Stage 2de année Master informatique

Confection d'emplois du temps pour cursus universitaires

Programmation par contraintes et méthodes approchées

Étudiant :
Cyril Grelier

Encadrants :
David Lesaint
Vincent Barichard
David Genest

24 août 2020

ENGAGEMENT DE NON PLAGIAT

Je, soussigné (e) ……, déclare être pleinement conscient(e) que le plagiat de documents ou d'une partie d'un document publiés sur toutes formes de support, y compris l'internet, constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée. En conséquence, je m'engage à citer toutes les sources que j'ai utilisées pour écrire ce rapport ou mémoire.

Signature :

Cet engagement de non plagiat doit être inséré en première page de tous les rapports, dossiers, mémoires.

Remerciements

Je voudrais remercier **David Lesaint**, directeur du LERIA, **Vincent Barichard**, responsable du Master 2 ACDI et **David Genest**, directeur du portail licences MPCIE pour leur aide tout au long du stage. Leurs connaissances sur le fonctionnement de l'université et sur la réalisation de plannings m'ont grandement aidé à mieux comprendre la difficulté du problème de génération d'emplois du temps. Comprendre tous les problèmes pouvant intervenir du point de vue de la conception comme du point de vue de la réparation ainsi que les différents scénarios et paramètres à prendre en compte m'ont permis de prendre plus de recul sur l'ampleur de la tâche et des enjeux de mettre en place un tel système. Je les remercie pour les conseils, leur expertise et le temps mis en oeuvre pour la réalisation des modèles de programmation par contraintes mais aussi sur les différentes recherches locales, ainsi que leur patience lors des réunions pour m'expliquer et débattre des différents points de ce stage. Merci pour leur aide et conseils pour la rédaction de ce rapport.

Je remercie également **Maryse Esnou** et **Yann Depauw** de la DDN pour les connaissances apportées sur l'organisation de l'université et le retour sur le travail effectué.

Pour finir, je voudrais remercier **Elsa Gobin** pour sa relecture du rapport.

Table des matières

Introduction

Dans le cadre du Master Intelligence Décisionnelle de l'Université d'Angers, nous sommes amenés à réaliser un stage de recherche pour compléter notre formation avec une expérience en laboratoire de recherche.

Mon stage c'est déroulé au LERIA, le Laboratoire d'Étude et de Recherche en Informatique d'Angers. Celui-ci portait sur la génération d'emplois du temps pour cursus universitaires dans le cadre du projet Thélème.

Le projet Thélème vise à modifier le cursus de Licence. L'aspect du projet Thélème qui nous concerne est l'abandon du rythme semestriel et des parcours imposés pour s'orienter vers un rythme de blocs de 6 semaines successives où les étudiants assisteront à des cours qu'ils auront sélectionnés en fonction de leurs envies.

Dans la littérature du domaine de la confection d'emplois du temps, il existe deux approches principales. La première consiste au fait de proposer à des élèves de s'inscrire à des cours puis de créer l'emploi du temps en fonction de ces inscriptions est nommé "Post Enrolment based Course Timetabling" que l'on pourrait traduire par "confection d'emplois du temps après inscription". Cette version s'oppose à la seconde, le "Curriculum based course timetabling" qui consiste à proposer aux étudiants des cursus prédéfinis après avoir créé l'emploi du temps.

Autour de ce problème de confection d'emplois du temps gravite une multitude de critères à prendre en compte : le nombre d'heures pour chaque cours et le découpage de ceux-ci, les disponibilités des enseignants donnant cours, le nombre d'étudiants inscrits et leur répartition dans les différents groupes. Ce dernier problème se nomme le "student sectionning", trouver comment diviser des étudiants en groupes d'élèves. Il faudra favoriser des groupes stables et de la taille la plus grande possible. Favoriser ces groupes stables a deux objectifs : encourager au partage entre étudiants pour éviter des étudiants isolés et plus enclins à abandonner mais aussi à faciliter la création d'emplois du temps pour ce qui est du placement des créneaux.

Comme tout système faisant interagir un grand nombre de personnes et de moyens, nous nous retrouvons inéluctablement à rencontrer des imprévus : une salle de classe inutilisable pour cause de travaux, un enseignant indisponible pour arrêt maladie ou conférence, un événement ponctuel qui s'ajoute au planning des étudiants ou encore une épidémie mondiale forçant étudiants comme enseignants à travailler à distance. C'est pourquoi le concept de réparation d'emplois du temps est un point majeur à prendre en compte dans la confection d'emplois du temps.

Pour réaliser ce projet, nous nous sommes grandement inspirés des concours ITC, International Timetabling Competition, compétitions récompensant les meilleurs algorithmes générateurs d'emplois du temps depuis de nombreuses années et offrant des données pour réaliser des tests. Deux concours ITC ont attiré notre attention, l'ITC2007 et l'ITC2019, tous deux portant sur la résolution du problème d'emplois du temps pour les universités. En 2007, le concours durait environ 6 mois et portait sur une représentation relativement simple des données et peu de contraintes différentes, l'objectif étant de placer les séances dans les salles et créneaux disponibles (45 créneaux en tout). En 2019, le concours durait environ 15 mois et proposait des données issues de différentes universités représentées de manière plus complète en termes de possibilités de contraintes et plus malléable pour la gestion des semaines, jours et créneaux ainsi que sur la structure des cours, le but n'étant plus seulement de placer les cours sur des créneaux ou des salles mais aussi de gérer le sectionnement des étudiants en groupes. Le concours ITC2019 étant encore très récent et dû à la covid-19, la fin du concours a été repoussée, c'est pourquoi l'état de l'art sur ce problème est encore très léger au moment de l'écriture de ce rapport.

Dans la suite du rapport, vous trouverez, le contexte du stage avec une brève présentation du LERIA et de Thélème ainsi que des objectifs du stage. Ensuite, vous seront présentés l'état de l'art autour de l'ITC2007 avec la représentation utilisée et les différentes méthodes employées. Puis la représentation de l'ITC2019 ainsi que la problématique du student sectionning. Nous vous proposerons une courte présentation du vocabulaire utilisé dans le rapport, certains termes pouvant porter à confusion. Nous développerons ensuite les outils utilisés, la reproduction réalisée sur l'ITC2007 et le travail réalisé sur l'ITC2019 suivi des résultats obtenus. Enfin, nous conclurons ce rapport.

Chapitre 1

Contexte

Dans cette partie vous seront présentés le contexte de ce stage, ses objectifs ainsi que l'état de l'art sur le domaine du problème d'emplois du temps.

1.1 LERIA – Thélème

Mon stage s'est déroulé au LERIA, le Laboratoire d'Étude et de Recherche en Informatique d'Angers (`http://blog.univ-angers.fr/leria/`). Les thématiques de recherche étudiées au LERIA relèvent des différents aspects de l'intelligence artificielle. On y trouve, entre autres, la logique propositionnelle, la programmation par contraintes, la représentation des connaissances, le traitement des langues naturelles, l'apprentissage automatique, les métaheuristiques et bien d'autres méthodes. L'Université d'Angers à répondu à l'offre de projet du Programme d'Investissements d'Avenir (PIA3) pour modifier l'organisation des Licences en travaillant sur l'organisation de la formation en allant du BAC-3 au BAC+3. L'objectif de ce projet, Thélème (`https://www.univ-angers.fr/fr/universite/actualites/theleme.html`), est d'améliorer la réussite des étudiants en cherchant à réduire le taux d'abandon, en valorisant la poursuite d'études et en aidant à l'insertion professionnelle.

Le projet Thélème impose une profonde refonte du fonctionnement en Licence. L'objectif est de quitter notre fonctionnement actuel sous forme de semestre, pour s'orienter vers des périodes de 6 semaines de cours où les étudiants recevrons des enseignements sur des matières qu'ils auront sélectionnées parmi une liste, avec bien sûr, un tronc commun en fonction de la discipline choisie. Par exemple, un étudiant en informatique devra suivre les cours d'algorithmie et de web mais aura le choix de suivre le cours de synthèse d'image, statistiques avancées ou encore d'économie alors qu'un étudiant en mathématiques suivra les cours d'algèbre et d'analyse et aura le choix entre introduction à Matlab ou programmation en R.

Ce projet est encadré par David Lesaint, responsable du LERIA, Vincent Barichard, responsable du Master 2 ACDI et David Genest, directeur du portail licences MPCIE. Tous trois très habitués au problème de génération d'emplois du temps de manière manuelle, ils connaissent les différents concepts à prendre en compte et l'organisation générale du département informatique. De plus, Vincent Barichard et David Lesaint se sont spécialisés en

programmation par contraintes. Pour nous guider au niveau de l'organisation de la faculté en elle-même et des moyens techniques possibles, Yann Depauw, développeur et intégrateur d'applications à la DDN (Direction du développement du numérique) et Maryse Esnou, correspondante des applications du domaine scolarité à la DDN, nous ont rejoint lors de ce projet. Une application web a aussi été réalisé par Christophe Duveau lors de son stage de L3 pour nous aider à la fois à générer des données mais surtout pour visualiser les emplois du temps générés pour chaque élève et enseignant.

1.2 Objectifs

L'ajout du choix des cours pour les étudiants va augmenter la difficulté de la création des emplois du temps. En effet, ce choix va multiplier les cursus possibles, qu'il faudra prendre en compte lors du placement des horaires et salles. C'est pourquoi il devient impératif de passer à un système de génération automatique d'emplois du temps. Ce générateur d'emplois du temps devra répondre aux différentes contraintes inhérentes à leur création telles que le choix des salles et des horaires tout en s'assurant qu'enseignants comme élèves pourront être présents pour chacune de leurs séances. Il devra aussi prendre en compte différentes contraintes comme imposer le déroulement simultané de 3 groupes de TP pour un contrôle par exemple.

Comme tout système gérant des personnes, ce système doit être flexible, pouvoir proposer des solutions, des réparations, en cas d'absence d'enseignant, d'indisponibilité de salles ou encore d'échange de cours ou de séance entre deux enseignants sans modifier l'intégralité de l'emploi du temps déjà mis en place. Bien entendu un tel système doit fonctionner avec des informations complètes sur les différents paramètres entrant en jeu dès la génération d'une période de 6 semaines. Chaque enseignant doit savoir pour quelles séances il donnera cours, quelles seront ses disponibilités et préférences pour les horaires des cours et salles utilisées. De plus dans un problème où les emplois du temps sont générés en fonction des inscriptions des étudiants (Post Enrolment based Course Timetabling), il est nécessaire de connaître les choix des étudiants.

La création d'emplois du temps de manière automatique ajoute aussi la gestion des étudiants en groupes, le student sectionning ; l'objectif étant de chercher un groupement idéal d'étudiants, à la fois pour favoriser la cohésion de groupe, mais aussi pour simplifier le placement des heures de cours par la suite.

Pour l'état de l'art, nous nous baserons sur les concours ITC2007 et ITC2019 qui portaient sur la génération d'emplois du temps. Ces concours nous donneront une représentation du problème et des données pour réaliser des tests.

Les approches pour réaliser le générateur d'emploi du temps ou le sectionnement d'étudiants seront basées sur la programmation par contraintes ou/et les métaheuristiques.

1.3 Présentation de l'état de l'art

Concevoir des emplois du temps est une tâche récurrente chaque semestre dans toute école ou université. La création de ces emplois du temps demande un travail administratif considérable et de plus en plus difficile avec le nombre d'étudiants, de professeurs et de parcours possibles qui augmente chaque année en gardant un nombre de salles disponibles souvent constant ce qui rend la tâche de plus en plus ardue.

Ce problème a été abordé dans le cadre de nombreuses recherches, entre autres via les concours International Timetabling Competition (ITC) et leurs conférences Practice and Theory of Automated Timetabling (PATAT) depuis 2002 (`http://www.patatconference.org/communityService.html`). Les différentes compétitions les ont amenés à travailler avec différents types de problèmes d'organisation d'emplois du temps. Tout d'abord pour les universités, en divisant le problème en 3 parties : les planifications d'examens (Examination Timetabling), la planification basée sur les inscriptions (Post Enrolment based Course Timetabling) [1] et la planification basée sur le programme d'étude (Curriculum based Course Timetabling)[2]. Les concours des années suivantes portaient sur la gestion des infirmier.e.s dans les hôpitaux, sur la planification d'emplois du temps pour les lycées ou sur des problèmes plus généraux d'emplois du temps. Pour l'ITC2019, le thème est le problème d'emplois du temps pour les universités en ajoutant les problèmes de division d'étudiants en groupes (student sectioning).

Voici les objectifs des trois variantes de conception d'emplois du temps :

- les planifications d'examens :
  - espacer au maximum tous les examens,
  - possibilité de mélanger les étudiants de différentes filières dans les salles d'examens,
- le Curriculum Based Course Timetabling (CB-CTT) :
  - l'université propose différents programmes d'études,
  - les plannings sont générés en fonction des programmes,
  - les étudiants s'inscrivent aux cours en fonction de leurs disponibilités,
- le Post Enrolment based Course Timetabling (PE-CTT) :
  - différentes possibilités d'inscriptions à des cours sont proposées aux étudiants,
  - les emplois du temps sont générés après les inscriptions des étudiants.

Le problème que nous cherchons à résoudre dans le cadre de ce projet est donc basé sur les inscriptions, le Post Enrolment based Course Timetabling (PE-CTT).

Pour créer une instance du problème, il existe différentes solutions pour représenter les données. Nous en présenterons deux, celle utilisée pour l'ITC2007 puis celle utilisée pour l'ITC2019. Les contraintes et fonctionnements étant très différents d'une université à une autre, il est difficile d'imaginer une représentation générale qui rassemblerait toutes les possibilités, cependant, la représentation utilisée pour l'ITC2019 est très fexible et permet d'incorporer beaucoup de contraintes différentes.

1.4 ITC2007

Le concours de l'ITC2007 à commencé le 1 août 2007 pour finir le 25 janvier 2008. Des données étaient proposées pour les tests des modèles des différentes équipes. Le concours est divisé selon les trois variantes du problèmes cités précédemment. Ainsi, les participants disposent de 12 jeux de données de problème d'Examination Timetabling, 24 jeux de données de Post Enrolment based Course Timetabling et 21 jeux de données de Curriculum based Course timetabling (disponibles à l'adresse `http://www.cs.qub.ac.uk/itc2007/Login/SecretPage.php`). Les 5 premières équipes obtenant les meilleurs scores sur chaque type de problème étaient récompensées.

1.4.1 Représentation

La représentation de l'ITC2007 est basique mais permet de donner les informations sur les inscriptions aux cours, les salles disponibles pour chaque séance de cours (en fonction des caractéristiques de celles-ci, du nombre d'étudiants inscrits à la séance et de la taille de la salle de classe), les créneaux possibles pour chaque séance et la notion de précédence entre deux cours (ou plus). La notion de précédence correspond au fait de placer une séance avant une autre sur le planning.

Partant de cette représentation l'objectif était de satisfaire les contraintes strictes (hard constraints) suivantes :

1. les étudiants n'ont qu'une seule séance à la fois,
2. les séances ont lieu dans des salles, disponibles, pouvant accueillir tous les étudiants inscrits et répondent aux caractéristiques demandées,
3. une seule séance par salle, par créneau,
4. une séance ne peut avoir lieu que dans l'un de ces créneaux prédéfinis
5. quand spécifié, les événements doivent se dérouler dans le bon ordre dans la semaine

Dans un second temps, si une solution au problème existe, le solveur devait chercher à optimiser la solution afin de satisfaire au maximum les contraintes souples (soft constraints) suivantes :

1. éviter d'avoir cours sur les derniers créneaux de chaque jour
2. éviter d'avoir plus de deux cours à suivre le même jour
3. éviter d'avoir un seul cours par jour

Les données sont représentées dans un fichier de la manière suivante :

— sur la première ligne : nombre de séances, nombre de salles de classes, nombre de caractéristiques (exemple : vidéo-projecteur, tablettes, salle informatique,...), nombre d'étudiants (le nombre de créneaux est fixe, 5 jours de 9 heures soit 45 créneaux),
— une ligne par salle de classe : le nombre de place pour la salle de classe,
— une ligne par étudiant / séance : 1 si l'étudiant assiste à la séance, 0 sinon,
— une ligne par salle de classe / caractéristique : 1 si la salle possède la caractéristique demandée, 0 sinon,

— une ligne par séance / caractéristique : 1 si la séance demande la caractéristique, 0 sinon,
— une ligne par séance / créneau : 1 s'il est possible de faire la séance à ce créneau, 0 sinon,
— une ligne par séance / séance : 1 si la séance doit se dérouler avant la seconde séance, -1 si c'est l'inverse et 0 sinon.

Soit 4 nombres, 1 tableau et 5 matrices pour représenter le problème.

1.4.2 Méthodes

Voici les articles (ou extrait d'articles) et méthodes utilisées par les finalistes de la compétition :

— 1ère place : H. Cambazard et al. (Ireland) [3] remportent avec leur recherche locale mais ont aussi développé des modèles en programmation par contraintes hybridés avec une recherche locale dans le cadre d'un LNS (Large Neighborhood Search).
— 2ème place : M. Atsuta et al. (Japan) [4] ont représenté les instances sous forme de CSP (Constraint Satisfaction Problem) couplé à une recherche locale avec tabou avec un contrôle des poids des contraintes.
— 3ème place : M. Chiarandini et al. (Denmark) [5] ont divisé le problème en deux parties : dans un premier temps, une alternance de phase de construction et une phase de recherche locale pour donner une première solution valide puis une recherche locale pour améliorer les résultats.
— 4ème place : C. Nothegger et al. (Austria) [6] ont utilisé un algorithme de colonies de fourmis.
— 5ème place : T. Müller (USA) [7] utilise un système en plusieurs étapes, construction avec une IFS (Iterative Forward Search) puis recherche locale, ensuite un GD (Great Deluge) parfois alterné avec un recuit simulé jusqu'à arriver à une solution optimale.

D'autres méthodes ont étés utilisées pour tenter de résoudre ce problème :

— Méthodes exactes : SAT [8]
— Méthodes utilisant des populations (algorithmes génétiques, colonies de fournies,...) [9] [10] [11]
— Méthodes basées sur la recherche locale, recuit simulé ou tabou [3][12] [13]

L'article de A. Bashab et al. [14] (sorti en juin 2020) liste différentes méthodes utilisées parmi 131 publications. Entre autres, nous retrouvons :

— 41 algorithmes hybrides,
— 23 algorithmes génétiques (et 4 mémétiques),
— 7 recuits simulés (et 3 tabou),
— 6 algorithmes de colonies d'abeilles artificielles,
— 6 algorithmes de colonies de fourmis

Les approches exactes sont généralement moins utilisées que les méthodes approchées (recherche locale ou hybridation de métaheuristiques) car face à la taille et à la complexité des problèmes sur des cas réels, le temps de résolution est souvent plus long [3]. Cependant, une modélisation SAT implémentée lors de l'ITC2019 [8] a obtenu de bons résultats.

1.5 ITC2019

Le courcours ITC2019 a commencé le 30 août 2018 pour finir le 18 Novembre 2019 (le 1 janvier 2020 pour la date de rendu des algorithmes codés, la date de présentation des vainqueurs est repoussée à 2021 pour cause d'épidémie mondiale). Les articles des vainqueurs ne sont pas tous accessibles et peu de personnes ont encore travaillé sur ces données, il existe donc peu d'articles sur cette représentation et ces données au moment de l'écriture de ce rapport.

1.5.1 Vocabulaire

Faisons un petit point sur le vocabulaire avant de commencer, la représentation pouvant être compliquée à saisir au premier abord. Les *courses* représentent les matières telles que Programmation orientée objet en C++, Développement Web ou Développement mobile. Nous utiliserons le terme cours ou *course* (en italique). Un cours est divisé en différentes *configs* (configurations) elles-mêmes composés de *subparts*. Les *subparts* (sous-parties) divisent les cours en fonction des types de séances auxquels les étudiants doivent assister. Un étudiant inscrit à un cours devra participer à une séance par sous-partie d'une des configurations du cours. Le cours de Développement mobile est composé d'un ensemble de séances de CM et de deux ensembles de TP pour représenter les deux groupes. Il possèdera donc deux *subparts*, la première proposant une *class* (ensemble de séances de CM qui se répètent sur plusieurs semaines) et la seconde *subpart* composée de deux *class* où chacune des *class* a lieu de manière répétée sur plusieurs semaines. Les *class* représentent donc des séances de cours. Si la *class* se déroule sur plusieurs jours ou plusieurs semaines (plusieurs bits à 1 dans les bitsets *days* ou/et *weeks*) alors la *class* représente un ensemble de séances qui se répètent au fil des semaines, sinon elle ne représente qu'une seule séance.

Côté optimisation, nous avons deux types de contraintes. Les contraintes dures (*hard constraints*) représentent des contraintes qui doivent à tout prix être respectées, par exemple le fait qu'un enseignant ne doit pas donner cours dans deux classes différentes à la fois. Si une contrainte dure n'est pas respectée dans une solution proposée, alors l'emploi du temps n'est pas considéré comme valide. Les contraintes souples (*soft constraints*) donnent le score du planning généré, 0 étant le score parfait. Dans le cas de l'ITC2007 les contraintes souples sont : ne pas avoir de cours sur les derniers créneaux de chaque journée, de ne pas avoir deux cours à suivre et de ne pas avoir un seul cours sur une journée. Dans le cas de l'ITC2019, les contraintes souples sont divisées en 4 catégories dont chacune possède un poids via la balise *optimization*. Les contraintes *time* et *room* correspondent à la somme des pénalités des créneaux et salles de classes choisies, c'est-à-dire à quel point les préférences des enseignants sont respectées. Les contraintes *student* correspondent au fait qu'un étudiant ne se retrouve pas avec plus d'un cours à la fois. En général cette contrainte est plutôt considérée comme dure en France, cependant dans le concours, ils la considèrent comme souple et lui donnent un poids très fort. Les contraintes *distribution* correspondent aux contraintes de répartition souples non respectées. Les contraintes de *distribution* s'appliquent à un ensemble de *class*/séances et non à des *courses*/cours.

1.5.2 Représentation

Dans le cadre de l'ITC2019, 36 fichiers de données sont mis à disposition au format XML. Ces fichiers sont extraits de données réelles de différentes universités afin de pouvoir réaliser des benchmarks lors du concours et tester les différents solveurs. Ces fichiers partagent une structure commune, voir [15] ou `https://www.itc2019.org/format` pour la formulation originale proposée pour le concours.

La représentation nous propose une une liste de salles de classes, avec des informations de disponibilités et de temps de voyage entre salles, une liste de cours, chaque cours étant composé de configurations, elles-mêmes composées de sous-parties et composées à leurs tour de séances de cours. Les séances de cours possèdent des créneaux et des salles possibles. Les étudiants sont représentés dans une liste qui nous donne aussi les cours qu'ils suivent. Un étudiant inscrit à un cours devra assister à une séance de chaque sous-partie de l'une des configurations du cours. À cela s'ajoute les contraintes de répartitions imposant aux séances concernées de se dérouler dans la même salle ou en même temps par exemple. De plus, nous disposons de poids sur le choix des créneaux, des salles, de la disponibilité des étudiants et sur les contraintes de distribution pour guider l'optimisation de la solution.

<table>
<tr><td>Cours</td><td colspan="5">Développement Python</td></tr>
<tr><td>Configurations</td><td colspan="2">Niveau Intermédiaire</td><td colspan="3">Niveau Débutant</td></tr>
<tr><td rowspan="3">Sous-parties</td><td>CM</td><td></td><td>CM</td><td colspan="2"></td></tr>
<tr><td></td><td>TP</td><td></td><td>TD</td><td></td></tr>
<tr><td colspan="4"></td><td>TP</td></tr>
<tr><td rowspan="7">Séances</td><td>CM-NI</td><td></td><td>CM-ND</td><td colspan="2"></td></tr>
<tr><td></td><td>TP-NI Grp A</td><td></td><td>TD-ND Grp A</td><td></td></tr>
<tr><td></td><td>TP-NI Grp B</td><td colspan="2"></td><td>TP-ND Grp A1</td></tr>
<tr><td colspan="4"></td><td>TP-ND Grp A2</td></tr>
<tr><td colspan="3"></td><td>TD-ND Grp B</td><td></td></tr>
<tr><td colspan="4"></td><td>TP-ND Grp B1</td></tr>
<tr><td colspan="4"></td><td>TP-ND Grp B2</td></tr>
</table>

Le schéma ci-dessus vous présente l'organisation du cours de Développement Python. Celui-ci est divisé en deux configurations, Une configuration pour les étudiants connaissant déjà le Python, Niveau Intermédiaire (NI), et une seconde pour les autres, Niveau Débutant (ND). La configuration NI propose deux sous-parties, un ensemble de CM suivis d'un ensemble de TP divisés en deux groupes. La configuration ND propose trois sous-parties, CM, TD et TP. Le CM ne possède qu'une séance, les TD sont divisés en deux groupes A et B qui sont eux-mêmes divisés en deux pour les TP, le TD groupe A étant désigné comme *parent* des TP groupe A, 1 et 2.

Voir figure 1.1 pour une représentation du fichier xml sous forme de graphe que nous détaillerons dans cette partie en adaptant les explications à l'Université d'Angers.

Le problème *problem* est divisé en 6 parties, les salles de classes *rooms*, les cours *courses*, les poids des différentes contraintes *optimization*, les étudiants *students*, les contraintes de répartition *distributions* et la balise solution optionnelle *solution*.

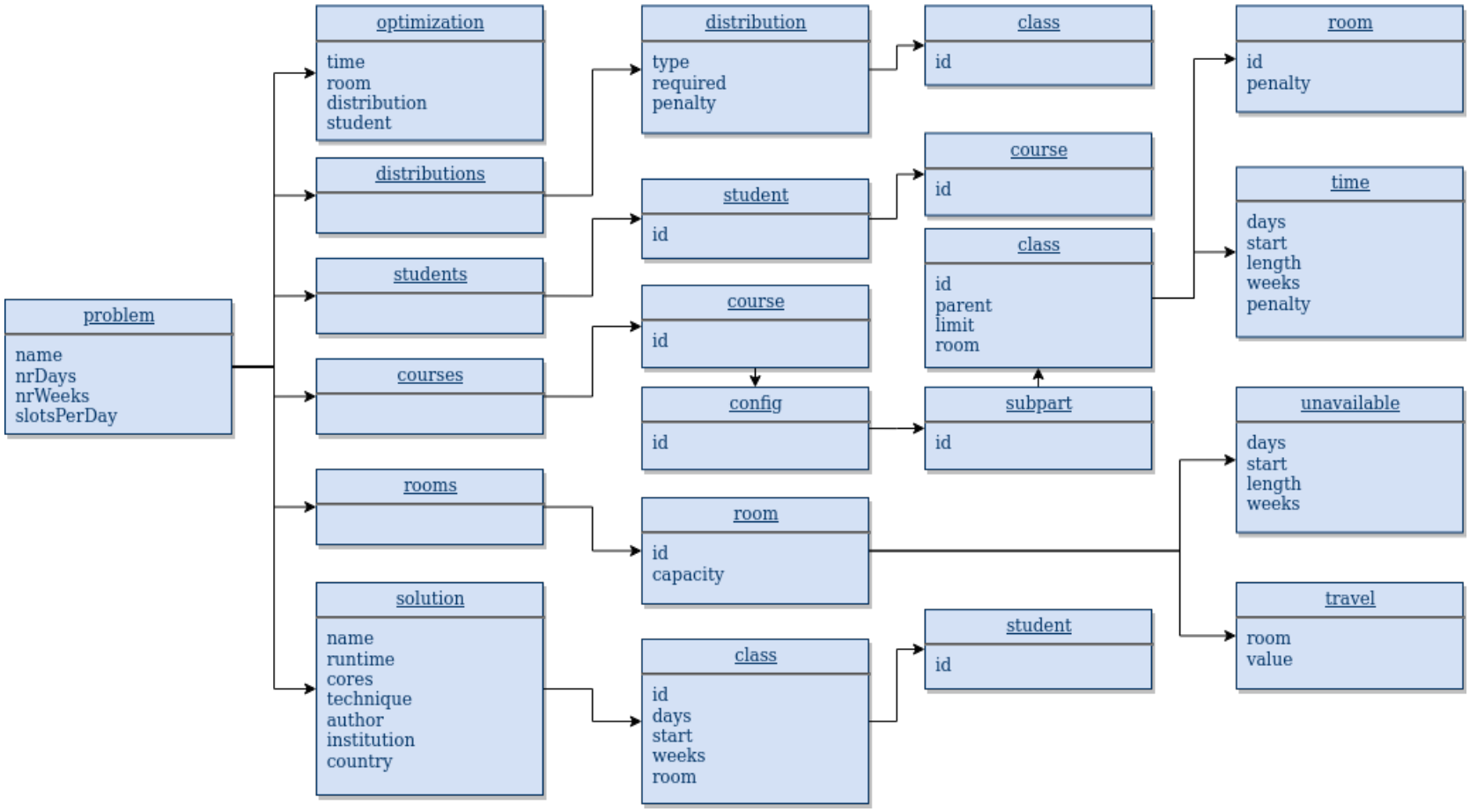

FIGURE 1.1 – L'instance du problème est représentée sous la forme d'un *problem* composé des balises *optimization*, *rooms*, *distribution*, *students*, *courses* et éventuellement une balise *solution* pour l'envoi du résultat pour l'ITC2019.

problem* et *optimization

```
<?xml version="1.0" encoding="UTF-8"?>
<problem      name="UA"
              nrDays="7"
              nrWeeks="6"
              slotsPerDay="288">
  <optimization time="2" room="1" distribution="1" student="2"/>
  <rooms> ... </rooms>
  <courses> ... </courses>
  <distributions> ... </distributions>
  <students> ... </students>
</problem>
```

problem Le problème traité possède un nom, un nombre de jours dans la semaine (généralement 7), un nombre de semaines à traiter pour le problème (6 pour Thélème) et un nombre d'emplacement par jour, en divisant une journée par tranches de 5 minutes nous obtenons 288 emplacements. Tous les temps seront ensuite considérés en fonction de la tranche choisie. Il possède les balises *optimisation*, *rooms*, *distribution*, *students*, *courses* et éventuellement *solution* que nous décrirons ensuite.

optimization L'optimisation possède les poids des pénalités pour les placements d'horaires *time*, salles de classes *room*, contraintes de répartitions non respectées *distribution* et nombre de conflits d'étudiants *student*.

rooms

```
<rooms>
    <room id="AMPHI-A" capacity="100"/>
    <room id="AMPHI-B" capacity="100"/>
    ...
    <room id="A014" capacity="40"/>
    ...
    <room id="L231" capacity="40">
        <travel room="A014" value="2"/>
        <unavailable days="100000" start="102" length="24" weeks="111111"/>
        <unavailable days="010100" start="102" length="24" weeks="101010"/>
    </room>
    ...
  </room>
</rooms>
```

rooms Les salles de classes (*room*) sont gérées dans la balise *rooms*. Elles possèdent les attributs *id* et *capacity* (pour le nombre de places disponibles dans la salle). Les balises *room* disposent de balises *travel*, indiquant la distance jusqu'à une autre salle en nombre de créneaux de 5 minutes (s'il n'y a pas d'informations nous considérons un temps nul), et de balises *unavailable* indiquant les périodes où la salle n'est pas disponible, dans l'exemple, la salle L231 est indisponible le lundi de 8h30 à 10h30 toutes les semaines et les mardis et jeudis des semaines impaires de 8h30 à 10h30.

courses

Pour les *courses*, prenons l'exemple du cours de programmation logique. Un premier enseignant s'occupe des CM des trois premières semaines (CM-1) et des TD et TP A et un second des 3 derniers CM (CM-2) ainsi que les TD-B et TP B et C. Nous verrons par la suite comment les classes sont affectées aux enseignants et comment les élèves s'inscrivent aux cours. Le contenu de la balise *courses* est un ensemble de balises *course* comme celle-ci :

```
<course id="Programmation␣Logique">
  <config id="PL-1">
    <subpart id="PL-CM1">
      <class id="Programmation␣Logique␣CM-1" limit="80">
        <room id="AMPHI-A" penalty="0"/>
        ...
        <!-- Lundi - 11h00-12h20 sur les 3 premieres semaines-->
        <time start="133" length="16" days="1000000" weeks="111000" penalty="1"/>
        <!-- Lundi - 14h00-15h20 sur les 3 premieres semaines-->
        <time start="169" length="16" days="1000000" weeks="111000" penalty="0"/>
        ...
        </class>
    </subpart>
    <subpart id="PL-CM2">
      <class id="Programmation␣Logique␣CM-2" limit="80">
        <room id="AMPHI-A" penalty="0"/>
        <room id="AMPHI-B" penalty="0"/>
        ...
        <!-- Mardi - 11h00-12h20 sur les 3 dernieres semaines-->
        <time start="133" length="16" days="0100000"
              weeks="000111" penalty="1"/>
        <!-- Mardi - 14h00-15h20 sur les 3 dernieres semaines-->
        <time start="169" length="16" days="0100000"
              weeks="000111" penalty="0"/>
        ...
        </class>
    </subpart>
    <subpart id="TD">
      <class id="Programmation␣Logique␣TD-A" limit="40">...</class>
      <class id="Programmation␣Logique␣TD-B" limit="40">...</class>
    </subpart>
    <subpart id="TP">
      <class id="Programmation␣Logique␣TP-A" limit="20">...</class>
      <class id="Programmation␣Logique␣TP-B" limit="20">...</class>
      <class id="Programmation␣Logique␣TP-C" limit="20">...</class>
    </subpart>
</config>
</course>
```

courses Les cours (*course*) (au sens large, UE ou matière) sont contenus dans la balise *courses*. Chaque cours a un *id* et possède une ou plusieurs configurations (*config*) auxquelles les étudiants peuvent assister. L'utilisation des différentes configurations étant très spécifique nous ne l'avons pas détaillée dans le rapport mais elle est prise en compte dans le travail

réalisé. Dans la suite du rapport, pour simplifier les explications, nous allons considérer que chaque cours ne possède qu'une seule configuration. Les sous-parties (*subpart*) et leur *id* permettent de différencier les séances de cours tel que CM, TD, TP. Les étudiants inscrits au cours doivent être présents à une séance de chaque sous-partie. Les séances possèdent un *id* et éventuellement une séance parente et une limite d'étudiants. Si la séance se déroule en télétravail ou n'utilise pas de salle, un attribut *room* doit être mis à False. Les étudiants doivent être divisés sur les différentes séances de manière à ce que les limites de taille ne soient pas dépassées. Dans l'exemple un étudiant devra assister au CM-1 puis au CM-2, puis au TD-A ou B puis au TP A, B ou C.

class La balise *class* contient différentes balises dont *time* permettant de gérer les horaires disponibles pour chaque cours. Les jours disponibles se gèrent avec des 0 et 1 dans l'attribut *days* (premier 0/1 correspond au lundi, le second au mardi,...), *start* correspond à l'heure de début du cours (0 correspondant à minuit, 90 à 7h30), *length* correspond au nombre d'emplacement de temps (le découpage du temps étant réalisé par tranches de 5 min, 10 correspond à 50 min), *weeks* correspond aux semaines où le cours a lieu sur le même fonctionnement que les jours (premier 0/1 correspond à la première semaine et ainsi de suite...). Ainsi en fonction de ses balises *time* une *class* peut représenter une seule séance ou un ensemble de séances qui se répètent plusieurs fois dans la semaine et/ou sur plusieurs semaines. Chaque *time* et *room* possède une *penalty*, celle-ci correspond à la préférence de l'enseignant pour ces horaires de cours ou ces salles, 0 étant le meilleur score.

students

```
<students>
    <student id="Alice">
        <course id="securite_reseaux"/>
        <course id="programmation_logique"/>
        <course id="developpement_web"/>
    </student>
    <student id="Bob">
        <course id="algebre_lineaire"/>
        <course id="programmation_logique"/>
        <course id="cryptographie"/>
    </student>
</students>
```

students Les étudiants (*student*) sont contenus dans la balise *students*. Chaque étudiant est inscrit aux cours repérés par leurs *id* dans les balises *course*. Il devra donc participer à une séance (*class*) de chaque *subpart* pour chacun des cours auquel il est inscrit.

distributions

```
<distributions>
  <distribution type="SameAttendees" required="true">
    <class id="Programmation_Logique_CM-1"/>
    <class id="Programmation_Logique_TD-A"/>
    <class id="Programmation_Logique_TP-A"/>
  </distribution>
```

```
  <distribution type="SameAttendees" required="true">
    <class id="Programmation Logique CM-2"/>
    <class id="Programmation Logique TD-B"/>
    <class id="Programmation Logique TP-B"/>
    <class id="Programmation Logique TP-C"/>
  </distribution>
  <distribution type="SameStart" penalty="2">
    <class id="Programmation Logique TD-A"/>
    <class id="Programmation Logique TD-B"/>
  </distribution>
</distributions>
```

distribution En plus des contraintes de temps et disponibilité des salles, nous disposons de différentes contraintes de répartition (listées ci-dessous). Les contraintes de répartition possèdent un *type* correspondant au type de contrainte présenté dans le chapitre suivant. Nous avons ici l'exemple *SameAttendees* utilisé pour gérer les enseignants. L'enseignant n'est pas représenté avec une balise *teacher* (ou autre) mais est présent grâce à la contrainte *SameAttendees*. On informe le modèle que le CM-1, TD-A et TP-A de programmation logique ne devront pas se chevaucher sur le planning et que le temps de déplacement entre les salles utilisées devra permettre d'atteindre la séance suivante à temps. Elles disposent aussi de, soit l'attribut *required* a vrai, qui signifie que la contrainte est stricte (hard constraint), soit l'attribut *penalty*, qui signifie que la contrainte est souple (soft constraint) avec la pénalité indiquée si celle-ci n'est pas respectée.

Les contraintes de répartition sont au nombre de 19, vous pouvez trouver la liste sur ce lien `https://www.itc2019.org/format#distributions` avec le détail de chaque contrainte. Vous trouverez ici un résumé de chacune :

- SameStart : Les séances concernées doivent commencer à la même heure (peu importe le jour ou la semaine).
- SameTime : Les séances concernées doivent se dérouler durant les créneaux de la plus longue séance de la contrainte (peu importe le jour ou la semaine).
- DifferentTime : Les séances concernées ne doivent pas se chevaucher.
- SameDays : Les séances doivent se dérouler sur les mêmes journées ou sur un sous-ensemble de jours de la classe qui se déroule sur le plus de jours.
- DifferentDays : Les séances ne doivent pas se dérouler pendant les mêmes jours.
- SameWeeks : Les séances doivent se dérouler sur les mêmes semaines ou sur un sous-ensemble des semaines de la classe qui se déroule sur le plus de semaines.
- DifferentWeeks : Les séances ne doivent pas se dérouler pendant les mêmes semaines.
- Overlap : Les séances doivent se chevaucher.
- NotOverlap : Les séances ne doivent pas se chevaucher.
- SameRoom : Les séances doivent se dérouler dans la même salle de classe.
- DifferentRoom : Les séances doivent se dérouler dans des salles différentes.
- SameAttendees : Les séances ne doivent pas se chevaucher et les salles attribuées aux séances doivent permettre aux personnes présentes à la séance d'avoir assez de temps pour aller d'une salle à une autre.
- Precedence : Les séances listées doivent se dérouler dans l'ordre listé. Si des séances

possèdent plusieurs rencontres dans la semaine ou sur plusieurs semaines, c'est l'horaire le plus tôt qui est pris en compte.
- WorkDay(S) : Si les séances se déroulent sur des jours en commun, alors le temps entre la fin de la séance la plus tardive et le début de la séance qui commence le plus tôt doit être inférieur ou égal à S.
- MinGap(G) : Si les séances se déroulent sur des jours en commun, alors le temps entre la fin de la séance qui commence le plus tôt et entre le début de celle qui commence le plus tard doit être inférieur à G.
- MaxDays(D) : Les séances ne doivent pas se dérouler sur plus de D jours (peu importe la semaine).
- MaxDayLoad(S) : Les séances ne doivent pas se dérouler sur plus de S unités de temps par jour sur toute la période.
- MaxBreaks(R,S) : Cette contrainte limite le nombre de pauses de durée S entre les séances à R fois maximum par jour.
- MaxBlock(M,S) : Cette contrainte limite le temps M entre des séances consécutives si les séances ne sont pas séparées par au moins S unités de temps.

solution

```
<solution name="UA"
          runtime="20.20" cores="8" technique="Local Search"
          author="Cyril Grelier" institution="UA"
          country="France">
  <class id="Programmation Logique CM-1" days="1000000"
         start="169" weeks="111000" room="AMPHI-A">
    <student id="Alice"/>
    <student id="Bob"/>
  </class>
  <class id="Programmation Logique TD-A" days="0100000"
         start="86" weeks="111111" room="H001">
    <student id="Alice"/>
  </class>
  <class id="Programmation Logique TD-B" days="0100000"
         start="86" weeks="111111" room="H002">
    <student id="Bob"/>
  </class>
</solution>
```

solution Dans le cadre du concours ITC2019, les participants doivent donner leurs résultats sur les différentes instances d'emplois du temps. Il est aussi possible de soumettre un résultat sans faire partie du concours pour valider la solution. Il faudra indiquer le temps de calcul, les CPU utilisés, la méthode employée, l'auteur, son organisme et pays suivis du planning proposé avec les positions des différentes classes dans le temps et leurs étudiants inscrits. Cette représentation est aussi utilisée pour afficher l'emploi du temps sur l'application web développée pour ce projet.

1.5.3 Student Sectionning

L'ITC2019 ajoute, en plus du placement de créneaux et de salles, la problématique de *student sectionning*, c'est-à-dire le fait de répartir les étudiants en groupes. Le student sectionning peut être traité avec différentes méthodes. Voir [16], [17] et [18] pour plus d'informations sur cette partie. Comme vous l'avez remarqué dans la représentation des données de 2019, les étudiants sont inscrits à des *courses* composés de différentes *subpart*, elles-mêmes composées de *class* auxquelles les étudiant devront assister. Prenons l'exemple avec les matières suivantes :

— Développement web (DW) composé d'un CM puis d'un TD divisé en 3 groupes puis d'un TP divisé en 3 groupes.
— Programmation fonctionnelle (PF) composée d'un CM puis deux groupes de TD et deux groupes de TP
— Synthèse d'image (SI) composée d'un CM puis d'un TP sur un seul groupe
— Interfaces graphiques (IG) composée d'un CM puis d'un TP sur un seul groupe

Le DW et PF représentent le tronc commun aux étudiants de L3 informatique, SI et IG sont deux options. L'objectif du *student sectionning* sera de répartir les étudiants sur les différentes matières en conservant les mêmes groupes d'étudiants. Favoriser ces groupes apporte certains avantages, comme la cohésion de groupe, faciliter les projets de groupe en raison des horaires sans cours des étudiants qui tombent en même temps et tous les avantages que l'on peut voir à avoir des étudiants en groupes. Mais surtout, cela facilite le placement des créneaux des cours. Si tous les étudiants de SI sont dans les mêmes groupes dans les matières du tronc commun, alors placer les créneaux sera simplifié comparé au cas où les étudiants sont répartis sur tous les groupes possibles. Comme vous pouvez le remarquer, les groupes ne font pas systématiquement la même taille, il existe 3 groupes clairement identifiés pour le DW, 3 groupes en PF mais les étudiants d'un des groupes de TP devront être répartis sur deux groupes pour les TD. Nous trouvons une grande quantité d'exceptions pour chaque cours. Par exemple il faudra prendre en compte les redoublants qui modifieront la taille d'un groupe dans une matière mais qui ne seront pas présents dans une autre, etc.

Le student sectionning est généralement abordé de trois manières différentes : avant, pendant ou après la génération de l'emploi du temps à proprement parler, le placement des créneaux et l'attribution des salles. Certains alternent les trois possibilités quitte à remodifier plusieurs fois l'emploi du temps. Le fait de sectionner les étudiants avant réduit l'espace de recherche avant le placement des créneaux et salles. Dans le pire cas l'emploi du temps sera impossible à réaliser et certaines possibilités seront bloquées à cause de groupes mal formés. Dans le meilleur cas, le nombre de conflits entre étudiants avec des profils différents sera si faible que la génération de l'emploi du temps sera grandement simplifiée. Pour le cas de la gestion pendant la recherche, l'espace de recherche sera le plus grand possible, ce qui permettra d'être très souple pendant la recherche. Cependant il sera plus long d'arriver à une solution faisable et une modification d'un seul groupe d'étudiants peut grandement dégrader la solution courante. Gérer les groupes après le placement des créneaux demandera de relancer la recherche à de multiples reprises afin d'arriver à un placement des créneaux idéal.

Chapitre 2

Travail Réalisé

2.1 Outils et architecture du projet

Pour réaliser le travail présenté dans ce chapitre, nous avons utilisé les outils résumés ci-après. Minizinc (`https://www.minizinc.org/`), un langage de modélisation par contraintes open-source. Minizinc permet la modélisation de nos modèles, puis la compilation vers du FlatZinc, langage d'entrée pour de nombreux solveurs tel que Gecode (`https://www.gecode.org/`). Gecode est un outil open-source utilisable en C++ pour le développement de systèmes à base de contraintes.

Nos tests sur Gecode sont réalisés en C++. Dans le cas de Minizinc, il existe une bibliothèque en cours de développement en Python. À ce jour l'ensemble des fonctionnalités n'est cependant pas encore totalement implémenté. La majorité du code est développé en Python, la base du projet (environnements de tests de couverture, intégration, validation du code, création de la documentation) est générée par CookieCutter (`https://github.com/cookiecutter/cookiecutter`) et la documentation est générée automatiquement avec Sphinx (`www.sphinx-doc.org`). Les graphiques sont générés avec l'aide de l'outil Matplotlib [19]. L'ensemble des tests sur le format de l'ITC2019 sont réalisés avec l'interprète et compilateur Python, PyPy (`https://www.pypy.org/`), qui permet une exécution beaucoup plus rapide comparé au classique CPython. Le code est hébergé sur github (`https://github.com/Cyril-Grl/EmploisDuTemps`) pour le travail sur les données de l'ITC2007 puis sur gitlab (`https://gitlab.com/Cyril_Grl/emploisdutemps`) pour les données de l'ITC2019, gitlab permettant une intégration des tests facilement avec leur outil GitLab CI/CD (Continuous Integration (CI), Continuous Delivery (CD), Continuous Deployment (CD)). Les caractéristiques de l'ordinateur sur lesquelles les tests sont réalisés sont les suivantes : Quad core Intel Core i5-8400H, 2.50GHz $\times$ 8, 16Go de RAM.

Le schéma 2.1 présente l'architecture du projet pour le travail réalisé sur les données de l'ITC2019. Christophe Duveau a réalisé pour son stage de L3 Informatique, le gestionnaire d'emplois du temps avec une interface web, nous avons pu alors créer une instance du problème en se basant sur le planning de la L3 informatique d'Angers. Nous disposons d'un outil pour gérer le student sectionning qui génère en sortie un fichier XML que nous utilisons ensuite sur nos modèles avec l'instance de base pour résoudre le problème. Nous avons aussi

développé l'aspect réparation pour proposer des modifications après génération du planning.

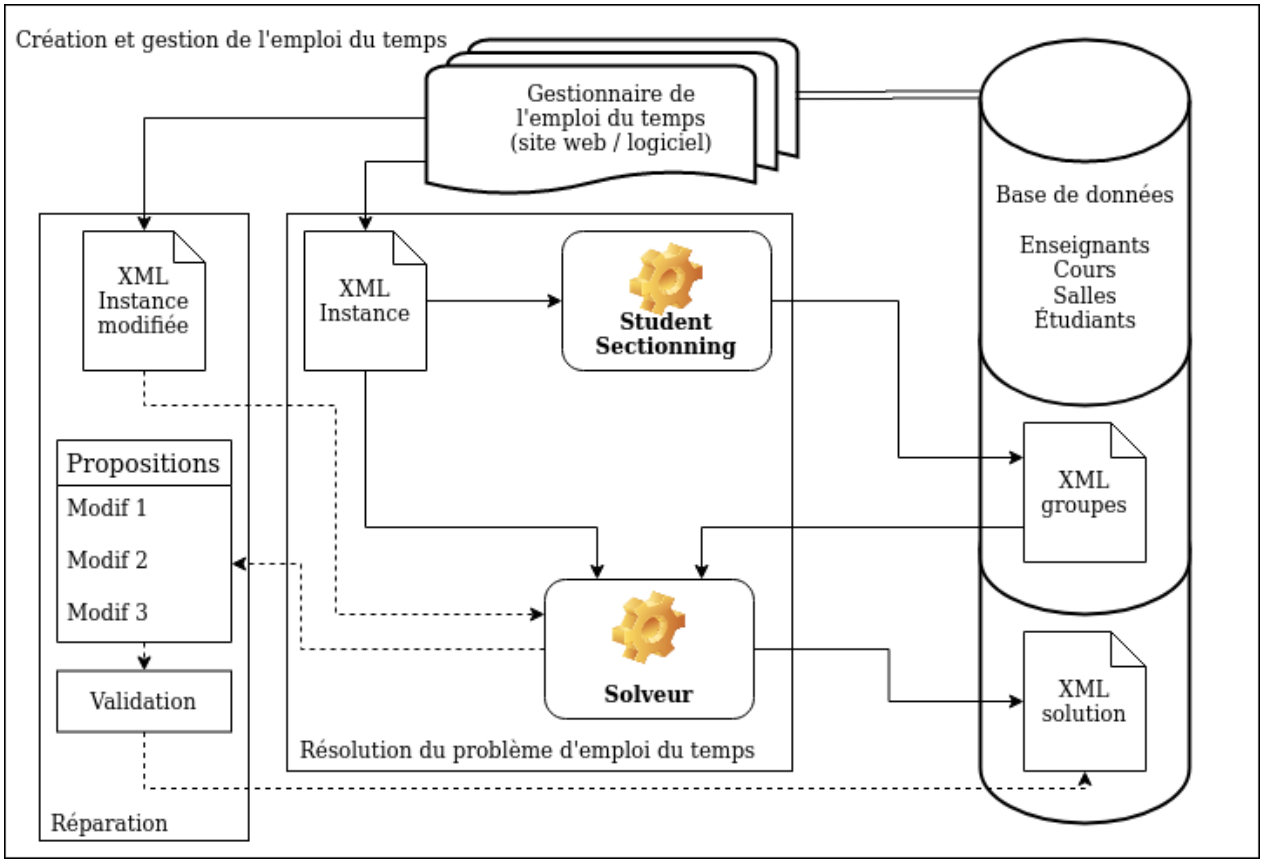

FIGURE 2.1 – Fonctionnement général du projet pour tout l'aspect ITC2019.

2.2 Reproduction état de l'art ITC2007

Pour débuter et s'initier au problème de génération d'emploi du temps, nous nous sommes attaqués au concours de l'ITC2007 en se basant sur l'article de H. Cambazard et al. [3], vainqueurs de l'ITC2007. Dans leur article, ils présentent la recherche locale qui leur a permis de remporter le concours mais aussi 3 modèles de programmation par contraintes (PPC) qu'ils ont testés sur les différentes instances ainsi qu'une Large Neighborhood Search (LNS). La PPC a l'avantage d'offrir un planning faisable s'il en existe un mais reste difficile à utiliser pour réduire le nombre de contraintes souples. C'est pourquoi ils ont implémenté une LNS (initialisée avec leur recherche locale), afin de réduire la complexité et de tendre vers un modèle plus simple (nous vous présenterons dans la partie suivante la LNS que nous avons implémentée sur le modèle de l'ITC2019).

Dans le cadre de cette reproduction nous avons reproduit leurs modèles 1 et 2 en Minizinc et le modèle 2 en Gecode.

2.2.1 Modèle 1

Le modèle 1 est composé de :

— Données :
  — n : le nombre de séances.
  — m : le nombre de salles de classe.
  — R_i : l'ensemble des salles de classe possibles pour la séance i (en fonction du nombre d'étudiants inscrits à la séance et des caractéristiques de la salle).
  — T_i : l'ensemble des créneaux possibles pour la séance i.
  — $student(i)$: l'ensemble des ensembles des étudiants devant assister à la séance i.
  — $prec$ l'ensemble des paires de séances possédant une contrainte de précédence.
— Variables :
  — $eventTime_i \in \{1, ..., 45\}$ pour le créneau choisi pour la séance i (les cours se déroulant sur 5 jours de 9 heures).
  — $eventRoom_i \in \{1, ..., m\}$ pour la salle choisie pour la séance i.
— Contraintes :
  1. $\forall i, j \leq n$ tel que $student(i) \cap student(j) \neq \emptyset$ $eventTime_i \neq eventTime_j$
     Si deux séances ont des étudiants en commun alors leurs créneaux doivent être différents.
  2. $\forall i \leq n$ $eventRoom_i \in R_i$
     Une séance doit avoir lieu dans une salle qui lui est attribuée.
  3. $\forall i, j \leq n$ $(eventTime_i \neq eventTime_j) \vee (eventRoom_i \neq eventRoom_j)$
     Deux séances ne peuvent pas se dérouler dans la même salle au même créneau.
  4. $\forall i \leq n$ $eventTime_i \in T_i$
     Une séance doit avoir lieu dans une salle qui lui est attribuée.
  5. $\forall (i, j) \in prec$ $(eventTime_i < eventTime_j)$
     Les séances avec une contrainte de précédence doivent se dérouler dans le bon ordre.

Ce modèle est divisé en deux parties : les contraintes 1, 3 et 5 forment une résolution de problème de coloration et les contraintes 2 et 4 assurent le bon positionnement des créneaux et salles. H. Cambazard et al. [3] ont remarqué que dû à la disjonction en 3, tant que les créneaux et salles ne sont pas placés, il y a beaucoup de pertes lors de la recherche de solution. C'est pourquoi ils ont cherché à diminuer ces pertes avec le modèle suivant.

2.2.2 Modèle 2

Pour diminuer les pertes dues au placement des séances sur les salles, ils ont cherché à supprimer la variable *eventRoom*. Le placement dans les salles sera maintenant représenté par un graphe biparti $G = (V_1, V_2, E)$ où $V_1 = \{1, ..., n\}$ (l'ensemble des séances), $V_2 = \{(1,1), ..., (45, m)\}$ (l'ensemble des paires $(salles, creneaux)$), un arc $(i, (j, k))$ est présent si la séance i peut avoir lieu au créneau j dans la salle k.

— Données :
  — n : le nombre de séances.
  — (R_i, T_i) : l'ensemble des paires de $(salles, creneaux)$ auxquelles la séance i peut avoir lieu.
  — T_i : l'ensemble des créneaux possibles pour la séance i.
  — $student(i)$: l'ensemble des ensembles des étudiants devant assister à la séance i.
  — $prec$ l'ensemble des paires de séances possédant une contrainte de précédence.
— Variables :
  — $eventTime_i \in \{1, \ldots, 45\}$ pour le créneau choisi pour la séance i (les cours se déroulant sur 5 jours de 9 heures).
  — $event_i \in (R_i, T_i)$ pour la paire $(salles, creneaux)$ choisie pour la séance i.
— Contraintes :

1. $\forall i, j \leq n : student(i) \cap student(j) \neq \emptyset \; eventTime_i \neq eventTime_j$
   Si deux séances ont des étudiants en commun alors leurs créneaux doivent être différents.

4. $\forall i \leq n \; eventTime_i \in T_i$
   Une séance doit avoir lieu dans une salle qui lui est attribuée.

5. $\forall (i, j) \in prec \; (eventTime_i < eventTime_j)$
   Les séances avec une contrainte de précédence doivent se dérouler dans le bon ordre.

6. $\forall i \leq n \; event_i \in (R_i, T_i)$
   Toute séance est placée sur une paire $(salles, creneaux)$ possible.

7. $\forall i \leq n \; eventTime_i = event_i[0]$
   Le créneau de la séance i est celui de la paire $(salles, creneaux)$ attribuée.

8. $Alldifferent(\{event_1, \ldots, event_n\})$
   Toutes les valeurs des *event* doivent être différentes, c'est-à-dire, toutes les séances se déroulent soit dans des salles différentes soit à des créneaux différents.

Ce second modèle gère donc les salles plus facilement et peut inférer plus rapidement grâce à l'arc-consistance sur les paires de créneaux et salles et à la contrainte ALLDIFF.

2.2.3 Tests

Nous avons réalisé des tests sur les 24 instances de l'ITC2007, l'objectif étant de satisfaire toutes les contraintes dures sans optimiser les contraintes souples. MZN 1 correspond au modèle 1 en minizinc, MZN 2, au modèle 2 en minizinc, pour la représentation en Gecode, nous avons représenté les paires $(salles, creneaux)$ de deux manières, en fonction des salles, G 2 % salles et en fonction des créneaux, G 2 % créneaux. Les -O0, -O2 et -O5 pour les modèles minizinc correspondent à la compilation des modèles, -O0, aucune optimisation à la compilation, -O2, deux passages de compilation et -O5 deux passages de compilation avec Gecode ainsi qu'une singleton arc consistency. Le temps maximum de recherche est fixé à 10 minutes.

Instance	MZN 1	MZN 2	G 2 % salles	G 2 % créneaux
3	18s (-O0)	3s (-O0/O2)	-	-
5	-	30s (-O2)	-	-
6	-	6s (-O0)	-	<1s
8	37s (-O2)	<1s (-O0)	<1s	<1s
11	-	<1s (-O0)	55s	-
15	-	16s (-O2)	<1s	<1s
16	<1s	<1s	<1s	<1s
17	<1s	<1s	<1s	<1s
20	-	1s (-O0)	-	-
21	-	5s (-O0)	<1s	-
24	89s (-O2)	2s (-O0)	-	-

FIGURE 2.2 – Récapitulatif des exécutions sur les instances de l'ITC2007.

Dans le tableau 2.2, nous représentons 11 des 24 instances qui peuvent être résolues. Les modèles Minizinc vont généralement un peu moins vite que les modèles Gecode mais trouvent plus souvent une solution. Vis-à-vis des tests sur la compilation en Minizinc, le temps de compilation en -02 et -05 est parfois relativement long, en revanche le temps de résolution ensuite est beaucoup réduit. Cependant la perte de temps lors de la compilation n'est pas rattrapée par le gain de temps lors de la résolution. Après ces tests, nous n'avons pas reproduit le modèle 3 de H. Cambazard et al. car celui-ci était utilisé après une recherche locale qui donnait un état valide de la solution avant une utilisation dans une LNS. Nous sommes ensuite passé à l'ITC2019 afin de travailler avec une représentation des données plus complète. Pour la suite nous utiliserons Minizinc avec une compilation -00, les modèles Minizinc étant beaucoup plus faciles à produire et le temps d'exécution raisonnable.

2.3 Contribution sur les données de l'ITC2019

Notre travail réalisé sur les données de l'ITC2019 se divise en 4 parties. Afin de travailler avec des données connues et plus aisées à gérer, nous avons généré un fichier XML sur les normes de l'ITC2019 partant des données de la L3 informatique. Nous avons travaillé sur le student sectionning tout d'abord avec une recherche locale puis avec un algorithme glouton. Pour résoudre les instances nous avons travaillé sur 3 types de modèles : des modèles PPC, des LNS pour optimiser les solutions des modèles PPC et des recherches locales. Nous nous sommes aussi attaqués au problème de réparation d'emplois du temps que nous détaillerons en dernière partie.

2.3.1 Données de la L3 informatique d'Angers

Les données de l'ITC2019 étant anonymisées et représentant des instances de plusieurs centaines d'étudiants, nous avons produit une instance avec des données connues, celles de la L3. Partant de la feuille de services des enseignants et du placement des cours de l'année passée, nous avons reconstitué une instance.

Celle-ci est constituée des cours d'Anglais (3 groupes), Bases de données (2 groupes de TD et 3 groupes de TP), Développement Web (3 groupes), Images de Synthèse (1 groupe), Production Automatisée de Documents (1 groupe), Programmation Fonctionnelle (2 groupes de TD et 3 groupes de TP), Programmation Logique ((2 groupes de TD et 3 groupes de TP), Qt Avancé (1 groupe) et Systèmes Intelligents (1 groupe). Soit 9 cours différents pour 18 étudiants types. Où chaque étudiant type correspond à un des triplets possibles (groupe TD et TP). Les élèves sont divisés en 3 profils, le tronc commun composé des cours d'Anglais, Bases de Données, Développement Web, Images de Synthèse, Programmation Fonctionnelle et Programmation Logique et les 3 options qui différencient les profils étant Systèmes Intelligents, Qt Avancé et Production Automatisée de Documents.

Nous y avons ajouté différents types de contraintes de répartition (Precedence, SameRoom, SameDays, SameAttendees,...) dures et souples.

Pour les créneaux, nous avons donné 1h20 à toutes les séances de cours, réparties sur 10 semaines. Au niveau des créneaux au sein de la semaine, ils sont tous disponibles pour tous les cours. Nous avons simplement donné une pénalité de 0 aux créneaux où les cours avaient lieu sur le planning originel et la pénalité augmentait en s'écartant de ce créneau.

Pour les salles, nous disposions de deux salles de CM (amphi A et B) et trois salles de TD/TP (H001, 2 et 3).

Le fichier XML peut être généré à la main ou grâce au site développé par Christophe Duveau pendant son stage de L3. La lecture sur le site permet de modifier l'instance, créer de nouveaux cours, regarder les emplois du temps des étudiants, des salles, des cours et des enseignants (grâce à une base de données informant quel enseignant dispense quel cours). Il permet aussi de regarder des statistiques telles que les types de cours dispensés/assistés, la répartition des cours dans la semaine et dans la journée. Nous avons aussi la possibilité de créer un cours, une salle, un étudiant et apporter diverses modifications.

2.3.2 Student Sectionning

Un ajout important pour l'ITC2019 est le student sectionning. Comme nous l'avons présenté plus tôt, il existe trois possibilités pour gérer les effectifs : avant la résolution, pendant la résolution et après la résolution. Nous avons travaillé sur deux possibilités, pendant la résolution avec le modèle PPC qui vous sera présenté dans la partie suivante et avant la résolution que nous allons vous présenter ici. Le student sectionning évite aux enseignants ou au personnel administratif de devoir gérer les groupes manuellement. Avec un nombre de profils différents de plus en plus important, il devient nécessaire d'utiliser des outils informatiques pour assister la gestion des étudiants afin d'arriver à un bon résultat rapidement et pouvant être modifié aisément si besoin (demande d'échange de la part d'un étudiant, modification du nombre de groupes, etc...). Pour ce faire le modèle nous fournit plusieurs informations. Pour chaque étudiant, les cours auxquels il assiste, pour chaque cours, les différentes séances possibles divisées dans les différentes sous-parties. Chaque séance possède une limite d'effectif (qui doit être inférieure ou égale aux capacités des salles possibles).

Pour répartir les étudiants dans les séances des différents cours, nous avons cherché, partant de l'organisation en sous-parties, à créer des structures au sein des cours. Nous nous sommes alors basés sur les limites de chaque cours et du nombre de séances par sous-partie.

Pour simplifier, prenons l'exemple du cours de Bases de données composé de un groupe de CM, CM1 (18 places), deux groupes de TD, TD1 (9 places) et TD2 (9 places) et trois groupes de TP, TPA (6 places), TPB (6 places) et TPC (6 places). Nous allons chercher à créer la structure du cours en fonction des places disponibles pour chaque séance et de l'organisation du cours. Chaque étudiant devra assister au CM1 donc il sera présent dans chaque structure du cours, ensuite il y a deux possibilités, soit assister au TD1 soit au TD2, les étudiants seront donc divisés en deux et pour finir, divisés en trois pour les TP. On constate donc que les étudiants devront être divisés en autant de groupes qu'il existe de séances différentes dans la sous-partie présentant le plus de séances. Mais comment constituer les groupes pour les TD ? Pour simplifier mais aussi forcer les étudiants à suivre les mêmes cours ensembles, dans ce genre de cas nous divisons les élèves du TPC en deux, une partie assistera au TD1 avec les élèves du TPA et l'autre au TD2 avec les élèves du TPB. Nous nous retrouvons donc avec le cours de base de données avec l'organisation suivante :

- Structure 1 : [CM1, TD1, TPA] (6 places disponibles)
- Structure 2 : [CM1, TD2, TPB] (6 places disponibles)
- Structure 3 : [CM1, TD1, TPC] (3 places disponibles)
- Structure 4 : [CM1, TD2, TPC] (3 places disponibles)

Nous avons donc pu, avec ce système, organiser les cours des différentes instances en structures. Il existe cependant des blocages sur certaines instances, les limites étant fixées à 0 étudiant sur certains cours. Nous n'avons pas su comment traiter ce cas. Est-ce dû à une erreur de saisie ou existe-t-il réellement des cours sans élèves ? Aucun élève ne pourra nous le dire...

Notons que si la personne qui créée un cours renseigne dans le XML les liens de parentés entre les séances, les structures suivront ce choix (et la génération en sera aussi simplifiée).

Lorsque l'on cherche à sectionner les étudiants, nous pouvons les représenter les inscrip-

tions de ceux-ci sous la forme d'un graphe biparti $(etudiants, cours)$. En analysant ce graphe nous remarquons des parties du graphes où il n'existe aucune connexion. En effet un étudiant en L1 Lettre n'a aucun cours en commun avec un étudiant de M2 Informatique par exemple. En revanche nous trouvons des composantes fortement connexe entre les étudiants de L1 Informatique et L1 Mathématiques. Si nous divisons notre graphe par blocs de composantes connexes, nous pouvons ainsi traiter les étudiants ayant des cours en commun plus aisément. Certains étudiants forment une biclique maximale car ils suivent les mêmes cours, on obtient alors les différents profils d'étudiants en réunissant ces bicliques en un seul point. Ainsi nous pouvons simplifier le sectionnement en travaillant avec ces différents profils.

Pour cette partie nous avons cherché à travailler avec des données de taille plus importante que les données de la L3. Nous avons donc cherché parmi les instances de l'ITC2019 une qui ressemblait à une instance de la faculté des sciences, à savoir "muni-fsps-spr17c". En voici les caractéristiques : 116 matières divisées en 650 séances et 395 étudiants que l'on peut diviser en 23 profils, que l'on peut rassembler en 7 blocs en fonction de leurs inscriptions (ci dessous, 2 des 7 blocs).

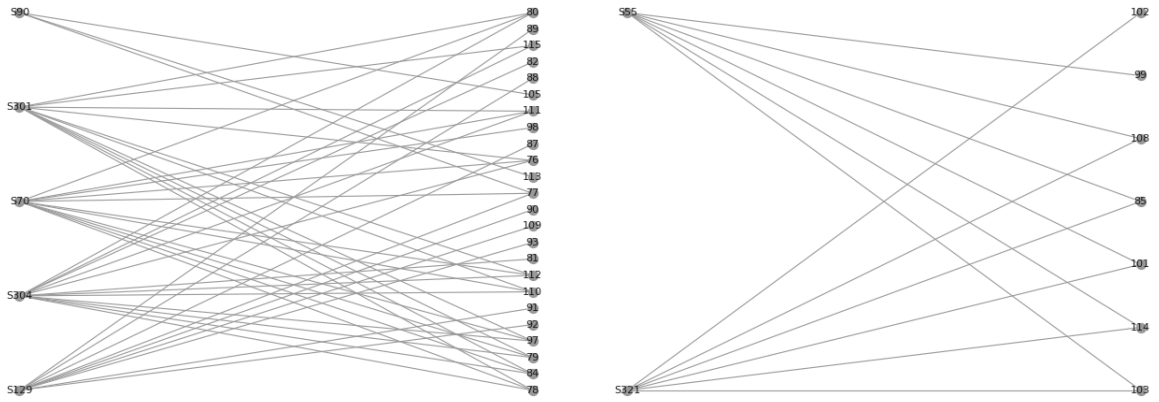

FIGURE 2.3 – Sur ces deux graphes nous retrouvons des graphes bipartis de deux blocs d'étudiants. À droite les cours et à gauche les profils d'étudiants inscrits aux cours.

Sur les graphes 2.3, sur le graphe de droite nous retrouvons donc 2 profils et 7 cours, la différence entre ces profils étant leur inscription au deux cours du haut du graphique. Ce graphe pourrait ressembler à une classe de M1 informatique avec 2 options au choix et un tronc commun. Dans le graphe de gauche nous avons 5 profils pour une vingtaine de cours. Celui-ci peut s'apparenter à une L2 informatique liée pour certains cours à la L2 mathématiques avec certains profils possédant moins de cours pour cause de redoublement ou autre. Ces blocs sont créés pour diviser l'instance en fonction des profils d'étudiants

Une fois ces blocs d'étudiants inter-connectés grâce aux cours et les structures des cours identifiés, nous devons placer ces étudiants dans les différentes structures des cours. Nous avons alors deux objectifs : bien diviser les étudiants dans les structures, par exemple sur un cours avec 5 structures de 10 places chacune et 42 étudiants inscrits, ne pas avoir 4 groupes de 10 étudiants et un groupe de 2 étudiants ; mais aussi de faire en sorte que les étudiants suivent leur cours avec les mêmes étudiants le plus souvent possible pour favoriser la cohésion

de groupe et simplifier la génération de l'emploi du temps en diminuant le nombre de conflits entre séances ne pouvant pas avoir lieu en même temps.

Nous avons testé trois méthodes pour répartir les étudiants. Tout d'abord, utiliser la partie gérant les inscriptions dans le modèle PPC et l'orienter vers un problème non plus de satisfaction mais d'optimisation. Malheureusement l'optimisation sur des données de cette taille était difficile sur le modèle produit. Nous nous sommes donc dirigés vers une recherche locale. Puis après un certain temps à bloquer dans des minimums locaux ou un temps d'exécution beaucoup trop long, nous avons cherché à initialiser la recherche locale avec un algorithme glouton qui a donné de bons résultats seul.

Recherche locale

La recherche locale [20] est une métaheuristique utilisée sur des problèmes d'optimisation. Partant d'un état du problème initial (aléatoire ou non), l'algorithme sélectionne une partie du problème puis réalise une modification telle qu'un échange de valeurs, afin d'obtenir un voisin de l'état courant. Tous les voisins possibles sont notés pour finalement choisir le meilleur voisin comme nouvel état avant de recommencer un cycle de sélection, modification, calcul de score, choix du nouvel état. Il existe différentes variantes de la recherche locale, l'utilisation d'une liste tabou permet d'éviter de toujours visiter les mêmes voisins et de diversifier la recherche. Le recuit simulé permet d'autoriser de prendre comme nouvel état un voisin dégradant le score afin de quitter des minimums locaux.

Algorithme 1 : Algorithme de recherche locale général (maximisation)

```
Données : état_courant
initialisation de l'état courant;
tant que nombre d'itérations maximal non atteint faire
    sélection d'un voisin de l'état courant;
    calcul du score du voisin;
    si score du voisin plus élevé que l'état courant alors
        le voisin devient l'état courant;
```

Dans notre cas, nous avons travaillé de deux points de vue, à savoir, organiser les étudiants dans les structures d'un seul cours puis organiser les étudiants dans les structures de tous les cours à la fois. Dans le premier cas, la progression n'était pas idéale, le score d'un état se basait sur le nombre de profils différents dans une structure mais n'était pas performant. De plus, au niveau d'un cours, l'algorithme n'a pas d'information sur les structures dans les autres cours, c'est pourquoi nous sommes passés à une gestion sur tous les cours à la fois.

Nous partons d'une situation initiale aléatoire, les étudiants inscrits à un cours sont répartis entre les différentes structures du cours (après quelques tests, placer les étudiants en fonction de leur profil ne répartissait pas correctement les étudiants dans toutes les structures). Nous créons une matrice carrée de rencontres de la taille du nombre d'étudiants.

Voisinage À chaque tour nous sélectionnons un étudiant. Pour chacun des cours où il est inscrit, on échange sa place dans sa structure avec un étudiant d'une autre structure. Toutes ces permutations entre tous les cours forment notre voisinage pour cet étudiant. Ce premier

principe de permutation fonctionne très bien et donne de bons résultats, cependant le temps de calcul de tous les voisins est beaucoup trop long pour être applicable sur des cours avec beaucoup de structures ou/et beaucoup d'étudiants. Afin de réduire l'espace de recherche, nous ne pouvons permuter qu'avec un seul étudiant de même profil, qu'avec une seule place vide par structure. Nous avons aussi essayé de réduire les permutations avec des étudiants d'autres profils s'il existait déjà une permutation avec ce profil, sans vraiment de succès. Soit le temps de calcul est beaucoup trop long, soit la qualité des voisins explorés est trop faible.

Score Lors du calcul du score, la matrice de rencontres est remise à zéro puis on parcourt tous les cours et toutes les structures des cours. Pour chaque étudiant se rencontrant, on incrémente leurs positions dans la matrice de rencontres. Le score est ensuite calculé en prenant la moyenne des $(nb\_etudiants/3)$ plus grandes valeurs sur chaque ligne de la matrice. Ce résultat permet de maximiser les cours où les étudiants rencontrent des étudiants du même profil. Afin d'accélérer le calcul du score lors de la recherche parmi les voisins, nous retirons les lignes et colonnes de la matrice correspondant aux étudiants permutés. Puis nous recalculons leurs rencontres. Malheureusement, nous sommes tout de même obligés de recalculer les moyennes sur chaque ligne car le nombre de rencontres sur les colonnes change systématiquement.

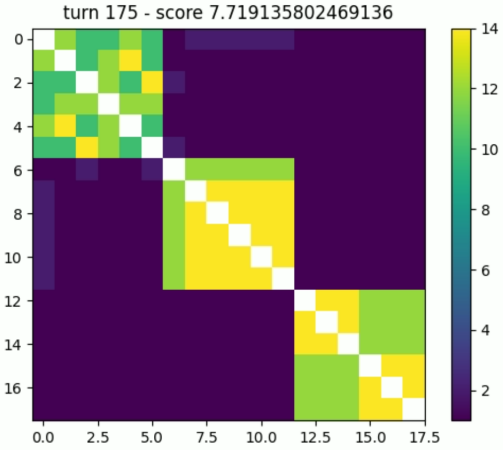

FIGURE 2.4 – Exemple de matrice de rencontres sur les données de la L3.

Chaque ligne et colonne représente les étudiants et l'intensité de la couleur le nombre de rencontres (jaune 14 rencontres dans les structures). On remarque bien 3 blocs correspondants aux trois profils différents mais le placement n'est pas encore idéal. Pour les étudiants de 0 à 11 il existe encore beaucoup de mélanges entre les groupes.

Cet algorithme fonctionne relativement bien, sur les petites instances. Sur les instances plus grandes (grand nombre de structures par cours et plus d'étudiants), le nombre de possibilités dans le voisinage rend le temps d'exploration et de calcul du score très long. C'est pourquoi nous avons cherché à réduire la taille du voisinage. Cette réduction de la taille n'était pas idéale, nous avions un dilemme : soit la recherche était trop longue car trop de possibilités, soit elle n'était pas efficace car les bons voisins n'étaient pas dans le voisinage.

Nous avons donc cherché à initialiser de manière plus efficace notre problème. Nous avions déjà testé de mettre les étudiants par profil sans faire une découpe claire en cas de structure trop petite ou d'étudiants de profils très variés. Nous avons donc réalisé un algorithme glouton pour résoudre ce problème.

Algorithme glouton

Notre algorithme se déroule en 4 étapes. Prenons un exemple avec 20 étudiants avec le profil A, 40 avec le profil B, 10 avec le profil C et 10 avec le profil D, à mettre dans 4 structures de 20 étudiants chacune.

1. placer les profils d'étudiant dans les structures si la taille est exactement égale à la taille de la structure.

   Les 20 étudiants A sont placés dans la première structure le reste n'est pas placé.

2. placer les étudiants dans les structures si la taille de la structure est supérieure au nombre d'étudiants.

   Les 40 avec le profil B ne passent pas, les 10 avec le profil C rentrent dans la 2ème structure puis les 10 avec le profil D rentrent dans la 2ème structure aussi.

3. découpe des blocs d'étudiants en fonction des tailles des groupes et des tailles de structures avec leurs PGCD.

   Il nous reste [20 (taille structure 3), 20 (taille structure 4), 40 (étudiants profil B)] le pgcd est de 20, on découpe donc les étudiants du profil B en deux groupes de 20. On place le 1er dans la structure 3 et le 2ème dans la structure 4.

4. remplacer les profils d'étudiants dans les structures par des vrais étudiants.

Nous pensions devoir faire une étape 5 où nous devrions réaliser une recherche locale pour affiner le résultat. Cependant, après ses 4 étapes les résultats étaient optimaux dans la majorité des cas. Dans le cas des très grandes instances du problème (plusieurs centaines d'étudiants à placer sur un très grand nombre de cours avec beaucoup de structures différentes) l'optimalité n'est en revanche pas garantie. L'algorithme glouton permet donc de gérer par construction ce que produisait la recherche locale. Cependant cette dernière étant dépassée par le nombre de voisin et le temps de calcul du score montre des difficultés à obtenir des bonnes solutions. De plus l'algorithme glouton profite au maximum des différents profils d'étudiants alors que la recherche locale devait principalement travailler avec des étudiants "réels".

Après la génération des groupes, les inscriptions des étudiants sont enregistrées dans un fichier XML sous la même forme que le fichier XML solution, sans les horaires et salles des séances, simplement la liste des séances et les étudiants inscrits pour chaque séance. Ainsi nous pouvons recharger les groupes et les modifier à loisir avant de lancer le solveur.

Comme vous pouvez le constater en figure 2.5, le modèle PPC réuni les étudiants par petits blocs de deux étudiants très régulièrement dans les mêmes séances et 4 autres assistant souvent aux mêmes séances. Alors que l'algorithme glouton permet de réunir tous les étudiants d'un même profil ensemble et réduit au maximum les rencontres avec des étudiants de d'autres profils. On voit donc bien les 3 profils distincts, avec une coupure en deux due aux cours avec deux groupes. Ce sectionnement permet de réduire au maximum les croisements entre étudiants différents, ce qui réduit ensuite la complexité de la résolution.

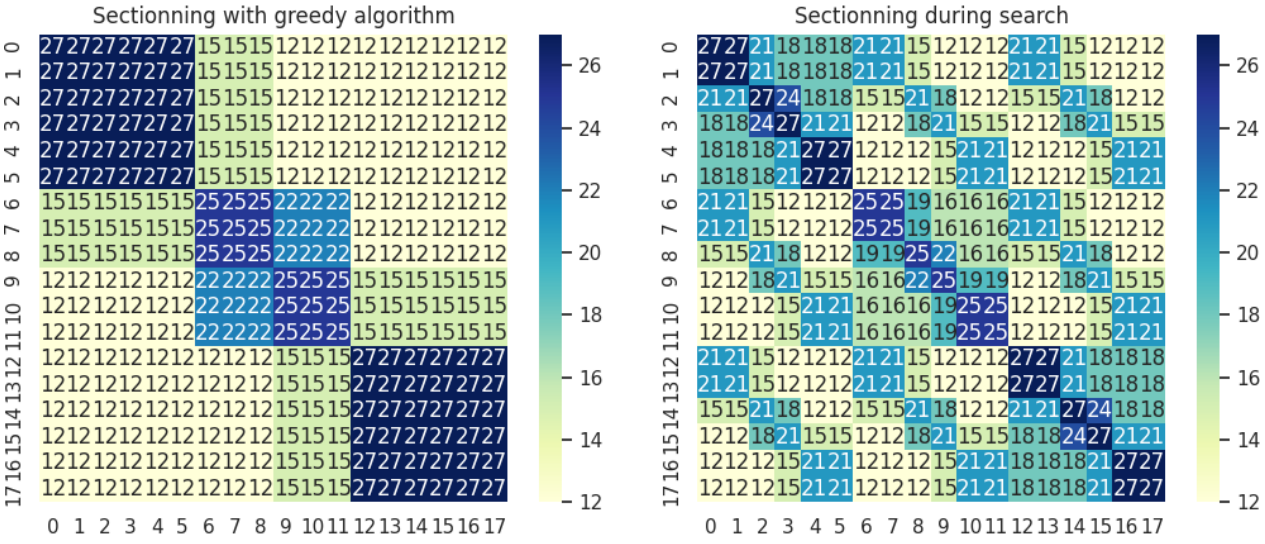

FIGURE 2.5 – Ces deux matrices représentent les rencontres entre les 18 étudiants de l'instance de la L3 informatique d'Angers.

Dans les matrices de la figure 2.5, à chaque fois qu'un étudiant est dans une séance avec un autre, on incrémente les cases leur correspondant. Ainsi, les parties claires correspondent à peu de rencontres (12) et en foncés au maximum de rencontres possibles étant donné les cours (27). La diagonale représente le nombre de rencontres d'un étudiant avec lui-même, ce qui indique le nombre de séances auquel il assiste. À gauche nous retrouvons la matrice des rencontres avec l'algorithme glouton, à droite en réalisant le student sectionning pendant la recherche avec le modèle PPC que nous présenterons dans la partie suivante.

2.3.3 Modèles

Nous allons maintenant vous présenter les modèles développés pendant ce stage. Nous avons tout d'abord réalisé un modèle PPC (Programmation Par Contraintes). Puis, afin d'optimiser les résultats du modèle PPC, nous avons implémenté une LNS (Large Neighborhood Search). Enfin, nous avons conçu une recherche locale.

Modèle PPC

La programmation par contraintes (PPC) permet de résoudre des problèmes combinatoires grâce à des modèles composés de variables, de domaines pour ces variables et de contraintes. Elle permet non seulement de résoudre des problèmes de satisfaction mais offre aussi des outils dédiés à l'optimisation.

Lors de l'ITC2007, la génération s'apparentait plutôt à un problème d'affectation, le cours était placé sur un créneau, dans une salle et pouvait être permuté avec d'autres créneaux, tous de même durée. Dans le cas de l'ITC2019, nous avons à faire à un problème d'ordonnancement. Le format des données est bien plus malléable, une séance peut en réalité représenter

plusieurs séances au fil des semaines. De plus, nous ne fonctionnons plus avec un système de créneaux dans la journée mais avec des heures de départ et des durées. Ces deux points complexifient beaucoup le problème. D'autant plus du fait que nous ne travaillons plus avec 5 jours de 9 heures (soit 45 créneaux) mais avec plusieurs semaines en quittant le système de créneaux.

Nous avons donc décidé de simplifier le problème en autorisant 5 créneaux par jour, les créneaux durant tous 1h20 (comme prévu dans Thélème), 8h-9h20, 9h30-10h50, 11h-12h20, 14h-15h20 et 15h30-16h50. Pas de créneau double de 2h40 ou plus prévu pour le moment.

Le modèle complet est disponible dans les sources du projet.

Inscriptions

Pour gérer la partie inscription des étudiants aux groupes, nous avons retranscrit l'organisation des cours de l'ITC2019. Les inscriptions des élèves sont représentées dans une liste d'ensembles de nombres entiers. Chacun de ces nombres représente un cours qui propose lui aussi un ensemble de configurations, qui eux-mêmes donnent les indices des sous-parties représentées de la même manière. Nous créons ensuite une variable de décision x_class, une matrice de taille $nb\_etudiants * nb\_sousparties$ de domaine $[0, ..., nb\_seances]$. Ainsi, chaque étudiant aura, s'il est inscrit à une séance, le numéro de la séance à l'indice de la sous-partie du cours auquel il est inscrit, sinon 0. Si un étudiant est inscrit à un cours alors il devra être inscrit à toutes les sous-parties d'une des configurations du cours. Nous limitons aussi le nombre d'étudiants inscrits par séance et forçons les inscriptions aux séances lorsqu'il y a des informations de parenté entre séances.

Ce fonctionnement permet de gérer la répartition des élèves pendant la recherche et est donc très flexible. En revanche il faut forcer la variable x_class a être assignée en début de recherche pour détecter rapidement les échecs évidents et éviter des calculs inutiles. Cette méthode donne en résultat le graphique 2.5.

Après avoir réalisé le student sectionning, nous avons pu modifier cette partie. Les étudiants étant déjà inscrits aux séances, nous avons pu retirer toute cette partie et remplacer les étudiants par des contraintes *SameAttendees*. Pour rappel, les contraintes *SameAttendees* permettent de gérer le fait que les enseignants ne peuvent pas dispenser deux séances à la fois et doivent avoir le temps d'aller d'une salle à l'autre quand les séances se suivent.

Contraintes de répartition

Pour les contraintes de répartition, nous utilisons une matrice de taille $nb\_contraintes * nb\_classes$ qui indique, dans l'ordre, l'indice des classes intervenants dans la contrainte ou 0. Dans un autre tableau, nous repérons le type de contrainte grâce à une chaîne de caractères et dans un autre la pénalité, si la contrainte est dure, 0 sinon la valeur de la pénalité. 16 des 19 contraintes de l'ITC2019 sont représentées (certaines simplifiées vis-à-vis de notre version avec 5 créneaux). Nous utilisons ensuite le nom de la contrainte pour choisir quel prédicat utiliser. Si la contrainte est dure, le prédicat doit être satisfait. Sinon nous multiplions la valeur de vérité du prédicat par le nombre de séances de la contrainte et par la pénalité pour obtenir le score. Pour les prédicats nous avons au maximum cherché à utiliser des contraintes globales telles que AllDifferent, AllEqual, ... permettant de réaliser des coupes dans l'arbre de recherche plus rapidement.

Placement des créneaux et salles

Pour les salles et créneaux, nous utilisons deux variables, x_room et x_time, chacune de taille $nb\_seances$. Pour renseigner les possibilités, chaque séance dispose d'un ensemble d'indices vers des temps $times_i$ et vers des salles $rooms_i$.

Ainsi $\forall i \in nb\_seances, \quad x\_room_i \in rooms_i \wedge x\_time_i \in times_i$.

Pour gérer l'occupation des salles aux créneaux choisis, nous avons utilisé les contraintes globales *disjonctive* et *cumulative*, la contrainte *disjunctive* permettant d'éviter les chevauchements sur les créneaux et la contrainte *cumulative* de respecter la contrainte de ressources sur les salles. Afin de simplifier le problème, nous divisons l'instance semaine par semaine, ce qui nous a permis de grandement accélérer la résolution.

Exploration de l'arbre de recherche et coût

Afin d'accélérer la résolution, lorsque nous recherchons avec la partie inscription gérée dans le modèle PPC, nous indiquons au solveur d'affecter en premier la variable x_class pour placer les étudiants dans les séances puis placer les créneaux et . Les coûts sont calculés pendant la recherche mais ne sont pas optimisés. Nous demandons seulement au modèle de satisfaire les contraintes dures, l'optimisation sur ce problème étant difficile. C'est pourquoi nous avons développé une LNS.

LNS

La LNS (Large Neighborhood Search) est une méthode de recherche utilisée pour explorer des espaces de recherche de très grande taille. Son fonctionnement consiste à bloquer une partie de l'espace de recherche afin de travailler de manière intensive sur les parties libres. Il faut donc une solution initiale valide (pour la PPC) puis bloquer des valeurs et en laisser d'autres libres, aléatoirement ou non. Partant de cette solution partiellement bloquée, nous recherchons avec le modèle une nouvelle solution améliorante. La recherche devient alors plus rapide et simple pour satisfaire et optimiser le problème. Voir schéma du fonctionnement en figure 2.6. Dans l'article de H. Cambazard [3], leur LNS se déroulait en trois étapes :

1. construction d'une solution sans contraintes dures non satisfaites avec leur recherche locale,
2. blocage des valeurs dans la LNS,
3. passage à leur modèle PPC pour optimisation puis répétition des étapes 2 et 3.

Dans notre cas, partant d'un planning valide généré par le modèle PPC, nous bloquons les affectations des salles et créneaux aléatoirement pour chaque séance. Puis nous donnons au modèle PPC cette instance partiellement immobilisée. Son objectif est alors d'améliorer le score précédant d'un certain pourcentage. Il existe alors trois cas de figures :

— soit le modèle nous rend une nouvelle solution du planning améliorant le score,
— soit il nous informe que le problème n'est pas satisfiable, c'est-à-dire qu'il ne peut pas, avec le degré de liberté sur les valeurs des variables, satisfaire la contrainte sur le score,
— soit il n'arrive pas à trouver une solution dans le temps imparti.

Peu importe le résultat, nous recommençons la phase de blocage des valeurs et de recherche jusqu'à ce que le modèle ne soit plus capable de donner de meilleurs résultats plusieurs fois de suite.

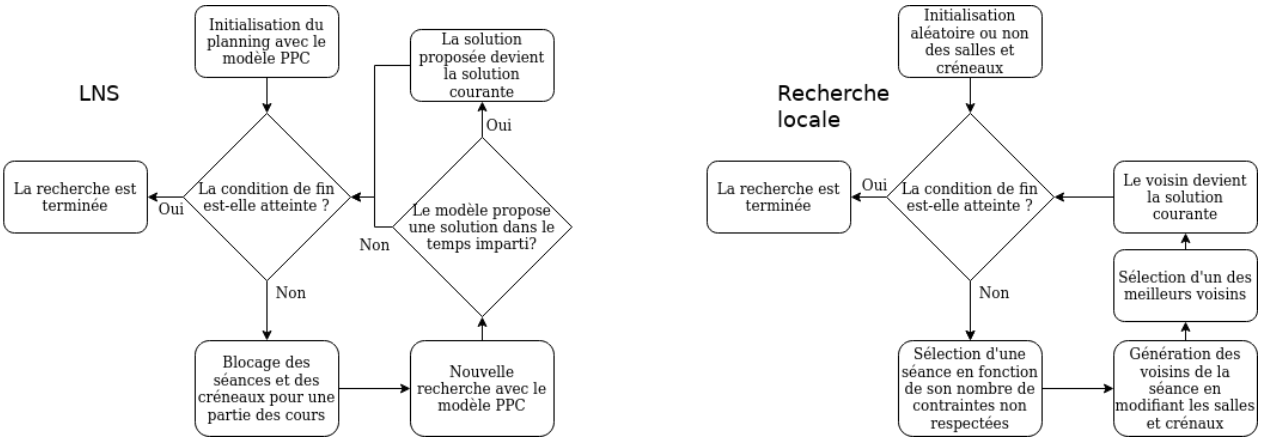

FIGURE 2.6 – Fonctionnement des algorithmes implémentés

Recherche locale

Pour la génération d'emploi du temps, notre recherche locale se déroule en deux parties. Dans un premier temps, nous initialisons l'emploi du temps aléatoirement ou par construction en satisfaisant principalement les contraintes strictes. Dans un second temps, nous effectuons une recherche locale avec tabou pour optimiser le résultat. Voir schéma du fonctionnement ci-dessus (figure 2.6).

Modélisation

L'emploi du temps est représenté par l'ensemble des séances $seances$. Chaque séance s possède une salle $s.room$ et un créneau choisis $s.time$, une liste de salles $s.rooms$ et de créneaux possibles $s.times$, la liste des contraintes $s.constraints$ auxquels la séance participe ainsi que différents compteurs pour le nombre de contraintes non respectées par la séance $s.room\_clashes$, $s.hard\_clashes$, $s.soft\_clashes$, $s.student\_clashes$.

Initialisation

La recherche locale commence avec un état initial. Celui-ci peut être aléatoire ou construit avec diverses méthodes telles que les algorithmes gloutons. Nous avons commencé avec une initialisation des créneaux et des salles aléatoire lors des tests sur les données de la L3. La recherche locale permettait de résoudre l'instance sans trouver le résultat optimum mais arrivait souvent à satisfaire toutes les contraintes strictes. Après avoir testé notre algorithme sur des instances de la taille d'une université, une initialisation autre que aléatoire était nécessaire, un nombre de contraintes strictes et souples très élevé en début de recherche étant difficile à résoudre. Notre nouvelle initialisation commence donc par une phase de recherche locale tabou. Le voisinage et le calcul du score sont les mêmes que pour la recherche locale utilisée ensuite, au détail près que nous ne prenons en compte que les séances dont le créneau et la salle sont placés. La recherche commence avec une seule séance placée et ajoute à chaque itération de l'algorithme une nouvelle séance. Pour les vingts premières séances nous ne réalisons qu'une itération puis passons à deux itérations et incrémentons le nombre d'itérations de un toutes les cent séances.

Voisinage

Pour générer le voisinage, nous sélectionnons le cours à modifier en fonction du nombre de contraintes dures non respectées. Plus un cours en possède, plus il a de chances d'être choisi (s'il n'est pas dans la liste tabou). Une fois le cours à modifier sélectionné, nous parcourons sa liste de salles et créneaux possibles et testons toutes les combinaisons.

$$Voisins(s) = \{(r, t) \mid r \in s.rooms, t \in s.times\}$$

Score

Le score est calculé grâce à trois types de contraintes :

- les contraintes sur les salles : à chaque fois que deux séances se déroulent dans la même salle en même temps, leur compteur $room\_clashes$ est incrémenté,
- les contraintes de répartition : pour chaque contrainte de répartition stricte non respectée, on incrémente le compteur $hard\_clashes$ des séances concernées ($soft\_clashes$ pour les contraintes de répartitions souples),
- les contraintes sur les étudiants : grâce au student sectionning, nous savons exactement combien d'étudiants assistent à quelles séances, si deux des séances en question se chevauchent ou ne permettent pas d'accéder de l'une à l'autre parce que les salles sont trop éloignées, alors le compteur $student\_clashes$ des séances est incrémenté.

Il existe aussi les compteurs $time\_penalties$ pour les pénalités sur les créneaux choisis et $room\_penalties$ pour les pénalités sur les salles choisies. Ainsi, avec les poids d'optimisation renseignés dans l'instance du problème, w_time pour les choix de créneaux, w_room pour les choix de salles, $w\_distribution$ pour les contraintes de répartition et $w\_students$ pour les contraintes sur les étudiants :

$$score_{hard} = \sum_{s}^{seances} s.room\_clashes + s.hard\_clashes$$

$$\begin{aligned} score_{soft} = \sum_{s}^{seances} & s.soft\_clashes * w\_distribution + \\ & s.student\_clashes * w\_students + \\ & s.time.penalty * w\_time + \\ & s.room.penalty * w\_room \end{aligned}$$

$$score_{total} = score_{hard} * 10^5 + score_{soft}$$

Pour rappel, dans le format de la compétition, les contraintes sur les étudiants (le fait qu'ils puissent assister à toutes leurs séances) est une contrainte souple. Son poids est en revanche généralement élevé.

Lors de l'analyse du voisinage, la seule modification étant le créneau et la salle d'une seule séance, nous ne calculons pas tous les points cités ci-dessus, mais simplement la différence apportée sur chacun des points pour la séance traitée.

Nous avons aussi ajouté un système fonctionnant comme un recuit simulé, permettant d'être plus souple sur le score minimal toléré pour qu'une solution puisse être acceptée.

$$seuil = 1 + ((turn_{max} - turn) * 4/turn_{max})$$

Dans le cas de l'initialisation tabou, $turn_{max}$ est égal au le nombre d'itérations de l'algorithme. Pour la recherche locale, $turn_{max} = 3 * nb\_seances$ après $turn_{max}$ itérations le seuil de tolérence passe à 1.

Algorithme 2 : Recherche locale ITC2019

Données : séances : l'ensemble des séances de l'instance
tabou : booléen, vrai si initialisation tabou
voisins(séances, séance) : retourne la liste des voisins possibles sur la séance
applique(séances, voisin) : applique le voisin à la solution
score_voisins = $\emptyset$: ensemble des couples $(voisin, score)$
s : solution courante du planning
si *tabou* **alors**
  s = initialisation_tabou(séances);
sinon
  s = initialisation_aléatoire(séances);
tant que *nombre de tour maximal non atteint* **faire**
  séance_à_modifier = sélection_séance(séances);
  pour *v in voisins(séances, séance_à_modifier)* **faire**
    applique(s, v);
    scores_voisins <- (v,score(v));
  score_min = min_score(scores_voisins);
  v =
   sélection_aléatoire($\{v \mid (v, sc) \in scores\_voisins\ ,\ sc <= score\_min * seuil\}$);
  applique(s, v);

Tests et résultats

Nous allons maintenant vous présenter les tests de nos modèles. La durée des tests comprend uniquement le temps de recherche, le chargement de l'instance et le calcul des groupes étant déjà réalisés. Dans un premier temps nous analyserons les résultats du modèle PPC, de la LNS et de la recherche locale sur les données de la L3. Puis nous traiterons les tests de la recherche locale sur des instances de l'ITC.

Données de la L3

Conditions des tests :

- pour le modèle PPC : le résultat étant déterministe, nous ne réalisons qu'une exécution, ('SS' : groupes gérés par le student sectionning, 'FREE' : groupes gérés par la PPC)
- pour la LNS : 50 itérations. La recherche s'arrête si le score ne bouge plus après 10 itérations ou s'il dépasse les 200 itérations,
- pour la recherche locale (LS) : 50 itérations avec et sans l'initialisation tabou et avec et sans le recuit simulé. L'algorithme s'arrête lorsqu'il stagne depuis 20 tours ou lorsque le temps d'exécution dépasse 2h. ('RAND' : initialisation aléatoire, 'TABOU' : initialisation tabou, 'RC' : recuit simulé)

Méthode	LS				LNS		PPC	
	RAND	TABOU	RAND RC	TABOU RC	SS	FREE	SS	FREE
temps min	**0.34**	0.49	0.46	0.57	154	263		
temps moyen	**0.43**	0.52	0.66	0.80	491	658	2	7
temps max	**0.52**	0.61	0.86	0.97	875	1149		
strict min	**0**	**0**	**0**	**0**				
strict moyen	2.38	0.72	0.64	0.14	**0**	**0**	**0**	**0**
strict max	9	3	6	4				
souple min	61	39	110	35	33	**28**		
souple moyen	330.92	**54.22**	399.70	128.70	133.88	129.44	893	1017
souple max	960	**73**	1030	693	368	654		

TABLE 2.1 – Tableau résumant les 50 exécutions sur l'instance de la L3. Les temps sont en secondes.

Dans le tableau 2.1, un récapitulatif des résultats des tests effectués. Comme vous pouvez le constater, niveau temps d'exécution, la LS RAND est généralement plus rapide, le fait d'ajouter le recuit augmente légèrement le temps de recherche. Les LNS sont plus de cent fois moins rapides dans les meilleurs cas. On remarque tout de même que réaliser les groupes avant la recherche accélère celle-ci. Niveau score, les LNS et modèles PPC ont l'avantage de n'avoir aucune contrainte stricte. Ils présentent aussi les deux meilleurs résultats en termes de score souple minimum suivis de près par la LS TABOU RC. En moyenne la LS TABOU est la meilleure pour ce qui est du score souple. Cependant, étant donné qu'il reste parfois des contraintes dures non satisfaites, le résultat n'est pas idéal.

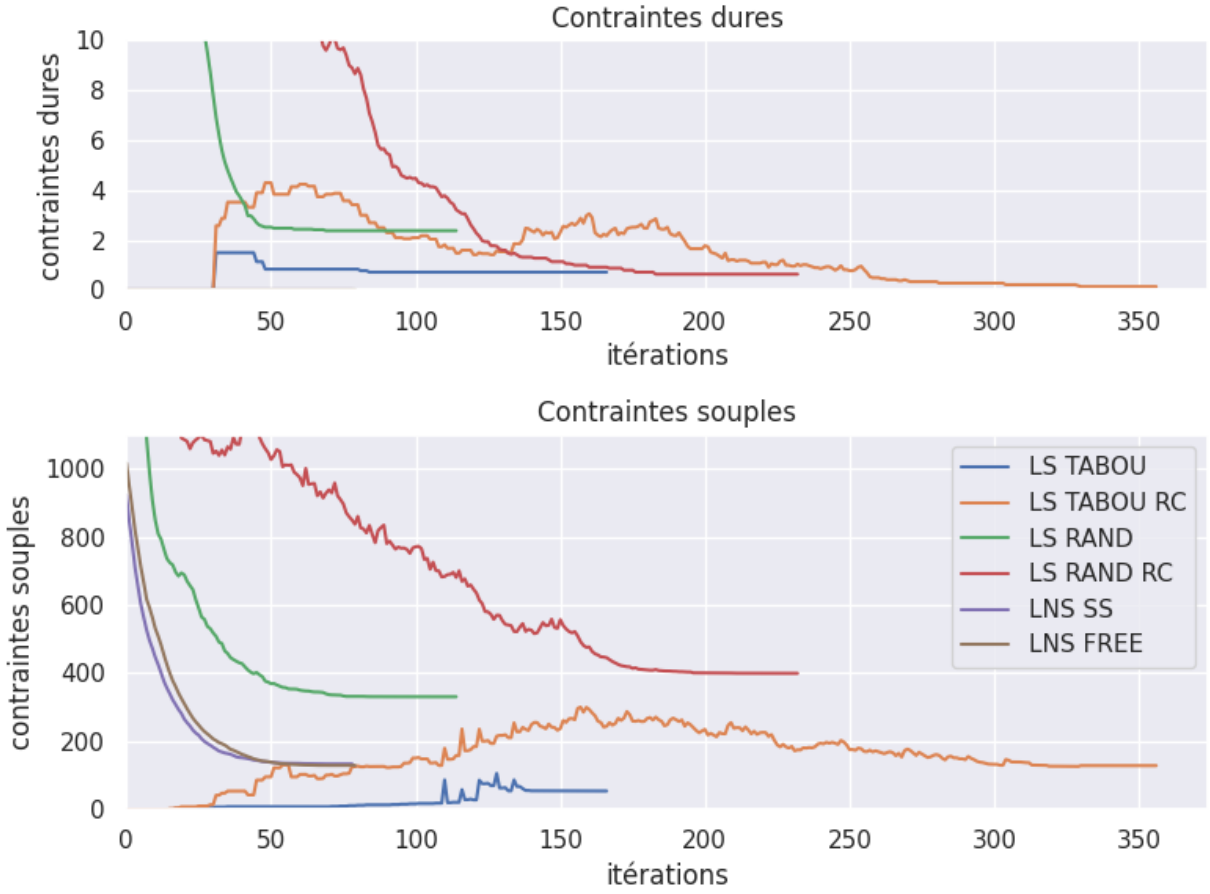

FIGURE 2.7 – Voici un graphique des 50 exécutions moyennes pour chaque méthode. Les recherches locales avec initialisation aléatoire commencent à un peu plus d'une centaine de contraintes strictes et presque 3000 souples.

Sur les graphiques 2.7, un résumé des 50 exécutions de test. En vert, la LS RAND s'améliore très rapidement sur les deux types de contraintes mais tombe dans des minimums locaux rapidement. Utiliser le recuit simulé, en orange et rouge permet d'éviter les minimums locaux jusqu'à un certain point. Les LNS, violet et marron, n'ont aucune contrainte dure tout au long de la recherche mais stagnent assez rapidement à une centaine de contraintes souples non respectées. Les LS TABOU, bleu et orange, voient leur nombre de contraintes augmenter au fil de la recherche lorsque de nouvelles séances s'ajoutent à l'espace de recherche. Cependant, sans le recuit simulé, orange, la recherche ne parvient pas à retirer toutes les contraintes dures pour s'orienter vers optimum global. La TABOU RC, orange, fonctionne très bien pour les contraintes dures mais à plus de mal à corriger les contraintes souples. On peut cependant supposer qu'elle tend vers le minimum possible pour les contraintes souples dans le cas où toutes les contraintes dures sont satisfaites. Alors que la TABOU, bleu, obtient de meilleurs résultats sur les contraintes souples mais ne parvient pas à satisfaire toutes les contraintes dures.

Instances ITC

Pour les tests sur les données de l'ITC, nous n'avons pas testé toutes les méthodes. En effet, les LNS et PPC ne gérant que des instances avec 5 créneaux de taille unique, ceux-ci ne s'adaptent qu'aux données générées pour la L3. Nous n'avons pas non plus testé la LS RAND car celle-ci n'est pas assez efficace, en dehors du temps, sur les données de la L3 qui sont relativement simples comparées aux instances de la taille d'une université comme dans le cas de l'ITC.

Dans le tableau 2.2, un récapitulatif des données des instances de l'ITC que nous avons traitées. Lorsqu'il n'y a aucun étudiant dans le récapitulatif, cela signifie que l'instance est traitée comme un problème de Curriculum based course timetabling où les étudiants pourront s'inscrire après génération de l'emploi du temps.

Instance	Étudiants	Salles	Cours	Séances	Contraintes de répartition
tg-fal17	0	24	36	711	503
tg-spr18	0	25	44	676	426
iku-spr18	0	209	1290	2782	3488
lums-spr18	0	74	313	487	518
lums-fal17	0	98	328	502	599
bet-sum18	0	47	48	127	144
lums-sum17	0	63	19	20	2
iku-fal17	0	215	1206	2641	2903
muni-fspsx-fal17	1152	34	515	1623	1361
agh-ggis-spr17	2116	45	272	1852	2690
muni-fi-spr17	1469	36	186	516	710
wbg-fal10	19	8	21	150	0
muni-pdfx-fal17	5651	87	1635	3717	3501
yach-fal17	821	34	91	417	645
muni-fsps-spr17c	395	30	116	650	709
pu-cs-fal07	2002	14	44	174	102
muni-pdf-spr16	3443	84	881	1515	1012
agh-ggos-spr17	2254	85	406	1144	1689
l3-angers	*18*	*6*	*9*	*52*	*97*

TABLE 2.2 – Résumé des caractéristiques des intances.

Les tests n'ont pas pu être réalisés sur un grand nombre d'exécutions mais sur uniquement 5 exécutions, faute de temps. Dans le tableau des tests, les temps sont en secondes et les nombres représentent la moyenne de toutes les exécutions.

Méthode	RAND RC			TABOU RC			TABOU		
	temps	strict	souple	temps	strict	souple	temps	strict	souple
tg-fal17	16	1227	25730	25	1197	25230	**6**	**1014**	**23358**
tg-spr18	26	1185	37484	41	1185	53146	**11**	**1147**	**9892**
iku-spr18	385	2100	115949	1049	1343	124119	**379**	**780**	**76427**
lums-spr18	23	16	615	23	23	456	**8**	**0**	**436**
lums-fal17	26	**15**	2126	30	51	1580	**10**	79	**1621**
bet-sum18	1	**24**	3887	0	38	4022	**0**	32	**3183**
lums-sum17	**0**	**0**	**4**	**0**	**0**	**4**	**0**	**0**	**4**
iku-fal17	**241**	1540	101066	637	1308	92944	283	**572**	**60981**
muni-fspsx-fal17	783	733	**109515**	1360	640	128760	**350**	**495**	116693
agh-ggis-spr17	**775**	1761	97481	2028	1477	108933	1010	**898**	**93354**
muni-fi-spr17	57	96	19896	86	**68**	**19298**	**32**	111	19345
wbg-fal10	1	**0**	96	**0**	**0**	163	**0**	4	**55**
muni-pdfx-fal17	7200	7554	984410	7402	3405	821973	**6846**	**1004**	**322874**
yach-fal17	29	**20**	11218	39	27	11754	**13**	36	**10434**
muni-fsps-spr17c	177	210	54128	258	171	**43994**	**117**	**168**	61156
pu-cs-fal07	1	**0**	1780	1	3	1760	**0**	2	**1462**
muni-pdf-spr16	458	231	152379	726	114	130055	**367**	**76**	**107664**
agh-ggos-spr17	139	422	26105	153	**256**	26348	**69**	330	**18150**

TABLE 2.3 – Récapitulatif des 5 exécutions sur les instances de l'ITC. Les temps sont en secondes, les nombres sont les moyennes des exécutions.

Dans le tableau 2.3, un résumé des exécutions sur les instances de l'ITC. Au niveau du temps, la seule instance impactée par la limite de deux heures est muni-pdfx-fal17. C'est aussi l'instance avec le plus d'élèves, de cours, de séances et de contraintes de répartition avec un nombre de salles qui semble limité comparé aux autres instances. La recherche locale avec initialisation tabou est généralement plus rapide que les autres méthodes.

Du point de vue des contraintes, nous n'arrivons malheureusement pas à des résultats parfaits. Seules les plus petites instances nous permettent de trouver une solution valide, toutes les contraintes dures respectées, comme le demande le concours.

L'initialisation aléatoire fonctionne mieux que ce à quoi nous nous attendions et l'initialisation tabou avec recuit moins bien vis-à-vis des résultats obtenus sur l'instance de la L3.

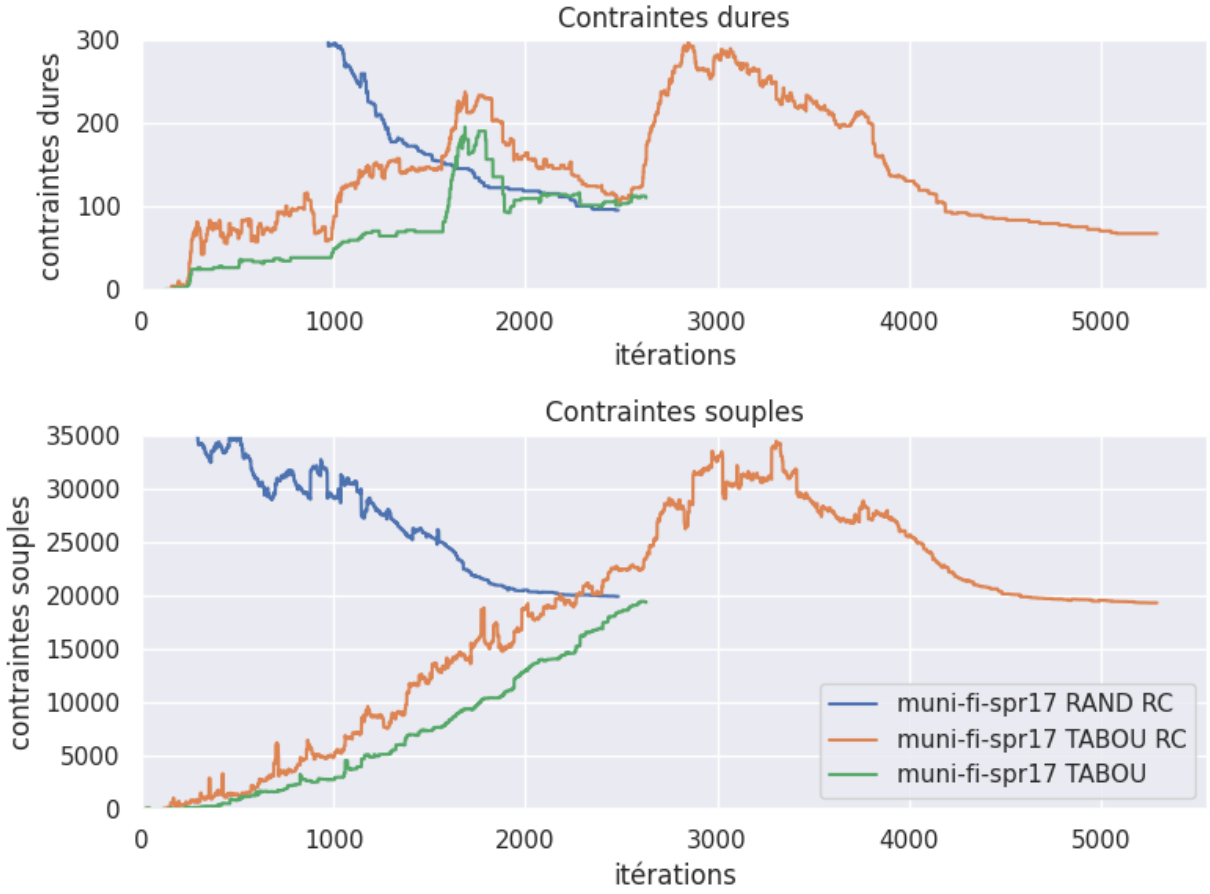

FIGURE 2.8 – Exemple d'exécution avec l'instance muni-fi-spr17. La RAND RC commence à environ 5000 contraintes dures et presque 100000 souples.

En figure 2.8, les courbes moyennes des trois méthodes utilisés. La RAND RC commence avec beaucoup de contraintes non respectées et tend rapidement vers un minimum local. Dans le cas des initialisation TABOU, les courbes augmentent au fur et à mesure de l'ajout des séances à la recherche, avec des pics lors de l'ajout d'un ensemble de séances très contraintes. Sans le recuit, la courbe tend rapidement vers un minimum. Une fois l'initialisation terminée, sans le recuit, la recherche stagne rapidement, avec le recuit, le seuil de tolérance du recuit repart à zéro et autorise à nouveau des solutions dégradantes qui font rapidement augmenter le score pour diminuer lentement vers un minimum local. Le recuit avec tabou ne permet pas d'éviter totalement les minimums locaux comme dans le cas des données de la L3. Cette fois c'est la LS TABOU qui obtient les meilleurs résultats. Repartir à zéro avec le recuit après une initialisation tabou augmente de manière très significative le score, il serait probablement plus pertinent de réduire le seuil après l'initialisation.

2.3.4 Réparation d'emploi du temps

Dans le cadre de ce stage, nous avons aussi abordé la problématique de réparation. La réparation consiste à modifier l'emploi du temps après sa construction car un évènement imprévu survient. Nous avons traité trois cas, l'annulation d'horaire pour une séance, une salle devenant indisponible pour certains horaires et l'échange de séances entre deux enseignants. Ces trois évènements sont relativement basiques mais couvrent beaucoup de cas utiles de réparation. Les différentes réparations sont nombreuses mais surtout, font intervenir des inconnues du problème initial. En effet, après qu'un enseignant propose des créneaux et préférences pour ses séances, il ne redonne pas ses disponibilités. Au moment de la réparation, les disponibilités ne sont probablement plus d'actualité, nous arrivons donc à un problème avec des informations incomplètes. De plus, dans la pratique, bien peu d'enseignants voudront mettre à jour régulièrement leurs disponibilités. Aussi, modifier le planning d'un enseignant car un autre ne peut plus assurer son cours peut être source de conflits. Ainsi, nos réparations se présentent plus comme un outil d'aide à la décision qu'un agent autonome. Du côté du XML, une séance peut en représenter plusieurs en réalité. Pour simplifier la réparation, il faudrait développer un système qui, lors de la génération, utilise des blocs de séances pour obtenir un planning répétitif, puis pour la réparation, divise ces blocs en séances uniques.

Annulation de l'horaire prévu pour une séance

Ce cas arrive lorsqu'un horaire pour une séance est supprimé, l'enseignant est finalement indisponible ou les élèves ont un évènement imprévu qui s'ajoute à leur planning. Dans l'état actuel, nous traitons les différents cas en fonction des horaires que l'enseignant a donnés avant la création du planning, nous testons tous les horaires et les proposons en liste triée par rapport au nombre de contraintes non respectées qui sont ajoutées au planning. L'enseignant peut alors choisir sur quel créneau il veut déplacer sa séance. Dans le cas où le créneau chevauche une autre séance, nous pouvons répéter l'action pour trouver un autre placement. Dans un cas réel, l'enseignant modifiera probablement ses disponibilités, celles-ci ayant très probablement changées depuis la création de l'instance. Puis, si aucun créneau n'est disponible sans fortes contraintes avec les disponibilités données, il regardera dans la liste des propositions celle qui apporte le moins de contraintes et demandera à l'enseignant impacté par le changement s'il est possible pour lui de décaler son cours.

Salle indisponible pour certains horaires

Dans le cas où une salle se retrouve indisponible pour cause de travaux ou autre problème, nous récupérons les séances impactées par ce changement de salle et relançons une recherche en laissant uniquement à ces séances la liberté d'être modifiées. Encore une fois, au moment de la réparation, il sera possible d'ajouter des salles possibles pour les séances impactées. S'il n'y a pas de solution viable il faut alors relâcher d'autres séances pour satisfaire le problème.

Échange de séances entre deux enseignants

Il peut arriver que deux enseignants échangent une séance de leurs cours car l'un est indisponible ou pour diverses raisons. Dans ce genre de cas, nous modifions donc les contraintes SameAttendees qui concernaient les séances en question, puis lançons une nouvelle recherche en bloquant les autres séances pour éviter au maximum les modifications. S'il n'y a pas de solution, il faudra alors rajouter des créneaux ou modifier d'autres cours.

Chapitre 3

Perspectives et améliorations

Il reste beaucoup de choses à améliorer sur ce projet mais aussi beaucoup de développements à réaliser pour créer une application utilisable par le personnel de l'université pour gérer les plannings. Il faudra créer une base de données plus complète pour gérer les cours et enregistrer les cours des années précédentes ainsi que mémoriser les parcours des étudiants. Nous pourrions imaginer ensuite un système capable de prédire les inscriptions dans chaque cours afin de prévoir au plus tôt le nombre de groupes nécessaires. Le site devra aussi être capable de proposer de manière ergonomique une création de cours et de séances ainsi qu'une gestion facilitée des enseignants donnant cours. La gestion des préférences des étudiants lors de l'inscription serait aussi un bonus intéressant. Il permettra de faciliter l'acceptation d'un étudiant sur un cours et la répartition des étudiants pour les enseignants.

Du point de vue de la modélisation, la représentation de l'ITC2019 est extrêmement complète et nous permet de gérer librement notre instance. L'absence des enseignants dans la modélisation peut être géré grâce à l'application qui produira le fichier XML et leur préférences d'horaires se feront grâce aux pénalités et les contraintes de répartition qu'ils pourront donner pour leurs séances. Il faudra cependant peut-être chercher à bloquer certaines possibilités, comme le placement des créneaux devant être des multiples d'une heure vingt (sauf exception). Pour représenter les évènements tel que le Campus Day ou les portes ouvertes, il faudrait étudier la possibilité de modifier les instances de manière à diviser les blocs de séances impactées par l'évènement. Par exemple, si le Campus Day se déroule un jeudi et qu'une séance est prévue pour avoir lieu soit tous les vendredis, soit tous les jeudis, il faudrait éviter de lui interdire tous les jeudis (comme c'est fait à l'heure actuelle), mais proposer à l'enseignant d'ajouter une séance 'volante' à placer dans le cas où le choix se porte sur le jeudi. Lors de la pandémie et de la mise en place des cours à distance, l'université devait gérer les connexions aux salles à distance pour alléger le nombre de connexions au serveur. Dans le cas d'un autre confinement ou d'une augmentation du nombre de cours à distance, il faudra probablement prendre en compte, dès la création de l'instance, une limite de connexions simultanées pour les salles à distance.

Du point de vue du solveur, il reste beaucoup à améliorer, la recherche locale arrive trop rapidement dans des minimums locaux et le modèle PPC ou la LNS auront beaucoup de mal à gérer les instances de l'université avec uniquement cinq créneaux par jour et l'impossibilité, à l'heure actuelle, de faire des créneaux de 2h40. De plus la mise à l'échelle sur des données plus importantes sera sûrement difficile en terme de temps d'exécution. Pour améliorer la recherche locale, nous avons pensé à diviser l'instance, à la manière de la division réalisée lors du student sectionning mais cette fois en prenant aussi en compte les enseignants (via les contraintes SameAttendees). Ceci pourrait permettre de diviser la difficulté en résolvant l'instance partie par partie. L'inconvénient serait en revanche les salles partagées par plusieurs départements. Nous pouvons alors imaginer laisser les séances concernées ouvertes à la modification dans le cas d'un partage de la même salle. Les tests réalisés sur les instances d'ITC ont montré à quel point notre solveur doit être améliorer, nous ne pouvons le comparer au maigre état de l'art existant à l'heure actuelle car ils demandent un score de contraintes dures de 0. Le travail réalisé sur la recherche locale ayant eu lieu relativement tard lors du stage, nous n'avons pas eu le temps de chercher des paramètres idéaux pour la taille de la liste tabou, de la fonction gérant le recuit simulé ou la condition de fin, qui était bonne pour l'instance de la L3 mais probablement trop restrictive pour des instances de taille plus importante. Développer une autre forme de voisinage, comme un échange de salles entre cours pourrait aussi permettre d'explorer l'espace de recherche plus aisément.

Du point de vue de la réparation, les réparations proposées sont encore très sommaires. Il faudra bien entendu développer cet aspect avec des méthodes permettant de proposer directement des modifications facilement en modifiant le moins de cours possibles. Cependant, la modification restera très dépendante des informations que les enseignants voudront bien laisser au solveur.

Conclusion

L'objectif de ce stage était d'étudier le problème d'emplois du temps en prévision de l'arrivée du projet Thélème, en recherchant à utiliser des méthodes exactes et des méthodes approchées. Grâce à l'ITC2007 nous avons pu partir avec un grand ensemble d'articles et l'ITC2019 nous a offert une modélisation très souple et des instances complètes. Les concours durant respectivement six mois et presque un an et demi, nous n'avons pas obtenu des résultats parfaits en six mois mais il nous reste des pistes à explorer pour les améliorer.

Pour les modèles implémentés, ceux-ci ne sont pas encore assez performants, côté méthodes exactes comme approchées. Dans l'état de l'art, les méthodes exactes sont rarement utilisées seules, il existe très généralement une recherche locale, ou autre, en parallèle, pour aider à optimiser. Il pourrait être intéressant de regarder du coté des méthodes SAT si les modèles peuvent être plus performants qu'en PPC. Pour les méthodes approchées, sur celles implémentées, plus de travail sur les hyper-paramètres permettrait probablement de beaucoup améliorer les résultats. Regarder du côté des algorithmes génétiques / mémétiques pourrait aussi permettre d'explorer plus aisément l'espace de recherche.

Pour le problème en lui-même, comme le disent Michael W. Carter et al. dans la conclusion de leur article[17], nous recherchons au maximum à obtenir un planning "parfait" sur l'aspect qualité. Mais, avec du recul, ils ont remarqué que dans l'utilisation quotidienne, certes la qualité de l'emploi du temps a joué un rôle très important mais que leur solveur réponde en temps réel était le point primordial.

Les emplois du temps sont, de base, un problème très politique et pouvant être source de conflits au sein d'une équipe, par exemple si les mêmes personnes ont toujours des créneaux qui se suivent et n'ont jamais cours le vendredi ou à l'inverse des créneaux très espacés ou simplement jamais aux créneaux préférés. La gestion de la génération est donc très délicate, encore plus dans le cas de la réparation car les enseignants ayant renseignés le plus de créneaux de disponibilités se retrouveront probablement plus souvent à voir leurs créneaux modifiés. C'est pourquoi il sera important que la réparation ne soit pas totalement automatique et prenne au maximum en compte les avis des enseignants impactés par la modification.

D'un point de vue personnel, ce stage m'a permis de travailler plus en profondeur sur Minizinc et de découvrir Gecode. J'ai aussi pu découvrir les tests d'intégrations et différentes méthodes d'optimisation en Python, utiliser PyPy accélérant significativement la vitesse d'exécution, car passer de presque 3 jours à 14h de tests sur les données de l'ITC était un plus. Ce stage m'a aussi conforté dans mon choix de poursuivre dans le monde de la recherche, en particulier pour travailler sur différentes métaheuristiques.

Bibliographie

[1] Lewis, R., Paechter, B. & Mccollum, B. Post enrolment based course timetabling : A description of the problem model used for track two of the second international timetabling competition. *Cardiff University, Cardiff Business School, Accounting and Finance Section, Cardiff Accounting and Finance Working Papers* (2007).

[2] Di Gaspero, L., Mccollum, B. & Schaerf, A. The second international timetabling competition (itc-2007) : Curriculum-based course timetabling (track 3) (2007).

[3] Cambazard, H., Hebrard, E., O'Sullivan, B. & Papadopoulos, A. Local search and constraint programming for the post enrolment-based course timetabling problem. *Annals of Operations Research* **194**, 111–135 (2012). URL `https://doi.org/10.1007/s10479-010-0737-7`.

[4] Atsuta, M., Nonobe, K. & Ibaraki, T. Itc2007 track 2 : An approach using general csp solver (2008).

[5] Chiarandini, M., Fawcett, C. & Hoos, H. A modular multiphase heuristic solver for post enrolment course timetabling (2008).

[6] Nothegger, C., Mayer, A., Chwatal, A. & Raidl, G. Solving the post enrolment course timetabling problem by ant colony optimization. *Annals OR* **194**, 325–339 (2012).

[7] Müller, T. Itc2007 solver description : A hybrid approach. *Annals of Operations Research* **172**, 429–446 (2008).

[8] Lemos, A., Monteiro, P. T. & Lynce, I. Minimal perturbation in university timetabling with maximum satisfiability. EasyChair Preprint no. 2821 (EasyChair, 2020).

[9] Jaradat, G. & Ayob, M. A comparison between hybrid population-based approaches for solving post-enrolment course timetabling problems. *International Journal of Computer Science and Network Security* **11**, 116–123 (2011).

[10] Jat, S. N. & Yang, S. A hybrid genetic algorithm and tabu search approach for post enrolment course timetabling. *Journal of Scheduling* **14**, 617–637 (2011). URL `https://doi.org/10.1007/s10951-010-0202-0`.

[11] Rezaeipanah, A., Abshirini, Z. & Zade, M. Solving university course timetabling problem using parallel genetic algorithm. *Journal of Scientific Research and Development* **7**, 5–13 (2019).

[12] Taylor, L. A. Local search methods for the post enrolment-based course timetabling problem. In *Local Search Methods for the Post Enrolment-based Course Timetabling Problem* (2013).

[13] Chiarandini, M., Fawcett, C. & Hoos, H. A modular multiphase heuristic solver for post enrolment course timetabling. *Track Two of the ITC2007* (2008).

[14] Bashab, A. *et al.* A systematic mapping study on solving university timetabling problems using meta-heuristic algorithms. *Neural Computing and Applications* (2020). URL `https://doi.org/10.1007/s00521-020-05110-3`.

[15] Tomáš Müller, Z. M., Hana Rudová. University course timetabling and international timetabling competition 2019. *In Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling* (2018).

[16] Müller, T. & Murray, K. Comprehensive approach to student sectioning. *Annals of Operations Research* **181**, 249–269 (2010).

[17] Carter, M. W. A comprehensive course timetabling and student scheduling system at the university of waterloo. In Burke, E. & Erben, W. (eds.) *Practice and Theory of Automated Timetabling III*, 64–82 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2001).

[18] Schindl, D. Student sectioning for minimizing potential conflicts on multi-section courses. *Proceedings of the 11th International Conference of the Practice and Theory of Automated Timetabling (PATAT 2016)* pp. 327–337 (2016). URL `http://hesso.tind.io/record/1781`.

[19] Hunter, J. D. Matplotlib : A 2d graphics environment. *Computing in Science & Engineering* **9**, 90–95 (2007).

[20] Hao, J.-K., Galinier, P. & Habib, M. Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Revue d'Intelligence Artificielle* **13**, 283–324 (1999).