TER 2019 - Rapport

Maxime Gonthier - Benjamin Guillot - Laureline Martin

29 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 infrastructures de mobilité afin d'améliorer 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 des moyens de mobilité 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. L'objectif sera ici de minimiser le nombre d'étudiants par bus pour ainsi éviter de fortes congestions. Pour cela, nous influerons sur les horaires de début de chaque cours pour un emploi du temps journalier. 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 avous utilisé dans notre projet. Puis, nous définirons les objectifs à atteindre et les stratégies de résolutions envisagées afin d'atteindre ces objectifs. Nous comparerons ensuite pour ces différentes stratégies les résultats obtenus afin d'identifier la meilleure stratégie.
Afin de faciliter la compréhension de ce rapport, nous allons expliciter chaque étape de notre algorithme sous la forme d'un exemple simple situé en annexes.

2 Les données initiales

Cette section fait l'état des données utilisées pour le bon fonctionnement de l'application.
Nous avons pour but de créer une planification pour un emploi du temps, c'est-à-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 dûe aux arrivées d'étudiants au début de leurs cours. Il nous faut donc un emploi du temps de base sur lequel travailler. Pour créer cet emploi du temps, nous aurons besoins des données suivantes :

- Probabilité de liens entre deux cours (un lien représente au moins un étudiant en commun entre deux cours)
- Le nombre cours
- Le nombre de salles

— Le type de chaque cours
— L'heure de début et de fin de journée
— Le lapse de temps entre la fin d'un cours et le début du suivant

La probabilité de liens entre deux cours est défini entre 0 et 1. Les cours sont considérés comme les sommets d'un graphe. Si nous voulons un degré moyen K dans notre graphe, alors on doit mettre une probabilité de lien de K/N avec N le nombre cours.
Une fois les liens générés, le graphe est représenté 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, nous pouvons voir que le cours 1 et le cours 2 ont au moins un étudiant en commun.
Nous avons besoin de cette information afin de pouvoir fournir une planification réaliste.
Le nombre de salles sera dans notre graphe le nombre couleurs utilisées au maximum pour colorier les sommets. Chaque cours est également défini par son type :

— un CM
— un TD

Ces informations seront utilisées pour connaitre le nombre d'élèves arrivés à chaque cours. En effet, chaque type de cours a un nombre différent d'élèves.
L'heure de début de journée correspond à l'horaire minimale de début d'un cours.
Le lapse de temps entre deux cours correspond au temps laissé entre deux cours, dans notre cas d'étude, cette valeur est fixée à 15 minutes.

Avec les données initiales décrites ci-dessus nous pouvons créer un emploi du temps initial.

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 et dont la métrique est minimale.

3.1 Contraintes

1. Deux cours qui utilisent la même salle ne doivent pas avoir des horaires qui se chevauchent.
2. Deux cours qui ont des élèves en communs ne doivent pas avoir des horaires qui se chevauchent.
3. Le temps laissé entre deux cours doit respecter un lapse de temps minimum donné dans les données initales.

3.2 Métriques sur les contraintes

Nous évaluons le dépassement du seuil de confort du bus. Dans notre cas d'étude, le seuil de confort est fixé à 50 personnes. Au delà de ce seuil, 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. Cette capacité maximale est fixée à 60 dans notre cas d'étude. Ainsi, s'il y a plus de 60 élèves pour un bus, le surplus d'élèves empruntera le bus précédent l'horaire du bus surchargé.
Pour calculer le nombre d'élèves par bus, on utilise l'emploi du temps initial (voir section "Les données initales"). Nous pouvons ainsi récupérer pour chaque tranche horaire le nombre d'étudiants censé arriver pour suivre leur cours.
Nous posons l'hypothèse que les élèves arrivent dans le bus précédant leur cours, de ce fait on peut savoir précisément le nombre d'élèves qu'il y aura dans un bus avant un cours.
Pour calculer le nombre d'élèves par bus, nous prenons aussi en compte la situation où un élève à plusieurs cours dans la même journée. Ainsi, il a emprunté uniquement le bus l'ammenant à son cours le plus tôt.
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 informations, nous pouvons savoir si un bus atteint son seuil de confort. Si c'est le cas, nous avons comptabilisé ce bus dans notre calcul de congestion totale. La métrique est la congestion totale, c'est-à-dire le nombre de bus pour lesquels le seuil de confort a été dépassé pour au moins un arrêt sur le trajet.

3.3 Objectif à atteindre

La métrique décrite dans le point précédent va nous permettre de donner un score à chaque planification que nous allons réaliser.
En effet, l'objectif étant de réduire la congestion des bus, nous aurons tendance à 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.
L'objectif final est de trouver une planification qui minimise la congestion des bus tout en respectant les trois contraintes vu précédemment.

4 Stratégies de résolution

4.1 Planification initiale

Dans un premier temps, nous allons créer une planification initiale qui sera utilisée 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, représenté dans la matrice d'adjacence vu dans le premier point de ce rapport.
En pseudocode on a :

```
Sommet de degre entrant nul d indice minimale a l horaire minimale
Pour chaque sommet sauf le premier sommet :
  Mettre l horaire le plus tot respectant les contraintes.
```

4.2 Les algorithmes

4.2.1 L'algorithme glouton

Cet algorithme est basé sur une heuristique simple : à partir d'une planification initiale, on fait varier les cours un par un sur toutes 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 toutes les horaires possibles afin de choisir celui qui améliore le plus la congestion. Si aucun n'améliore la congestion, le cours est placé à son horaire initial.
Une fois le premier sommet fixé, le sommet suivant est traité et ainsi de suite.

En pseudocode on a :

```
Pour chaque sommet :
  Pour chaque horaire :
        Si la planification a cette horaire respecte les contraintes :
        Calculer la nouvelle congestion.
        Si la congestion est meilleur qu'avant le changement,
        on fixe ce sommet a ce nouvel horaire et on reitere.
    Sinon, on replace le sommet a son horaire initial
      et on passe au sommet suivant.
```

4.2.2 L'algorithme tabou dur

Cet algorithme est basé sur le même principe que pour l'algorithme glouton mais d'une façon différente.
Pour chaque cours, l'heure de début varie tout en respectant les contraintes et une nouvelle valeur pour la congestion est calculée. La meilleure modification, c'est à dire l'horaire qui baisse le plus la congestion, pour chaque sommet est gardée en mémoire.
Après avoir effectué cette recherche sur tous les cours de la planification, le sommet ayant la meilleure amélioration de congestion est modifié.
Ce procédé est ensuite répété mais en interdisant l'optimisation d'un cours dont l'horaire a déjà été modifiée.
Ainsi on explore un voisinage restraint de la solution initiale. Restraint car on ne prend pas en compte les solutions non valides. Le voisinage est exploré en choisissant toujours la meilleure solution.
En pseudo code on a :

```
Pour chaque sommet :
  Pour chaque horaire :
        Si la planification a cette horaire respecte les contraintes :
        Calculer la nouvelle congestion.
        Si la congestion est meilleur qu'avant le changement,
        on la conserve.
  On selectionne la meilleure congestion.
  On modifie l'horaire du sommet associe a cette congestion
                                              pour l'ameliorer
```

4.2.3 L'algorithme tabou roulette

Cet algorithme ressemble à l'algorithme tabou dur, à ceci près qu'il laisse une part de hasard dans le choix de la modification à apporter dans la planification.

En effet, cet algorithme traite toutes les horaires respectants les contraintes pour tous les cours. La congestion est ensuite calculée pour chaque horaire. Ensuite, parmi toute les solutions de tous les sommets, une seule est choisie aléatoirement. Cependant, plus une solution améliore la congestion, plus elle aura de chance d'être sélectionnée. C'est le même fonctionemmment qu'une roulette dont les sous parties ne sont pas de tailles égales.
L'opération est ensuite réitérée tout en interdisant l'optimisation d'un cours déjà modifié, comme pour l'algorithme précédent.
Ici on explore le voisinage d'une manière un peu différente que l'algorihtme tabou dur, ce qui peut permettre de tomber sur de meilleures solutions si on a un peu de chance.
En pseudocode on a :

```
Pour chaque sommet :
  Pour chaque horaire :
        Si la planification a cette horaire respecte les contraintes :
        Calculer la nouvelle congestion.
        Si la congestion est meilleur qu'avant le changement,
        on la conserve.
  Chaque valeur de congestion est associe a une valeur proportionelle
        au taux de modification par rapport a la planification initiale
  On choisit un nombre au hasard. Le sommet associe a la congestion
        choisis change son horaire a l horaire donnant cette congestion.
```

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 montées et de descentes à chaque arrêt. Pour obtenir les données des arrêts intermédiaires nous choisirons une valeur aléatoire, choisie 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 étudiants en communs. Pour plus de clarté nous considérons ici que chaque professeur est disponible sur toute la durée de la journée.
Les sommets du graphe sont également coloriés, chaque couleur représentant une salle.

1. On crée 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 (étudiants en commun). Pour cela on crée une matrice d'adjacence avec un degré moyen choisit 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. Les arêtes s'oriente, 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. Le graphe est colorié, chaque couleur représente une salle.
6. L'emploi du temps est agencé 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 choisit 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 dû les générer faute de données concrètes, 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, ils 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.
— *le nombre de salles* : le nombre de salles disponibles pour les cours.

— *heure maximum* : l'heure limite à laquelle on peut placer un cours. Dans notre cas d'étude, nous représentons le temps par tranche de 15 minutes. Il va de 8h à 18h, on a donc 42 quart d'heure.

5 Présentation et analyse des résultats

Nous allons tester nos algorithmes en choisissant différentes valeurs de degré moyen. Nous fixons le nombre de cours à 40. Par exemple pour un degré moyen de 3 on calcule la probabilité de lien entre deux cours à utiliser : 40/3 = 0.075. Le nombre de salles est fixé à 30. On itère les algorithmes tabou sur 10 itérations. Les valeurs affichées sont le nombre de bus congestionés, c'est à dire dont le seuil de confort est dépassé à un moment du trajet.
Nous testerons chaque degré 15 fois ainsi nous pourrons calculer la moyenne de congestion totale de chaque méthode. On effectue un grand nombre de test car les liens entre cours sont différents à chaque itération du programme car ils sont générés aléatoirement. L'objectif est de savoir quelle méthode est la plus efficace.

Ce tableau représente le nombre de bus congestionés par méthode. Le degré moyen du graphe est 1 (0.025 probabilité de lien entre deux cours).

Tirage numéro	Initiale	Algo glouton	Algo tabou dur	Algo tabou roulette
1	9	9	9	9
2	10	10	10	10
3	11	8	8	8
4	10	10	10	10
5	10	10	10	10
6	10	8	9	8
7	10	10	10	10
8	12	12	12	12
9	10	10	10	10
10	16	9	10	11
11	9	9	9	9
12	10	10	10	10
13	11	11	11	11
14	10	10	10	10
15	10	10	10	10
Moyenne	10.5	9.7	9.9	9.9
Amélioration	X	7.6%	5.7%	5.7%

Avec un degré moyen de 1 on observe que tous les résultats sont très similaires. On a 10.5 bus congestionné pour la solution initiale contre 9.9 et 9.7 pour les autres solutions. Cela améliore tres peu le nombre de bus congestionné, à peine 7% au mieux. Ainsi on peut en conclure que pour un graphe dont le degré moyen est 1, il y a trop peu de liens entre les cours pour laisser la place à une amélioration.

On remarque qu'avec un degré de 1 les horaires de la planification sont tous très tôt. On obtient par exemple : 2 2 15 22 15 2 2 15 2 15 15 2 2 2 2 35 15 22 15 15 28 2 22 15 15 15 2 2 28 15 15. Chaque chiffre représente l'horaire d'un cours. On remarque qu'il y en a beaucoup à 2 et 15, respectivement 8h et 12h15. Avec cette planification tous les bus du matin sont donc congestionnés.

Ainsi quand les autres algorithmes partent de cette solution, ils prennent les sommets 1 par 1 et cherchent une meilleure pour ce sommet. Or de cette manière la congestion n'est pas amélioré car il y a beaucoup de cours le matin et donc modifier l'horaire de un seul sommet n'améliore pas du tout la congestion. Donc l'algorithme va replacer le cours à son horaire initiale.

Cela se remarque d'autant plus si on regarde les résultats ligne par ligne. En effet souvent les trois algorithmes prennent la même valeur de congestion que celle de la planification initiale. Ainsi on observe que notre méthode est limité si le degré est très faible. En effet puisque l'on améliore les sommets 1 à 1, on ne pourra pas beaucoup s'éloigner de la planification initiale. Dans ce cas il faudrait un algorithme qui etudie les sommets 3 par 3 ou 4 par

4. Ainsi modifier plusieurs sommets d'un coup peut permettre de modifier la congestion sur une itération et donc nous permet de fixer ces cours à de nouveaux horaires.

Ce tableau représente le nombre de bus congestionnés par méthode. Le degré moyen du graphe est 5 (0.125 probabilité de lien entre deux cours).

Tirage numéro	Initiale	Algo glouton	Algo tabou dur	Algo tabou roulette
1	13	6	11	7
2	11	9	7	8
3	12	7	7	7
4	10	9	8	8
5	14	9	10	8
6	12	7	7	8
7	13	8	9	6
8	12	10	12	10
9	11	8	9	8
10	12	7	7	7
11	10	9	9	8
12	11	10	10	10
13	16	7	8	10
14	10	7	6	6
15	10	7	7	8
Moyenne	11.8	8	8.5	7.9
Amélioration	X	32.2%	28.0%	33.1%

On observe que c'est l'algorithme glouton et l'algorithme tabou roulette qui améliore le plus la congestion avec environ 32% d'amélioration. Dans ce cas on choisira donc d'utiliser l'algorithme tabou roulette. Il fait 10 itérations, c'est à dire qu'il parcourt l'ensemble des sommets 10 fois. Alors que l'algorithme glouton fais 40 itérations, il parcourt l'ensemble des sommet autant de fois qu'il y a de sommets dans le graphe. L'algorithme roulette étant 4 fois moins coûteux, il sera privilégié dans ce cas.

Ce tableau représente le nombre de bus congestionnés par méthode. Le degré moyen du graphe est 7 (0.175 probabilité de lien entre deux cours).

Tirage numéro	Initiale	Algo glouton	Algo tabou dur	Algo tabou roulette
1	8	6	6	5
2	10	9	9	9
3	11	7	8	9
4	10	10	7	9
5	13	8	8	11
6	10	8	8	8
7	9	8	8	8
8	12	8	9	9
9	13	4	5	5
10	11	7	7	8
11	9	8	8	9
12	12	7	8	7
13	10	9	9	9
14	13	8	10	8
15	9	8	7	8
Moyenne	10.7	7.7	7.8	8.1
Amélioration	X	28.0%	27.1%	24.3%

Avec un degré moyen de 7, on observe que l'on modifie moins la congestion que pour un degré de 5. On passe par exemple de 32% à 28% pour l'algorithme glouton. Ici l'algorithme glouton et l'algorithme tabou dur sont aussi efficace. On choisira donc l'algorithme tabou dur car il est moins coûteux.

Ce tableau représente le nombre de bus congestionnés par méthode. Le degré moyen du graphe est 10 (0.25 probabilité de lien entre deux cours).

Tirage numéro	Initiale	Algo glouton	Algo tabou dur	Algo tabou roulette
1	12	8	7	7
2	13	8	8	8
3	11	7	6	8
4	13	8	10	9
5	12	8	9	7
6	10	8	6	6
7	13	7	8	8
8	11	10	9	10
9	15	8	8	9
10	11	8	9	7
11	10	6	9	6
12	12	9	6	6
13	8	6	6	7
14	9	7	7	7
15	9	8	9	7
Moyenne	11.3	7.7	7.8	7.4
Amélioration	X	31.9%	31.0%	34.5%

Pour un degré moyen de 10, on constate que les différents algorithmes utilisés on amélioré la congestion d'au moins 30% chacun. Si on fais la moyenne des trois algorithmes ce sont les meilleurs taux de modifications que l'on a obtenu. On peut donc emmettre l'hypothèse que sur un degré moyen fort, sois 1) La planification initiale est très mauvaise et donc les algorithmes améliorent facilement la solution sois 2) les algorithmes sont très efficace pour trouver des solutions quand il y a un grand nombre de liens. On constate également que l'algorithme le plus efficace est l'algorithme roulette. On aura tendance a le préférer aux autres dans ce cas.

5.1 Modifier le nombre d'itérations des algorithmes tabou

Dans le cas précedent on itérait les algorithmes tabou sur 10 itérations pour 40 sommets. Puisque l'algorithme glouton parcourt l'ensemble des sommets autant de fois qu'il y a de sommets, alors les algorithmes tabou était 4 fois moins coûteux. Ici nous allons tester nos résultats en faisant varier le nombre d'itérations ded algorithmes tabou. Le degré moyen est fixé à 5.

Itérations algo tabou	Initiale	Algo glouton	Algo tabou dur	Algo tabou roulette
1	11	9	10	10
2	11	9	10	11
4	10	9	10	10
6	11	8	9	9
8	14	7	8	9
10	11	8	9	8
12	10	10	9	9
14	11	8	8	8
16	11	8	8	9
18	12	8	8	9
20	11	9	10	8
22	9	9	9	9
24	16	8	11	8
26	13	8	9	9
28	11	7	7	7
30	14	8	9	7
Moyenne	11.6	8.5	8.8	8.7
Amélioration	X	26.7%	24.1%	25.0%

On remarque ici que augmenter le nombre d'itérations des tabou n'augmente pas significativement leurs efficacité. On constate qu'ils deviennent aussi bon ou meilleur que l'algorithme glouton à partir de 10 itérations, mais au-delà on observe pas de différences importantes. On peut donc en conclure que pour 40 sommets, itérer sur un quart des sommets est largement suffisant pour l'algorithme tabou.

5.2 Conclusion sur les résultats

Tableau récapitulatif des améliorations de la congestion :

Degré moyen	Initiale	Algo glouton	Algo tabou dur	Algo tabou roulette
1	X	7.6%	5.7%	5.7%
5	X	32.2%	28.0%	33.1%
7	X	28.0%	27.1%	24.3%
10	X	31.9%	31.0%	34.5%
Moyenne totale	X	24.9%	23.0%	24.4%

On observe que globalement ce sont l'algorithme glouton et l'algorithme tabou roulette qui améliore le plus la congestion. On choisira donc l'algorithme tabou roulette pour résoudre notre problème car il est moins coûteux.
On a également remarqué que pour un degré moyen tres faible, les algorithmes présenté ici sont inefficaces. La raison etant que la planification initiale place tout les cours tres tôt, ainsi quand les algorithmes utilisent cette planification comme point de départ, ils ne peuvent pas améliorer la congestion car ils considèrent les sommets 1 par 1. En effet modifier un seul

sommet n'améliore pas la congestion quand tous les cours sont aux mêmes horaire, donc l'algorithme remet le cours à son horaire initial.
On peut aussi conclure que la méthode tabou roulette est plus avantageuse que la méthode tabou dur, inclure une part d'aléatoire est benefique dans notre problème. Donc s'éloigner de la meilleure solution possible à chaque itération peut permettre d'arriver sur un meilleur résultat. On peut même faire l'hypothèse que passer par des solutions non valides nous permettrait de trouver de meilleures solutions.

6 Conclusion

6.1 Ce que nous avons obtenu

Notre application permet de créer une planification valide à partir des données initiales entrées par l'utilisateur (Probabilité de lien entre deux cours, nombre cours, nombre de salles, type de chaque cours, heure de début et de fin de journée, lapse de temps entre la fin d'un cours et le début du suivant). Nous créons une planification initiale, une planification suivant un algorithme glouton et deux planifications suivant des algorithmes tabou. Tous ces résultats sont consignés dans des fichiers (resultats.txt et congestion_bus.txt) facilement compréhensibles pour un utilisateur. Nous avons également conclu que l'algorithme le plus efficace est l'algorithme tabou roulette.

6.2 Pistes de réflexion abandonnées

Lors de nos recherches préliminaires, nous avions pensé 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 fonctions procédurales (les algorithmes par exemple). Donc, nous avons opté pour l'utilisation du langage C++ qui est un langage hybride
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.
De plus nous souhaitions au début du projet donner à chaque professeur une plage horaire sur laquelle il serait disponible pour donner des cours. Cela aurait apporté plus de réalisme à notre projet. Cependant cela était difficile

à implémenter n'apportait au final pas grand chose de plus quand à l'étude de la meilleure méthode possible pour améliorer la congestion.

6.3 Conclusion sur les objectifs initiaux

Le problème d'optimisation pour un bureau des temps n'a 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 choisie. De plus, nous ne pouvons pas affirmer à 100% que les planifications fournies soient les meilleures. Cependant, nous pouvons essayer de s'en approcher.
Nous pouvons cependant conclure dans notre cas que l'algorithme tabou roulette est le plus efficace globalement, c'est à dire quel que soit le degré moyen du graphe.

6.4 Ouverture

Nous avons quelques pistes de réflexions qui pourraient améliorer l'efficacité de notre application.
Il serait possible par exemple de donner aux algorithmes la possibilité d'inverser les arcs de la matrice d'adjacence des cours. Ainsi on ne serait pas limité à l'ordre établi lors de la planification initiale. En effet inverser les arcs pourrait permettre d'inverser les dépendances entre les cours. Par exemple un cours ayant des liens avec 5 autres cours qui eux n'on pas de contraintes entre eux oblige a modifier ce cours pour pouvoir modifier les suivants. En inversant les arcs on pourrais d'abord modifier les 5 autres cours.
De plus nous avons observé que pour un degré faible, nos algorithmes sont très peu efficaces, pour résoudre ce problème il faudrait des algorithmes qui modifient plusieurs sommets à la fois.
Il serait également possible d'essayer d'utiliser d'autres heuristiques que celles que nous avons choisi, comme un recuit simulé par exemple, on pourrait alors en s'autorisant a prendre des solutions ou la congestion est mauvaise sortir d'un optimum local pour essayer de trouver l'optimum global avec une bonne probabilité. On pourrait aussi changer la variable de décision de l'algorithme glouton et choisir d'organiser la planification de façon à ce que les cours avec le plus d'étudiants soit les plus éloignés les uns des autres. On trie les cours par nombre d'élèves décroissant, puis on place le premier à l'horaire le plus tôt, le deuxième cours à l'horaire le plus tard et ainsi de suite afin de les repartir sur la journée.

7 Annexes

7.1 Exemple d'utilisation

7.1.1 Représentation des montées et descentes du bus

Nous considérons que tous les étudiants montent au premier arrêt (gare des chantiers) et descendent au terminus (l'université). On ajoute aussi à chaque arrêt les montées et descentes des non étudiants. Dans notre exemple nous avons :

Numéro de l'arrêt	1	2	3	4	5
montées	50	15	10	2	0
Descentes	0	5	10	8	54

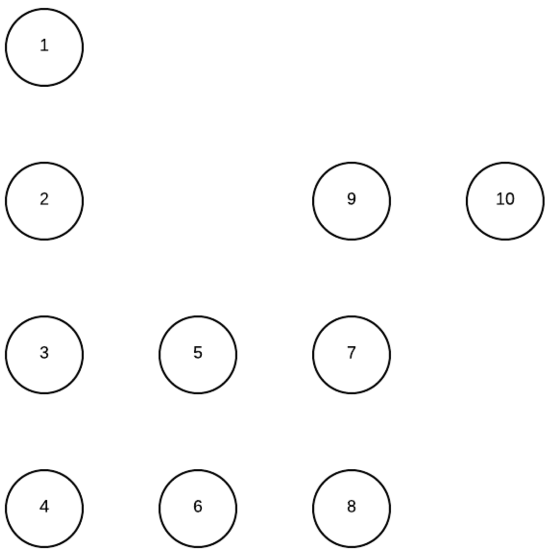
Pour obtenir les données des arrêts intermédiaires de montées et descentes nous choisirons une valeur aléatoire, choisis dans un intervalle différent en fonction de l'heure, l'objectif est de représenter la congestion forte des heures de pointes de manière un peu plus précise :

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]

7.1.2 L'emploi du temps

Voici un exemple de création de la planification initiale :

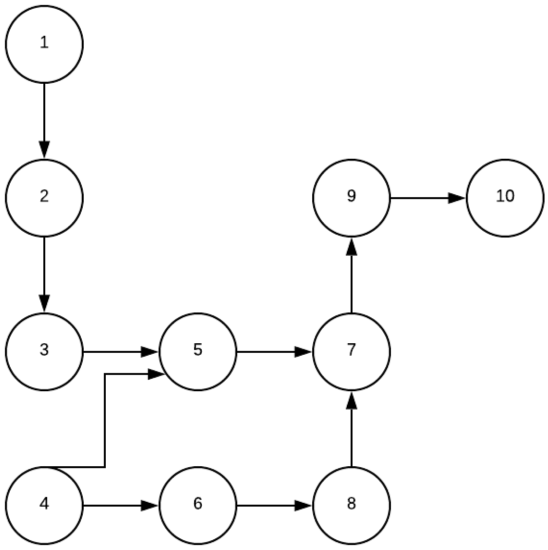
1. On créer un graphe non orienté en numérotant les sommets de 1 à 10.



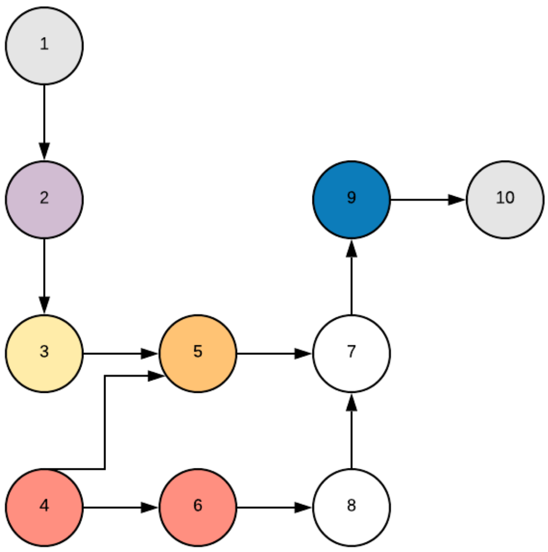
2. On relie les sommets entre eux lorsque qu'il y a une contrainte (même

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'ily ai une arête à insérer dans chaque case est K/N.

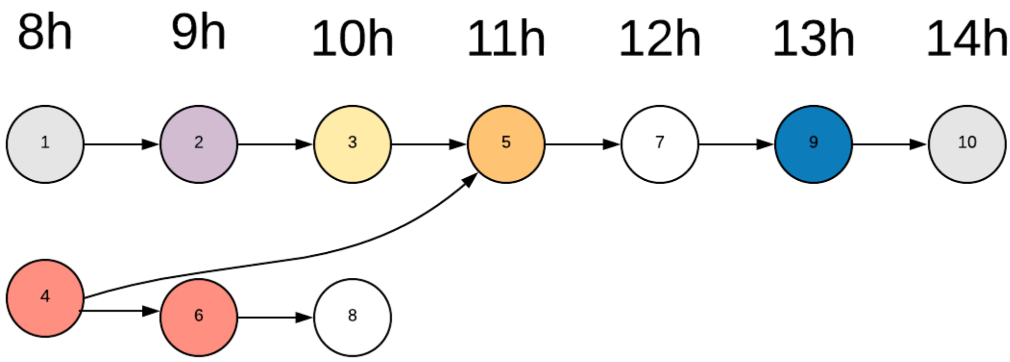
3. On oriente les arêtes désormais des arcs du plus faible au plus fort



4. On remarque dans notre exemple qu'il y a deux sommets de degré entrant nul, ces deux 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 agencons 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 le plus faible, ici le numéro 1. C'est la planification initiale :



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. Ainsi dans notre exemple nous avons :

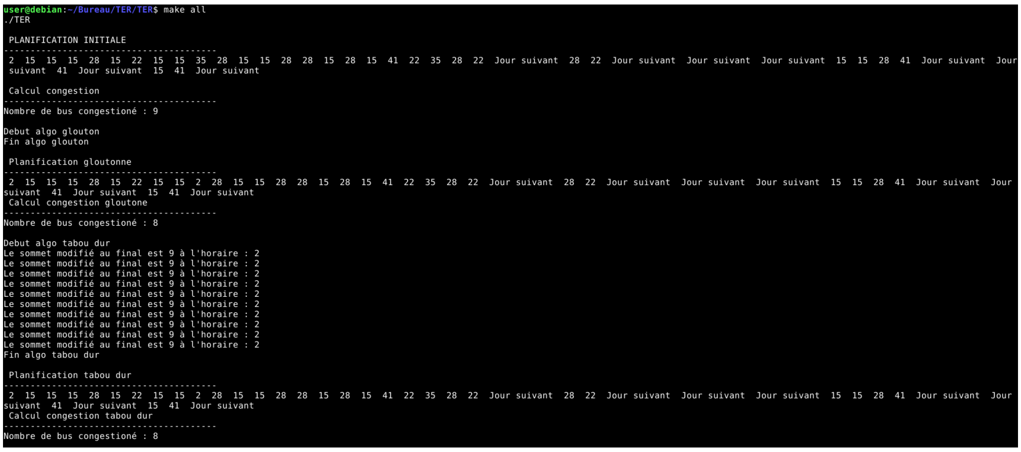
Numéro du sommet	1	2	3	4	5	6	7	8	9	10
Type de cour	TD	CM	CM	TD	TD	TD	TD	TD	TD	CM
Nombre d'élèves	16	70	60	30	25	20	16	30	14	50

7.1.3 Calcul de la congestion

A l'aide du tableau ci dessus nous allons pouvoir déterminer à chaque horaire le nombre d'étudiants arrivé au terminus et donc en conclure à l'aide du tableau des montées et des descentes le nombre de personnes à chaque arrêt. On considère que les etudiants prennent le bus arrivant 15 min avant leurs premier cour de la journée. Dans notre exemple nous avons pour 8h le cours 1 et 4, ce qui représente 46 élèves. Ainsi le bus de arrivant a 7h45 ressemble à cela :

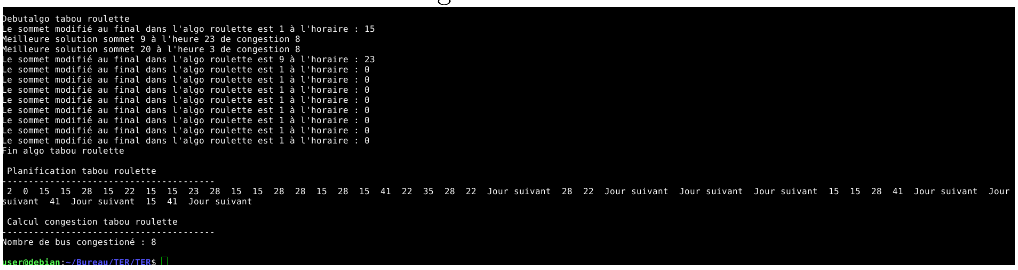
Numéro de l'arrêt	1	2	3	4	5
Différence montées/descentes	10	7	10	11	0
Nombre de personnes dans le bus	56	53	60	56	57

Ce tableau sera générés pour chaque bus. On mesurera dans chaque cas le dépassement à chaque arrêt en mettant un 1 si il y a dépassements, un 0 sinon. Sachant que la limite de confort est de 50 dans le bus que nous étudions, ce bus nous donne une valeur de congestion de 5.
A partir de la on applique l'algorithme glouton et les deux algorithmes tabou pour essayer d'améliorer notre fonction objectif. Quand on execute le code sur un terminal on obtient cela :



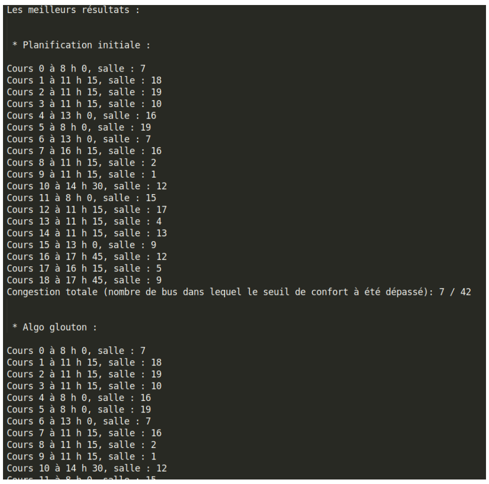
```
user@debian:~/Bureau/TER/TER$ make all
./TER

 PLANIFICATION INITIALE
---------------------------------------
 2  15  15  15  28  15  22  15  15  35  28  15  15  28  28  15  28  15  41  22  35  28  22  Jour suivant  28  22  Jour suivant  Jour suivant  Jour suivant  15  15  28  41  Jour suivant  Jour
 suivant  41  Jour suivant  15  41  Jour suivant

 Calcul congestion
---------------------------------------
Nombre de bus congestioné : 9

Debut algo glouton
Fin algo glouton

 Planification gloutonne
---------------------------------------
 2  15  15  15  28  15  22  15  15  2  28  15  15  28  28  15  28  15  41  22  35  28  22  Jour suivant  28  22  Jour suivant  Jour suivant  Jour suivant  15  15  28  41  Jour suivant  Jour
suivant  41  Jour suivant  15  41  Jour suivant
 Calcul congestion gloutone
---------------------------------------
Nombre de bus congestioné : 8

Debut algo tabou dur
Le sommet modifié au final est 9 à l'horaire : 2
Le sommet modifié au final est 9 à l'horaire : 2
Le sommet modifié au final est 9 à l'horaire : 2
Le sommet modifié au final est 9 à l'horaire : 2
Le sommet modifié au final est 9 à l'horaire : 2
Le sommet modifié au final est 9 à l'horaire : 2
Le sommet modifié au final est 9 à l'horaire : 2
Le sommet modifié au final est 9 à l'horaire : 2
Le sommet modifié au final est 9 à l'horaire : 2
Le sommet modifié au final est 9 à l'horaire : 2
Fin algo tabou dur

 Planification tabou dur
---------------------------------------
 2  15  15  15  28  15  22  15  15  2  28  15  15  28  28  15  28  15  41  22  35  28  22  Jour suivant  28  22  Jour suivant  Jour suivant  Jour suivant  15  15  28  41  Jour suivant  Jour
suivant  41  Jour suivant  15  41  Jour suivant
 Calcul congestion tabou dur
---------------------------------------
Nombre de bus congestioné : 8
```

Chaque nombre correspond à l'horaire d'un cours, on marque "Jour suivant" pour les cours que l'algorithme n'a pas pu mettre sur la journée. En dessous on calcule le nombre de bus congestioné.

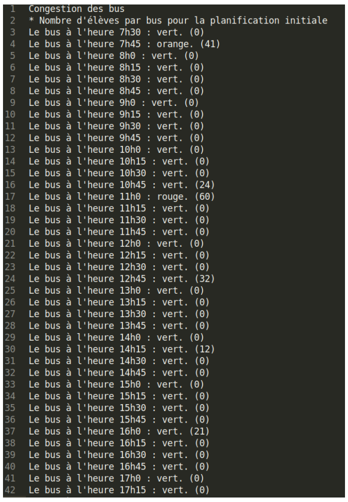


```
Debutalgo tabou roulette
Le sommet modifié au final dans l'algo roulette est 1 à l'horaire : 15
Meilleure solution sommet 9 à l'heure 23 de congestion 8
Meilleure solution sommet 20 à l'heure 3 de congestion 8
Le sommet modifié au final dans l'algo roulette est 9 à l'horaire : 23
Le sommet modifié au final dans l'algo roulette est 1 à l'horaire : 0
Le sommet modifié au final dans l'algo roulette est 1 à l'horaire : 0
Le sommet modifié au final dans l'algo roulette est 1 à l'horaire : 0
Le sommet modifié au final dans l'algo roulette est 1 à l'horaire : 0
Le sommet modifié au final dans l'algo roulette est 1 à l'horaire : 0
Le sommet modifié au final dans l'algo roulette est 1 à l'horaire : 0
Le sommet modifié au final dans l'algo roulette est 1 à l'horaire : 0
Le sommet modifié au final dans l'algo roulette est 1 à l'horaire : 0
Fin algo tabou roulette

 Planification tabou roulette
---------------------------------------
 2  0  15  15  28  15  22  15  15  23  28  15  15  28  28  15  28  15  41  22  35  28  22  Jour suivant  28  22  Jour suivant  Jour suivant  Jour suivant  15  15  28  41  Jour suivant  Jour
suivant  41  Jour suivant  15  41  Jour suivant

 Calcul congestion tabou roulette
---------------------------------------
Nombre de bus congestioné : 8

user@debian:~/Bureau/TER/TER$
```

Les résultats sont égalements dans les fichiers resultats.txt et congestion-bus.txt.



```
Les meilleurs résultats :


 * Planification initiale :

Cours 0 à 8 h 0, salle : 7
Cours 1 à 11 h 15, salle : 18
Cours 2 à 11 h 15, salle : 19
Cours 3 à 11 h 15, salle : 10
Cours 4 à 13 h 0, salle : 16
Cours 5 à 8 h 0, salle : 19
Cours 6 à 13 h 0, salle : 7
Cours 7 à 16 h 15, salle : 16
Cours 8 à 11 h 15, salle : 2
Cours 9 à 11 h 15, salle : 1
Cours 10 à 14 h 30, salle : 12
Cours 11 à 8 h 0, salle : 15
Cours 12 à 11 h 15, salle : 17
Cours 13 à 11 h 15, salle : 4
Cours 14 à 11 h 15, salle : 13
Cours 15 à 13 h 0, salle : 9
Cours 16 à 17 h 45, salle : 12
Cours 17 à 16 h 15, salle : 5
Cours 18 à 17 h 45, salle : 9
Congestion totale (nombre de bus dans lequel le seuil de confort à été dépassé): 7 / 42


 * Algo glouton :

Cours 0 à 8 h 0, salle : 7
Cours 1 à 11 h 15, salle : 18
Cours 2 à 11 h 15, salle : 19
Cours 3 à 11 h 15, salle : 10
Cours 4 à 8 h 0, salle : 16
Cours 5 à 8 h 0, salle : 19
Cours 6 à 13 h 0, salle : 7
Cours 7 à 11 h 15, salle : 16
Cours 8 à 11 h 15, salle : 2
Cours 9 à 11 h 15, salle : 1
Cours 10 à 14 h 30, salle : 12
```



```
Congestion des bus
* Nombre d'élèves par bus pour la planification initiale
Le bus à l'heure 7h30 : vert. (0)
Le bus à l'heure 7h45 : orange. (41)
Le bus à l'heure 8h0 : vert. (0)
Le bus à l'heure 8h15 : vert. (0)
Le bus à l'heure 8h30 : vert. (0)
Le bus à l'heure 8h45 : vert. (0)
Le bus à l'heure 9h0 : vert. (0)
Le bus à l'heure 9h15 : vert. (0)
Le bus à l'heure 9h30 : vert. (0)
Le bus à l'heure 9h45 : vert. (0)
Le bus à l'heure 10h0 : vert. (0)
Le bus à l'heure 10h15 : vert. (0)
Le bus à l'heure 10h30 : vert. (0)
Le bus à l'heure 10h45 : vert. (24)
Le bus à l'heure 11h0 : rouge. (60)
Le bus à l'heure 11h15 : vert. (0)
Le bus à l'heure 11h30 : vert. (0)
Le bus à l'heure 11h45 : vert. (0)
Le bus à l'heure 12h0 : vert. (0)
Le bus à l'heure 12h15 : vert. (0)
Le bus à l'heure 12h30 : vert. (0)
Le bus à l'heure 12h45 : vert. (32)
Le bus à l'heure 13h0 : vert. (0)
Le bus à l'heure 13h15 : vert. (0)
Le bus à l'heure 13h30 : vert. (0)
Le bus à l'heure 13h45 : vert. (0)
Le bus à l'heure 14h0 : vert. (0)
Le bus à l'heure 14h15 : vert. (12)
Le bus à l'heure 14h30 : vert. (0)
Le bus à l'heure 14h45 : vert. (0)
Le bus à l'heure 15h0 : vert. (0)
Le bus à l'heure 15h15 : vert. (0)
Le bus à l'heure 15h30 : vert. (0)
Le bus à l'heure 15h45 : vert. (0)
Le bus à l'heure 16h0 : vert. (21)
Le bus à l'heure 16h15 : vert. (0)
Le bus à l'heure 16h30 : vert. (0)
Le bus à l'heure 16h45 : vert. (0)
Le bus à l'heure 17h0 : vert. (0)
Le bus à l'heure 17h15 : vert. (0)
```

7.2 Explication de la programmation

7.2.1 Algorithme glouton

```
//N est le nombre de cours.
for (int i = 0; i < N; i++)
{
        //Horaires_glouton est un tableau contenant la planification
        //initiale. Chaque case representant l'horaire d'un sommet.
        //Meilleure solution et meilleure congestion representent
        //respectivement la meilleure horaire pour le sommet i et
        //la congestion associee a cette horaire.
        //Meilleure horaire signifie horaire qui minimise au plus la
        //congestion. Avant la boucle on les initialises a l'heure
        //deja attribue au sommet i.
        sommet_modifie = i;
        meilleure_solution = Horaires_glouton[sommet_modifie];
        meilleure_congestion = calcul_congestion_totale(
                Horaires_glouton,Nb_eleves,
                heure_max,N,TO);
        //Le while test tous les horaires possibles pour le sommet i.
        while(j < heure_max - 1)
        {
                Horaires_glouton[sommet_modifie] = j;
                //Si l'horaire j est valide
                if (test_solution_valide(Horaires_glouton, N, couleur,
                        Type, TO) == true)
                {
```

```
                        temp = calcul_congestion_totale(Horaires_glouton,
                                Nb_eleves,heure_max,N,TO);
                        //Si la congestion a l'horaire j est meilleure
                        //que la meilleure congestion du sommet i,
                        //alors on remplace la meilleure congestion et
                        //la meilleure horaire par j.
                        if(temp < meilleure_congestion)
                        {
                                meilleure_solution =
                                        Horaires_glouton[sommet_modifie];
                                meilleure_congestion = temp;
                        }
                }
                j++;
        }
        //j commence a 2 car 2 correspond a 8h du matin.
        j = 2;
        Horaires_glouton[sommet_modifie] = meilleure_solution;
}
```

7.3 Algorithme tabou dur et roulette

Nous ne décrivons pas l'algorithme tabou dur car il est quasiment identique. La différence est explicité ci dessous.

```
//Horaires_tabou_roulette est un tableau contenant la planification
//initiale. Chaque case representant l horaire d un sommet.
int *Horaires_tabou_roulette = (int*)malloc(N*sizeof(int*));

//Tableau qui servira a verifier si un sommet a deja ete modifie
//et ainsi l interdire
int *Sommets_interdits = (int*)malloc(Nb_iterations*sizeof(int*));

//Tableau contenant la meilleure horaire pour un sommet donne
int *Meilleure_solution = (int*)malloc(N*sizeof(int*));

//Tableau contenant la meilleure solution pour un sommet donne
        int *Meilleure_congestion = (int*)malloc(N*sizeof(int*));

//Tableau contenant les probabilite qu'un sommet sois choisis
//par la roulette
int *Valeurs_roulette = (int*)malloc(N*sizeof(int*));

//La boucle k permet d'iterer l'algo plusieurs fois
for(int k = 0; k < Nb_iterations; k++){
```

```
//On initialise la congestion initiale
congestion_initiale = calcul_congestion_totale
        (Horaires_tabou_roulette,Nb_eleves,heure_max,N,TO);
//La boucle i parcoure l'ensemble des sommets
for(int i = 0; i < N; i++){
        //On cherche a savoir si le sommet courant est un sommet
        //deja modifie, si oui on ne le regarde pas
        for(int l = 0; l < k+1; l++){
                if(i == Sommets_interdits[l]) {i++; l = 0;}
        }
        //Initialisation des meilleures solution pour le sommet i
        Meilleure_solution[i] = Horaires_tabou_roulette[i];
        Meilleure_congestion[i] = calcul_congestion_totale
                (Horaires_tabou_roulette,
                Nb_eleves,heure_max,N,TO);
        horaire_initiale = Horaires_tabou_roulette[i];

        //La boucle j parcours l'ensemble des horaires possibles
        while(j < heure_max - 1){
                Horaires_tabou_roulette[i] = j;
                //On verifie que l'horaire est bien valide
                if (test_solution_valide(Horaires_tabou_roulette,
                        N, couleur, Type, TO) == true){
                        temp = calcul_congestion_totale
                                (Horaires_tabou_roulette,
                                Nb_eleves,heure_max,N,TO);
                        //Si la congestion de cette horaire est
                        //meilleure, on met a jour le meilleur
                        //horaire et la meilleur congestion
                        // du sommet i
                        if(temp < Meilleure_congestion[i]){
                                Meilleure_congestion[i] = temp;
                                Meilleure_solution[i] = j;
                        }
                }
                j++;
        }
        //j commence a 2 car 2 correspond a 8h du matin
        j = 2;
        //On remet l'horaire du sommet i a son horaire de base
        //car on ne sait pas encore quel sommet sera modifie a
        //cette iteration de l'algo tabou
        Horaires_tabou_roulette[i] = horaire_initiale;
        //On remplit le tableau des proba de cette maniere :
```

```
        //On marque la valeurs de modification par rapport a
        //la congestion initiale. Ainsi si on a 10 de congestion
        //initiale et 8 en congestion pour le sommet i,
        //Alors on marque 2 dans le tableau. Pour le sommet
        // suivant si il modifie de 3 on marque 5.
        //Ainsi ca donne un tableau comme cela :
        //2 4 8 15 16 17 17.
        if(Meilleure_congestion[i] < congestion_initiale){
                Valeurs_roulette[i] =
                 congestion_initiale - Meilleure_congestion[i]; }
        else { Valeurs_roulette[i] = 0; }
}
for (int p = 1; p < N; p++) {
        if (Valeurs_roulette[p] == 0) {
                Valeurs_roulette[p] = Valeurs_roulette[p-1];}
        else { Valeurs_roulette[p] += Valeurs_roulette[p-1]; }
}

if (Valeurs_roulette[N-1] > 0) {
        choix_roulette = rand()%Valeurs_roulette[N-1];
}
else { choix_roulette = 0; }

//On prend au hasard une valeur et on regarde ou elle tombe
//dans le tableau, ainsi plus on modifie la congestion,
//plus on a de chance d'etre choisi.
//La difference avec l algo tabou dur ce fait ici
//L algo tabou dur va a la place de faire ca
//fixer horaires_tabou_dur a la meilleure horaire
//possible, celle qui ameliore le plus la congestion.
if ((choix_roulette > 0) &&
         (choix_roulette <= Valeurs_roulette[0]))
{
        Horaires_tabou_roulette[0] = Meilleure_solution[0];
        Sommets_interdits[k] = 0;
}
else{
        for (int m = 1; m < N; m++){
          if ((choix_roulette >= Valeurs_roulette[m-1])
             && (choix_roulette <= Valeurs_roulette[m])){
                //Mise a jour de la planification
                //et des sommets interdits
                Horaires_tabou_roulette[m]=Meilleure_solution[m];
                Sommets_interdits[k] = m;
```

```
                                m = N;
                        }
                    }
            }
}
```

7.3.1 Planification initiale

```
//N le nombre de cours.
//HEUREMAX le nombre de quart d'heure dans une planification
for(int i = 1; i < N; i++){
        //On va regarder tous les liens de i vers k pour tout i et tout k
        //Si il y a un lien, le cours i est place avant le cours k.
        //C'est le sens des arcs qui veut cela.
 for(int k = 0; k < N; k++){
 //Si il y a un lien entre le cours i et le cours k
  if(T[i][k] == 1){
   if(TYPE[i] == 0){//si le cours est un td
    for(int j = 0; j <= HEUREMAX; j++){
        //Si la solution est valide a l'horaire k +
        //la duree du cours + une constante j incremente
        //pour tester toutes les possibilites d'horaire.
         if(test_coloration(Horaires, couleur, N, TYPE,
                Horaires[k] + 13 + j, i) == true){
                 if(test_lien(Horaires, N, T, Horaires[k]
                        + 13 + j, i, TYPE) == true){
                //On met l'horaire de i avant celle de k.
                        Horaires[i] = Horaires[k] + 13 + j;
                        break;
                   }
            }
         }
    }
        else //si le cours est un cm, le code est le meme.
        {
                for(int j = 0; j <= HEUREMAX; j++)
                {
                        if(test_coloration(Horaires, couleur, N, TYPE,
                                Horaires[k] + 7 + j,i) == true)
                        {
                          if(test_lien(Horaires, N, T, Horaires[k] + 7
                                + j, i, TYPE) == true)
                          {
```

```
                                        Horaires[i] = Horaires[k] + 7 + j;
                                        break;
                                  }
                            }
                          }
                  }
}
//Si il n'y a aucun lien vers le sommet k.
else {
 if(testConnnexe(T, N, i) == true){
  if(TYPE[k] == 0)//si le cours est un td
  {
        for(int j = 0; j <= HEUREMAX; j++)
        {
          if(test_coloration(Horaires, couleur, N, TYPE,
                Horaires[k] + 13  + j, i) == true)
          {
                if(test_lien(Horaires, N, T, Horaires[k] +
                        13 + j, i, TYPE) == true)
                {
                        Horaires[i] = Horaires[k] + 13 + j;
                        break;
                }
        }
   }
  }
  else //si le cours est un cm.
  {
        for(int j = 0; j <= HEUREMAX; j++)
    {
                if(test_coloration(Horaires, couleur, N, TYPE,
                        Horaires[k] + 7 + j,i) == true)
                {
                        if(test_lien(Horaires, N, T, Horaires[k] +
                                7 + j, i, TYPE) == true)
                        {
                                Horaires[i] = Horaires[k] + 7 + j;
                            break;
                        }
                }
        }
   }
  }
  }
```

```
  }
}
```