TER 2019 - Rapport

Maxime Gonthier - Benjamin Guillot - Laureline Martin

17 mai 2019

Table des matières

1 Introduction

Le projet décrit dans ce rapport est la réalisation d'Algorithmes d'optimisation pour un bureau des temps.
La mission d'un tel bureau est de permettre une décongestion des moyens et infrastructure de mobilité afin d'améliorée la qualité de vie des utilisateurs et de diminuer l'impact environnemental de ces structures. Il s'agira donc d'influer sur les causes de la congestion de mobilités en repensant les horaires d'activités.
Dans notre projet, les horaires d'activités sont celles de l'Université Versailles-Saint-Quentin, et plus précisément de l'UFR des Sciences, basé à Versailles. Ainsi en repensant ces horaires, on pourra minimiser la congestion sur la ligne de bus R, allant de la gare transilien Versailles-Chantier à l'UFR.

Ce rapport a pour objectif de décrire dans un premier temps les données que nous allons utiliser dans notre projet. Ensuite nous définirons les objectifs à atteindre puis les stratégies de résolutions envisagées afin d'atteindre ces objectifs. Nous comparerons ensuite pour ces différentes stratégies les résultats obtenus afin d'identifier la meilleur stratégie.
Afin de facilité la compréhension de ce rapport, nous allons expliciter les étapes sous la forme d'un exemple simple.

2 Les données initiales

Cette section fait l'état des données utiliser pour le bon fonctionnement de l'application.
On a pour but de créer une planification pour un emploi du temps, c'est a dire un déroulement sur une journée des cours dans une université, ici l'UFR de Versailles.

L'idée étant de modifier un emploi du temps afin de minimiser la congestion dans les bus du aux arrivées d'étudiant au débuts de leurs cours, il nous faut donc un emploi du temps de base sur lequel travailler.
On à donc comme donnée initiale différents cours, représentés sous la forme d'une matrice d'adjacence.
Cette représentation nous permet de facilement identifier quels cours ont des étudiants en communs.

	Cours 1	Cours 2	Cours 3
Cours 1	0	1	0
Cours 2	1	0	0
Cours 3	0	0	0

Dans cette exemple on peut voir que le cours 1 et le cours 2 ont au moins un étudiant en commun.

On a besoin de cette information afin de pouvoir fournir une planification réaliste.

Chaque cours à également un type, c'est soit :
- un CM : dans ce cas il aura au plus 100 élèves.
- un TD : dans ce cas il aura au plus 34 élèves.

Ces information seront utilisé pour savoir combien d'élèves arrives à chaque cours.

Enfin parmi les données initiales il y a les bus. Ils sont représenté par une capacité ainsi qu'une heure.

3 Objectif

L'objectif de notre projet est de générer un emploi du temps, c'est à dire une planification qui respecte toutes les contraintes, dont la métrique est minimale.

3.1 Contraintes

3.1.1 Entre deux cours

1. Deux cours utilisent la même salle ne doivent pas avoir des horaires qui se chevauchent.
2. Deux cours qui ont des horaires qui se chevauchent ne doivent pas avoir d'élèves en commun.
3. Le temps laissé entre deux cours doit être de 15 minutes minimum.

3.2 Métriques sur les contraintes

Ce que nous évaluons c'est le dépassements du seuil de confort du bus. Dans notre cas le seuil de confort est fixé à 50 personnes, au delà le score de congestion est incrémenté à chaque arrêt de bus. Le nombre d'étudiants dans un bus ne doit pas dépasser la capacité $capacité_{max}$ du bus (fixé à 60 dans notre cas). Ainsi si il y a plus de 60 élèves pour un bus, le surplus d'élèves sera envoyé dans le bus précédent l'horaires du bus surchargé.
Pour calculer le nombre d'élèves par bus, on utilise l'emploi du temps des données initiales. On peut ainsi récupéré pour chaque tranches horaires le nombre d'étudiants censé arriver pour suivre leur cours.
On à émit l'hypothèse que les élèves arrivent dans le bus précédant leurs cours, de ce fait on peut savoir précisément le nombre d'élève qu'il y aura dans un bus avant un cours.
On prend également en compte le fait que chaque bus comporte une part de non-étudiants, qui ne sera donc pas variable dans notre application. En

effet une modification de la planification ne changera pas leur nombre.
A partir de ces information, on va pouvoir savoir si un bus atteint son seuil de congestion.

3.3 Objectif

La métrique décrite dans le point précédent va nous permettre de donner un score a chaque planification que nous allons réaliser.
En effet, l'objectif étant de réduire la congestion des bus, on aura tendance a préférer une planification pour laquelle le score de congestion est faible.
C'est donc dans la modification de la planification que les contraintes vu plus haut interviennent.
On va donc devoir trouver une planification qui minimise la congestion des bus tout en respectant les trois contraintes vu précédemment.

4 Stratégies de résolution

4.1 Les algorithmes

4.1.1 L'algorithme glouton

Cette algorithme est basé sur une heuristique simple : à partir d'une planification initiale, on fait varier les cours un par un sur tout les horaires possibles respectant les contraintes.
Pour chaque modification on recalcule la congestion des bus. Si la congestion est améliorée, alors on défini cet horaire comme étant le nouvel horaire du cours.
On parcours cependant tout les horaires possible afin de choisir celui qui améliore le plus la congestion. Si aucun n'améliore la congestion, on replace le cours a son horaire initial.
Enfin une fois qu'on a fixé le premier sommet, on passe au suivant et on recommence le même processus.

4.1.2 L'algorithme tabou dur

Dans cet algorithme, on utilise le même principe que pour l'algorithme glouton mais d'une façon différente.
Pour chaque cours, on fait varier l'heure de début tout en respectant les contraintes et on calcule la nouvelle congestion. On va également garder en mémoire la meilleure modification par sommet.
Après avoir effectué cette recherche sur tout les cours de la planification, on choisi de modifier le sommet ayant la meilleur amélioration de congestion.
On recommence ensuite ce procédé mais en interdisant l'optimisation d'un cours dont l'horaire à déjà été modifié.

4.1.3 L'algorithme tabou roulette

Cette algorithme ressemble à l'algorithme vu précédemment à ceci près qu'il laisse une part de hasard dans le choix de la modification a apporter dans la planification.
En effet on va dans cette algorithme pour chaque cours les horaires respectant les contraintes. Tout les horaires qui améliore la congestion sont ensuite gardé en mémoire.
Ensuite, parmi toute les solutions de tout les sommets, une seule est choisi aléatoirement. Cependant, plus une solution améliore la congestion, plus elle aura de chance d'être sélectionnée.
On réitère ensuite l'opération tout en interdisant l'optimisation d'un cours déjà modifié, comme pour l'algorithme précédent.

4.2 Planification initiale

Dans un premier temps nous allons créer une planification initiale qui sera utilisé comme point de départ par chaque algorithme. Cette solution va simplement mettre le sommet initial à la première horaire de la journée puis placer les autres sommets en respectant les contraintes et en les mettant le plus tôt possible. Le sommet initial est le sommet de degré entrant nul d'indice le plus faible. Ces sommets sont les cours de la matrice d'adjacence vu dans le premier point de ce rapport.

4.3 Données utilisées

Voir l'exemple en annexe pour plus de clarté.

4.3.1 Le nombre de personnes par bus

Dans notre cas d'étude, nous considérons que tous les étudiants montent au premier arrêt (gare des chantiers) et descendent au terminus (l'université).
Nous allons aussi créer le nombre de montée et de descentes à chaque arrêt. Pour obtenir les données des arrêts intermédiaires nous choisirons une valeur aléatoire, choisis dans un intervalle différent en fonction de l'heure, l'objectif est de représenter la congestion forte des heures de pointes de manière un peu plus précise :

Horaire	7-8h	8-9h	9-10h	10-11h	11-12h	12-13h	13-14h	14-15h	15-16h
Montées	[5 :15]	[5 :15]	[3 :10]	[2 :8]	[1 :5]	[1 :5]	[1 :5]	[2 :8]	[3 :10]
Descentes	[0 :5]	[0 :5]	[1 :6]	[1 :6]	[1 :5]	[1 :5]	[0 :5]	[0 :5]	[1 :5]

4.4 L'emploi du temps

L'emploi du temps est un graphe dont les sommets sont des cours et les liens des contraintes entre les cours. Pour plus de clarté nous considérons ici que chaque professeur est disponible sur toute la durée de la journée.

1. On créer un graphe non orienté en numérotant les sommets de 1 jusqu'au nombre de cours.
2. On relie les sommets entre eux lorsque qu'il y a une contrainte (même étudiant sur des horaires qui se chevauchent). Pour cela on créé une matrice d'adjacence avec un degré moyen choisis arbitrairement. Pour N sommets et K le degré moyen, la probabilité qu'il y ai une arête à insérer dans chaque case est K/N.
3. On oriente les arêtes, désormais des arcs, du sommet d'indice le plus faible au plus fort
4. Il y a obligatoirement un ou plusieurs sommets de degré entrant nul, ces sommets de notre DAG seront des points de départs possible pour notre planification.
5. On colorie le graphe, chaque couleur représente une salle.
6. Nous agençons l'emploi du temps en respectant le fait que deux couleurs et deux sommets reliés par un arc ne peuvent pas être sur la même plage horaire. Par défaut le sommet duquel nous partirons est le sommet de degré entrant nul d'indice le plus faible.

Chaque cours possède un nombre d'étudiants choisis aléatoirement entre 16 et 32 pour un TD et entre 16 et 100 pour un cours magistral. les TDs représentent trois quart des cours.

Cependant comme nous avons du les générer faute de données concrète, nous inclurons également dans cette section l'ensemble des variables utilisées pour les générer.

4.5 Les variables

les noms des variables sont simplifiés par rapport à leur nom réel dans l'application afin de les rendre plus explicite pour le lecteur.

— *probabilité de lien* : la probabilité qu'il y ai un lien entre 2 cours. Un lien entre 2 cours signifie qu'ils ont au moins un étudiant en commun, il ne peuvent donc pas avoir lieu en même temps.
— *nombre de cours* : le nombre de cours que nous allons représenter lors du déroulement de l'application. C'est le nombre de cours que nous allons devoir modifier pour la planification.
— *le nombre de salles* : le nombre de salles disponibles pour les cours.

— *heure maximum* : l'heure limite a laquelle on peut placer un cours. Dans nôtre application on représente le temps par tranche de 15min. Il va de 7h30 à 16h, on à donc 34 quart d'heure.

5 Présentation des résultats

comparer les algos, comparer en fct des données en entrées

6 Conclusion

6.1 Pistes de reflexion abandonnées

Au commencement du développement de ce projet, nous avions penser représenter chaque éléments (cours, étudiants, professeurs, salles) par des classes. Ainsi, un langage de programmation orienté objet nous semblait tout indiqué. Mais nous savions également que nous allions avoir besoin de fonction procédurales donc nous avons opté pour l'utilisation du langage C++.
Au cours de nos recherches nous nous sommes rendu compte que nous pouvions modéliser notre problème en utilisant simplement des matrices, nous avons donc abandonnée l'idée de représenter les éléments du projets par des objets.
Cependant comme nous avions déjà commencé à programmer une partie du projet, nous avons décidé de garder le C++ comme langage de programmation, sont aspect hybride entre le procédural et l'orienté objet nous permettant de continuer a l'utiliser.

6.2 Conclusion sur le problème

Le problème d'optimisation pour un bureau des temps n'as pas de solution simple. Nous avons eu recourt à différentes heuristique afin de pouvoir fournir des planifications viables.
Ce problème est donc assez coûteux en temps en fonction de l'heuristique choisi. De plus nous ne pouvons pas affirmer a 100% que nos planification sont les meilleurs possibles, cependant on peut essayer de s'en approcher.

6.3 Ouverture

Nous avons quelques pistes de réflexions qui pourrait améliorer l'efficacité de notre application.
On pourrait par exemple donner la possibilité d'inverser les arcs de la matrice représentant les cours lors des modifications de la planification. On peut également essayer d'utiliser d'autres heuristiques que celles que nous avons

choisit, comme un recuit simulé par exemple, ou alors changer la variable de décision de l'algorithme glouton.

7 Annexes

7.1 Exemple d'utilisation

7.2 Algo en code clair et commenté