UNIVERSITÉ D'AVIGNON
ET DES PAYS DE VAUCLUSE

CENTRE
D'ENSEIGNEMENT
ET DE RECHERCHE
EN INFORMATIQUE

MASTER 1 Informatique
Ingénierie Logiciel
UE PROJET

>>>

Tournée de véhicule dans les réseaux tolérants aux délais

Omar SY / MESSAS KOUSEILA // Tutrice : ROSA FIGUEIREDO

9 janvier 2017

CERI - LIA
339 chemin des Meinajariès
BP 1228
84911 AVIGNON Cedex 9
France

Tél. +33 (0)4 90 84 35 00
Fax +33 (0)4 90 84 35 01

http://ceri.univ-avignon.fr

Sommaire

1 Introduction

La planification de tournées de véhicules est un problème qui fait l'objet de nombreuses recherches scientifiques, dans le domaine de la recherche opérationnelle.

Nous allons mettre en place au cours de cette année une application permettant de résoudre une version récente du problème de tournée de véhicule, qui apparaît dans les réseaux tolérants aux délais par des méthodes exactes mais aussi par des méthodes heuristiques.

Une application sera développée durant le projet pour nous permettre une meilleure compréhension des résultats, une utilisation plus simple de nos algorithmes notamment grâce à une interface graphique qui proposera plusieurs fonctionnalités visant à simplifier l'obtenions de résultats, leurs analyses et leurs comparaisons afin de déterminer l'efficacité des algorithmes de résolution développée.

1.1 Contexte générale

La communauté scientifique s'est heurtée à de nombreux problèmes face à l'optimisation des activités liées à l'industrialisation. Les problèmes étaient principalement comment réduire le coût et le temps des activités sur les différentes chaînes de production.

Au niveau du transport des ressources et des marchandises plusieurs questions ont été posées. En effet comment faire pour réduire le temps de trajet mais aussi comment réduire les frais liés au trajet. En étudiant le problème du tournée de véhicule (*Vehicle Routing Problem* (VRP)), De plus prés ils ont remarqué que ce problème avait plusieurs points identiques avec le problème du voyageur de commerce. En recherche opérationnelle, ce problème est représenté par un graphe $G = (V, A, w)$. En effet dans ce problème, on considère un ensemble $V = \{1, 2, \ldots, n\}$ de n villes et un ensemble $A = \{(i, j) \mid i, j \in V\}$ de chemins direct entre quelques paires des villes. Chaque chemin $(i, j) \in A$ est caractérisé par une distance w_{ij}. Le problème consiste à trouver le plus court cycle hamiltonien (un cycle passant par chaque sommet une et seule fois).

Le problème de tournée de véhicule peut être vue comme une généralisation du problème du voyageur de commerce.

Comme pour le problème du voyageur de commerce, le problème VRP [2] est un problème NP-Complet [], c'est-à-dire un problème pour lequel les méthodes exacts de solutions connues sont exponentiel dans la taille du problème. Ces méthodes sont donc inexploitables en pratique même pour des instances de taille modérée. En effet on peut résoudre le problème que pour des petites instances. Pour pouvoir résoudre ce problème pour des grandes instances nous devons passer par des heuristiques ainsi on peut noter des heuristiques comme l'algorithme de Clarke and Wright [4], qui permet de résoudre le problème de tournée de véhicule, avec un dépôt centrale et avec un nombre de véhicules non fixé. Mais aussi on note les heuristiques de recherche locale et certaines méta-heuristiques permettent aussi de résoudre le problème. Selon l'environnement où on applique le problème et les contraintes liées à cet environnement, on note plusieurs types de problèmes liés au vehicle routing.

Nous avons plusieurs variantes de ce problème.On peut citer les problèmes liés aux contraintes :

-**De Capacité** : Aussi appelé *Capacited Vehicle Routing Problem* (CVRP) [1]. Ces types de problèmes cherchent à effectuer une tournée pour un seul véhicule avec une capacité finie. On note des différences sur l'autonomie du véhicules : certains chercheurs traitent le problème en gérant l'autonomie du véhicule, d'autres considèrent l'énergie du véhicule illimitée.

- **De temps** : Aussi appelé *Vehicle Routing Problem with Time Windows* (VRPTW) [6]. Au niveau de ce problème on doit respecter le temps de visite des clients.Pendant la visite on peut arriver plutôt que la date de visite, mais lorsqu'on arrive avant la date de visite, on est obligé d'attendre jusqu'à la date de visite soit atteinte. On ne peut plus livrer un client dont sa date de livraison est dépassée et ce client est considéré comme insatisfait.

-**De collecte et livraison** : Aussi appelé *Vehicle Routing Problem with Pick-up and Delivery (VRPPD)*[5].L'objectif est de minimiser le parc de véhicules et la somme du temps de déplacement, avec la restriction que le véhicule doit avoir une capacité suffisante pour transporter les marchandises à livrer et celles qui sont ramassées chez les clients pour les renvoyer au dépôt.

-**Temps et communication d'information** : Aussi appelé *Vehicle Routing for the communication of time-dependent information* [3]. On cherche à maximiser dans ce problème la quantité d'information récoltée. Dans ce problème on peut autoriser de retourner de temps en temps vers la base, mais lorsque la date finale est atteinte, on doit retourner vers la base.

1.2 Définition du problème

Le problème de la tournée de véhicule dans les réseaux tolérants aux délais. (*Vehicle Routing with Delay-Tolerant Networks*), consiste à trouver une route pour le véhicule capable de maximiser la quantité l'information pour un temps finit T . Mais aussi de garantir de temps en temps que les informations recueillies seront envoyées à l'extérieur (selon la fonction objective utilisée).

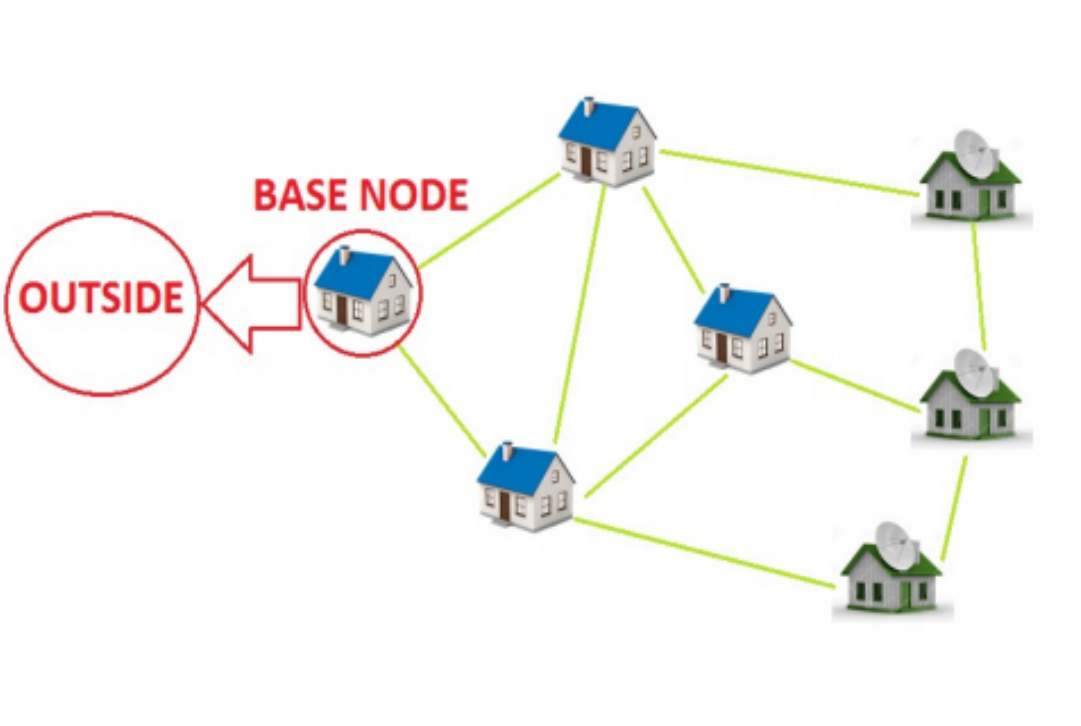

Figure 1. Environnement.

Le problème de tournée de véhicule est défini par un graphe non orienté G=(V,A) où V est l'ensemble des sommets $i \in V$ et A l'ensemble des arcs.

Pour chaque arc $(i, j) \in A$, nous avons un temps t_{ij} qui représente la durée pour aller de i à j.

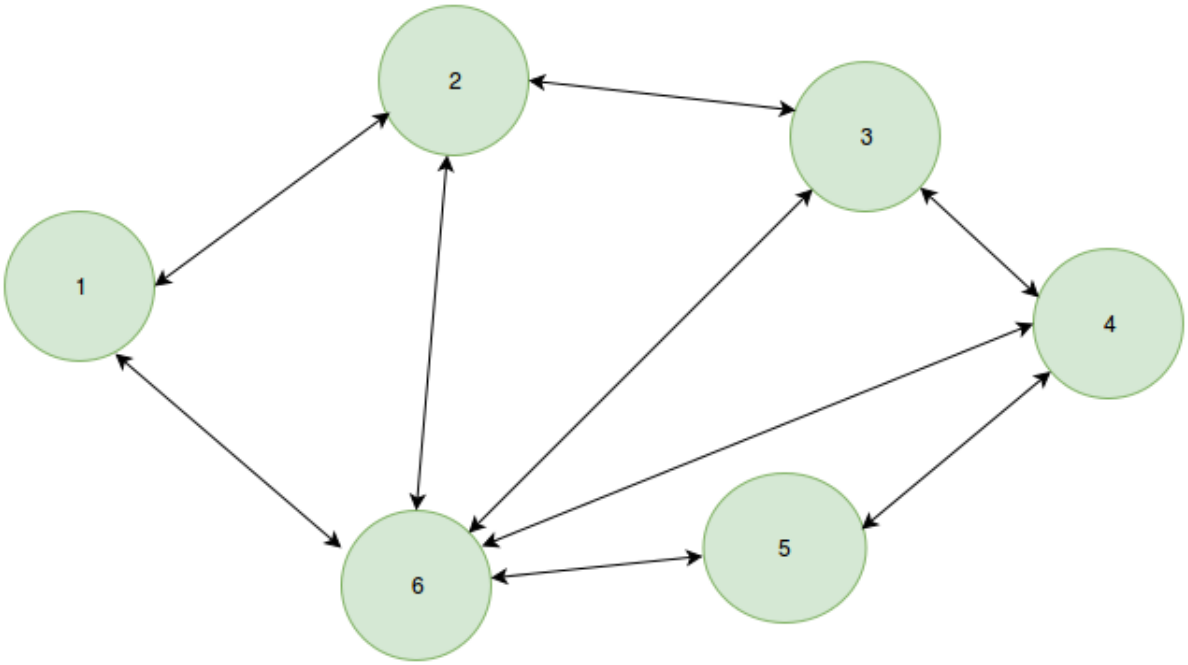

Figure 2. Exemple graphe non orienté.

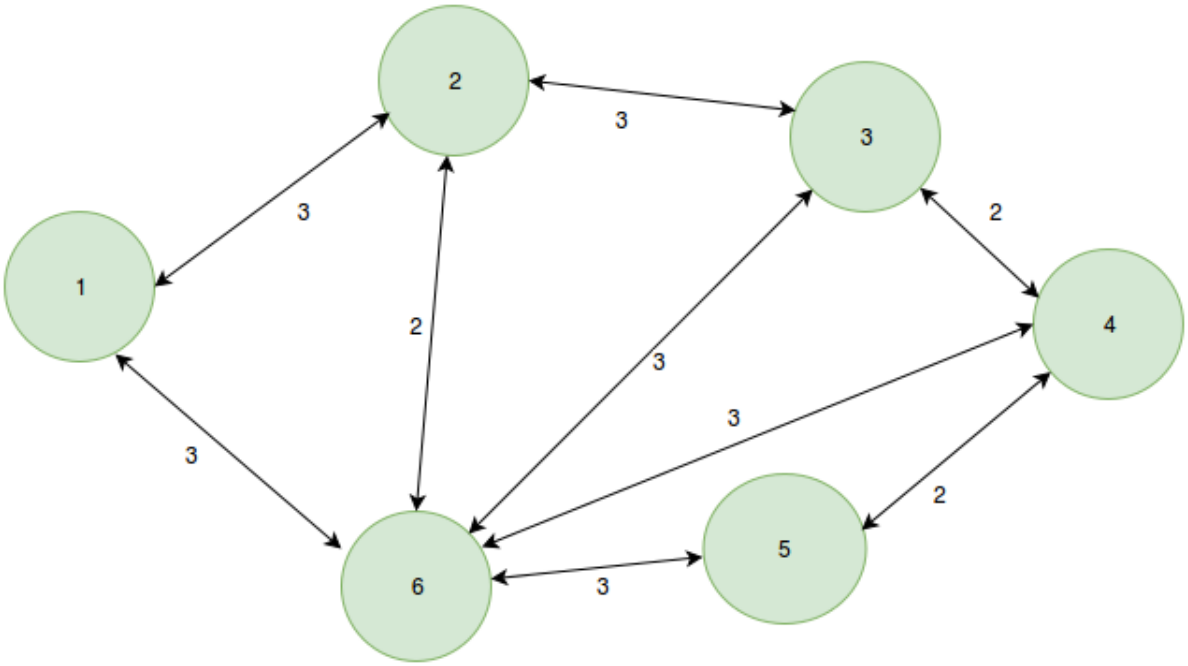

Figure 3. Exemple graphe avec temps de déplacement entre deux nœuds.

Chaque sommet ou station i est caractérisé par une quantité d'informations au temps k q_{jk} et C_{j1} est la quantité initiale.

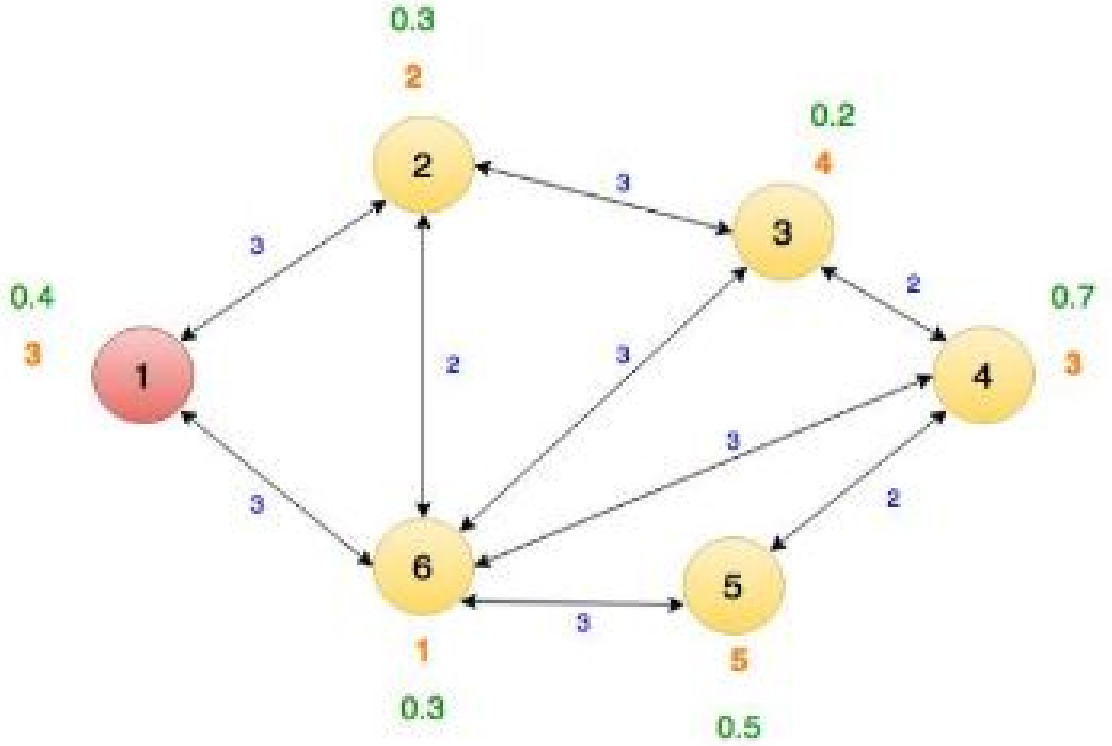

Figure 4. Exemple graphe avec temps de déplacement entre deux nœuds et leur quantité initiale.

L'objectif est de minimiser l'ensemble des informations présentes dans G.

Pour minimiser l'ensemble des informations présentent dans G on peut s'autoriser des retours vers la base de temps en temps.

1.3 Objectifs du projet

objectif final :

A la fin de l'année, l'objectif sera de livrer une application (ORPA pour Optimisation Routing Problem and Analysis) complète permettant de répondre au problème du VRP, en implémentant plusieurs fonctionnalités permettant de :

- Charger une instance sous forme de fichier.
- Générer automatiquement une instance.
- Construire une instance via l'interface graphique.
- Afficher des statistiques concernant l'instance à résoudre.
- Afficher l'instance généré sous forme de graphe.
- Lancer la résolution de l'instance via un solver (gurobi).
- Lancer la résolution de l'instance via des heuristiques.
- Afficher la solution sous forme de graphe.
- Afficher les statistiques sur la solution générer.
- Sauvegarder les solutions produites.
- Un gestionnaire de projets (instance, solution).
- Produire une analyse de la solution générée.

Objectifs par semestre

- **Semestre 1 :**
  1. Se documenter sur le problème
  2. Présentation du problème.
  3. Rédiger un cahier des charges pour l'application.
  4. Définir une maquette pour l'interface graphique.
  5. Implémenter l'interface graphique.
  6. Définir la structure des données manipuler notamment pour une instance.
  7. Implémenter le générateur d'instance.
  8. Permettre le chargement d'instance via un fichier.
  9. Permettre la construction d'instance via l' IU.
  10. Permettre la communication entre l' IU et le solver.
  11. Étudier un modèle de résolution du problème.
  12. Étudier Gurobi.
  13. Implémenter le modèle avec Gurobi.
- **Semestre 2 :**
  1. Implémentation d'un analyseur de solution.
  2. Implémentation d'une fonctionnalité pour les statistiques.
  3. Documentation sur les heuristiques de résolution d'instance.
  4. Développement d'heuristiques pour la résolution d'instance.
  5. Fournir une documentation complète sur l'application.

1.4 Conduite et organisation du projet

1.4.1 Planning :

Nom	Date de début	Date de fin
Réunion de lancement du projet	03/10/16	03/10/16
Étude du problème	04/10/16	10/10/16
présentation d'une maquette	04/10/16	10/10/16
Amélioration du descriptif du problème	13/10/16	17/10/16
Amélioration de la maquette	13/10/16	17/10/16
Implémentation de l'interface graphique	18/10/16	31/10/16
Compléter l'état de l'art et la description du problème	18/10/16	31/10/16
Implémetantion d'un générateur d'instance	01/11/16	28/11/16
Étude de Gurobi (Solver)	01/11/16	28/11/16
Interaction du solver avec l'IU	29/11/16	06/01/17
Implémentation du model avec le solver Gurobi	29/11/16	06/01/17
Finalisation du rapport	06/01/17	09/01/17
Soutenance	16/01/17	16/01/17
Finalisé l'implémentaion du modél	30/01/17	13/02/17
Finalisé l'implémentation du générateur	30/01/17	13/02/17
Finalisé l'implémentation de l'affichage des graphes	30/01/17	13/02/17
Intégration	14/02/17	27/02/17
Étude des méthodes heuristiques	28/02/17	13/03/17
Implémentation d'heuristiques pour la résolution du problème	14/03/17	10/04/17
Implémentation de fonctionnalités pour les statistiques	11/04/17	25/04/17
Implémenation d'un analyseur de solution	26/04/17	09/05/17
Intégration	10/05/17	16/05/17
Rédaction d'une documentation de l'application	16/05/17	23/05/17

Figure 5. Diagramme de Gant 1.

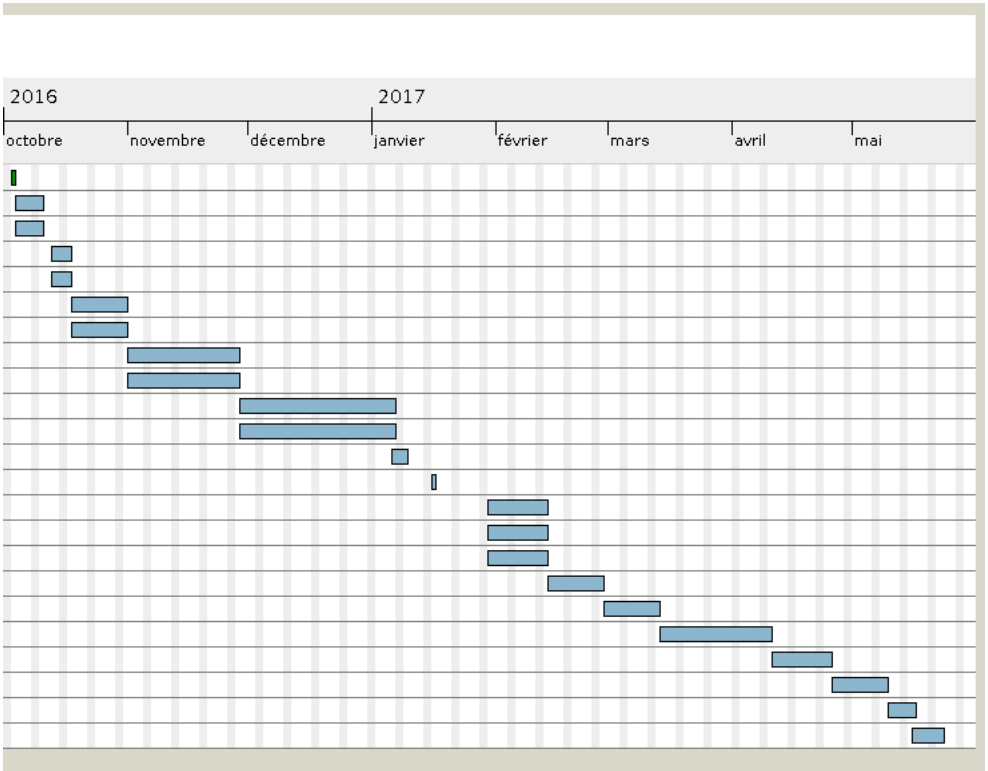

Figure 6. Diagramme de Gant 2.

1.4.2 Partage des taches :

Pour atteindre nos objectifs du premier semestre, nous avons partagés les taches de la manière suivante :

- **Omar Sy :**
  1. Étude du problème : faire l'état de l'art ; concernant les différents problèmes qui sont en rapport avec le sujet étudier.
  2. Présentation du problème : Présenté le problème du VRP en détaillant spécifiquement les données qui le caractérisent et présenter un exemple détaillé.
  3. Implémentation du modèle via un solveur (Gurobi) : codé via l'API Gurobi le modèle afin de résoudre le problème.
- **MESSAS Kouseila :**
  1. Développement de l'interface graphique : implémentation d'une interface graphique en langage Java via l'API Swing de Java, qui puisse simplifier la lecture, saisir, traiter les instances via le solveur.
  2. communication entre interface graphique et solveur : permettre de lancer la résolution d'une instance via l'interface graphique en appelant le solveur.
  3. générateur d'instance : implémenté un générateur d'instance qui permet de générer aléatoirement des données représentatives d'un cas qui pourra être traité par le solveur ou les heuristiques.
  4. affichage graphique des instances générées : implémenté la fonctionnalité qui permet de dessiner sur une fenêtre un graphe représentant une instance.

1.4.3 Communication :

Pour mener à bien notre projet, nous avons fixé un rendez-vous hebdomadaire avec notre tutrice Figueiredo Rosa, qui se déroule généralement chaque lundi. durant la réunion, nous faisons le point sur l'état d'avancement du projet de manière générale, chacun de nous expose les travaux qu'il a réalisés durant la semaine, et prend compte des remarques de la tutrice sur le travail effectué, puis nous fixons des objectifs à réaliser pour la prochaine réunion.

Nous communiquons avec notre tutrice par mail, un compte rendu de réunion est effectué chaque semaine, ou nous détaillons les travaux effectués, les remarques et les objectifs pour la prochaine réunion.

2 Modélisation par PLNE

2.1 Constantes

- Le temps pour aller du nœud i au nœud j :

$$t_{ij}$$

- La distance entre deux nœuds i et j :

$$d_{ij}$$

- Le flux d'information du station i :

$$r_i$$

- Le temps de la simulation :

$$T$$

- La quantité d'information maximale qu'une station peut recupérer en un instant k :

$$R$$

- L'influence des facteurs d'équipement, des obstacles qu'a sur la transmission des données de la station i vers la station j :

$$\alpha_{ij}$$

2.2 Variables

- On doit pouvoir décider en instant k la station ou on doit aller :

$$x_{ijk} = \begin{cases} 1 & \text{Si la voiture va aller du nœud i au nœud j au temps k} \\ 0 & \text{sinon.} \end{cases}$$

- En un instant k on doit pouvoir décider à quelle station ou on se trouve :

$$z_{jk} = \begin{cases} 1 & \text{Si la voiture se trouve au nœud i au temps k} \\ 0 & \text{sinon.} \end{cases}$$

- En un instant k on doit décider si on envoie des information de la station i vers la station j :

$$\Theta_{ijk} = \begin{cases} 1 & \text{Si la station envoie des informations, via réseau, du nœud j au noeud i au temps k} \\ 0 & \text{sinon.} \end{cases}$$

- Quantité d'information du nœud i au temps k :

$$q_{ik}$$

- Quantité d'information envoyée du nœud i au nœud j au temps k :

$$f_{ijk}$$

2.3 Objective fonctions

$$\text{minimise} \sum_{j \in V} q_{jT}$$

Cette fonction a pour objective de minimiser la quantité d'informations de chaque station. Le but de cette fonction est de minimiser la quantité d'informations des nœuds afin de viser l'ensemble, ainsi chaque nœud ayant une quantité minimale, donnera un ensemble c'est-à-dire un graphe ayant une quantité minimale.

L'avantage de cette fonction est de ne délaisser aucune station. Cette fonction permet une meilleure résolution des problèmes où l'on considère à partir d'un certain temps T l'information devient inutile.

2.4 Contraintes

Pour pouvoir étudier le problème nous devrions prendre en compte :

ou le véhicule se trouve au commencement, dans notre modèle nous considérons que le véhicule commence au nœud 1.

Le véhicule doit envoyer des informations vers une base, dans notre modèle nous allons considérés que le véhicule doit retourner les informations au nœud 1.

À chaque temps k la station a une quantité d'informations qui augmente lorsqu'on n'envoie pas d'information, mais cette quantité diminue seulement lorsqu'on envoie des informations.

Il faut noter que les stations ne peuvent faire qu'une seule chose en un temps k. En effet l'envoie des informations se font d'une station vers un et un seul récepteur en un temps k.

Il faut que le véhicule visite récupère le maximum d'informations avant que le temps T soit atteint.

Si le temps T'est atteint le véhicule doit retourner à la base pour remettre l'information.

Le véhicule ne peut quitter une station qu'après avoir reçu l'ensemble des informations de cette station.

Il envoie des informations d'une station à un autre en un temps k vont prendre un temps l qui est lié à la quantité d'informations stockées.

2.4.1 Contraintes sur la route du véhicule

Au début nous nous trouvons au sommet 1, et on ne peut que se déplacer vers le voisin j du sommet 1. Le déplacement va nous un prendre un temps t_{1j} qui est la durée pour aller de 1 vers j :

$$\sum_{(i,j) \in A} x_{1jt_{1j}} = 1$$

Le dernier sommet à visiter est le sommet 1. En effet c'est le sommet qu'on va utiliser pour l'envoi des informations vers l'extérieur :

$$\sum_{(i,j) \in A} x_{1jT} = 1$$

Si on est dans un nœud j à un instant k on peut pas prévoir de visiter au même instant le nœud j :

$$z_{jk} + \sum_{(i,j) \in A} x_{ijk} \leq 1, \forall j \in V, \forall k \in T$$

Si on est dans une station j dans un instant k, soit on reste dans la station soit on prévoit d'aller dans une autre station i voisine à j :

$$z_{jk} + \sum_{(i,j)\in A} x_{ijk} = \sum_{(j,p)\in A} x_{jp(k+t_{jp})} + z_{j(k+1)}$$

2.4.2 Contrainte sur le transfert d'information

On peut envoyer à une station j des informations si le véhicule se trouve à la station j :

$$\sum_{j\in range(i)} \Theta_{jik} \leq M z_{ik} \forall i \in V, \forall k \in T$$

Si le véhicule se trouve à une station j à l'instant k, la quantité d'information qu'il va envoyer dépend du wifi et de la distance entre i et j :

$$f_{jik} \leq \alpha_{ji}(\frac{1}{1+d_{ij}^2}) r_j \Theta_{jik}, \forall j \in V, \forall i \in range(j), \forall k \in T$$

La quantité d'information qu'on peut envoyer vers un noeud i ne peut pas dépasser à la quantité d'information qu'il peut recevoir :

$$\sum_{j\in range(i)} f_{jik} \leq R$$

2.4.3 Contrainte sur la quantité d'informations

La quantité d'informations d'une station j à l'instant k+1 doit était égale à la quantité d'informations à l'instant k plus le flux d'informations de la station j moins la quantité d'informations envoyées :

$$q_{j(k+1)} = q_{jk} + r_j - \sum_{i\in range(j)} f_{jik}, \forall \in V, \forall k \in T$$

La quantité d'informations ne peut pas être inférieure au flux d'information :

$$q_{jk} \geq r_j, \forall j \in V, \forall k \in T$$

$$x_{ijk}, z_{jk}, \Theta_{jik} \in 0, 1$$

2.4.4 Inégalité valable

On a droit d'envoyer des informations au nœud i que si le véhicule à la station i

$$\Theta_{jik} \leq zik, \forall \in V, \forall k \in T$$

Si le véhicule se trouve à la station i, la quantité d'informations reçues ne doit pas être supérieur à 0, sinon la quantité doit être nulle

$$\sum_{j\in range(i)} f_{jik} \leq R zik, \forall_i \in V, \forall k \in T$$

L'environnement du problème est constitué de base, les bases sont chargées de collecter les données, et de stations, chargées d'envoyer les données.

3 Modélisation est mise en œuvre :

Dans cette partie nous présentant la modélisation réalisée pour mener à bien l'implémentation de l'application, puis la mise en œuvre de cette dernière.

3.1 Modélisation :

3.1.1 Maquette :

Voici la maquette validée par le tuteur, elle représente l'interface graphique qu'on souhaite implémenter.

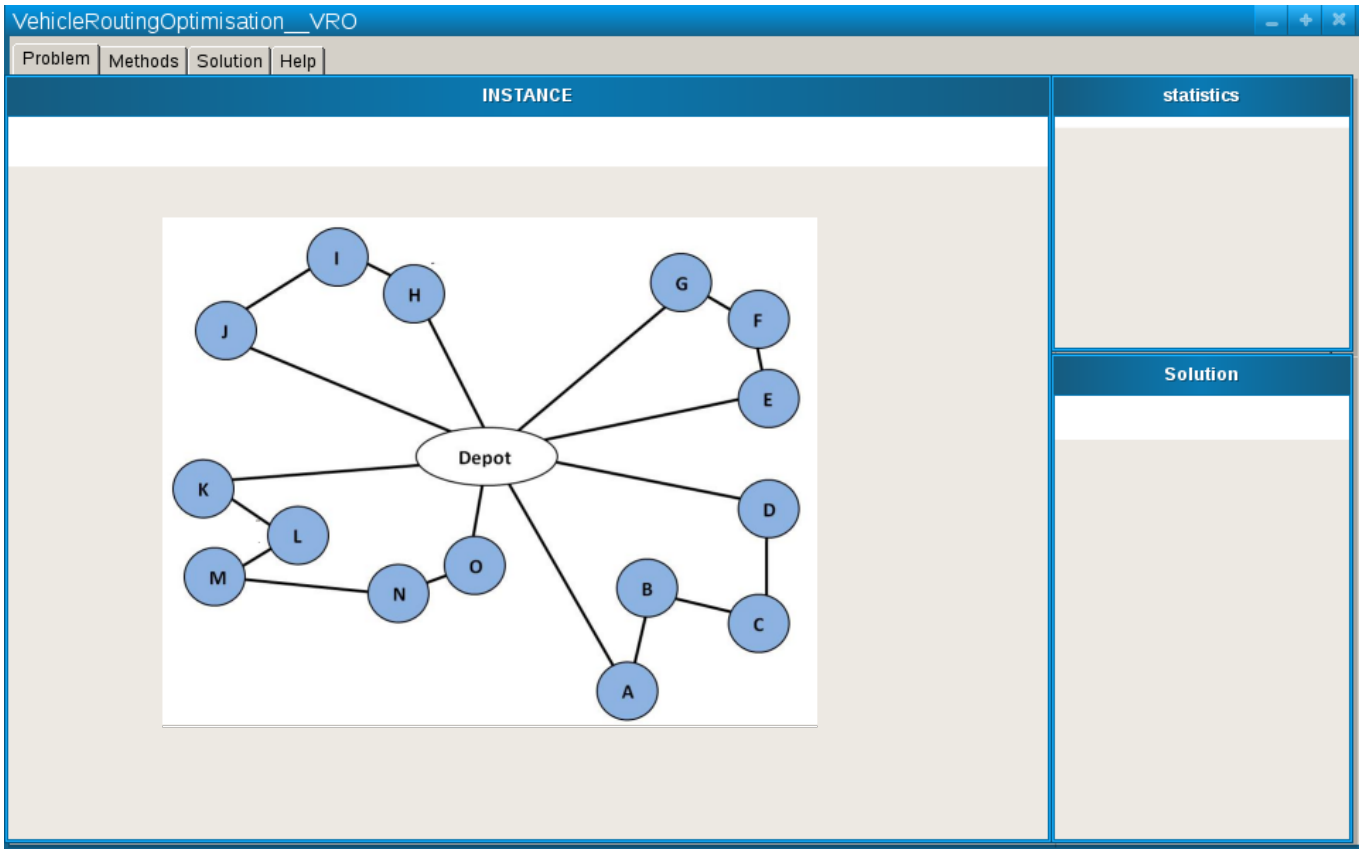

Figure 7. Interface Graphique.

Description de l'interface graphique :

- **Instance :**
  fenêtre représentant le graphe sur lequel les algorithmes d'optimisation vont s'appliquer, c'est la représentation graphique de l'instance.
- **statistics :**
  fenêtre contenant des informations sur les nœuds du graphe, détails sur la solution.
- **Solution :**
  fenêtre contenant le graphe solution, indiquant le chemin sélectionner par l'algorithme.
- **Menu :**

  1. Problèm : contient les fonctionnalités suivantes :
     - Charger une instance.
     - Générer une instance.
     - sauvegarder les résultats.
     - Quitter.

2. Methods :

   Sélectionner une méthode de résolution, on peut choisir soit la résolution via le modèle implémenté en Gurobi (Solver), ou la méthode heuristique via une des heuristiques qui seront développées.

3. Solution :

   Permets d'effectuer des comparaisons sur les résultats, analysé la solution obtenue.

4. Help :

   contient une présentation sur l'application et ses différentes fonctionnalités ainsi qu'un tutoriel sur son utilisation.

3.1.2 Diagramme de cas d'utilisation :

Les diagrammes de cas d'utilisation sont des diagrammes UML utilisés pour donner une vision globale du comportement fonctionnel d'un système logiciel. Voici une liste descriptive des différents cas d'utilisation

1. Chargé une instance :

   l'utilisateur peut charger une instance via un fichier de données contenant des informations sur le graphe et les nœuds, arêtes qui le composent.

2. Générer une instance :

   l'utilisateur peut demander la création d'une instance en donnant des paramètres (exemple : densité du graphe), l'application générera une instance.

3. Sauvegarder :

   Permets de sauvegarder une solution obtenue, une instance générer.

4. Charger une solution :

   Permets à l'utilisateur de charger une solution déjà sauvegardée, afin de faire une comparaison, ou de revoir les résultats.

5. Résoudre :

   lancer la résolution de l'instance via une des méthodes disponible, via le solver grâce au modèle, ou via une heuristique.

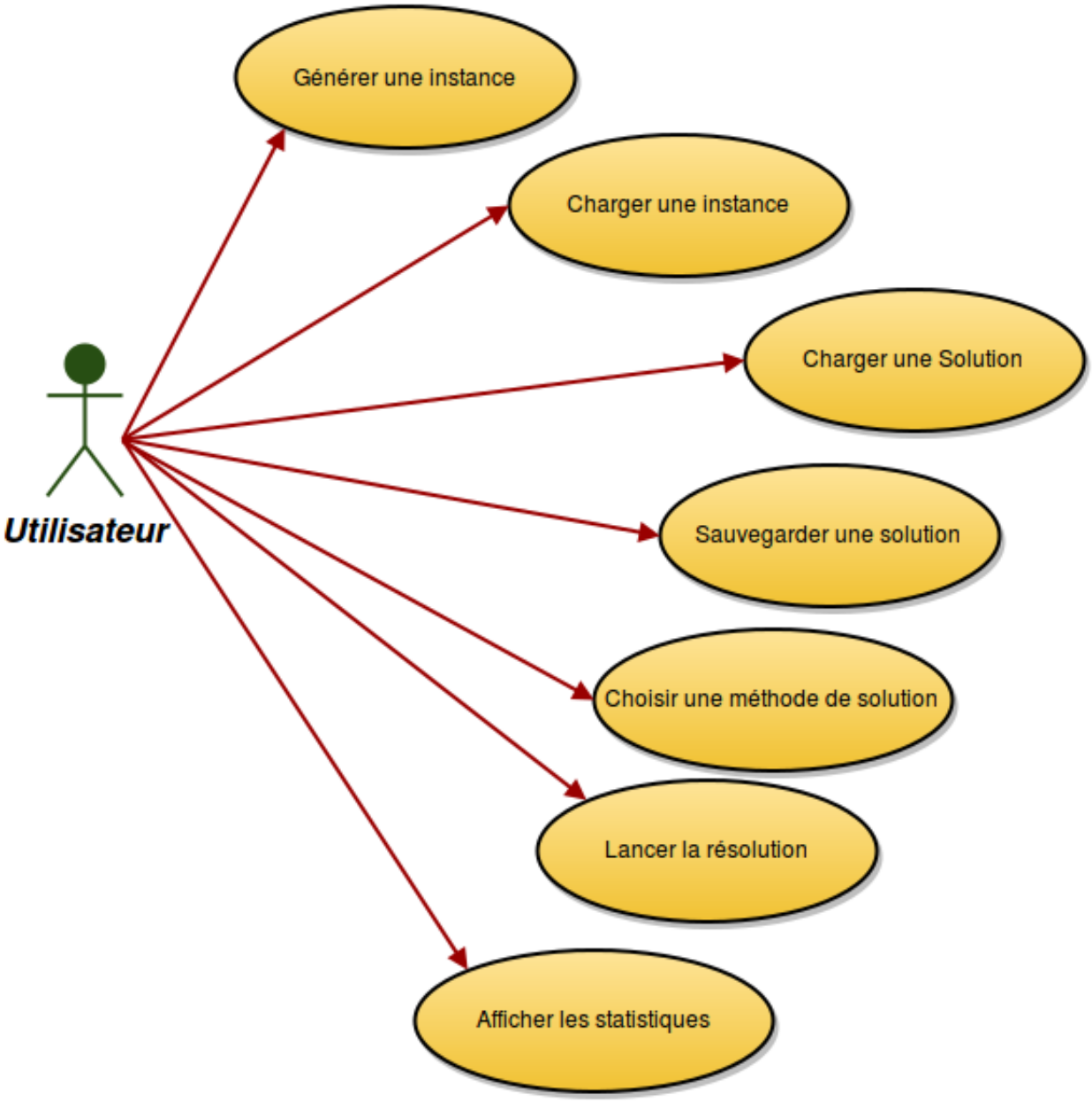

Figure 8. Diagramme de cas d'utilisation.

3.1.3 Diagramme de classe :

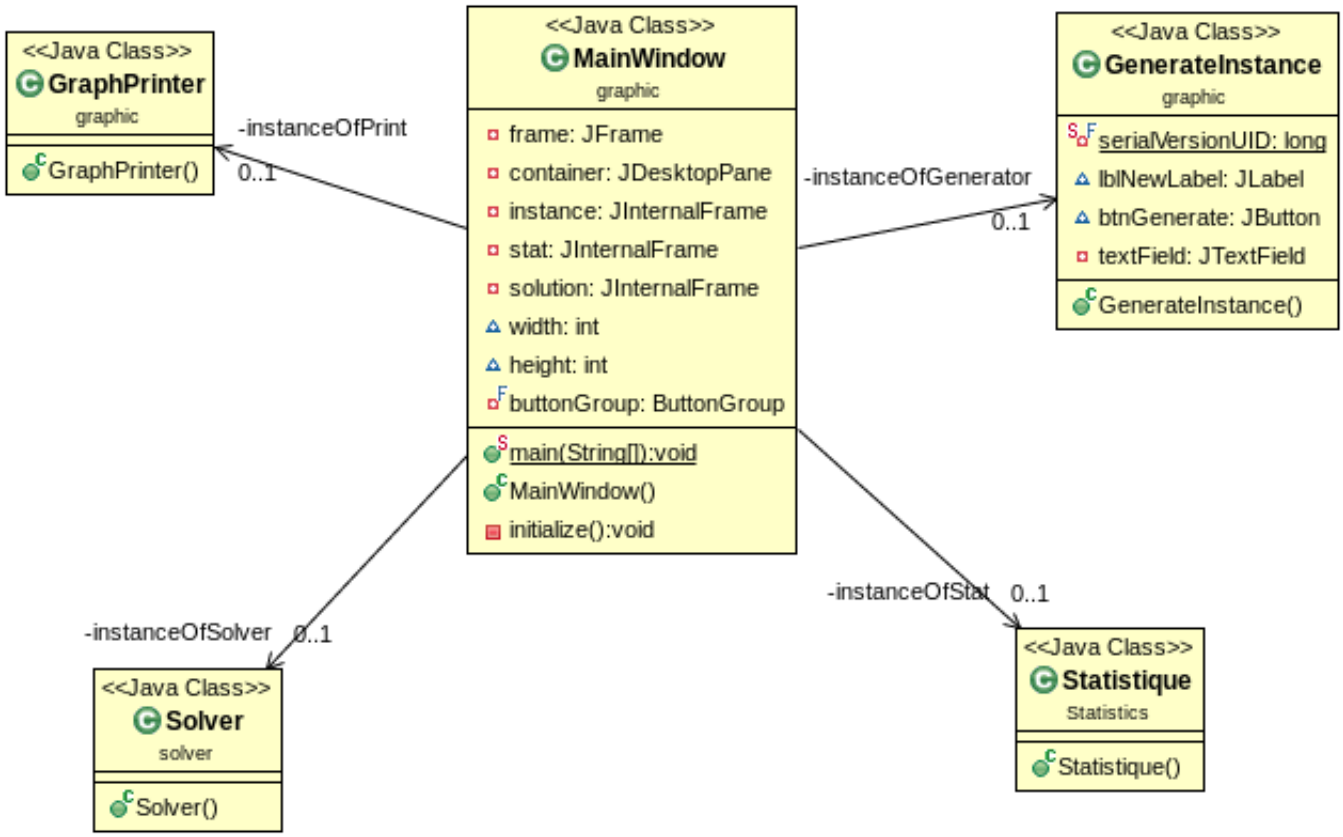

Figure 9. Interface Graphique.

1. **MainWindow :**
   Classe principale contenant le point d'entrée du programme (Main), elle initialise l'ensemble des instances nécessaires au fonctionnement du programme, c'est aussi la classe qui implémente l'interface graphique principale.
2. **GraphPrinter :**
   Classe utilisant l'API JgraphX afin de pouvoir afficher les graphes sur la fenêtre de l'application.
3. **Solver :**
   Classe contenant l'implémentation du modèle avec le solver codé via l'API Gurobi.
4. **Statistique :**
   Classe permettant des données une analyse sur la solution trouvée, elle sera implémentée au prochain semestre.
5. **GenerateInstance :**
   Classe permettant de gérer la génération d'instances, elle crée aléatoirement des données quelconques.

3.2 Mise en œuvre

3.2.1 outils et environnement utilisés :

Eclipse :

environnement de développement (IDE) particulièrement puissant. Il permet en particulier de développer en java et en C++, mais offre de nombreuses autres possibilités grâce à un système de plugins.

Linux mint 18 :

Système d'exploitation GNU/Linux libre, basé sur Ubuntu, Nous avons développé l'interface graphique de l'application sur cette distribution.

Windows 10 :

Système d'exploitation propriétaire développé par la société Microsof, nous avons implémenté le modèle avec gurobi sur cet environnement.

3.2.2 Langages utilisés :

Pour l'implémentation de l'application nous avons choisie d'utiliser le langage **Java SE**, qui est un langage compatible avec toutes les plateformes, très puissant pour la conception d'interface graphique. Java met aussi à disposition plusieurs API permettant d'effectuer des traitements spécifiques à plusieurs domaines tels que Gurobi.

API utilisés :

- **Swing :**
  Bibliothèque incluse dans la librairie standard de Java, elle permet la conception d'interfaces graphiques qui sont indépendants du système sur lequel l'application est lancée.
  Nous avons eu recours à cette bibliothèque car elle dispose d'une documentation conséquente, mais aussi du fait qu'on ait déjà eu à travailler avec durant d'autre projets.
- **JGraphX :**
  API qui permet la représentation des graphes sur une fenêtre en Java, en offrant plusieurs méthodes de dessin sur les éléments d'un graphe, telle que le nœud, l'arc, etc ...
- **Gurobi :**
  Gurobi est un API mais aussi un solveur capable de résoudre les programmes linéaires en nombres entiers.
  À la différence des autres solveur Gurobi est gratuit pour les étudiants ; c'est la raison que notre choix est porté sur Gurob.

3.3 Travail Réalisé :

Nous détaillons ici les implémentations réalisées durant le premier semestre

3.3.1 Interface graphique :

Voici une capture d'écrans de l'interface graphique telle qu'implémenté :

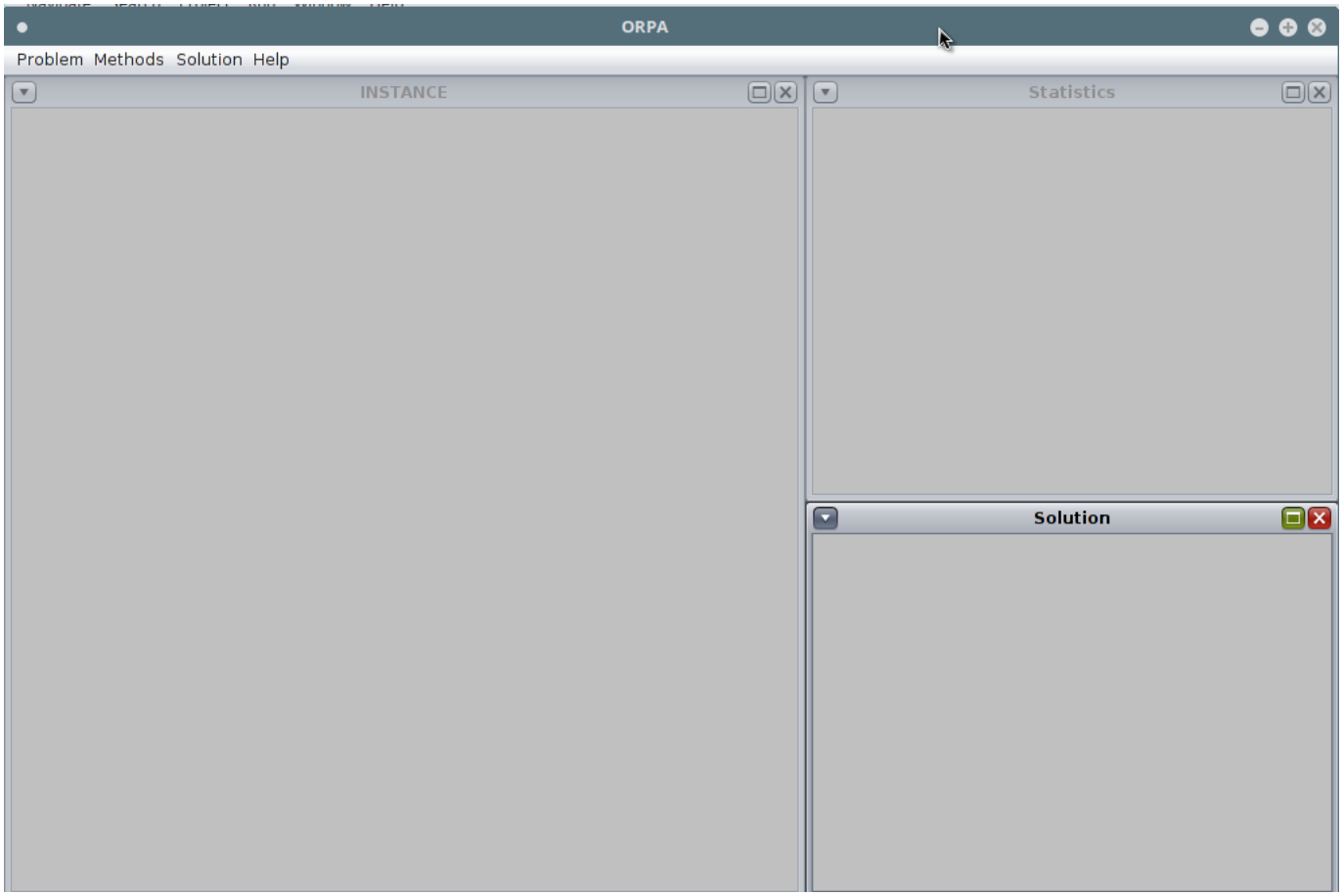

Figure 10. Interface Graphique.

L'IU est composé de trois sous-fenêtres et d'une barre de menus donnant l'accès aux fonctionnalités de l'application.

les fonctionnalités implémentés :

- redimensionnement des fenêtres :
  les fenêtres de l'application s'adaptent automatiquement à la taille de la fenêtre principale après redimensionnement.
- génération d'instances :
  la génération d'instances se fait via un script R (développer par Luis Flores) pour le moment, qui est lancée via l'application, il se charge de générer des fichiers data et un fichier PDF contenant les graphes.
  chaque fichier data correspond aux données d'une instance.

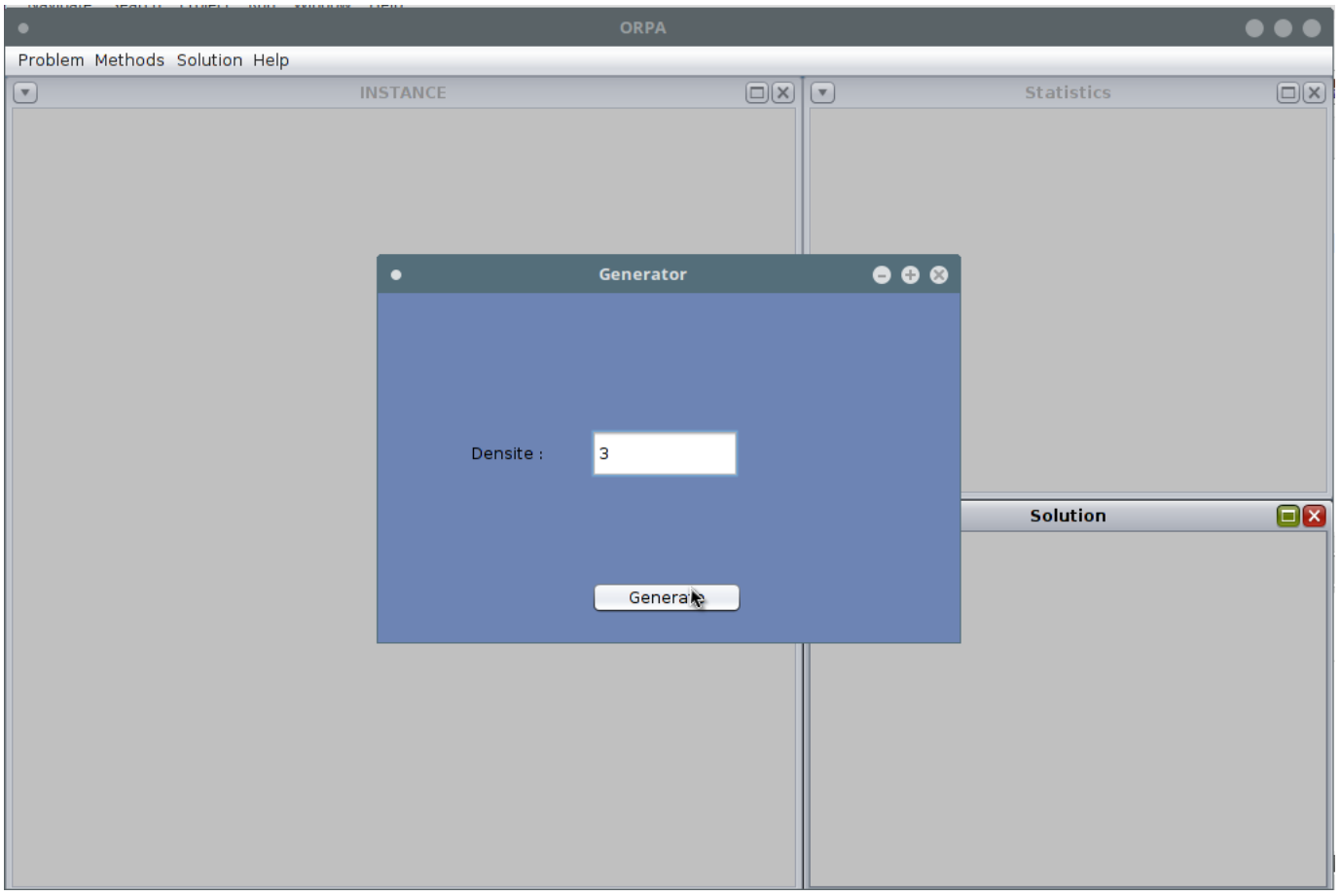

Figure 11. Générateur d'instance.

Nom	Taille	Type	Date de m
ejemplo_5_80_100_1.dat	5,8 ko	document texte brut	lun. 09 janv
ejemplo_5_80_100_2.dat	5,8 ko	document texte brut	lun. 09 janv
ejemplo_5_80_100_3.dat	5,8 ko	document texte brut	lun. 09 janv
ejemplo_5_80_100_4.dat	5,8 ko	document texte brut	lun. 09 janv
ejemplo_5_80_100_5.dat	5,8 ko	document texte brut	lun. 09 janv
ejemplo_5_80_100_6.dat	5,8 ko	document texte brut	lun. 09 janv
ejemplo_5_80_100_7.dat	5,8 ko	document texte brut	lun. 09 janv
ejemplo_5_80_100_8.dat	5,8 ko	document texte brut	lun. 09 janv
ejemplo_5_80_100_9.dat	5,8 ko	document texte brut	lun. 09 janv
ejemplo_5_80_100_10.dat	5,8 ko	document texte brut	lun. 09 janv
generateInstance.R	3,2 ko	document texte brut	lun. 21 nov
installLib.R	85 octets	document texte brut	lun. 21 nov
Rplots.pdf	13,8 ko	document PDF	lun. 09 janv

Figure 12. Résultats du générateur.

3.3.2 Implémentation du modèle en Gurobi :

L'implémentation du modèle se fait en Gurobi du fait qu'il nous permet de résoudre un modèle dans un langage (python, java, c++....), mais aussi le fait que la licence est gratuite pour les étudiants.

Le modèle a été développé en Java afin de permettre une meilleure lisibilité du programme, nous avons séparé l'implémentation des contraintes et de la fonction objective en plusieurs fonctions.

3.3.2.0.1 Entrées : Si on se base des données suivantes pour tester l'implémentation de notre modèle :

- $$T = 7$$
- $$M = 2$$
- $$R = 10$$
- $$t_{ij} = \begin{pmatrix} 0 & 3 & 0 & 0 & 0 & 3 \\ 3 & 0 & 3 & 0 & 0 & 2 \\ 0 & 3 & 0 & 2 & 0 & 3 \\ 0 & 0 & 2 & 0 & 2 & 3 \\ 0 & 0 & 0 & 2 & 0 & 3 \\ 3 & 2 & 3 & 3 & 3 & 0 \end{pmatrix}$$
- $$d_{ij} = \begin{pmatrix} 100 & 300 & 200 & 400 & 400 & 400 \\ 300 & 0 & 3 & 5 & 6 & 7 \\ 200 & 3 & 0 & 2 & 3 & 3 \\ 400 & 5 & 2 & 0 & 2 & 3 \\ 400 & 6 & 2 & 2 & 0 & 3 \\ 400 & 7 & 3 & 3 & 3 & 0 \end{pmatrix}$$
- $$\alpha_{ij} = \begin{pmatrix} 0 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0 & 0.01 & 0.01 & 0.01 \\ 0.1 & 0.1 & 0.1 & 0 & 0.1 & 0.1 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0 \end{pmatrix}$$
- $$r_i = \{0.4, 0.3, 0.2, 0.7, 0.5, 0.3\}$$

3.3.2.0.2 Sorties : Les résultats obtenus sont :

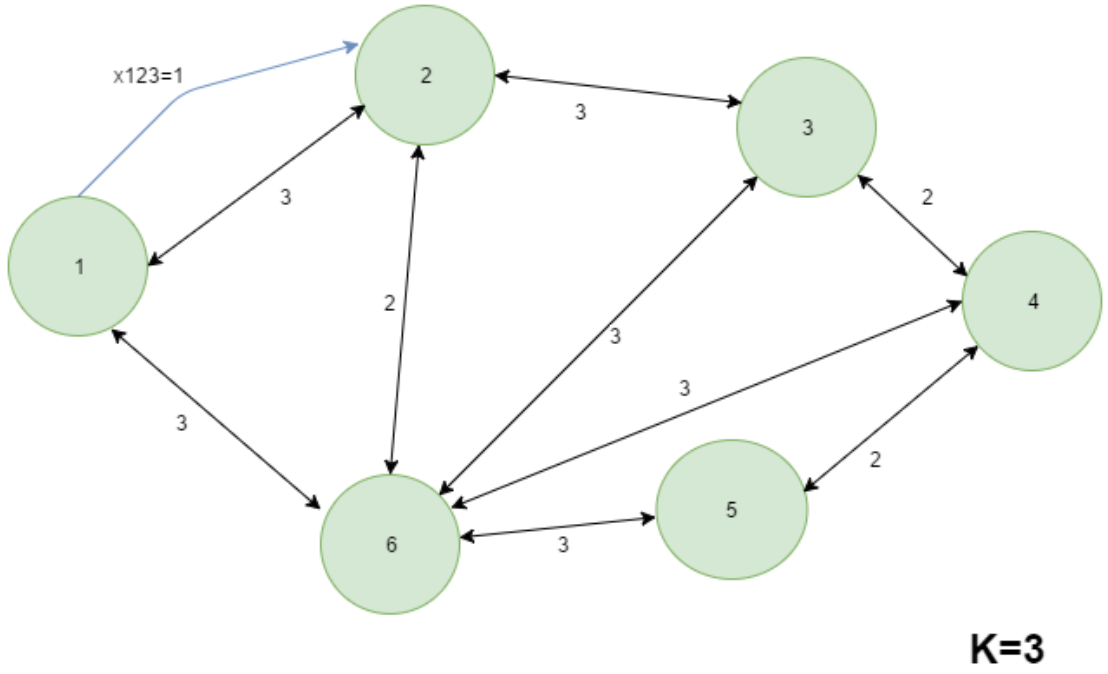

Figure 13. chemin pris à l'instant 3.

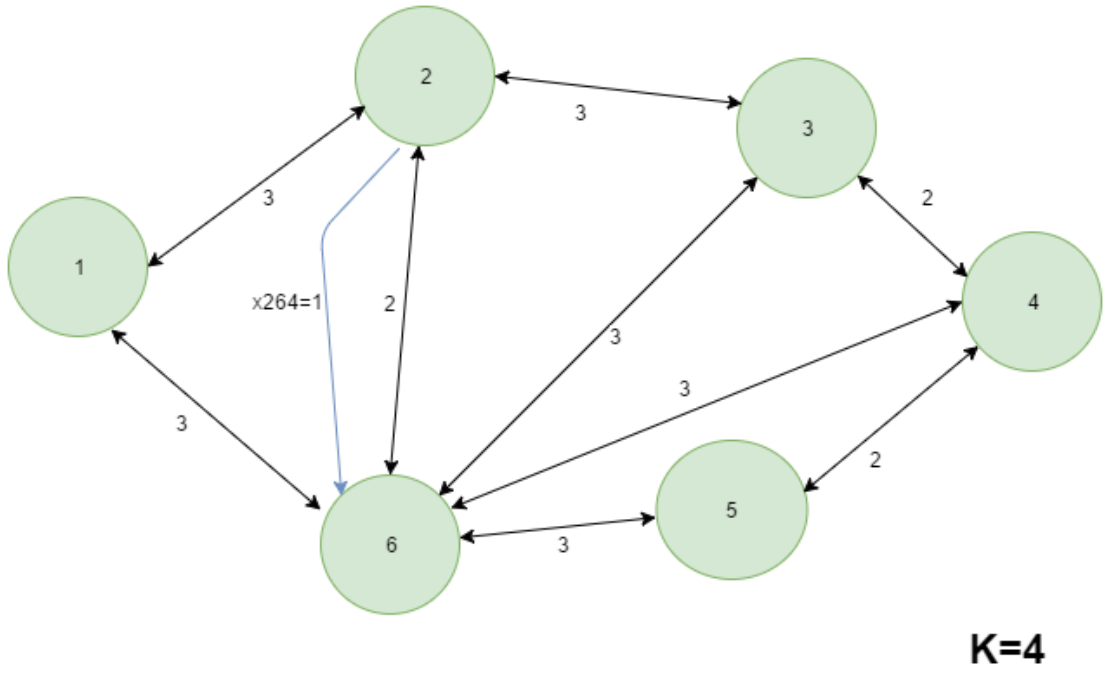

Figure 14. chemin pris à l'instant 4.

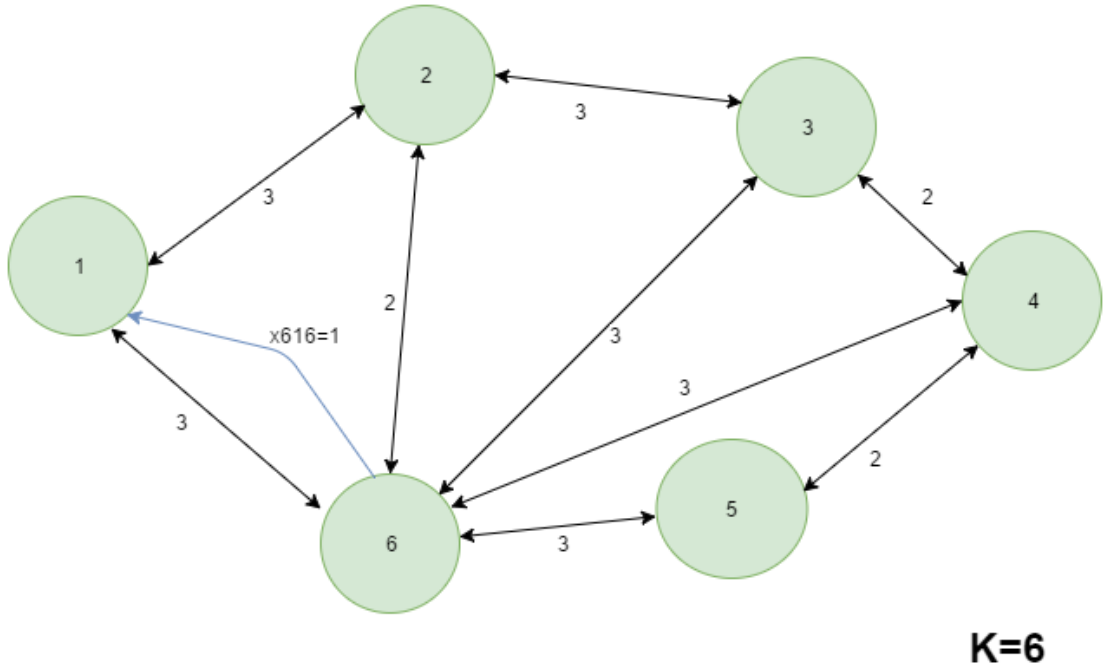

Figure 15. chemin pris à l'instant 6.

3.3.2.0.3 Commentaires : Nous remarquons qu'il retourne à l'instant T à la base. Mais nous remarquons des incohérences, en effet pour aller du sommet 2 au sommet 6 cela devrait lui coûter deux unités de temps et comme on le voit dans la figure 14 cela lui coûte 1 unité de temps.

Mais aussi nous avons des incohérences au niveau de la variable z_{ik} ou l'ensemble des variables sont égales à 1, à l'exception de :

- $z_{11} = 0$
- $z_{12} = 0$
- $z_{13} = 0$
- $z_{23} = 0$
- $z_{14} = 0$
- $z_{64} = 0$
- $z_{15} = 0$
- $z_{16} = 0$

Les incohérences au niveau du z_{ij} va rendre l'ensemble du modèle faux, puisque l'ensemble des contraintes utilisent cette variable pour la résolution du problème.

Il manque le résultat des autres variables, que n'ont pas mis du fait des incohérences.

3.4 Objectifs non atteints :

Plusieurs objectifs fixés pour le premier semestre n'ont pas été atteints, pour plusieurs raisons parmi elles :

- **Manque de temps :**
  Durant ce semestre nous avons eu beaucoup de projets à gérer, mais aussi de nombreux travaux pratiques à remettre.
- **Mauvaise gestion :**
  Une meilleure gestion des projets, Tp, examens durant ce semestre nous airaient permis d'avoir plus de temps et d'avancée plus sur ce projet.
- **Mauvaise coordination :**
  Une meilleure synchronisation de notre équipe de projet aurait permis de mieux intégrés les fonctionnalités.

Les objectifs prévus pour le premier semestre qui n'ont pas été atteints sont les suivants :

1. **Affichage des graphes :**
   La fonctionnalité permettant de dessiner les graphes sur les fenêtres n'ont pas été implémentés.
2. **Intégration du Solver :**
   L'interface graphique ne communique pas avec le solver, l'implémentation du générateur n'étant pas intégrée correctement, notamment la récupération des données de l'instance générée, la communication de ces données ne peut être faite au solver.
3. **Implémentation du Solver :**
   l'implémentation du solver a été effectuée mais lors de la résolution d'un exemple simple, des erreurs sont apparus que nous n'arrivons pas encore à expliquer.

4 Perspectives

Durant ce premier semestre plusieurs objectifs non pas été atteints, et d'autres ne sont pas réalisés comme nous le souhaitons, pour les prochains semestres voici quelques perspectives :

- **Générateur d'instance :**
  1. le générateur doit pouvoir créer une instance et récupérer les données de l'instance.
  2. charger une instance à partir d'un fichier de données.
- **Affichage des graphes :**
  cette fonctionnalité n'étant pas implémentée durant le premier semestre, nous devrons réaliser son implémentation de manière à ce qu'elle puisse afficher un graphe sur la fenêtre, mais qu'elle puisse aussi donnés accès à des fonctionnalités telle que :
  1. Sélection d'un nœud du graphe, et affichage des informations le concernant.
  2. Possibilité pour l'utilisateur de créer une instance, en ajoutant des nœuds et des routes via l'interface graphique.
- **Implémentation du solveur :**
  1. Trouver l'origine des incohérences
  2. Analyser le résultat des exemples dont on a donné comme entré au solveur

Bibliographie

[1] S. Raghavan Bala Chandran. *Modeling and Solving the Capacitated Vehicle Routing Problem on Trees*. PhD thesis, University of Maryland ,University of California. URL https://pdfs.semanticscholar.org/df49/d9948416f0dbe7277bbccae6aacb37272d90.pdf.

[2] Martin W.P. Savelsbergh Daniele Vigo Jean-François Cordeau, GGilbert Laporte. *Vehicle Routing*. PhD thesis. URL http://dis.unal.edu.co/~gjhernandezp/TOS/ROUTING/VRP1.pdf.

[3] P. Michelon E. Ocaña Luis Flores Luyo, R. Figueiredo. *Vehicle routing for the communication of time-dependent information*. PhD thesis, Universidad Nacional de Ingenieria, Lima-Perú IMCA Instituto de Matemáticas y Ciencias Afines, Université d'Avignon, Avignon-France LIA Laboratoire Informatique d'Avignon. URL http://roadef2016.utc.fr/papiers/ROADEF_2016_paper_236.pdf.

[4] Jens Lysgaard. *Clarke and Wright's Savings Algorithm*. PhD thesis. URL http://pure.au.dk/portal-asb-student/files/36025757/Bilag_E_SAVINGSNOTE.pdf.

[5] Gábor Nagy Niaz A. Wassan. *Vehicle Routing Problem with Deliveries and Pickups : Modelling Issues and Meta-heuristics Solution Approaches*. PhD thesis, University of Kent. URL http://www.sersc.org/journals/IJT/vol2_no1/6.pdf.

[6] MICHEL GENDREAU OLLI BRÄYSY. *Vehicle Routing Problem with Time Windows, Part I : Route Construction and Local Search Algorithms*. PhD thesis, Université de Montréal. URL http://cepac.cheme.cmu.edu/pasi2011/library/cerda/braysy-gendreau-vrp-review.pdf.