement d'un choix approprié des paramètres qui contrôlent la recherche. Les deux paramètres essentiels sont la structure de voisinage et le schéma de refroidissement (c'est-à-dire la façon dont la température est ajustée). C'est exactement là où réside notre contribution.

- Dans ce travail, nous proposons trois structures de voisinage. Le premier, appelé Exam-Shifting, considère un examen et essaie de l'affecter à une autre période. Le deuxième, appelé Timeslots-Interchange, considère la permutation des examens de deux périodes différentes. Le troisième est Kempe chain, qui tente de permuter des sous-ensembles d'examens dans deux périodes différentes. L'échange est bien connu et Kempe chain est largement utilisée [21, 45, 67, 98]. Il est bien connu que la Kempe chain est le meilleur choix ([40, 67]). Si Exam-Shifting se révèle attrayant, l'exploration de Kempe chain prend du temps. Pour permettre un grand nombre de transitions, il est exploré seulement 20% du temps, tandis que les deux autres sont considérés pendant 40% du temps chacun. Au stade de nos connaissances, c'est la première fois que le premier voisinage est proposé pour la recherche dans le contexte de planning d'examen. En outre, les auteurs considèrent généralement un seul voisinage pour la recherche. Nous ne connaissons qu'un seul article de Aycan et Ayav (2009) [15] qui considère simultanément deux voisinages. Les auteurs rapportent que l'exploration simultanée de deux voisinages est préférable à leur recherche séparée. Dans cet article, nous considérons trois structures différentes simultanément.
- Conformément aux recommandations de Dowsland et Thompson [40] et Thompson et Dowsland [97, 98], où plusieurs études empiriques sont menées sur des implémentations réussies de SA, nous avons adopté un schéma de refroidissement géométrique robuste et très lent, mais consacrant moins de temps à explorer les voisinages à haute température et à laisser plus de temps progressivement.

4.3 Méthodologie

Cette section présente l'approche proposée. Des détails sont donnés pour rendre l'algorithme reproductible (Algorithme 4.1). Rappelons que l'espace de solution S de UESP est l'ensemble de toutes les θ-colorations légales. La structure de données adoptée est assez simple. Une solution est codée sous la forme d'un θ-tableau $[P_1, \cdots, P_\theta]$ où l'entrée P_i est un pointeur vers une liste chaînée dont les nœuds représentent les examens appartenant à la période t_i. La famille $P_1, \cdots, P_\theta$ constitue une partition de l'ensemble des examens et tout P_j (un ensemble stable du graphe G) représente la $j^{\text{ième}}$ période dont les éléments sont les examens assignés à la $j^{\text{ième}}$ période (sommets colorés de couleur j). La figure 5.1 illustre un exemple de la structure générale d'une solution pour UESP. Une représentation directe est utilisée. Chaque solution générée est représentée comme une matrice qui contient des informations sur les périodes et les examens. Par exemple les examens, E7, E11, E14, E19, E20 sont programmés dans la période P1.

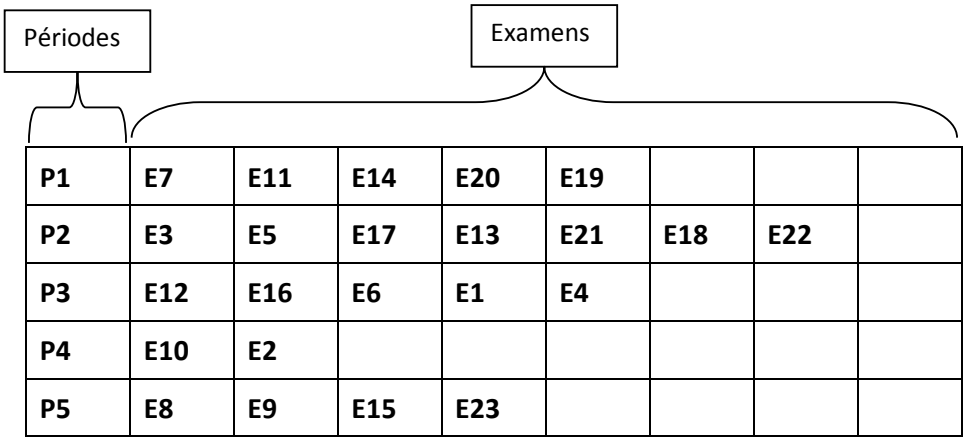

FIGURE 4.1: Représentation d'une solution

4.3.1 Solution initiale

La solution initiale (voir la ligne 1 du pseudo-code dans l'Algorithme 4.1), qui peut être quelconque, est obtenue en employant une heuristique de coloration de graphe

bien connue et largement utilisée “Saturation degree with Backtracking”au graphe mentionné dans l'introduction (voir [4, 18, 31] pour une présentation complète, et [61] pour des méthodes et des applications de coloration de graphe plus générales).

Algorithme 4.1 Pseudo-code de la méthode proposée

```
1  calculer un planning initial réalisable s_initiale
2  s_acceptée ⟵ s_initiale
3  s_meilleure ⟵ s_initiale
4  k ⟵ 0
5  Soit T_0 and T_f respectivement la température initiale et la température finale
6  Tant que ( T_k ≤ T_f  et le temps écoulé < 7200 )
7     Pour it = 1, ··· , nb(k)
8        si ( random [0,1[ ≤ 0.2 ) choisir s ∈ N_1 (s_acceptée)
9        sinon si ( 0.2 < random [0,1[ ≤ 0.6 )  choisir s ∈ N_2 (s_acceptée)
10       sinon si ( 0.6 < random [0,1[ ≤ 1 )  choisir s ∈ N_3 (s_acceptée)
11       δ ⟵ f (s) − f (s_acceptée)
12       si ( δ < 0 ) s_acceptée ⟵ s
13       sinon si ( random [0,1[ < exp (−δ/T_k) ) s_acceptée ⟵ s
14       si ( f (s_meilleure) > f (s_acceptée)) s_meilleure ⟵ s_acceptée
15    fin pour
16    k ⟵ k + 1
17    T_k ⟵ α (T_k)
18 Fin tant que
19 Retourner s_meilleure
```

Saturation degree with Backtracking consiste à colorier d'abord les examens avec le plus petit nombre de couleurs disponibles (périodes). Cette heuristique est dynamique dans le sens où le nombre de couleurs légales disponibles pour les sommets non coloriés doit être recalculé après chaque coloration de sommet. Si aucune période valide n'est disponible pour un examen donné, Backtracking tente de lui faire de la place en libérant des examens déjà attribués. Des règles doivent être imposées pour éviter le cyclage. Burke et Newall [18] soulignent l'efficacité élevée de cette heuristique pour produire des plannings de départ de bonne qualité.

4.3.2 Configurations de voisinage

Une préoccupation cruciale dans la conception de toute heuristique de recherche locale est la façon dont le voisinage est défini. Dans ce travail, nous avons trois propositions.

1) Exam-Shifting : une solution voisine est obtenue à partir de la solution courante en déplaçant un examen donné. Cela signifie que l'examen est éjecté de la période à laquelle il appartient et réaffecté à une autre. Malheureusement, cela n'est pas toujours possible. Pour que l'examen k puisse incorporer la période t_j il faut s'assurer que $c_{ki} = 0$ pour tout $i \in t_j$, ce qui signifie que l'examen k est compatible (ou non conflictuel) avec tous les examens dans la période j. Comme il existe n examens et θ périodes, la taille de ce voisinage est clairement $O(n \times \theta)$.
2) Timeslots-Interchange : Un voisinage est obtenu en échangeant deux périodes. Évidemment, cela est toujours possible sans affecter la faisabilité. La taille du voisinage est $O(\theta^2)$ puisqu'il existe $\theta(\theta - 1)/2$ manières de choisir deux périodes distinctes.
3) Kempe chain : Considérons deux periodes arbitraires t_i et t_j et considérons le graphe biparti $B = (t_i, t_j, E)$ où E est l'ensemble des arêtes reliant deux examens en conflit, l'un appartenant à t_i et l'autre à t_j. Kempe chain désigne toute composante connexe $C = C_i \cup C_j$ dans le graphe biparti B où $C_i \subset t_i$ (resp. $C_j \subset t_j$). Dans cette structure de voisinage, un voisin est obtenu en échangeant les ensembles C_i et C_j. Un exemple est donné dans la figure 4.2, où $C_i = \{e_3^i\}$ et $C_j = \{e_2^j, e_4^j, e_5^j\}$. Il est facile de voir que cette transition préserve la faisabilité. La taille du voisinage dépend de l'instance. Il existe $\theta(\theta - 1)/2$ manières de choisir deux périodes, mais, compte tenu de ces derniers, le nombre de composantes connexes dans le graphe biparti sous-jacent peut être exponentiel en la cardinalité de $t_i \times t_j$. Ainsi, Kempe chain définit de grands voisinages. Il est bien établi que le voisinage est d'autant meilleur qu'il est grand, mais aussi il faut plus de temps pour l'explorer. En fait, la recherche menée par Thompson et Dowsland [98] conclut que Kempe chain est le meilleur choix [6].

Alors que les deux structures de voisinages basées sur la permutations des périodes et Kempe chain satisfont la propriété d'accessibilité, malheureusement, nous ne disposons pas de preuve formelle si oui ou non la première structure de voisinage, Exam-Shifting, remplit la propriété. Dans le négatif, un biais peut être

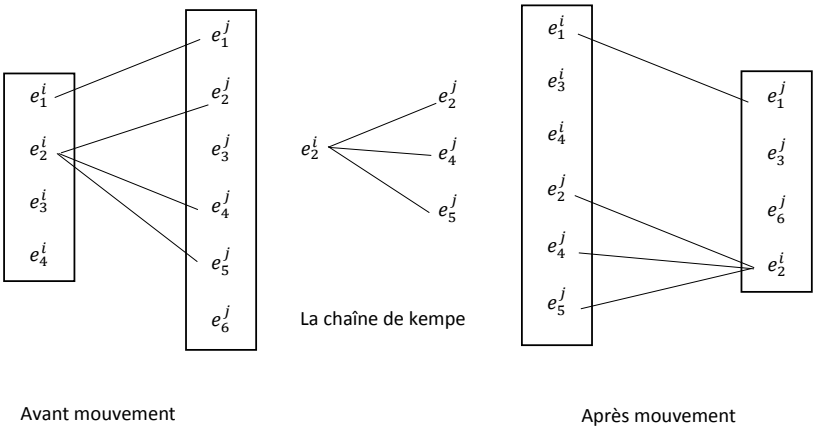

FIGURE 4.2: Exemple de déplacement en “kempe chain”

introduit dans la recherche qui peut empêcher l'algorithme SA de converger vers une solution optimale globale.

Les lignes 8-14 de l'Algorithme 4.1 constituent une transition, acceptée ou non. La ligne 14 met à jour le meilleur planning trouvé jusqu'à présent, qui sera retourné à la fin de la méthode. En raison de la surcharge causée par la recherche, le stockage et la mise à jour des Kempe chains, nous avons adopté la stratégie suivante dans les lignes 8-10 du pseudo-code de l'algorithme de SA. On lance une pièce à “trois faces”, les faces 2 et 3 avec une probabilité $0,4$ chacun, et la face 1 avec une probabilité $0,2$. Quel que soit la face apparaissant (la face 1 correspond au voisinage Kempe chain), nous choisissons le voisinage correspondant pour la recherche.

4.4 Expérimentation numérique

4.4.1 Design expérimental

La méthode proposée est codée en C et exécutée sur un Intel Core I3, 2,4 GHz. Le code est testé sur l'ensemble de données de l'université de Toronto. Les caractéristiques des instances sont présentées dans les cinq premières colonnes du

tableau 2.1.

4.4.2 Réglage des paramètres

Plusieurs essais sont effectués pour le réglage des paramètres de refroidissement. La température initiale (ligne 5 de l'Algorithme 4.1) est fixée à 1. A cette valeur, environ 80% en moyenne des transitions sont acceptées. Ensuite, le refroidissement commence jusqu'à ce qu'il atteigne une température finale de $0,1$. La fonction de réduction, ou taux de refroidissement, est déterminée en suivant les recommandations de Thompson et Dowsland [96, 98]. Nous avons adopté un programme de refroidissement géométrique en autorisant $\alpha(T) = 0.99 \times T$ dans la ligne 17 en Algorithme 4.1. Ceci conduit l'algorithme à effectuer environ 230 changements de température puisque $0.1 \simeq 0.99^{230}$.

Il est également recommandé dans [40] de passer moins de temps à explorer le voisinage à haute température où la plupart des transitions sont acceptées. Le mieux est de laisser plus de temps progressivement. Dans notre implémentation, nous avons adopté la fonction suivante $nb(k) = 100 \times 1.02^k$ pour le nombre de transitions à la température T_k (ligne 7 en Algorithme 4.1). Ainsi, au début, à la température 1, 100 itérations sont autorisées, et à la température finale, le nombre d'itérations est d'environ 9500. Cette stratégie, confirmée par l'expérience, est plutôt exigeante en termes d'effort de calcul [69].

Deux critères d'arrêt sont imposés. L'algorithme termine soit lorsque la température atteint sa valeur la plus basse $0,1$, soit lorsque le temps de calcul atteint une limite de temps de 7200 secondes.

4.4.3 Résultats

Les valeurs moyennes des solutions initiales (sur 10 essais) sont indiquées dans les sixième et septième colonnes du tableau 4.1, intitulées «coût» et «temps», qui se rapportent respectivement à la valeur de la fonction objectif 2.1 du planning

Table 4.1: Solutions initiale et finale

	Solution initiale		Meilleure solution
Instance	Coût	Temps	Coût
CAR91	11.01	0.68	5.26
CAR92	8.39	0.48	4.34
EAR83	57.18	0.17	34.17
HEC92	17.63	0.21	10.43
KFU93	44.88	0.28	14.36
LSE91	26.21	0.23	11.25
RYE93	30.96	0.31	9.47
STA83	182.49	0.23	157.13
TRE92	14.67	0.20	8.47
UTA92	6.60	0.48	3.56
UTE92	48.98	0.21	25.71
YOR83	52.36	0.21	36.92

initial obtenu en utilisant l'heuristique Saturation Degree with Backtracking, et le temps de calcul moyen. On voit que le planning initial est obtenu très rapidement, en moins d'une demi-seconde, mais a une très mauvaise qualité car il est obtenu sans se soucier de la contrainte molle.

Notre méthode est exécutée 10 fois sur chaque instance, chaque fois avec une nouvelle solution initiale. Les coûts des meilleures solutions trouvées sont enregistrés dans la dernière colonne du tableau 4.1. Comme on peut le voir, il existe une grande déviation (d'environ 50%) de la meilleure solution à partir de la solution initiale. Nous observons, dans les itérations précoces, à haute température, plusieurs améliorations successives de la valeur de la fonction objectif. Au contraire, dans les itérations finales, à basse température, la plupart des transitions sont rejetées, et nous obtenons des améliorations sporadiques.

Les temps de calcul atteignent toujours la limite de temps de 7200 secondes pour chaque exécution sur chaque instance. Cela est prévu. Étant donné que le programme de refroidissement est très lent, un grand nombre de transitions (acceptées ou non) $100 \times \sum_{k=0}^{k=230} 1.02^k$ sont autorisées. Sans imposer une limite de temps, les temps de calcul seraient quelquefois prohibitifs.

4.4.4 Comparaison avec 10 méthodes existantes

Pour évaluer la performance de notre algorithme (appelé CH), nous le comparons avec 10 heuristiques récentes. Ces méthodes sont nommées par l'acronyme des initiales des noms des auteurs.

- ABBMO [1] propose une approche constructive hyper-heuristique basée sur la décomposition adaptative et le classement de quatre heuristiques de coloration de graphe de bas niveau.
- AVZG [4] considère une "hybrid particle swarm optimisation"incorporée dans une structure hyper-heuristique, où une combinaison d'opérateurs de mutation, d'opérateur de recombinaison et de coloration de graphe est utilisée pour mettre à jour la position des particules.
- BKAA [16] présente un algorithme hybride de colonies d'abeilles artificielles.
- BMMPQ [22] est une hyper-heuristique multi-étapes où la recherche taboue est utilisée pour choisir entre les heuristiques de graphe.
- BQS [27] est une autre hyper-heuristique qui hybride des heuristiques de graphe de bas niveau.
- CDI [29] est un autre algorithme hybride.
- PA [73] suggère un système basé sur des règles d'inférence floues pour ajuster certains paramètres de la recherche taboue.
- PB [78] présente une hyper-heuristique basée sur une combinaison hiérarchique d'heuristiques.
- QB [83] est encore une autre hyper-heuristique qui hybride des heuristiques de coloration de graphe de bas niveau.
- SAQK [89] présente une hyper-heuristique constructive de coloration de graphe.

Les auteurs de toutes ces méthodes (à l'exception de CDI où seulement cinq essais sont autorisés) choisissent la meilleure solution parmi 10 exécutions pour chaque instance. Le tableau 4.2 montre le coût de la meilleure solution trouvée par chaque méthode (extraite de la littérature). La marque − dans une cellule

TABLE 4.2: Comparaison de notre approche avec les méthodes existantes du point de vue de la précision

	ABBMO	AVZG	BKAA	BMMPQ	BQS	CDI	PA	PB	QB	SAQK	CH
CAR91	5.17	5.22	**4.22**	5.41	5.19	6.6	5.46	4.97	5.16	5.14	5.26
CAR92	4.74	4.67	5.00	4.84	4.31	6.0	4.57	4.28	**4.16**	4.70	4.34
EAR83	40.91	35.74	34.08	38.19	35.79	**29.3**	33.50	36.86	35.86	37.86	34.17
HEC92	12.26	10.74	10.32	12.72	11.19	**9.2**	10.52	11.85	11.94	11.90	10.43
KFU93	15.85	14.47	13.91	15.76	14.51	**13.8**	14.05	14.62	14.79	15.30	14.36
LSE91	12.58	10.76	11.04	13.15	10.92	**9.6**	-	11.14	11.15	12.33	11.25
RYE93	10.11	9.95	9.18	-	-	**6.8**	9.11	-	-	10.71	9.47
STA83	158.12	157.10	**157.04**	157.38	157.18	158.2	157.29	158.19	158.33	160.12	157.13
TRE92	9.30	8.47	8.38	8.85	8.49	9.4	8.71	8.48	8.60	**8.32**	8.47
UTA92	3.65	3.52	**3.40**	3.88	3.44	3.5	3.71	**3.40**	3.42	3.88	3.56
UTE92	27.71	25.86	25.80	31.65	26.70	**24.4**	25.18	28.88	28.30	32.67	25.71
YOR83	43.98	38.72	36.53	40.13	39.47	**36.2**	39.08	40.74	40.24	40.53	36.92
	3.16	1.57	1.04[2]	2.94	1.60	0.55[1]	1.72	2.16	2.03	3.08	1.22[3]

signifie que la valeur correspondante n'est pas reportée. La meilleure valeur parmi les onze est en gras. On peut voir que notre méthode ne trouve jamais la meilleure. Les deux méthodes BKAA et CDI semblent les mieux qualifiés de ce point de vue. Mais si l'on considère la dernière rangée du tableau 4.2 qui rapporte l'écart moyen par rapport au meilleur (qui est, à notre avis, un paramètre plus objectif de comparaison), notre méthode prend le troisième rang derrière les méthodes BKAA et CDI. Notre méthode est compétitive avec BKAA, mais les deux sont dominés par CDI. Ceci est dû au fait que la méthode CDI trouve deux solutions extraordinairement bonnes pour les instances EAR83 et RYE93.

Le tableau 4.3 classe les 11 méthodes du point de vue de la meilleure solution trouvée. La dernière rangée de la même table donne un rang moyen pour les méthodes comparées. Encore une fois, nous pouvons voir dans ce tableau que notre méthode est compétitive avec les deux méthodes BKAA et CDI. Les deux paramètres de comparaison (écart moyen par rapport au mieux et au rang moyen) sont donc cohérents et donnent presque la même conclusion : les trois meilleures méthodes sont BKAA, CDI et CH, les deux premières dominant légèrement la dernière.

Table 4.3: Rang moyen des onze heuristiques

	ABBMO	AVZG	BKAA	BMMPQ	BQS	CDI	PA	PB	QB	SAQK	CH
CAR91	5	7	1	9	6	11	10	2	4	3	8
CAR92	8	6	10	9	3	11	5	2	1	7	4
EAR83	11	5	3	10	6	1	2	8	7	9	4
HEC92	10	5	2	11	6	1	4	7	9	8	3
KFU93	11	5	2	8	6	1	3	7	9	10	4
LSE91	9	2	4	10	3	1	-	5	6	8	7
RYE93	6	5	3	-	-	1	2	-	-	7	4
STA83	7	2	1	6	4	8	5	9	10	11	3
TRE92	10	3	2	9	6	11	8	5	7	1	3
UTA92	8	6	1	10	4	5	9	1	3	10	7
UTE92	7	5	4	10	6	1	2	9	8	11	3
YOR83	11	4	2	7	6	1	5	10	8	9	3
	8.58	4.58	2.92[1]	9.00	5.09	4.42[2]	5.00	5.91	6.55	7.83	4.42[2]

4.5 Conclusion

Dans cette partie, nous traitons un problème NP-dur, UESP, pour lequel une heuristique est conçue. Commençant par un premier placement obtenu à l'aide d'une heuristique de coloration de graphe simple, SA est utilisé pour améliorer itérativement sa qualité (dispersion des examens sur les intervalles de temps). Trois voisinages distincts sont proposés simultanément pour la recherche, et un programme de refroidissement géométrique robuste est conçu pour contrôler le changement de température. Les résultats obtenus sur l'ensemble de données de l'université de Toronto et la comparaison de deux points de vue (écart moyen par rapport au meilleur et classement) par 10 méthodes récentes montrent que notre méthode est compétitive avec l'état de l'art. La conclusion principale qui peut être tirée de cette étude est que SA, bien que prenant beaucoup de temps, demeure un bon outil, lorsqu'il est bien contrôlé, pour résoudre les problèmes d'optimisation combinatoire.

Chapitre 5

Hybridation de PPA avec LS pour UESP

5.1 Introduction

Puisque UESP est NP-dur, les approches de solutions exactes exigent souvent des temps de calcul prohibitifs. Il est donc souhaitable de se contenter d'une approche pratique et raisonnable capable de trouver de bons plannings dans un temps de calcul acceptable. La méthode que nous avons à l'esprit et que nous proposons dans ce travail consiste à hybrider PPA avec LS. Nos contributions peuvent être résumées comme suit :

- Nous présentons quelques améliorations de PPA et montrons comment l'adapter à UESP.
- LS est hybridé pour améliorer la qualité des plannings générés itérativement.
- Des tests approfondis sont réalisés sur l'ensemble de données de l'Université de Toronto.
- Notre méthode hybride améliore les résultats obtenus avec SA au chapitre précédent.
- Elle est comparée avec treize méta-heuristiques récentes.

5.2 Hybridation de LS

LS contribue de manière remarquable dans la résolution des problèmes d'optimisation combinatoire. Il existe différentes options pour LS. L'option choisie dans cette hybridation est la méthode de la descente la plus raide (steepest descent) qui accepte toujours la meilleure amélioration (quand elle existe). Le but de LS, dans ce travail, est d'améliorer chaque planning généré par l'algorithme, du point de vue de la fonction objectif dans 2.1. La caractéristique la plus importante de LS est la définition de la structure de voisinage. Dans cette étude, deux structures Exam-shifting et Kempe chain, déjà utilisées dans [34] (voir la section 4.3.2), sont proposées pour la recherche.

Une question qui est souvent posée est de savoir pourquoi certains voisinages conduisent-ils à de bons résultats qui souvent indépendants de la solution initiale, alors que d'autres conduisent à des solutions de moindre qualité et fortement corrélées à la première ? Si nous nous référons au premier type de voisinage comme bon, quelles sont les caractéristiques d'un bon voisinage ? Qu'est-ce qui rend une structure mieux qu'une autre ?

Toutes ces questions sont traitées dans l'excellente référence [60]. En considérant le problème d'emploi du temps, des études de cas et trois critères d'évaluation, la qualité de quatre voisinages est étudiée. La recherche conclut que Kempe chain est l'une des structures les plus puissantes.

Étant donné qu'un minimum local dans une structure n'est pas nécessairement un minimum local dans une autre, la combinaison de différentes structures peut être attrayante. Par conséquent, une autre question posée par Lü et al. [60] concerne la combinaison des structures et la façon de les explorer. Une fois de plus, l'étude conclut que Exam-Shifting et Kempe chain explorés avec une stratégie token-ring [38] constitue une bonne option. La recherche token-ring consiste à explorer un voisinage, et lorsqu'un optimum local est atteint, la recherche se poursuit au voisinage de ce dernier mais avec une autre structure. Ces choix, recommandés par [60], sont étroitement respectés dans ce travail.

Notons que la nature discrète de l'espace de recherche (plannings dans notre cas) permet de considérer les minimums locaux dans un voisinage comme une solution dans un voisinage différent. Cela ne fonctionnerait pas dans un espace euclidien continu. Le processus d'exploration des voisinages est illustré à la figure 5.1.

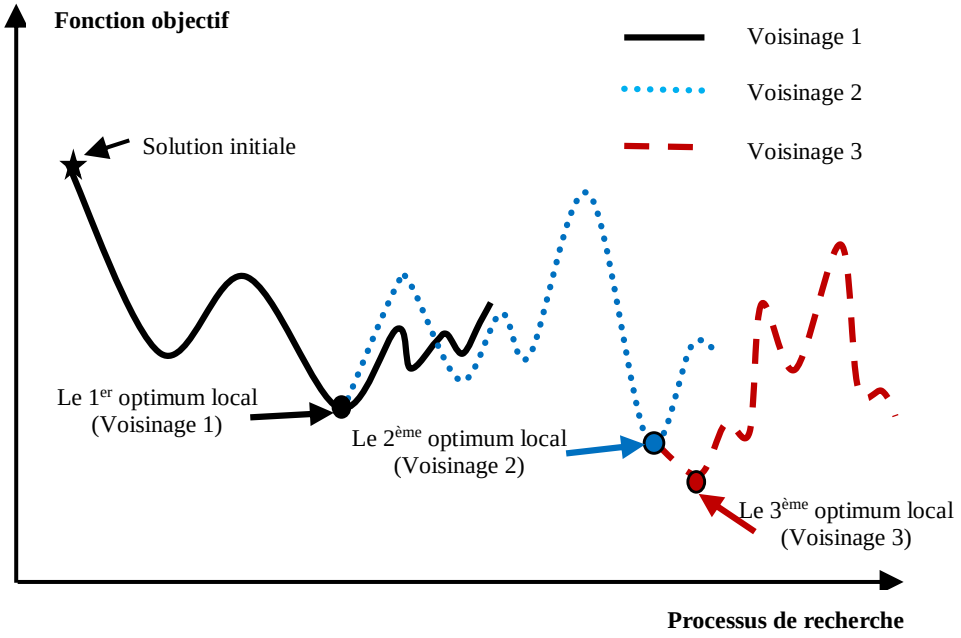

FIGURE 5.1: Stratégie "token-ring"

5.3 Algorithme proposé

Le pseudo-code de la méthode proposée est donné en Algorithme 5.1 3. Le critère de terminaison dépend du temps alloué à l'algorithme. A l'aide d'une heuristique de coloration de graphe, on commence en ligne 1 par générer une population initiale. Chaque planning généré (lignes 9 et 14) est amélioré (lignes 10 et 15) par LS. Le planning amélioré résultant est inséré dans la population à l'endroit approprié pour éviter d'avoir à trier la population de nouveau.

Algorithme 5.1 Algorithme hybride

// **Initialisation**
1 Générer N plannings distincts $T_1, ..., T_N$
2 Pour $i = 1, \cdots, N$ améliorer T_i // Soit $\{\tau_1, \cdots, \tau_N\}$ la population P améliorée
3 Trier P en ordre croissant en fonction des valeurs de la fonction objectif
4 $s \longleftarrow \%good \times N$
5 $Q \longleftarrow P$
// **Répéter**
6 **Tant que** $(Temps < Timelimit)$
7 **Pour** $i = 1, \cdots, s$
8 **Pour** $j = 1, \cdots, r$
9 - Générer un planning ρ_j à partir du planning τ_i
10 - Améliorer ρ_j
11 - $Q \longleftarrow Q \bigcup \{\rho_j\}$// Insérer le planning ρ_j à l'endroit approprié
12 **Pour** $i = s + 1, \cdots, N$
13 **Pour** $j = 1, \cdots, R$
14 - Générer un planning ρ_j à partir du planning τ_i
15 - Améliorer ρ_j
16 - $Q \longleftarrow Q \bigcup \{\rho_j\}$// Insérer le planning ρ_j à l'endroit approprié
17 Garder dans Q seulement N plannings selon un certain critère d'acceptation
18 $P \longleftarrow Q$ // Soit $P = \{\tau_1, ..., \tau_N\}$
19 **Fin tant que**
20 Retourner τ_1

5.3.1 Génération de stolons courts et longs

Les stolons courts sont mis en œuvre en utilisant une petite perturbation : Exam-Shifting. Ici seulement un examen est concerné. Étant donné un planning possible ρ_j dans la ligne 9 (plante mère), un nouveau planning réalisable τ_j (court stolon) est obtenu à partir de son parent en décalant un examen choisi au hasard de sa position dans une période à une autre période. Évidemment, la transition doit maintenir la faisabilité en évitant les conflits entre l'examen choisi et les examens appartenant à la nouvelle période.

Les longs stolons sont générés en utilisant une perturbation sensiblement plus importante : Period-Swapping. Étant donné un planning réalisable ρ_j dans la Ligne 14, un nouveau planning possible τ_j (long stolon) est obtenu à partir de son parent en choisissant au hasard deux périodes, et en les échangeant. Dans ce cas, de nombreux examens, ceux appartenant aux deux périodes, sont perturbés. Il va sans dire que cette mesure n'affecte pas la faisabilité.

5.3.2 Critère d'acceptation ou sélection

Le contenu de cette section constitue un changement par rapport aux versions précédentes de PPA. Premièrement, la taille de la population est maintenue constante. De plus, la version originale de PPA n'accepte que les bons plannings du début de la population triée. Cette règle sélectionne seulement les meilleurs pour se reproduire. Les généticiens objecteraient que cette règle réduit la diversité de la population. En effet, les «mauvais» parents peuvent encore engendrer de «bons» enfants (longs stolons) par des perturbations importantes. A partir de cette observation, une règle de sélection plus commune est adoptée, à savoir «Roulette Wheel». En ligne 17, pour ne garder que N plannings, les autres sont rejetés proportionnellement à leur valeur de la fonction objectif.

5.4 Expérimentation numérique

5.4.1 Design expérimental

Notre méthode est codée en C et exécutée sur un Intel Core I3, 2,4 GHz. Des tests sont effectués sur les bases de données de Toronto, dont les caractéristiques sont présentées dans le tableau 2.1 (voir [31] pour plus d'informations sur cet ensemble de données).

5.4.2 Réglage des paramètres

Il est bien connu que l'efficacité des méta-heuristiques repose fortement sur le paramétrage. Malheureusement, le réglage optimal des paramètres à priori est difficile et même impossible quand ils sont dépendants de l'instance. La manière commune de définir des valeurs de paramètres commence par considérer un petit échantillon d'instances. Les valeurs sont alors affectées aux paramètres et les instances du problème sont résolues avec ces paramètres. Ces derniers sont ajustés en conséquence pour de meilleurs résultats. Ces valeurs sont ensuite adoptées comme

valeur de paramètre par défaut pour une utilisation future. Il va sans dire que la recherche de ces valeurs par défaut est fastidieuse et prend beaucoup de temps.

Au regard de cette procédure générale, nous avons obtenu les valeurs suivantes pour nos paramètres. Pour avoir un temps d'exécution basé sur la taille de l'instance à résoudre, nous imposons $Timelimit = \max\{n \times \theta, 10800\}$ secondes. Cela signifie que, en général, le code fonctionne pendant $n \times \theta$ secondes. Mais certaines instances sont de taille si petite que cette limite est insuffisante. Dans ce cas, 10800 secondes, soit trois heures, sont imparties. La taille N de la population est maintenue constante avec une valeur 10 tout au long de l'exécution. Le pourcentage $\%good$ est fixé à 20%. Cela signifie que les 20% premières plantes dans la population triée est considéré comme vivant dans un bon fertile. Chacune de ces plantes génère alors $r = 3$ stolons courts, et chaque parent vivant dans un moins bon endroit génère $R = 1$ stolon long.

5.4.3 Résultats

Les résultats de calcul sont fournis au Tableau 5.1. Chaque instance est exécutée 10 fois avec les mêmes paramètres. Les en-têtes des colonnes sont explicites. Le coût moyen dans la troisième colonne est obtenu sur 10 exécutions. Le nombre de générations se réfère au nombre de fois où la boucle tant que est exécutée.

5.4.4 Comparaison avec l'état de l'art

Notre algorithme est comparé à treize méthodes publiées récemment (voir Tableau 5.2). La comparaison porte sur le point de vue de la précision et est proposée au Tableau 5.3, où les meilleures valeurs sont en gras. Dans cette perspective, on peut voir que les deux meilleures méthodes sont dues à Caramia et al. [30] et Alzaqebah et Abdullah [10], immédiatement suivies par notre approche, obtenant respectivement 5, 4 et 2 meilleurs résultats. Cette conclusion est confirmée par un autre critère. La dernière ligne du Tableau 5.3 donne l'écart moyen par rapport au meilleur. De ce point de vue aussi notre méthode gagne le troisième rang

TABLE 5.1: Résultats de l'algorithme hybride sur les instances benchmark

	Fonction de coût		
	Meilleure	Moyenne	Nombre de générations
CAR91	5.22	5.29	7
CAR92	4.34	4.39	9
EAR83	33.36	34.25	70
HEC92	10.08	10.19	693
KFU93	13.45	13.71	6
LSE91	10.76	11.25	7
RYE93	8.68	8.85	6
STA83	157.03	157.03	99
TRE92	8.14	8.43	25
UTA92	3.47	5.56	9
UTE92	24.88	24.98	95
YOR83	35.39	36.29	146

derrière les mêmes deux méthodes. Deux questions pertinentes se posent alors : cette conclusion est-elle mathématiquement justifiée ? Existe-t-il des différences significatives entre les méthodes ?

TABLE 5.2: Méthodes en compétition

#	Méthodologie	Auteurs	Référence
0	Notre approche		
1	Approche constructive	Abdul-Rahman et al (2014)	[1]
2	Optimisation par essaim de particules hybrides	Ahandani et al (2012)	[4]
3	Technique mémétique	Al-Betar et al (2014)	[6]
4	l'optimisation des colonies d'abeilles hybride	Alzaqebah and Abdullah (2015)	[10]
5	Combinaisons linéaires d'heuristiques	Burke et al (2012)	[26]
6	sélection adaptative des heuristiques	Burke et al (2014)	[27]
7	Approches basées sur la recherche locale	Caramia et al (2008)	[30]
8	Recuit simulé	Cheraitia and Haddadi (2016)	[34]
9	Algorithme évolutif hybride fonctionnant sur un GPU	Kolonias et al (2014)	[57]
10	Recherche tabou	Pais and Amaral (2012)	[73]
11	Les hybridations dans un hyper-heuristique à base de graphe	Qu and Burke (2009)	[83]
12	Hybridant des heuristiques dans un algorithme de distribution d'estimation	Qu et al (2015)	[86]
13	Un hyper-heuristic constructif de coloration de graphe	Sabar et al (2012)	[89]

5.4.5 Résultats du test de Friedman (Comparaison des k échantillons)

En essayant de donner une réponse, nous avons pris les quatre premières méthodes classées et utilisé le test de Friedman. Les résultats du test sont fournis par XLSTAT (Tableaux 5.4 et 5.5). Le test se termine par la négative. Il n'y a pas de différence significative entre l'efficacité des quatre premières méthodes classées.

Table 5.3: Comparaison avec l'état de l'art

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
CAR91	5.22	5.17	5.22	4.99	4.38	5.03	5.19	6.60	4.34	5.24	5.46	**4.16**	4.95	5.14
CAR92	4.34	4.74	4.67	4.29	**3.88**	4.22	4.31	6.00	5.26	4.47	4.57	5.16	4.09	4.70
EAR83	33.36	40.91	35.74	34.42	33.34	36.06	35.79	**29.30**	34.17	34.41	33.50	35.86	34.97	37.86
HEC92	10.08	12.26	10.74	10.40	10.39	11.71	11.19	**9.20**	10.43	10.39	10.52	11.94	11.11	11.90
KFU93	13.45	15.85	14.47	13.50	**13.23**	16.02	14.51	13.80	14.36	13.77	14.05	14.79	14.09	15.30
LSE91	10.76	12.58	10.76	10.48	10.52	11.15	10.92	**9.60**	11.25	11.06	-	11.15	10.71	12.33
RYE93	8.68	10.11	9.95	8.79	8.92	9.42	-	**6.80**	9.47	8.61	9.11	-	9.20	10.71
STA83	**157.03**	158.12	157.10	157.04	157.06	158.62	157.18	158.20	157.13	157.05	157.29	158.33	157.64	160.12
TRE92	8.14	9.30	8.47	8.16	**7.89**	8.37	8.49	9.40	8.47	8.51	8.71	8.60	8.27	8.32
UTA92	3.47	3.65	3.52	3.43	**3.13**	3.37	3.44	3.50	3.56	3.63	3.71	3.42	3.33	3.88
UTE92	24.88	27.71	25.86	25.09	25.12	27.99	26.70	**24.40**	25.71	24.87	25.18	28.30	26.18	32.67
YOR83	**35.39**	43.98	38.72	35.86	35.49	39.53	39.47	36.20	36.92	37.15	39.08	40.24	37.88	40.53
	0.90	3.36	1.77	1.04	0.78	2.29	1.82	0.75	1.42	1.49	1.52	2.25	1.53	3.29

Par conséquent, selon ce test, toute différence est simplement due au hasard. Fait intéressant, la méthode hybride proposée domine une mise en œuvre minutieuse du recuit simulé [34].

Interprétation des tests (tel que fourni par XLSTAT 2015.4.01.22368) :

— Hypothèse nulle : les échantillons proviennent de la même population ;

— Hypothèse alternative : les échantillons proviennent de populations différentes.

Puisque la valeur p est supérieure au niveau de signification statistique $\alpha = 0,05$, nous ne pouvons pas rejeter l'hypothèse nulle (les liens ont été détectés et les corrections appropriées ont été appliquées).

Table 5.4: Statistiques descriptives

Méthode	Taille	Minimum	Maximum	moyenne	Ecart-type
Caramia et al (7)	12	3.500	158.200	26.083	42.875
Alzaqebah and Abdullah (4)	12	3.130	157.060	26.113	42.712
Notre méthode (0)	12	3.470	157.030	26.233	42.622
Al-Betar et al. (3)	12	3.430	157.040	26.371	42.657

Table 5.5: Test de Friedman

Q (Valeur observée)	1.900
Q (Valeur critique)	7.815
DDL	3
p-value (bilatéral)	0.593
α	0.05

5.5 Conclusion

Une nouvelle méthode méta-heuristique hybride qui combine PPA et LS est présentée. Le concept le plus important introduit dans cette recherche est celui des stolons courts et longs, mis en œuvre par PPA imitant la façon dont le fraisier se propage.

Ce concept parallélise à celui de l'exploitation/exploration, vital pour toute méta-heuristique. Ce chapitre a été mis en œuvre pour résoudre UESP qui est notoirement difficile. Les résultats obtenus sur des exemples benchmark montrent que cette approche, combinée à une LS simple mais mise en œuvre avec une stratégie efficace de recherche des voisinages de l'espace de solution, produit des plannings de qualité. La comparaison avec un grand nombre de procédés connus démontre l'efficacité relative de l'algorithme hybride proposé.

Bien qu'il paraît surprenant que PPA avec LS puisse résoudre un problème difficile comme UESP, il l'est moins lorsque nous apprenons que PPA s'est avéré efficace en optimisation globale continue [94], et en optimisation combinatoire (problème du voyageur de commerce [93] et planification de trajets [104]). Nous n'en avons aucune preuve formelle. Mais le succès des plantes et leur capacité à optimiser leur croissance et à disperser efficacement leur progéniture dans leur environnement témoigne de l'efficacité de ce paradigme.

Pour conclure ce chapitre, une extension future de ce travail peut étudier l'exécution parallèle/distribuée de notre code afin de réduire le temps d'exécution.

Chapitre 6

ILS et VLSN pour un nouveau modèle UESP

6.1 Introduction

Dans ce chapitre, nous considérons un nouveau modèle de planification d'examens où la contrainte molle stipule que les examens doivent être répartis sur les périodes de façon à éviter autant que possible les conflits adjacents. La méthode proposée commence par construire un premier planning réalisable. Une méthode en deux phases est ensuite itérée. La première phase est une heuristique de recherche locale où deux voisinages, Exam-Shifting et Kempe chain, sont explorés à l'aide d'une stratégie token ring. La seconde phase prend comme entrée l'optimum local obtenu lors de la première phase et effectue une recherche VLSN.

Nous prouvons que la recherche dans le voisinage exponentiel est un problème NP-dur. Nous proposons ensuite une heuristique en temps polynomial fondée sur le réarrangement des périodes. Le résultat de VLSN est alors perturbé, et la méthode est itérée. L'algorithme proposé est testé sur les instances benchmark de l'Université de Toronto. Puisque c'est la première fois que le nouveau modèle est proposé, les expériences sont uniquement destinées à évaluer la faisabilité de notre approche.

6.2 Un nouveau modèle de planning d'examens

UESP a seulement deux contraintes, une dure et une molle, et néglige les contraintes de capacité. La contrainte dure exige que les examens avec des étudiants communs doivent être affectés à des périodes différentes. Habituellement, la contrainte molle recommande de disperser les examens sur les périodes données de manière à réduire le nombre d'étudiants qui passent des examens à courte proximité (voir [31] par exemple).

Deux examens sont considérés incompatibles (en conflit) s'ils concernent au moins un étudiant en commun. Deux examens incompatibles sont dits adjacents s'ils appartiennent à des périodes voisines. Dans ce travail, nous proposons une nouvelle forme de contrainte molle : les examens doivent être répartis sur les périodes afin d'éviter autant que possible des examens incompatibles adjacents.

Supposons que n examens, impliquant m étudiants, doivent être programmés. Les examens en conflit sont identifiés par la matrice de conflit $(c_{ij})_{n\times n}$, où c_{ij} est le nombre d'étudiants qui subissent les examens i et j . Supposons que p périodes sont autorisées.

Considérons le graphe $G = (V, E)$ où V est un ensemble de n sommets, associé aux examens, et où E est un ensemble d'arêtes de telle sorte que les sommets i Et j sont connectés si $c_{ij} \neq 0$. La p-coloration de G consiste à assigner p couleurs aux sommets de telle sorte que chaque sommet est colorié et deux sommets adjacents sont de couleurs différentes. Par conséquent, une p-coloration du graphe G offre un planning faisable.

L'espace de solutions, l'ensemble de tous les plannings possibles, est donc l'ensemble de toutes les p-colorations du graphe G, qui, bien que fini, croît de façon exponentielle avec la taille de l'instance. Par conséquent, UESP est de trouver une p-coloration qui satisfait la contrainte molle.

Soit $\pi_1, \cdots, \pi_p$ une organisation arbitraire des périodes. L'écriture $j \in \pi_k$ signifie que l'examen j est affecté à la période π_k (ou de manière équivalente, le

sommet j est coloré avec la couleur π_k). Le nouveau modèle UESP est de trouver une p-coloration du graphe G qui minimise

$$\sum_{k=1}^{p-1} \sum_{i \in \pi_k} \sum_{j \in \pi_{k+1}} c_{ij} \tag{6.1}$$

La fonction objectif (6.1) compte le nombre de conflits sur toutes les périodes adjacentes. Pour être précis, la valeur $\sum_{i \in \pi_k} \sum_{j \in \pi_{k+1}} c_{ij}$ compte le nombre d'étudiants (pas nécessairement distincts) assistant aux deux examens i et j, un dans chacune des deux périodes adjacentes, $i \in \pi_k$ et $j \in \pi_{k+1}$.

Si X est l'ensemble de tous les p-colorations du graphe G, UESP est alors un problème d'optimisation combinatoire de la forme générale : trouver $x^* \in X$ tel que $f(x^*) = \min_{x \in X} f(x)$ où $f(x)$ est la fonction objectif dans (6.1). UESP est NP-dur puisque le problème de décision qui cherche à savoir s'il existe un planning faisable est le problème bien connu de p-coloration qui est NP-complet .

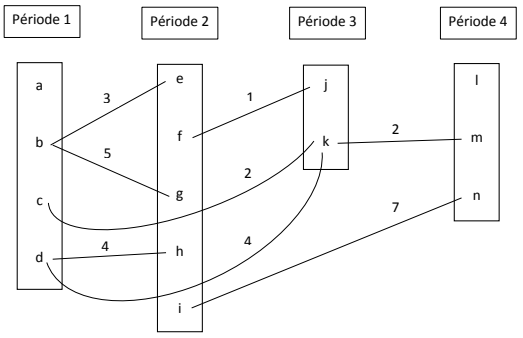

FIGURE 6.1: Exemple de planning

A titre d'illustration, la figure 6.1 propose un exemple de planning avec une valeur 20 de la fonction objectif. Quatorze examens sont attribués à quatre périodes. Les examens b et e, par exemple, sont en conflit. Ils sont également adjacents puisqu'ils appartiennent à des périodes adjacentes. Par conséquent, ils contribuent à la fonction objectif car il y a trois étudiants qui subissent les deux examens. Bien que les examens c et k soient en conflit, ils ne sont pas adjacents et n'ont aucune contribution à la fonction objectif.

6.3 Notre contribution

Un nouveau modèle réaliste de UESP est proposé. Alors que la méthode utilisée pour le placement initial, la sélection des voisinages (Exam-Shifting et Kempe chain) et la recherche locale sont des méthodes standard dans la littérature, l'originalité de ce chapitre réside dans VLSN, *i.e.* dans le voisinage exponentiel proposé et dans la méthode conçue pour l'explorer.

6.4 Méthodologie

UESP consiste à trouver simultanément une p-coloration du graphe induit (*i.e.* une attribution des examens aux périodes) et un réarrangement de ces dernières, afin de minimiser (6.1). Personne ne connaît une telle approche globale. Au contraire, il est commun de décomposer le problème en adoptant une approche typique : d'abord trouver une p-coloration (planning faisable) en utilisant une heuristique de coloration de graphe, ensuite effectuer une recherche locale basée sur deux voisinages (Exam-Shifting, Kempe chain) introduit dans Cheraitia et Haddadi [34] avec la stratégie token ring, puis chercher une permutation des périodes qui minimise (6.1).

Algorithme 6.1 Squelette de notre méthode

```
1   Calculer un planning faisable θ
    Répéter
2       θ' = ExamShifting ⟶ KempeChain(θ)
3       θ'' = PeriodReoder(θ')
4       θ = Perturb(θ'')
    Jusqu'à ce que le critère de terminaison est satisfait
    Retourner le meilleur planning trouvé
```

Nous commençons par esquisser notre méthode en algorithme 6.1, qui se décompose en quatre modules :

i) Construction d'un premier planning faisable en ligne 1 ;

ii) recherche locale en ligne 2 ;

iii) recherche VLSN en ligne 3 ;

iv) perturbation en ligne 4.

A titre d'exemple illustratif, la figure 6.2 montre une modification du planning de la figure 6.1 conduisant à un planning optimal (dont le coût est 0 car aucun étudiant ne subit deux examens consécutifs). La recherche locale résulte au décalage de l'examen c à la période 4, de l'examen f à la période 4 et de l'examen n à la période 1. La recherche VLSN réarrange les périodes comme indiqué sur la figure.

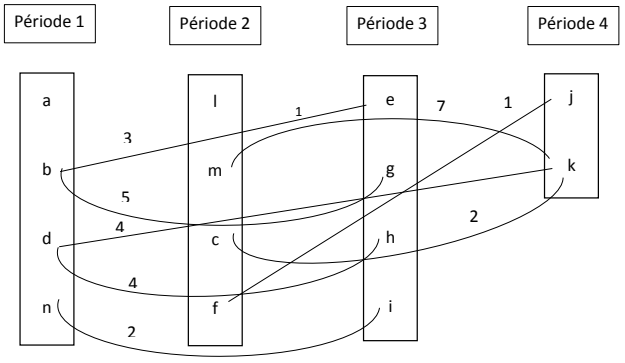

FIGURE 6.2: Exemple de planning optimal

6.4.1 Recherche VLSN

Nous disposons d'un planning θ', obtenu comme optimum local de la phase de la recherche locale. Le but de cette section est de trouver un réarrangement des périodes afin de trouver θ'', au voisinage de θ', qui minimise la fonction objectif 6.1. On observe que la recherche token ring est toujours adoptée (ligne 3 de l'algorithme 6.1), c'est-à-dire que le minimum local obtenu comme résultat de la recherche locale constitue la solution initiale de la recherche VLSN.

6.4.1.1 Voisinage de taille exponentielle

Nous considérons le voisinage $N(\theta')$ du planning θ' comme l'ensemble de tous les plannings qui peut être obtenu à partir de θ' en réarrangeant ou permutant les périodes. De toute évidence, le voisinage a une taille exponentielle, $O(p!)$.

L'exploration de $N(\theta')$ pour un optimum local est le problème d'optimisation combinatoire (appelons-le P) : trouver une permutation des périodes de façon à minimiser 6.1.

Lemma 6.1. *P est NP-dur.*

Démonstration. En fait, nous prouvons que même une restriction de P est un problème NP-dur. Considérons le cas où les n examens sont mutuellement incompatibles et où $c_{ij} \in \{1, 2\}$ pour tous i, j. Par conséquent, n périodes sont nécessaires. Tout planning est faisable dans la mesure où les examens sont en bijection avec les périodes. Dans ce cas particulier, le problème P devient celui de la recherche du chemin hamiltonien de longueur minimum dans le graphe complet dont les sommets sont les examens (ou les périodes) et où les longueurs des arêtes sont 1 ou 2. Ce problème est clairement NP-dur. □

Dans le cas général, définissons $d_{ij} = \sum_{r \in i} \sum_{s \in j} c_{rs}$ pour $i, j = 1, \cdots, p, i \neq j$ et $d_{ii} = 0, i = 1, \cdots, p$. Ici, l'écriture $r \in i$ signifie que l'examen r appartient à la période i. La $p \times p$-matrice $D = (d_{ij})$ est symétrique.

Si l'on considère le graphe complet dont les sommets sont les périodes, d_{ij} définit en quelque sorte une distance entre les périodes i et j. On peut alors rechercher directement un optimum local dans $N(\theta')$ en résolvant le problème du chemin hamiltonien de longueur minimum (MLHP) sur le graphe déjà défini avec la matrice de distance D. Même si ce n'est pas un moyen efficace d'explorer un voisinage, cette approche est satisfaisante puisque l'ordre du graphe, qui est le nombre de périodes p, est faible dans la pratique. Cependant, dans la section suivante, nous proposons une autre façon efficace d'aborder le problème même si la valeur de p est grande.

6.4.1.2 Une heuristique en temps polynomial pour la recherche VLSN

Considérons la $m \times p$-matrice binaire A dont les lignes et les colonnes sont indexées respectivement par les étudiants et les périodes, avec

$$a_{ij} = \begin{cases} 1 & \text{si l'étudiant } i \text{ subit un certain examen dans la période } j \\ 0 & \text{sinon} \end{cases}$$

Lemma 6.2. *$D = A^T A$ excepté les éléments diagonaux.*

Démonstration. Pour $i, j \in \{1, \cdots, p\}, i \neq j$ arbitraire, $a_i^T \cdot a_j$ compte le nombre d'étudiants impliqués dans les deux périodes i et j, où a_i et a_j sont des colonnes dans A. Chacun de ces étudiants est impliqué dans un examen (appelez-le r) dans la période i et un examen s dans la période j. Le nombre de ces étudiants est c_{rs}. Le total de tous les étudiants impliqués à la fois dans les examens prévus dans la période i et les examens prévus dans la période j donne $a_i^T \cdot a_j = \sum_{r \in i} \sum_{s \in j} c_{rs} = d_{ij}$. □

Étant donné une matrice binaire A, un bloc dans A est toute séquence maximale de 1 située sur la même ligne. Supposons que A ait p colonnes, et soit π une permutation de $1, \cdots, p$. On notera A_π la matrice obtenue à partir de la permutation des colonnes de A induites par π. Le problème de maximisation de blocs (CBM) cherche une permutation π maximisant le nombre de blocs dans A_π (des résultats positifs et négatifs sur la version de minimisation du problème peuvent être trouvés dans [47, 49]).

Lemma 6.3. *CBM est NP-dur.*

La preuve est analogue à ce qui a été fait en [49] pour la version de minimisation.

Lemma 6.4. *Le nombre de blocs dans A_π est*

$$f - \sum_{j=1}^{p-1} \sum_{i=1}^{m} a_{i\pi_j} \times a_{i\pi_{j+1}} \tag{6.2}$$

où f est le nombre de 1 dans A.

Démonstration. Pour des raisons de commodité, considérons A^0 la matrice obtenue à partir de A en concaténant à A une colonne artificielle nulle étiquetée $p+1$. De toute évidence, A^0 et A ont le même nombre de blocs. La position de la colonne artificielle est fixe pour toujours et ne change pas par permutation. Nous prouvons que le nombre de blocs dans A_π, qui est le nombre de blocs dans A^0_π, est

$$f - \sum_{j=1}^{p} \sum_{i=1}^{m} a^0_{i\pi_j} \times a^0_{i\pi_{j+1}} \tag{6.3}$$

Observons que le nombre de blocs dans A^0_π peut alternativement être défini comme le nombre de coefficients de A^0_π tels que $a^0_{i\pi_j} = 1$ et $a^0_{i\pi_{j+1}} = 0$ (ne pas omettre la dernière colonne nulle). Par conséquent, nous pouvons commencer notre comptage avec f et retrancher 1 dès qu'on rencontre une occurence telle que $a^0_{i\pi_j} = 1$ et $a^0_{i\pi_{j+1}} = 1$. C'est exactement ce qui es fait en (6.3). □

Lemma 6.5. *MLHP avec l'instance D est équivalent à CBM avec l'instance A.*

Démonstration. En vertu du lemme 6.2, le nombre de blocs dans A

$$f - \sum_{j=1}^{p-1} \sum_{i=1}^{m} a_{i\pi_j} \times a_{i\pi_{j+1}}$$

est aussi

$$f - \sum_{j=1}^{p-1} d_{\pi_j \pi_{j+1}}$$

Comme f est invariant par permutation, une permutation π maximise le nombre de blocs dans A si et seulement si elle définit un chemin Hamiltonien de longueur minimum dans le graphe complet avec des longueurs d'arête d_{ij}. □

Ainsi, en vertu du lemme 6.5, la recherche d'une p-coloration minimisant (6.1) est équivalente à la recherche d'une p-coloration de telle sorte que la matrice binaire A ait un nombre maximum de blocs.

Nous proposons maintenant une heuristique d'amélioration en temps polynomial pour rechercher une localement optimale dans le voisinage de taille exponentielle, sur la base de ce qui a été fait en [48] pour la version de minimisation du problème. Cela signifie que le voisinage de taille exponentielle est recherché lui-même localement via un voisinage de petite taille.

Nous commençons par définir le voisinage, appelé PeriodMove. Étant donné le planning θ' obtenu en tant que résultat de la recherche locale, nous définissons son voisinage comme l'ensemble de tous les plannings que nous pouvons obtenir à partir de θ' en déplaçant une certaine période à un autre endroit. La taille de PeriodMove est $O(p^2)$ car il y a p choix pour une période, et $p-1$ positions restées pour la déplacer.

Comment pouvons-nous obtenir une solution améliorée dans ce voisinage ? La réponse est basée sur l'observation suivante. Étant donné une $m \times p$-matrice binaire A, considérons trois colonnes adjacentes indexées $j-1, j, j+1$.

Lemma 6.6 ([48]). *La colonne j est responsable de*

$$\delta(j-1,j,j+1) = \sum_{i=1}^{m} (a_{ij} - a_{i,j-1} \times a_{ij} - a_{ij} \times a_{i,j+1} + a_{i,j-1} \times a_{i,j+1})$$

blocs.

En fait $\delta(j-1,j,j+1)$ compte le nombre d'occurrences de 101 et 010 dans les trois colonnes adjacentes. Donc, si nous supprimons la colonne j qui est entre les colonnes $j-1$ et $j+1$, nous éliminons $\delta(j-1,j,j+1)$ blocs de A. De la même manière, si nous insérons la colonne j entre les colonnes $k, k+1$, nous créons

$$\delta(k,j,k+1) = \sum_{i=1}^{m} (a_{ij} - a_{ik} \times a_{ij} - a_{ij} \times a_{i,k+1} + a_{ik} \times a_{i,k+1})$$

nouveaux blocs.

L'heuristique que nous proposons est basée sur le choix de la meilleure amélioration possible dans le voisinage PeriodMove (voir le pseudo-code en Algorithme 6.2).

Algorithme 6.2 Procédure $PeriodReorder(\theta')$.

```
1  Calculer la matrice binaire A induite par le planning θ'
2  Répéter
3      Trouver les colonnes j, k, j ≠ k, telle que δ(k, j, k+1) − δ(j−1, j, j+1) soit maximum
4      Retirer la colonne j de sa position et l'insérer entre les colonnes k et k+1
5  Jusqu'à ce qu'aucune amélioration ne soit plus possible
6  Soit θ'' le planning induit par le réarrangement dans la boucle
7  Retourner θ''
```

Le calcul de la matrice binaire A coûte $O(m \times p)$. Les instructions à l'intérieur de la boucle coûtent $O(m \times p^2)$ car il y a $O(p^2)$ façons de choisir les deux colonnes, et il prend $O(m)$ pour calculer $\delta(j-1, j, j+1)$ ou $\delta(k, j, k+1)$. Comme le nombre d'itérations d'amélioration est faible dans la pratique, on peut considérer la complexité de cette heuristique comme étant $O(m \times p^2)$.

A titre illustratif, considérons la matrice binaire A

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix} \qquad A_\pi = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

qui contient 6 blocs. Si nous choisissons $j = 4$ et $k = 1$, nous avons $\delta(3, 4, 5) = 0$ et $\delta(1, 4, 2) = 2$. Donc, on obtient une amélioration si on insère la colonne 4 entre les colonnes 1 et 2, ce qui donne 8 blocs dans la matrice A_π où $\pi = (14235)$.

6.4.2 Perturbation

Le planning θ'' obtenu en tant que résultat de la recherche VLSN, est perturbée pour l'obtention du planning θ avec lequel une nouvelle itération commence. La perturbation adoptée est assez simple : un examen choisi au hasard est décalé vers une période choisie au hasard, à condition qu'il ne génère aucun conflit. Ce processus est répété (voir la figure 6.3).

Algorithme 6.3 Procédure $Perturb(\theta'')$.

```
1  Considérer le planning θ''
2  Répéter un certain de fois // 10 dans notre implémentation
3      Choisir au hasard un examen i // soit k la période à laquelle i est affecté
4      Choisir au hasard une période j ≠ k
5      Réaffecter l'examen i à la période j à condition que cela ne crée pas de conflit
5  // Soit θ le planning induit
7  Retourner θ
```

6.5 Expérimentation numérique

La méthode proposée est codée en C et exécutée sur un Intel Core I3, 2,4 GHz. Le code est testé sur l'ensemble de données benchmark de l'Université de Toronto [31, 85]. Les caractéristiques des instances sont présentées au tableau 2.1.

Le fait que la méthode proposée n'a aucun paramètre est un avantage. Cependant, elle souffre de l'aléas de la phase initiale (heuristique de coloration du graphe) et de la phase de perturbation. Par conséquent, comme la plupart des méta-heuristiques, notre méthode est non reproductible, et chaque exécution peut donner lieu à un résultat différent. Nous avons choisi d'exécuter chaque instance cinq fois pour retenir le coût et le temps moyens.

Les résultats de la phase initiale sont donnés au tableau 6.1. Les en-têtes des colonnes sont explicites. Nous observons que le coût moyen est éloigné de celui du meilleur planning (tableau 6.2) du fait que l'heuristique de coloration ne se soucie pas de la contrainte molle.

Une limite de temps en fonction de la taille de l'instance est imposée dans le code. Elle est fixée à $n \times p$ en secondes. Par exemple, le temps de calcul sur l'instance CAR91 est $682 \times 35 = 23,870$ secondes.

Le rôle de la recherche VLSN peut être apprécié dans le Tableau 6.2. La colonne étiquetée "Itérations" Compte le nombre d'exécutions de la boucle répéter dans le pseudo-code 6.1. De ce point de vue, certaines des instances semblent plus difficiles que d'autres. Par exemple, bien que le temps accordé à l'exécution du code sur l'instance CAR91 soit important, la méthode proposée n'est itérée que 28

TABLE 6.1: Solution initiale trouvée en utilisant Saturation Degree with Backtracking (valeurs moyennes sur cinq essais)

	Solution initiale	
Instance	Coût moyen	Temps moyen
CAR91	5661.4	0.23
CAR92	5718.8	0.17
EAR83	2422.6	0.04
HEC92	2371.0	0.04
KFU93	7159.0	0.08
LSE91	2624.8	0.06
RYE93	11105.4	0.10
STA83	3883.8	0.04
TRE92	2061.2	0.05
UTA92	5033.2	0.20
UTE92	4943.6	0.04
YOR83	1763.4	0.06

fois. À partir de la table 6.2, nous pouvons voir que l'incorporation de la recherche VLSN dans la méthode entraîne une amélioration réelle de la fonction de coût d'environ 6% en moyenne.

TABLE 6.2: Résultats avec et sans recherche VLSN

	La recherche locale seulement			La recherche locale et VLSN		
Instance	Meilleur coût	Coût moyen	Itérations	Meilleur coût	Coût moyen	Itérations
CAR91	2318	2385.2	27	2034	2141.2	28
CAR92	1932	2063.4	46	1831	1852.8	47
EAR83	772	812.2	410	671	716.8	460
HEC92	436	474.4	6589	432	442.2	6740
KFU93	1072	1282.2	71	1079	1184.2	76
LSE91	589	667.0	126	637	646.8	132
RYE93	1262	1423.4	59	1159	1369.4	60
STA83	3078	3078.0	3442	3021	3021.0	4081
TRE92	849	907.0	295	750	776.2	309
UTA92	1822	1872.2	32	1751	1794.8	30
UTE92	549	688.8	968	554	695.2	1057
YOR83	756	771.8	878	626	704.6	946
Average	1286.25	1368.8		1212.08	1278.77	

6.6 Conclusion

Un nouveau modèle pour le problème de planification d'examen sans capacité est proposé. Étant NP-dur, une méthode heuristique itérant la recherche locale et la recherche VLSN est proposée. Puisque l'exploration du voisinage exponentiel

est NP-dur, une heuristique en temps polynomial basée sur le réarrangement des périodes est proposée. Les expériences numériques effectuées sur des instances benchmark montrent que l'intégration de la recherche VLSN dans notre méthode conduit à de meilleurs résultats. Malheureusement, c'est la première fois que le modèle est proposé. Par conséquent, il n'y a aucune comparaison possible.

Conclusion générale

Nous avons étudié le problème de planification d'examens sans contraintes de capacité qui est un problème d'optimisation combinatoire intraitable. Les trois premiers chapitres constituent des généralités et présentent respectivement le modèle du problème ainsi qu'un état de l'art, puis une introduction à l'optimisation combinatoire et à la théorie de la complexité, enfin l'arsenal des méthodes utilisées.

Notre contribution réside dans les trois chapitres suivants. Aux chapitres 4 et 5, on a proposé deux nouvelles méthodes fondées sur SA et PPA combinés, chacun, avec LS. Chacun de ces deux chapitres a fait l'objet d'une publication [34, 35]. Dans le dernier chapitre 6, on a introduit un nouveau modèle pour UESP, puis une méthode de résolution basée sur la recherche VLSN.

Chacune des trois méthodes a fait l'objet d'une expérimentation numérique rigoureuse, avec des comparaisons avec des méthodes existantes à l'issue desquelles nos méthodes ont montré leur efficacité et leur compétitivité.

Cette thèse permet d'ouvrir la voie à quelques perspectives :

1. Une extension future de ce travail peut s'intéresser à l'exécution parallèle/distribuée de nos codes afin de réduire le temps d'exécution.
2. Il serait intéressant de tester les données proposées dans "Track on of the 2nd International Timetabling Competition" [64]. Ces données représentent des instances du monde réel.
3. On essaiera de montrer que nos approches sont génériques, et peuvent être adaptées au problème général de l'optimisation combinatoire.

Bibliographie

[1] S Abdul-Rahman, EK Burke, A Bargiela, B McCollum, and E Ozcan. A constructive approach to examination timetabling based on adaptive decomposition and ordering. *Annals of Operations Research*, 218 :3–21, 2014.

[2] S Abdullah, H Turabieh, and B McCollum. A hybridization of electromagnetic-like machanism and great deluge for examination timetabling problems. In *Hybrid Metaheuristics*, volume 5818, pages 60–72, 2009.

[3] S Abdullah, H Turabieh, and B McCollum. A tabu-based memetic approach to the examination timetabling problem. In *Rough Set and Knowledge Technology*, volume 6401, pages 574–581, 2010.

[4] MA Ahandani, MTV Baghmisheh, MAB Zadeh, and S Ghaemi. Hybrid particle swarm optimization transplanted into a hyper-heuristic structure for solving examination timetabling problem. *Swarm and Evolutionary Computation*, 7 :21–34, 2012.

[5] S Akyol and B Alatas. Plant intelligence based metaheuristic optimization algorithms. *Artificial Intelligence Review*, 2016.

[6] MA Al-Betar, AT Khader, and IA Doush. Memetic techniques for examination timetabling. *Annals of Operations Research*, 218 :23–50, 2014.

[7] K Alutaibi, A Alsubaie, and J Marti. Allocation and scheduling of firefighting units in large petrochemical complexes. In *International Conference on Critical Infrastructure Protection*, pages 263–279. Springer, 2015.

[8] M Alzaqebah and S Abdullah. Hybrid artificial bee colony search algorithm based on disruptive selection for examination timetabling problems. In *Combinatorial Optimization and Applications*, volume 6831, pages 31–45, 2011.

[9] M Alzaqebah and S Abdullah. An adaptive artificial bee colony and late-acceptance hill-climbing algorithm for examination timetabling. *Journal of Scheduling*, 17 :249–262, 2014.

[10] M Alzaqebah and S Abdullah. Hybrid bee colony optimization for examination timetabling problems. *Computers and Operations Research*, 54 :142–154, 2015.

[11] MH Ammar, M Benaissa, and H Chabchoub. Seafaring staff scheduling. *International Journal of Services and Operations Management*, 19(2) :229–249, 2014.

[12] T Arbaoui. *Modeling and solving university timetabling.* PhD thesis, Université de Technologie de Compiègne, 2014.

[13] T Arbaoui. *Modeling and solving university timetabling.* PhD thesis, Université de Technologie de Compiègne, 2014.

[14] H Asmuni, EK Burke, JM Garibaldi, B McCollum, and AJ Parkes. An investigation of fuzzy multiple heuristic orderings in the construction of university examination timetables. *Computers and Operations Research*, 36 :981–1001, 2009.

[15] E Aycan and T Ayav. Solving the course scheduling problem using simulated annealing. In *Proceedings of the IEEE International Advance Computing Conference*, page 462–466, 2009.

[16] AL Bolaji, AT Khader, MA Al-Betar, and MA Awadallah. A hybrid nature-inspired artificial bee colony algorithm for uncapacitated examination timetabling problems. *Journal of Intelligent Systems*, 24 :37–54, 2015.

[17] A Brabazon, M O'Neill, and S McGarraghy. Plant-inspired algorithms. In *Natural Computing Algorithms*, pages 455–477. Springer Nature, 2015.

[18] EK Burke and JP Newall. Solving examination timetabling problems through adaptation of heuristic orderings. *Annals of Operations Research*, 129 :107–134, 2004.

[19] EK Burke and S Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research*, 140 :266–280, 2002.

[20] EK Burke, DG Elliman, PH Ford, and RF Weare. Examination timetabling in british universities. In *Practice and Theory of Automated Timetabling*, page 76–92. Springer, 1996.

[21] EK Burke, Y Bykov, JP Newall, and S Petrovic. A time-predefined local search approach to exam timetabling problems. *IIE Transactions*, 36 :509–528, 2004.

[22] EK Burke, B McColum, A Meisels, S Petrovic, and R Qu. A graph-based hyper heuristic for timetabling problems. *European Journal of Operational Research*, 176 :177–192, 2007.

[23] EK Burke, AJ Eckersley, B McCollum, S Petrovic, and R Qu. Hybrid variable neighbourhood approaches to university exam timetabling. *European Journal of Operational Research*, 206 :46–53, 2010.

[24] EK Burke, R Qu, and A Soghier. Adaptive selection of heuristics for improving constructed exam timetables. In *Proceedings of the 8th international conference on the practice and theory of automated timetabling PATAT'10*, page 136–151, 2010.

[25] EK Burke, R Qu, and A Soghier. An adaptive tie breaking and hybridisation hyper-heuristic for exam timetabling problems. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2011)*, volume 387, pages 205–223, 2011.

[26] EK Burke, N Pham, R Qu, and J Yellen. Linear combinations of heuristics for examination timetabling. *Annals of Operations Research*, 194 :89–109, 2012.

[27] EK Burke, R Qu, and A Soghier. Adaptive selection of heuristics for improving exam timetables. *Annals of Operations Research*, 218 :129–145, 2014.

[28] Y Bykov and S Petrovic. A step counting hill climbing algorithm applied to university examination timetabling. *Journal of Scheduling*, 19(4) :479–492, 2016.

[29] M Caramia, P Dell'Olmo, and GF Italiano. New algorithms for examination timetabling. *Algorithm Engineering*, 1982 :230–241, 2001.

[30] M Caramia, P Dell'Olmo, and G F Italiano. Novel local-search-based approaches to university examination timetabling. *INFORMS Journal on Computing*, 20(1) :86–99, 2008.

[31] MW Carter, G Laporte, and S Lee. Examination timetabling : Algorithmic strategies and applications. *Journal of the Operational Research Society*, 47 :373–383, 1996.

[32] D Cattaruzza, N Absi, D Feillet, and D Vigo. An iterated local search for the multi-commodity multi-trip vehicle routing problem with time windows. *Computers and Operations Research*, 51 :257–267, 2014.

[33] AA Ceder. Optimal multi-vehicle type transit timetabling and vehicle scheduling. *Procedia-Social and Behavioral Sciences*, 20 :19–30, 2011.

[34] M Cheraitia and S Haddadi. Simulated annealing for the uncapacitated exam scheduling problem. *International Journal of Metaheuristics*, 5(2) :156–170, 2016.

[35] M Cheraitia, S Haddadi, and A Salhi. Hybridizing plant propagation and local search for uncapacitated exam scheduling problems. *International Journal of Services and Operations Management*, in press, 2017.

[36] TB Cooper and JH Kingston. The complexity of timetable construction problems. In *Practice and Theory of Automated Timetabling*, pages 281–295. Springer, 1996.

[37] L Di Gaspero and A Schaerf. Neighborhood portfolio approach for local search applied to timetabling problems. *Journal of Mathematical Modelling and Algorithms*, 5(1) :65–89, 2005.

[38] L Di Gaspero and A Schaerf. Neighborhood portfolio approach for local search applied to timetabling problems. *Journal of Mathematical Modeling and Algorithms*, 5(1) :65–89, 2005.

[39] O Dominguez, D Guimarans, AA Juan, and I De la Nuez. A biased-randomised large neighbourhood search for the two-dimensional vehicle routing problem with backhauls. *European Journal of Operational Research*, 255(2) :442–462, 2016.

[40] KA Dowsland and JM Thompson. Simulated annealing. In *Handbook of Natural Computing*, page 1623–1655, 2012.

[41] YJ Dun, Q Wang, and YB Shao. A simulated annealing genetic algorithm for solving timetable problems. In *Fuzzy Information and Engineering and Operations Research and Management*, volume 211, page 365–374, 2014.

[42] M Eley. Some experiments with ant colony algorithms for the exam timetabling problem. In *Ant Colony Optimization and Swarm Intelligence*, volume 4150, pages 492–499, 2011.

[43] R Etemadi and NM charka. A novel combinational algorithm for solving the examination timetabling problem. In *International Conference on Industrial and Intelligent Information (ICIII 2012)*, Singapore 17-18 Mars, 2012.

[44] V François, Y Arda, Y Crama, and G Laporte. Large neighborhood search for multi-trip vehicle routing. *European Journal of Operational Research*, 255(2) :422–441, 2016.

[45] C Gogos, P Alefragis, and E E Housos. An improved multi-staged algorithmic process for the solution of the examination timetabling problem. *Annals of Operations Research*, 194 :203–221, 2012.

[46] P Grangier, M Gendreau, F Lehuédé, and LM Rousseau. An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. *European Journal of Operational Research*, 254(1) :80–91, 2016.

[47] S Haddadi and Z Layouni. Consecutive block minimization is 1.5-approximable. *Information Processing Letters*, 108(3) :132–135, 2008.

[48] S Haddadi, S Chenche, M Cheraitia, and F Guessoum. Polynomial-time local-improvement algorithm for consecutive block minimization. *Information Processing Letters*, 115(6-8) :612–617, 2015.

[49] S Haddadi. A note on the NP-hardness of the consecutive block minimization problem. *International Transactions in Operational Research*, 9(6) :775–777, 2002.

[50] JK Hao, P Galinier, and M Habib. Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Revue d'intelligence artificielle*, 13(2) :283–324, 1999.

[51] KL Hoffman, M Padberg, and G Rinaldi. Traveling salesman problem. In *Encyclopedia of operations research and management science*, pages 1573–1578. Springer, 2013.

[52] HH Hoos and T Stützle. *Stochastic Local Search : Foundations and Applications*. Elsevier, San Francisco, 2004.

[53] P Jittamai and T Kangwansura. A hospital admission planning model for operating room allocation under uncertain demand requirements. *International Journal of Services and Operations Management*, 23(2) :235–256, 2016.

[54] M Kaviani, H Shirouyehzad, SM Sajadi, and M Salehi. A heuristic algorithm for the university course timetabling problems by considering measure index : a case study. *International Journal of Services and Operations Management*, 18(1) :1–20, 2014.

[55] H Kellerer, U Pferschy, and D Pisinger. *Knapsack problems*. Springer, Berlin, 2004.

[56] S Kirkpatrick, CD Gellat, and MP Vecchi. Optimisation by simulated annealing. *Science*, 220 :671–680, 1983.

[57] V Kolonias, G Goulas, C Gogos, P Alefragis, and E Housos. Solving the examination timetabling problem in gpus. *Algorithms*, 7 :295–327, 2014.

[58] SY Lee, YD Kim, and HJ Leeo. Heuristic algorithm for a military training timetabling problem. *Asia Pacific Management Review*, 14(3) :289–299, 2009.

[59] H R Lourenço, O C Martin, and T Stützle. Iterated local search : Framework and applications. In *Handbook of Metaheuristics*, pages 363–397. Springer Nature, 2010.

[60] Z Lü, JK Hao, and F Glover. Neighborhood analysis : a case study on curriculum-based course timetabling. *Journal of Heuristics*, 17(2) :97–118, mar 2011.

[61] E Malaguti and P Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, 17 :1–34, 2010.

[62] M Malkawi, MA Hassan, and OA Hassan. A new exam scheduling algorithm using graph coloring. *The International Arab Journal of Information Technology*, 5 :80–86, 2008.

[63] R Masson, T Vidal, J Michallet, PHV Penna, V Petrucci, A Subramanian, and H Dubedout. An iterated local search heuristic for multi-capacity bin packing and machine reassignment problems. *Expert Systems with Applications*, 40(13) :5266–5275, 2013.

[64] B McCollum, A Schaerf, B Paechter, P McMullan, R Lewis, AJ Parkes, L Di Gaspero, R Qu, and EK Burke. Setting the research agenda in automated timetabling : The second international timetabling competition. *INFORMS Journal on Computing*, 22(1) :120–130, 2010.

[65] B McCollum. A perspective on bridging the gap between theory and practice in university timetabling. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 3–23. Springer, 2006.

[66] BGC McCollum. A perspective on bridging the gap between theory and practice in university timetabling. In *Practice and Theory of Automated Timetabling*, volume 3867, pages 3–23, 2007.

[67] L Merlot, N Boland, B Hughes, and P Stuckey. A hybrid algorithm for the examination timetabling problem. In *Practice and Theory of Automated Timetabling IV*, volume 2740, pages 207–231, 2003.

[68] SA MirHassani. Improving paper spread in examination timetables using integer programming. *Applied Mathematics and Computation*, 179 :702–706, 2006.

[69] AJ Monticelli, R Romero, and EN Asada. Fundamentals of simulated annealing. In *Modern Heuristic Optimization Techniques : Theory and Applications to Power Systems*, page 123–146, 2008.

[70] MM Nasiri and M Rahvar. A two-step multi-objective mathematical model for nurse scheduling problem considering nurse preferences and consecutive shifts.

[71] S Nesmachnow. An overview of metaheuristics : accurate and efficient methods for optimization. *International Journal of Metaheuristics*, 3 :320–347, 2014.

[72] F Neumann and C Witt. Combinatorial optimization and computational complexity. *Bioinspired Computation in Combinatorial Optimization*, pages 9–19, 2010.

[73] TC Pais and P Amaral. Managing the tabu list length using a fuzzy inference system : an application to examination timetabling. *Annals of Operations Research*, 194 :341–363, 2012.

[74] C Papadimitriou and K Steiglitz. Combinatorial optimization : Algorithms and complexity, 1982.

[75] PM Pardalos and MGC Resende. *Handbook of Applied Optimization*. Oxford University Press, 2002.

[76] S Petrovic and EK Burke. University timetabling. In *Handbook of Scheduling : Algorithms, Models, and Performance Analysis*. CRC Press, 2004.

[77] N Pillay and W Banzhaf. A genetic programming approach to the generation of hyper-heuristics for the uncapacitated examination timetabling problem. In *Progress in Artificial Intelligence*, volume 4874, pages 223–234, 2007.

[78] N Pillay and W Banzhaf. A study of heuristic combinations for hyper-heuristic systems for the uncapacitated examination timetabling problem. *European Journal of Operational Research*, 197 :482–491, 2009.

[79] N Pillay. A review of hyperheuristics for educational timetabling. *Annals of Operations Research*, 239 :3–38, 2014.

[80] D Pisinger and S Ropke. Large neighborhood search. In *Handbook of Metaheuristics*, pages 399–419. Springer Nature, 2010.

[81] J Puchinger and GR Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization : A survey and classification. In *Artificial Intelligence and Knowledge Engineering Applications : A Bioinspired Approach*, volume 3562, pages 41–53, 2005.

[82] R Qu and EK Burke. Adaptive decomposition and construction for examination timetabling problems. In *Multidisciplinary international scheduling : theory and applications (MISTA'07)*, pages 418–425, 2007.

[83] R Qu and EK Burke. Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems. *Journal of the Operational Research Society*, 60 :1273–1285, 2009.

[84] R Qu, EK Burke, and B McCollum. Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems. *European Journal of Operational Research*, 198 :392–404, 2009.

[85] R Qu, EK Burke, B McCollum, LGT Merlot, and SY Lee. A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*, 12 :55–89, 2009.

[86] R Qu, N Pham, R Bai, and G Kendall. Hybridising heuristics within an estimation distribution algorithm for examination timetabling. *Applied Intelligence*, 42 :679–693, 2015.

[87] S Abdul Rahman, A Bargiela, EK Burke, B McCollum, and E Özcan. Construction of examination timetables based on ordering heuristics. In *Proceedings of the 24th International Symposium on Computer and Information Sciences*, pages 727–732, 2009.

[88] S Abdul Rahman. *Search methodologies for examination timetabling*. PhD thesis, University of Nottingham, 2012.

[89] NR Sabar, M Ayob, R Qu, and G Kendall. A graph coloring constructive hyper-heuristic for examination timetabling problems. *Applied Intelligence*, 37 :1–11, 2012.

[90] A Salhi and ES Fraga. Nature-inspired optimisation approaches and the new plant propagation algorithm. In *Proc. of the International Conference on Numerical Analysis and Optimization*, pages K2.1–K2.8. 2011.

[91] A Schaerf and L Di Gaspero. Measurability and reproducibility in university timetabling research : discussion and proposals. In *Practice and Theory of Automated Timetabling*, volume 3867, pages 40–49, 2007.

[92] A Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13 :87–127, 1999.

[93] BI Selamoglu and A Salhi. The plant propagation algorithm for discrete optimisation : The case of the travelling salesman problem. In *Nature-Inspired Computation in Engineering*, pages 43–61. Springer Nature, 2016.

[94] M Sulaiman, A Salhi, BI Selamoglu, and OB Kirikchi. A plant propagation algorithm for constrained engineering optimisation problems. *Mathematical Problems in Engineering*, 2014 :10 pages, 2014.

[95] B Suman and P Kumar. A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society*, 57 :1143–1160, 2006.

[96] JM Thompson and KA Dowsland. General cooling schedules for a simulated annealing based timetabling system. In *Practice and Theory of Automated Timetabling*, volume 1153, page 345–363, 1995.

[97] JM Thompson and KA Dowsland. Variants of simulated annealing for the examination timetabling problem. *Annals of Operations Research*, 63 :105–128, 1996.

[98] JM Thompson and KA Dowsland. A robust simulated annealing based examination timetabling system. *Computers and Operations Research*, 25 :637–648, 1998.

[99] H Turabieh and S Abdullah. A hybrid fish swarm optimisation algorithm for solving examination timetabling problems. In *International Conference on Learning and Intelligent Optimization (LION 2011)*, Nizhny Novgorod, Russia 19-21 Juin, 2011.

[100] A Turing. On computable numbers with application to the entscheidungsproblem. In *Proc. London Math. Soc. Ser.*, pages 230–265 and 544–546, 1936.

[101] P Vansteenwegen and M Mateo. An iterated local search algorithm for the single-vehicle cyclic inventory routing problem. *European Journal of Operational Research*, 237(3) :802–813, 2014.

[102] I Wegener. *Complexity theory : exploring the limits of efficient algorithms.* Springer Science & Business Media, 2005.

[103] G Woumans, L De Boeck, J Beliën, and S Creemers. A column generation approach for solving the examination-timetabling system. *European Journal of Operational Research*, 253 :178–194, 2016.

[104] Y Zhou, Y Wang, X Chen, L Zhang, and K Wu. A novel path planning algorithm based on plant growth mechanism. *Soft Computing*, 21(2) :435–445, 2016.