



UNIVERSITÉ D'AVIGNON  
ET DES PAYS DE VAUCLUSE

C E N T R E  
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

19 mai 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

Titre . . . . .	1
Sommaire . . . . .	2
1 Introduction . . . . .	3
2 Présentation générale . . . . .	3
2.1 Contexte . . . . .	3
2.2 Définition du problème . . . . .	4
2.3 Domaine d'applications . . . . .	7
2.4 Objectifs du projet . . . . .	7
2.5 Organisation du projet . . . . .	7
2.5.1 Planning prévisionnel . . . . .	8
2.5.2 Partage des tâches . . . . .	9
3 Modèle vs heuristique . . . . .	9
3.1 Modèle . . . . .	9
3.2 Heuristique . . . . .	10
4 Modèle . . . . .	10
4.1 Constantes . . . . .	10
4.2 Variables . . . . .	11
4.3 Objective fonctions . . . . .	11
4.4 Contraintes . . . . .	12
4.4.1 Contraintes sur la route du véhicule . . . . .	12
4.4.2 Contrainte sur le transfert d'information . . . . .	12
4.4.3 Contrainte sur la quantité d'informations . . . . .	13
4.4.4 Inégalité valable . . . . .	13
4.5 Incohérence du modèle . . . . .	13
4.6 Modèle correction . . . . .	14
5 Heuristique . . . . .	14
5.1 Heuristique 1 . . . . .	14
5.2 Heuristique 2 . . . . .	17
6 Comparaison des méthodes de Résolutions . . . . .	18
6.1 Entrées . . . . .	18
6.2 Résultat . . . . .	18
6.2.1 Modèle . . . . .	18
6.2.2 Heuristique 1 . . . . .	18
6.3 Heuristique 2 . . . . .	18
6.4 Comparaison . . . . .	18
7 Mise en œuvre et réalisations . . . . .	19
7.0.1 outils et environnement utilisés : . . . . .	19
7.0.2 Langages utilisés : . . . . .	19
7.1 Réalisations . . . . .	19
7.1.1 Interface graphique : . . . . .	19
7.2 Objectifs non atteints : . . . . .	24
8 Perspectives . . . . .	25

## 1 Introduction

En informatique, le problème du voyageur de commerce, ou problème du commis voyageur, est un problème d'optimisation qui, étant donné une liste de villes, et des distances entre toutes les paires de villes, détermine un plus court chemin qui visite chaque ville une et une seule fois et qui termine dans la ville de départ.

Malgré la simplicité de son énoncé, il s'agit d'un problème d'optimisation pour lequel on ne connaît pas de méthode de résolution permettant d'obtenir des solutions exactes en un temps raisonnable pour de grandes instances (grand nombre de villes) du problème. Pour ces grandes instances, on devra donc souvent se contenter de solutions approchées, car on se retrouve face à une explosion combinatoire.

Le problème du voyageur de commerce un problème algorithmique célèbre, qui a généré beaucoup de recherches et qui est souvent utilisé comme introduction à l'algorithmique ou à la théorie de la complexité. Il présente de nombreuses applications que ce soit en planification et en logistique, ou bien dans des domaines plus éloignés comme la génétique (en remplaçant les villes par des gènes et la distance par la similarité).

Le problème de tournées de véhicules est une classe de problèmes de recherche opérationnelle et d'optimisation combinatoire. Il s'agit de déterminer les tournées d'une flotte de véhicules afin de livrer une liste de clients, ou de réaliser des tournées d'interventions (maintenance, réparation, contrôles) ou de visites (visites médicales, commerciales, etc.). Le but est de minimiser le coût de livraison des biens. Ce problème est une extension classique du problème du voyageur de commerce, et fait partie de la classe des problèmes NP-complet.

Nous avons mis 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.

L'application développée durant le projet nous permet 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'obtention de résultats, leurs analyses et leurs comparaisons afin de déterminer l'efficacité des algorithmes de résolution développée.

## 2 Présentation générale

### 2.1 Contexte

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 de la tournée de véhicule (*Vehicle Routing Problem* (VRP)), De plus près 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  $V = \{1, 2, \dots, n\}$  de  $n$  villes et un ensemble  $A = \{(i, j) \mid i, j \in V\}$  de chemins directs 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 vu comme une généralisation du problème du voyageur de commerce.

Comme pour le problème du voyageur de commerce, le problème VRP [?] est un problème NP-Complet [], c'est-à-dire un problème pour lequel les méthodes exactes de solutions connues sont exponentielles dans la taille du problème. Ces méthodes sont donc inexploitable 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 [?], qui permet de résoudre le problème de tournée de véhicule, avec un dépôt central et avec un nombre de véhicules non fixés. Mais aussi on note les heuristiques de recherche locale et certaines méta-heuristiques permet 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)* [?]. 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 véhicule : 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)* [?]. 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)* [?]. 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* [?]. On cherche à maximiser dans ce problème la quantité d'informations 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.

## 2.2 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).

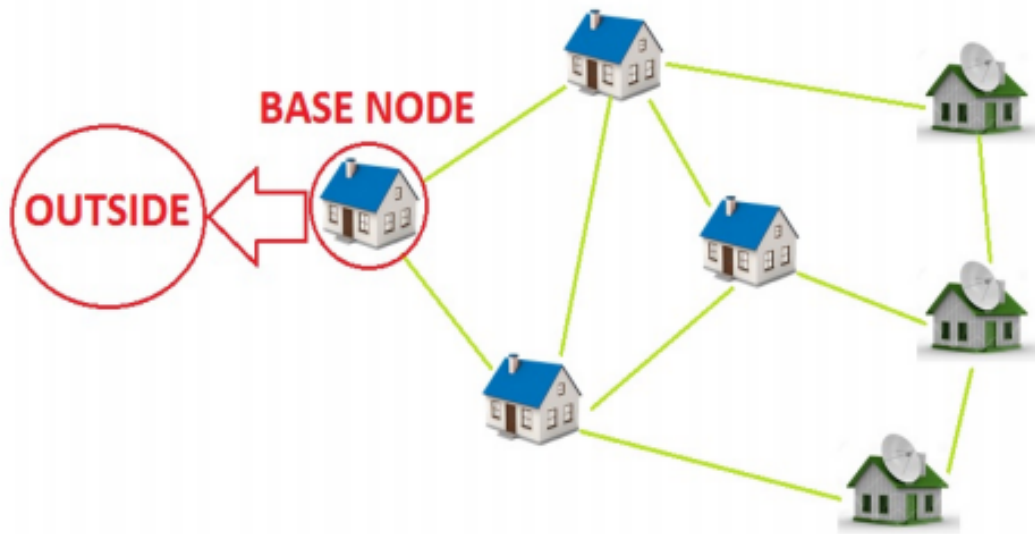


Figure 1. Environnement.

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.

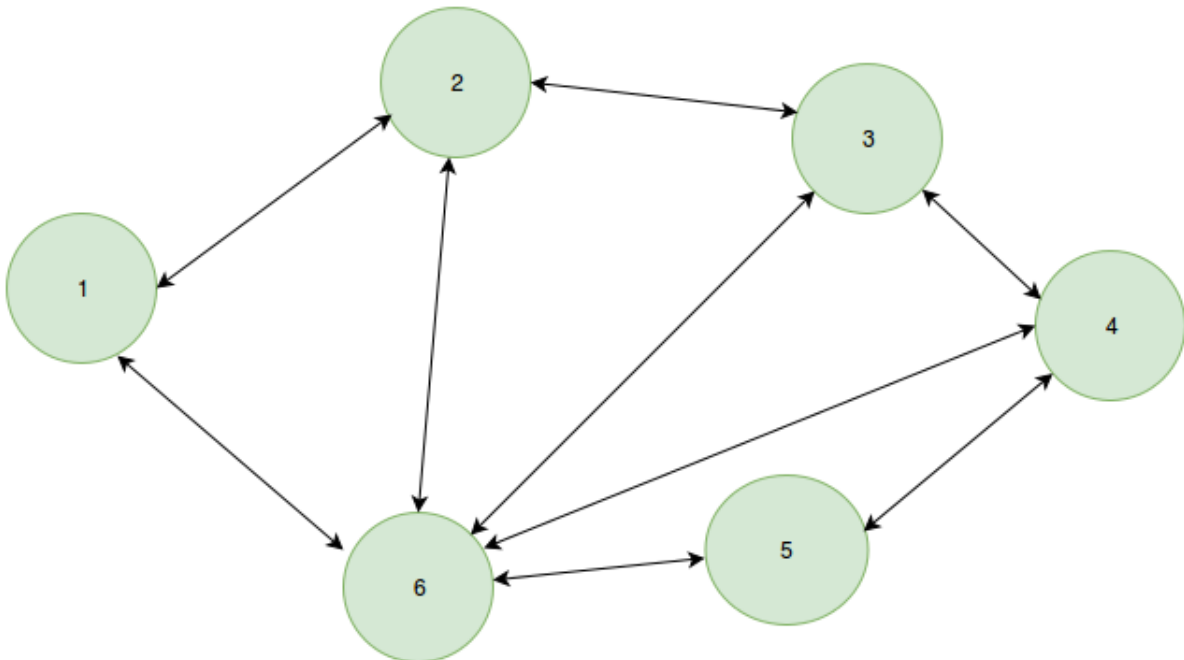
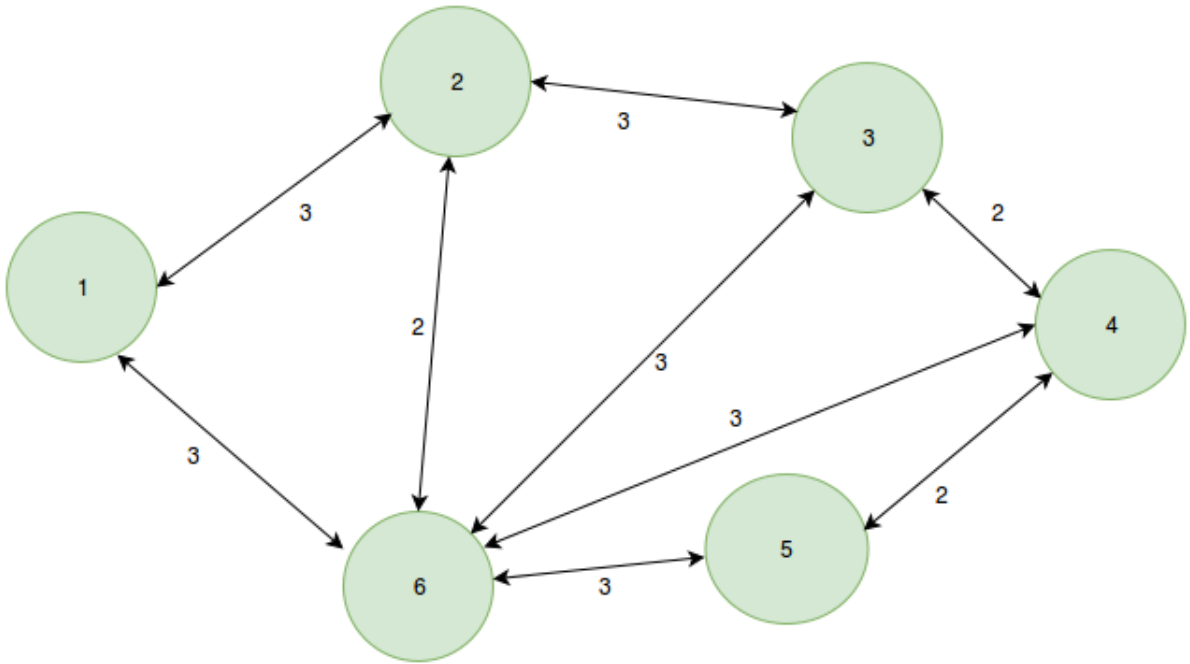


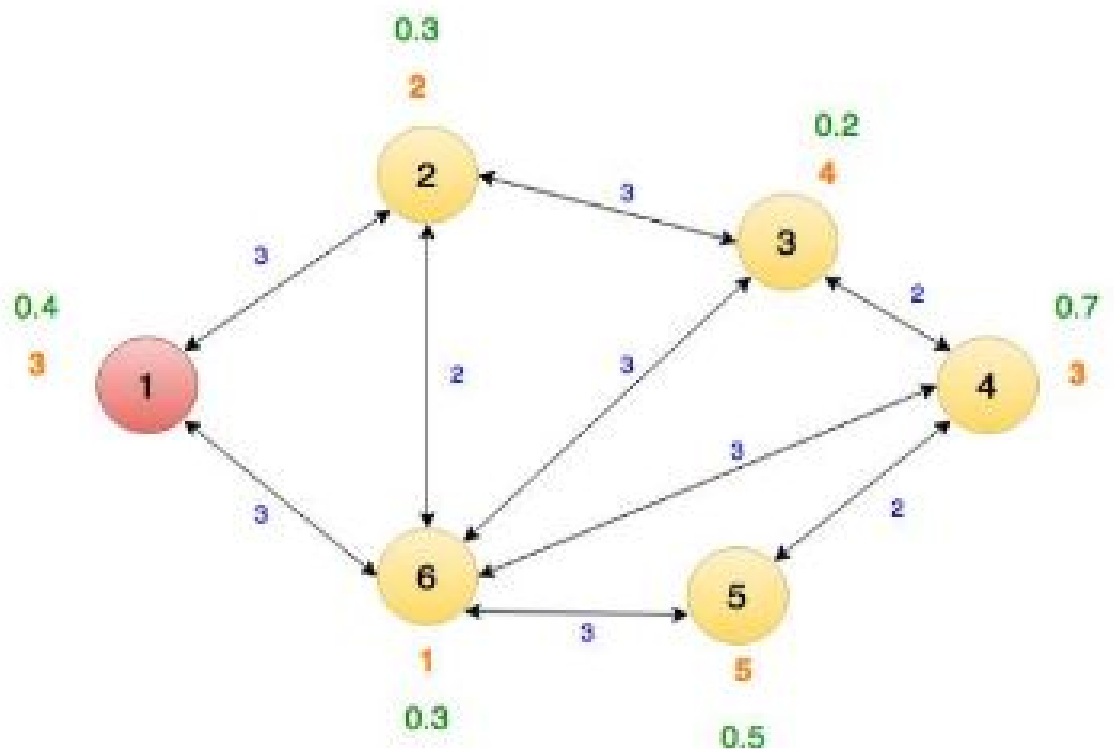
Figure 2. Exemple graphe non orienté.

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



**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.

### 2.3 Domaine d'applications

Les algorithmes de calcul de tournées sont utilisés dans les moteurs des logiciels d'optimisation de tournées. Ces solutions sont utilisées par des entreprises qui souhaitent rationaliser leur flotte de véhicules, réduire leurs coûts ou encore optimiser l'occupation de leur personnel mobile.

La fonction d'optimisation de tournée peut aussi être intégrée dans des solutions de planification de ressources mobiles. Ces solutions ont pour vocation de planifier des tâches ou missions avec ou sans prise de rendez-vous, et de les répartir entre les ressources en fonction de leurs contraintes (disponibilité, localisation, compétences requises, durées d'interventions, etc.) Les entreprises concernées par l'optimisation de tournées peuvent appartenir aux secteurs d'activités suivants :

- Livraison de biens à des entreprises ou à des particuliers : transporteurs, messageries (distribution de presse), grandes surfaces (livraison de marchandises des entrepôts aux magasins, livraison et installation d'équipement à domicile, etc.) ;
- Réparation et maintenance d'équipements de particuliers (électroménager, chaudières, informatique, etc.), d'équipements collectifs (ascenseurs, tapis roulants) ou d'entreprises (distributeurs automatiques, équipements industriels, informatique, etc.) ;
- Interventions d'expertises, de contrôle, d'audit (certification, prélèvements, etc.).
- Tournée d'affichage (publicitaire, campagne d'élection,...)

### 2.4 Objectifs du projet

A la fin de l'année, l'objectif est 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 informations concernant l'instance à résoudre.
- Afficher l'instance générée 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 informations sur la solution générée.
- Sauvegarder les solutions produites.
- Produire une comparaison des solutions générées.

### 2.5 Organisation du projet

### 2.5.1 Planning prévisionnel


<div>  </div>		
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
• Implémentation 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'UI	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 modèle	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
• Implémentation 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 Gantt.

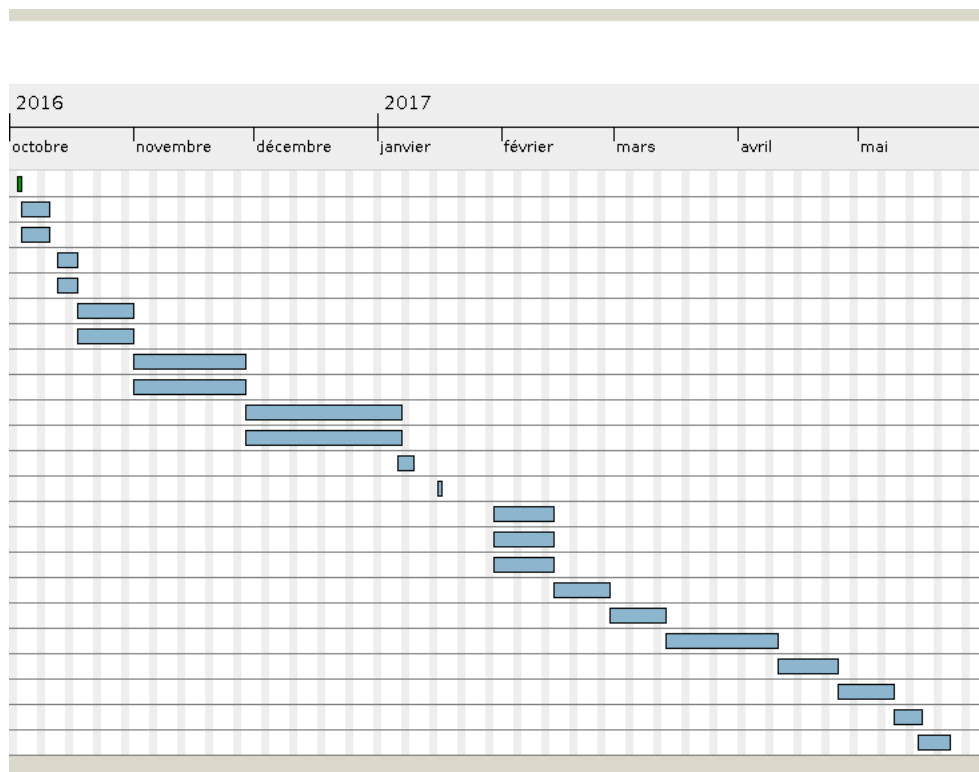


Figure 6. Diagramme de Gantt.



### 2.5.2 Partage des tâches

Pour atteindre nos objectifs du 2ème semestre, nous avons partagé les tâches de la manière suivante :

- **Omar Sy :**

1. Travailler sur la correction du modèle : après implémentation du modèle durant le 1er semestre, des incohérences ont été observées sur les résultats obtenus sur la résolution d'instance.
2. Présentation des méthodes heuristiques : détaillé le fonctionnement des méthodes heuristiques et l'intérêt de leurs utilisations dans la résolution des problèmes du type NP-Complet.
3. Implémentation des heuristiques
4. Implémentation du modèle

- **MESSAS Kouseila :**

1. Travaillé sur la correction du modèle : travailler sur le modèle afin de comprendre la source du problème des incohérences données par le modèle lors de la résolution d'instance.
2. Présentation des méthodes heuristiques : détaillé le fonctionnement des méthodes heuristiques et l'intérêt de leurs utilisations dans la résolution des problèmes du type NP-Complet.
3. communication entre interface graphique et heuristiques : permettre de lancer la résolution d'une instance via l'interface graphique en appelant une des méthodes heuristiques.
4. générateur d'instance : complété l'implémentation du générateur d'instance en proposant une solution en java inclus dans l'application.
5. affichage graphique des instances générées : implémenté la fonctionnalité qui permet de dessiner sur une fenêtre un graphe représentant une instance.
6. affichage graphique des solutions produites : implémenté la fonctionnalité qui permet de dessiner sur une fenêtre la solution produite par les différentes méthodes de résolution incluse dans l'application.
7. affichage graphique des détails sur les solutions produites : implémenté la fonctionnalité qui permet d'afficher sur la fenêtre "informations" les détails de la solution produite par une des méthodes de résolution.
8. Comparateur de solution : implémenté la fonctionnalité qui permet de comparer les solutions produites par les différentes méthodes de résolution

## 3 Modèle vs heuristique

### 3.1 Modèle

En optimisation combinatoire nous avons plusieurs méthodes de résolution exactes et à la différence des autres méthodes elles permettent d'obtenir une ou plusieurs solutions optimales.

Nous pouvons citer parmi ces méthodes l'algorithme du simplexe qui permet d'obtenir la solution optimale d'un problème en parcourant la fermeture convexe de l'ensemble de recherche (ensemble des solutions admissibles) et ce en passant de sommet en sommet. Malgré une complexité mathématique dans le pire des cas non polynomiale, il permet de résoudre la plupart des problèmes rapidement. Cependant il ne peut s'appliquer qu'aux problèmes ayant la propriété de convexité c'est-à-dire aux problèmes en variables continues ou à des problèmes en variables entières ayant une matrice des contraintes  $T$  unimodulaire (car dans ce cas, tous les sommets de l'ensemble de recherche sont entiers) comme les problèmes de transport ou d'affectation.

Pour les autres problèmes (ILP, MILP, 0-1ILP), il existe plusieurs méthodes : la programmation dynamique consistant à placer le problème dans une famille de problèmes de même nature mais de

difficulté différente puis à trouver une relation de récurrence liant les solutions optimales de ces problèmes. Le Branch & Bound consistant à faire une énumération implicite en séparant le problème en sous problèmes et en évaluant ceux-ci à l'aide d'une relaxation (continue ou lagrangienne principalement) jusqu'à ne plus avoir que des problèmes faciles à résoudre ou dont on sait avec certitude qu'ils ne peuvent pas contenir de solution optimale. Les méthodes polyédrales consistant à ajouter progressivement des contraintes supplémentaires afin de ramener le domaine des solutions admissibles à un domaine convexe (sans en enlever la ou les solutions optimales bien évidemment). Ces méthodes sont générales et demandent souvent une particularisation vis-à-vis d'un problème spécifique. Il existe aussi des applications génériques (AMPL, CPLEX, LINDO, MPL, OMP, XPRESS, GUROBI...) permettant de résoudre l'ensemble des problèmes pouvant s'écrire sous la forme algébrique d'un problème en variables binaires, entières ou mixtes.

Il faut aussi noter que la méthode consistant à effectuer une énumération explicite de toutes les solutions (c'est-à-dire de les tester une à une, méthode envisageable pour tous les problèmes à variables à valeurs bornées) montre très vite ses limites dès que le nombre de variables augmente puisque sa complexité est en  $k^n$  où  $k$  représente le nombre de valeurs que peut prendre une variable et  $n$  le nombre de variables du problème.

### 3.2 Heuristique

Dans certaines situations, où nous avons des contraintes de ressources (temps de calcul et/ou mémoire) limitées, il est nécessaire de disposer d'une solution de bonne qualité (c'est-à-dire assez proche de l'optimale). Dans ce cas l'optimalité de la solution ne sera pas garantie, ni même l'écart avec la valeur optimale. Cependant, le temps nécessaire pour obtenir cette solution sera beaucoup plus faible et pourra même être fixé (bien évidemment dans ce cas la qualité de la solution obtenue dépendra fortement du temps laissé à l'algorithme pour l'obtenir).

Typiquement ce type de méthodes, dites heuristiques est particulièrement utile pour les problèmes nécessitant une solution en temps réel (ou très court) ou pour résoudre des problèmes difficiles sur des instances numériques de grande taille. Elles peuvent aussi être utilisées afin d'initialiser une méthode exacte (Branch & Bound par exemple).

Parmi ces méthodes, il faut distinguer les heuristiques ciblées sur un problème particulier et les méta-heuristiques plus puissantes et adaptables pour résoudre un grand nombre de problèmes. Cependant une méta-heuristique, pour être suffisamment performante sur un problème donné nécessitera une adaptation plus ou moins fine.

Ces méthodes approchées peuvent se classer en différentes catégories :

- Constructives (algorithmes glouton, méthode Pilote, GRASP)
- Recherche locale (algorithmes de descente, multi-départs, recuit simulé, algorithme à seuil, recherche Tabou, méthode de bruitage)
- Évolutionnistes (algorithmes génétiques, algorithmes d'évolution, recherche dispersée, méthode des chemins, systèmes de fourmis)
- Réseaux de neurones (Modèle de Hopfield-Tank, machine de Boltzmann, réseau auto-adaptatif, réseau élastique)
- Heuristiques Bayésiennes (optimisation globale, optimisation discrète)
- Superposition (perturbation des données, perturbation des paramètres d'une heuristique).

## 4 Modèle

### 4.1 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 de la station  $i$  :

$$r_i$$

- Le temps de la simulation :

$$T$$

- La quantité d'information maximale qu'une station peut récupérer 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  $j$  :

$$\alpha_{ij}$$

## 4.2 Variables

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

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

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

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

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

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

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

$$q_{ik}$$

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

$$f_{ijk}$$

## 4.3 Objective fonctions

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

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.

#### 4.4 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'envoi 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.

##### 4.4.1 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 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  $j$  :

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

##### 4.4.2 Contrainte sur le transfert d'information

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

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

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  $j$  :

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

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

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

#### 4.4.3 Contrainte sur la quantité d'informations

La quantité d'informations d'une station  $j$  à l'instant  $k+1$  doit être 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 :

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

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

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

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

#### 4.4.4 Inégalité valable

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

$$\Theta_{jik} \leq z_{ik}, \forall j \in V, \forall k \in T$$

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

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

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

#### 4.5 Incohérence du modèle

L'implémentation du modèle ne nous a pas donné les résultats attendus. Après plusieurs recherches sur GUROBI et plusieurs tests sur l'implémentation, nous en avons déduit que le problème ne pouvait venir que du modèle. Nous avons constaté que le problème, était lié aux contraintes de route qui est incohérente avec la fonction objective. En effet si on cherchait à maximiser les informations, le modèle serait juste mais du fait qu'on cherche à minimiser les informations présentes dans le graphe, cela fait que les contraintes qui sont chargées de tracer la route du véhicule (c'est-à-dire que le véhicule va d'un point vers un point  $j$  voisin de  $i$  à l'instant  $k$  et que à l'instant  $k-1$  le véhicule était au point  $i$ .) sont incohérentes dans l'ensemble.

## 4.6 Modèle correction

Pour respecter toutes les contraintes liés au problème nous allons modifier le modèle afin qu'il puisse nous des résultats exacts. Ainsi seulement les contraintes liés à la route du véhicule seront sujet à être modifié.

- Le véhicule se trouve à l'instant 1 à la base

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

- Le véhicule retourne à la base, à la fin de la simulation

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

- On n'empêche que le véhicule se trouve dans une  $j$  à l'instant et se déplace vers ce même station :

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

- Contrainte permettant d'assurer la cohérence du chemin que le véhicule prend :

$$\sum x_{ijk} \leq \sum x_{jp(k+t_{jp})} + z_{jk}, \forall (i,j) \in A, \forall (j,p) \in A, \forall k \in T$$

- Forcé, les chemins inexistantes soient égaux à zéro :

$$\sum_{(ij) \notin A} x_{ijk} = 0$$

## 5 Heuristique

Nous utilisons la méthode relax and fix afin de résoudre le problème du vehicular routing en effet ces algorithmes à la différence des modèles vont nous permettre de résoudre l'heuristique pour des grandes combinaisons.

Ce papier va nous permettre de décrire la méthode relax and fix et comment on va l'utiliser dans notre problème.

Une heuristique est une méthode de calcul qui fournit rapidement une solution réalisable, pas nécessairement optimale ou exacte, pour un problème d'optimisation difficile. C'est un concept utilisé entre autres en optimisation combinatoire, en théorie des graphes, en théorie de la complexité des algorithmes et en intelligence artificielle. Une heuristique s'impose quand les algorithmes de résolution exacte sont de complexité exponentielle, et dans beaucoup de problèmes difficiles. L'usage d'une heuristique est aussi pertinent pour calculer une solution approchée d'un problème ou pour accélérer le processus de résolution exacte. Généralement, une heuristique est conçue pour un problème particulier, en s'appuyant sur sa structure propre, mais les approches peuvent contenir des principes plus généraux.

L'heuristique **Relax and fix** repose sur un partitionnement de ses variables entières qui définit un sous problème par sous-ensemble de la partition, ainsi que sur un ordre de traitement de ces sous-ensembles. A chaque itération, les contraintes d'intégrité sont relâchées pour tous les sous-ensembles sauf un. Le sous-problème résultant est résolu. Les variables entières du sous-ensemble courant sont fixées à leurs valeurs et le processus est répété pour les sous-ensembles restants.

### 5.1 Heuristique 1

Pour résoudre le problème avec un heuristique nous avons choisi d'utiliser la méthode "relax and fix".

Pour implémenter la méthode "relax and fix" au problème nous devons choisir la stratégie à utiliser afin de choisir les variables qu'on doit appliquer la méthode relax et les valeurs qu'on doit fixer.

Généralement dans les problèmes de VRP, un station a un nombre limité de voisins.

Si on se base sur cette observation nous pouvons éliminer le cas ou les liens entre les sommets vont poser un problème.

Mais la résolution avec un solveur va se heurter à un problème de temps. En effet puis la simulation dure plus le solveur a du mal à résoudre le problème.

En se basant sur ces observations nous avons décidé que seule la variable liée au temps va subir la méthode relax.

Ainsi les variables qui vont subir la méthode "fix" sont tous les variables qui sont liées au temps et qui font partie de la résolution précédente.

Maintenant la question qui se pose est comment on va faire l'implémentation c'est-à-dire la stratégie qu'on va utiliser.

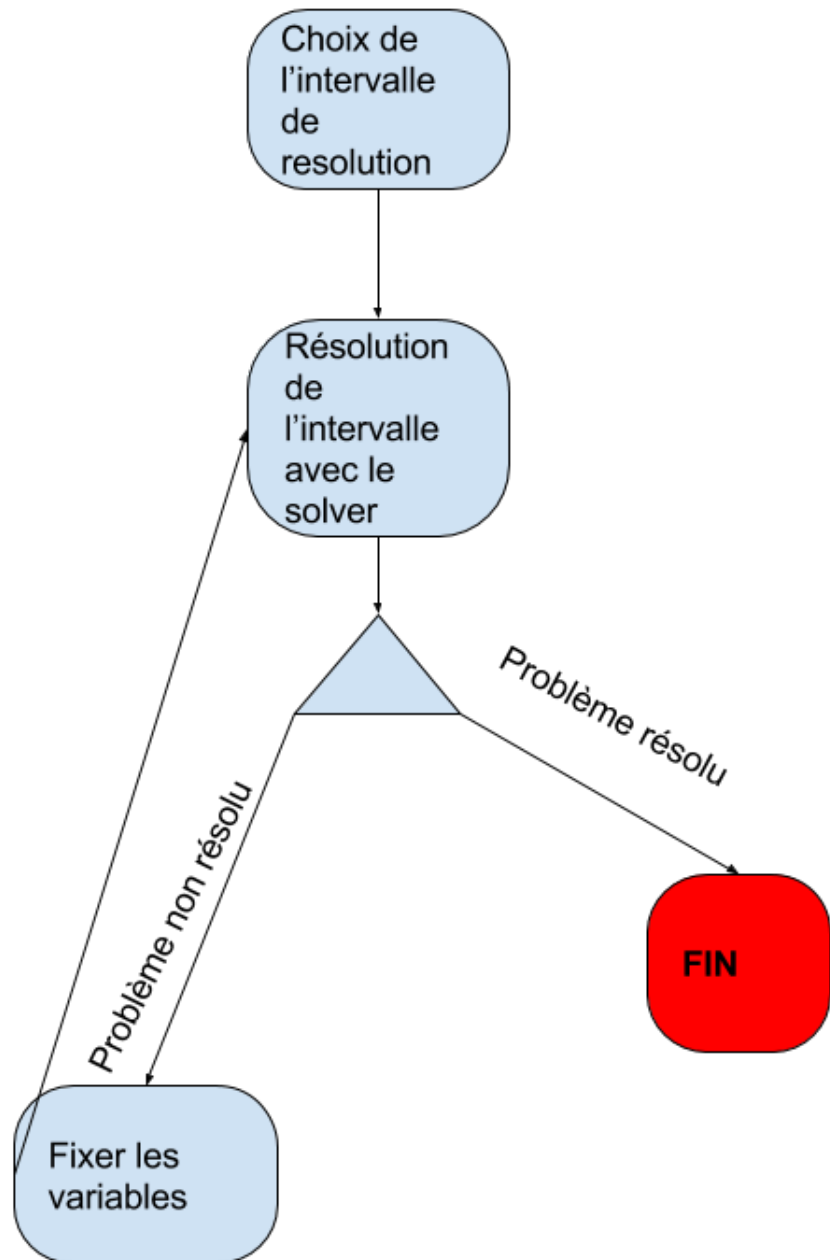
Est-ce-qu'on doit utiliser un algorithme pour pouvoir fixer les variables. Considérons que nous tentons de résoudre le problème avec un heuristique. Nous avons aucun moyen de savoir est ce que notre choix est le choix le plus optimal.

Pour remédier à ça nous sommes obligé de faire plusieurs fois la même expérience afin d'avoir le plus optimal de ces expérimentations qu'on va faire.

Nous avons pensé à un autre méthode de résoudre le problème ainsi cela consiste à diviser le problème en plusieurs sous-problèmes, que le solveur pourra résoudre. Ainsi par exemple nous avons une durée de simulation qui égale à 30, notre algorithme va traiter le problème de 0 à 10 puis il va fixer les valeurs qui sont liées à cet intervalle, puis on va traiter l'intervalle 0 à 20 on fixe les valeurs de cet intervalle et l'algorithme fini par l'intervalle 0 à 30.

La différence de ces méthodes est que si on considère le pire des cas. On aura pour la résolution avec les solveurs vers qui donnera un somme de choix optimal qui sera plus optimisé au somme de mauvais choix. Nous en concluons que la résolution via solveur est plus optimal que via heuristique.

Pour notre implémentation à raison de ces analyses nous avons choisis nous avons choisis la résolution via un solveur.



**Figure 7.** Heuristique 1.



## 5.2 Heuristique 2

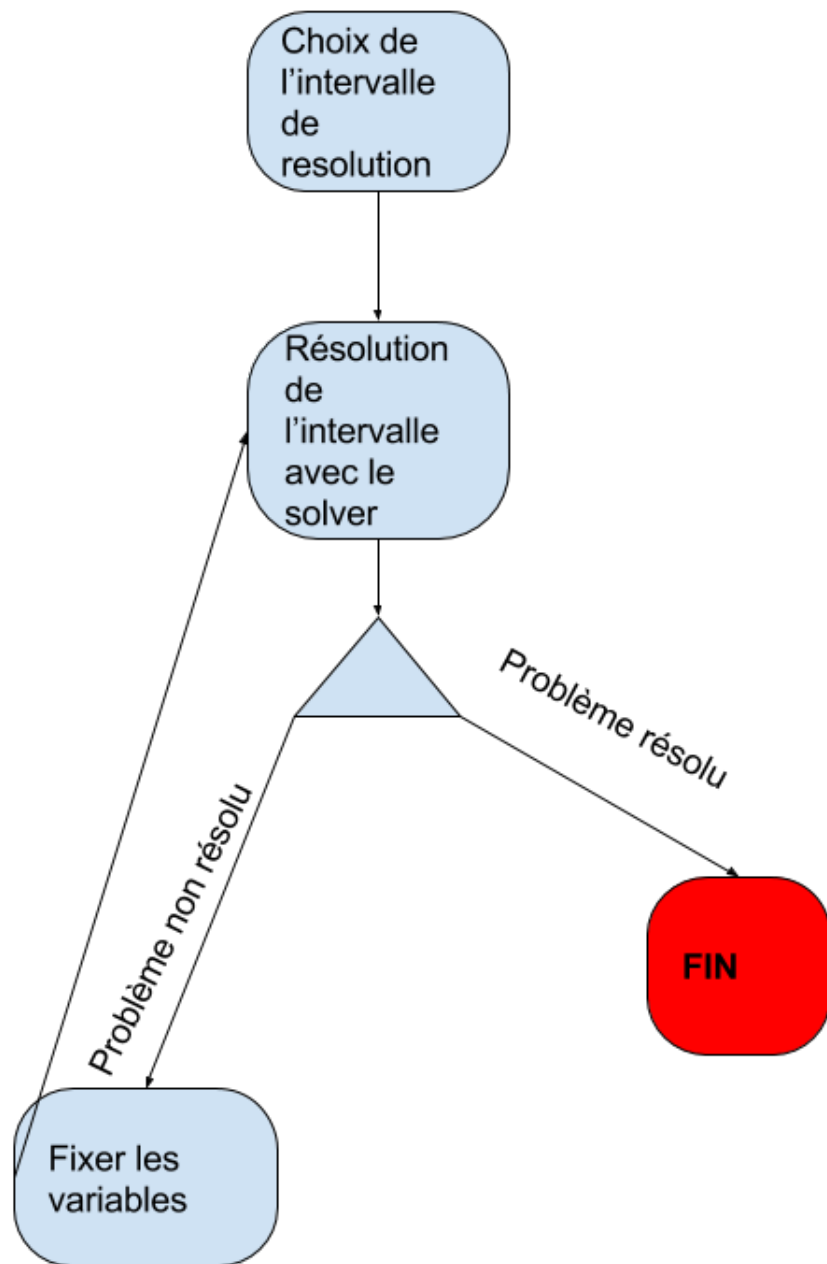


Figure 8. Heuristique 1.

Nous utilisons l'heuristique relax and fix pour pouvoir résoudre le problème. Pour que l'heuristique nous sorte des résultats cohérents nous allons suivre ces étapes :

- Étape 1 : Nous allons relaxer les valeurs
- Étape 2 : Nous allons chercher quelles voisins de  $i$  est plus intéressante et on vérifie en même temps est ce qu'on peut retourner à la base à la fin de l'expérience
- Étape 3 : On compare la quantité d'information minimiser si on restait et la quantité d'information minimiser si on se dirigeait vers une autre station
- Étape 4 : On fait une mise à jour du temps par rapport au choix qu'on a fait précédemment (soit  $t + 1$  si on choisit de rester ou  $t + t_{ij}$  si on choisit de se diriger vers  $j$  ou  $t$  est le temps de

l'itération précédente )

- Étape 5 : On fixe les variables liés au temps, de l'instant 0 à l'instant  $t$
- Étape 6 : On vérifie est qu'on est arrivé à la fin de l'expérience, si ce n'est pas le cas on retourne à la première itération

## 6 Comparaison des méthodes de Résolutions

### 6.1 Entrées

Pour pouvoir simuler toutes les heuristiques nous allons prendre comme durée  $T = 80$

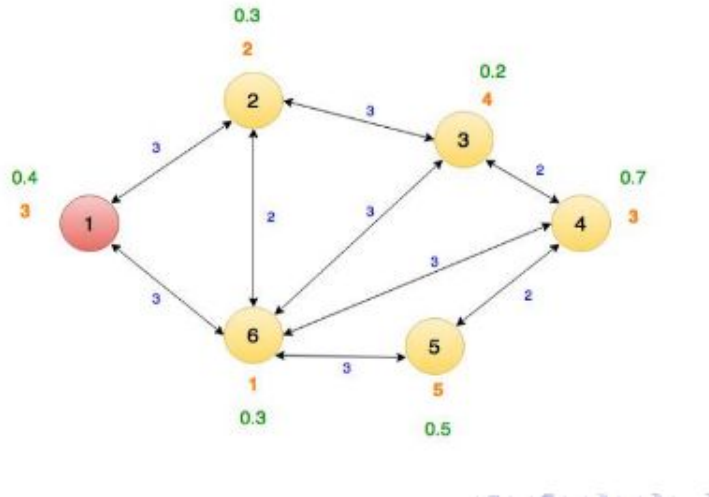


Figure 9. Entrées.

### 6.2 Résultat

#### 6.2.1 Modèle

- Quantité d'information restante : 180
  - Durée de la résolution : 30 minutes.
- NB : Le solveur était toujours entrain de résoudre.

#### 6.2.2 Heuristique 1

- Quantité d'information restante : 186
- Durée de la résolution en temps : 5 secondes

#### 6.3 Heuristique 2

- Quantité d'informations restantes : 190
- Durée de la résolution en temps : 20 secondes

### 6.4 Comparaison

Nous remarquons que le solveur était toujours mais malgré cela il a donné le meilleur résultat.

L'heuristique 2 donne le moins bon résultat du fait que la stratégie pour fixer n'est pas l'une des meilleurs.

## 7 Mise en œuvre et réalisations

durant le second semestre, nous avons entamer les grande implémentation de l'application, à savoir l'interface Utilisateur, les outils d'affichage et de génération d'instance, ainsi que l'intégration des algorithmes de résolution au logiciel ORPA.

### 7.0.1 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.

### 7.0.2 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.

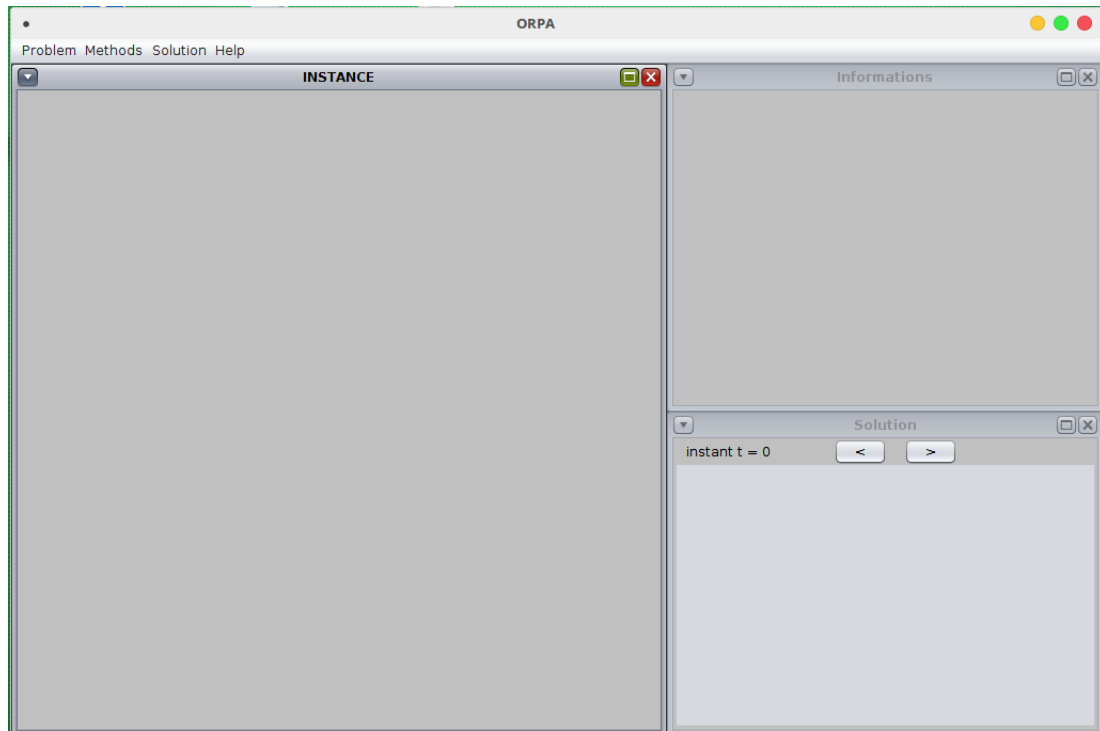
À la différence des autres solveur Gurobi est gratuit pour les étudiants ; c'est la raison que notre choix est porté sur Gurobi.

## 7.1 Réalisations

Nous détaillons ici les implémentations réalisées durant le second semestre

### 7.1.1 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.
- générateur d'instances :  
la génération d'instances se faisait au premier semestre via un script R (développé par Luis Flores), au second semestre, nous avons implémenté un générateur en Java qui est lancé via l'application, il se charge de générer une instance avec les éléments renseignés, c'est à dire, le nombre de stations qui composent l'instance et la durée de la simulation pour laquelle nous n'avons pas fixé d'unité de temps.

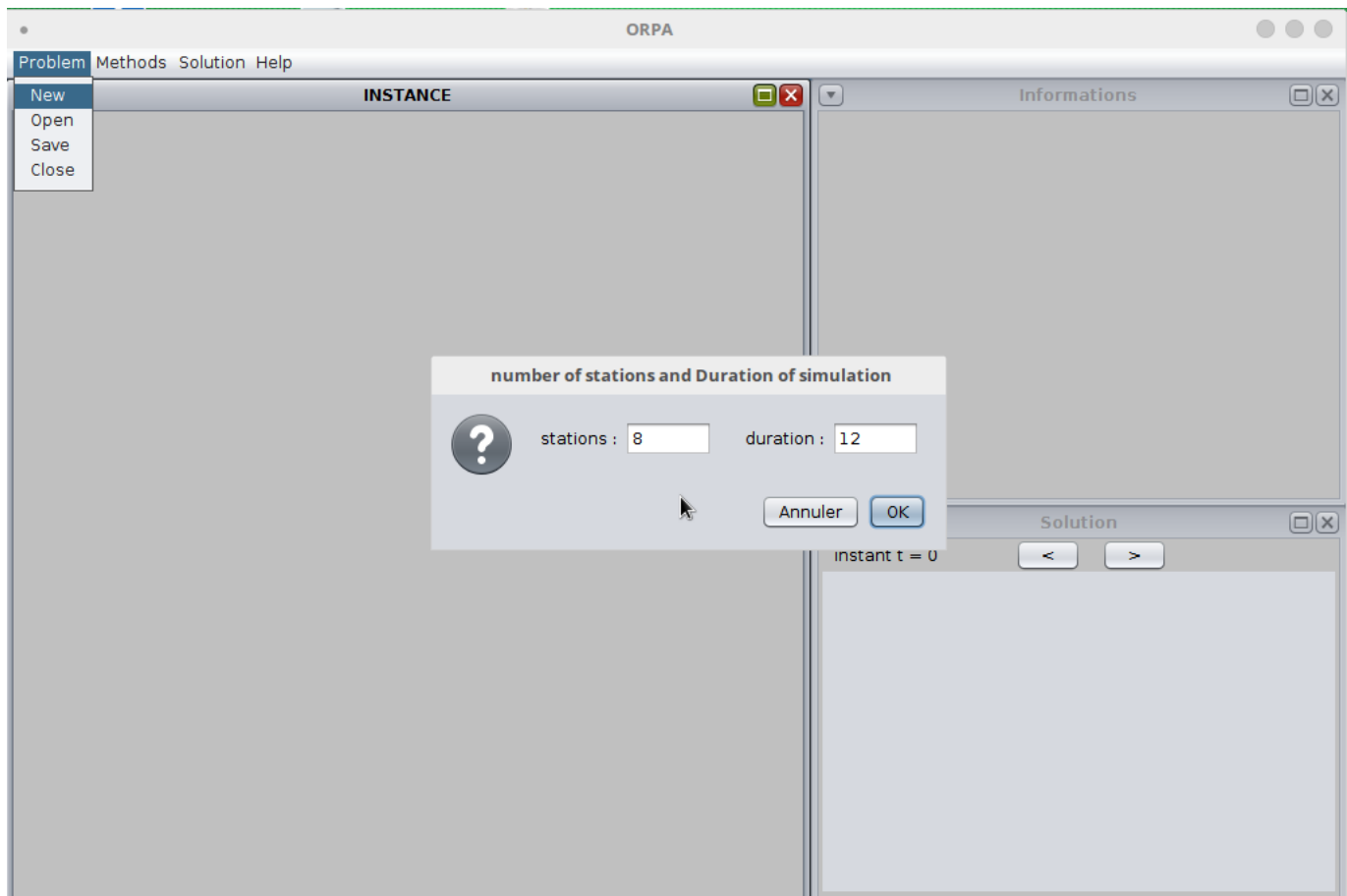
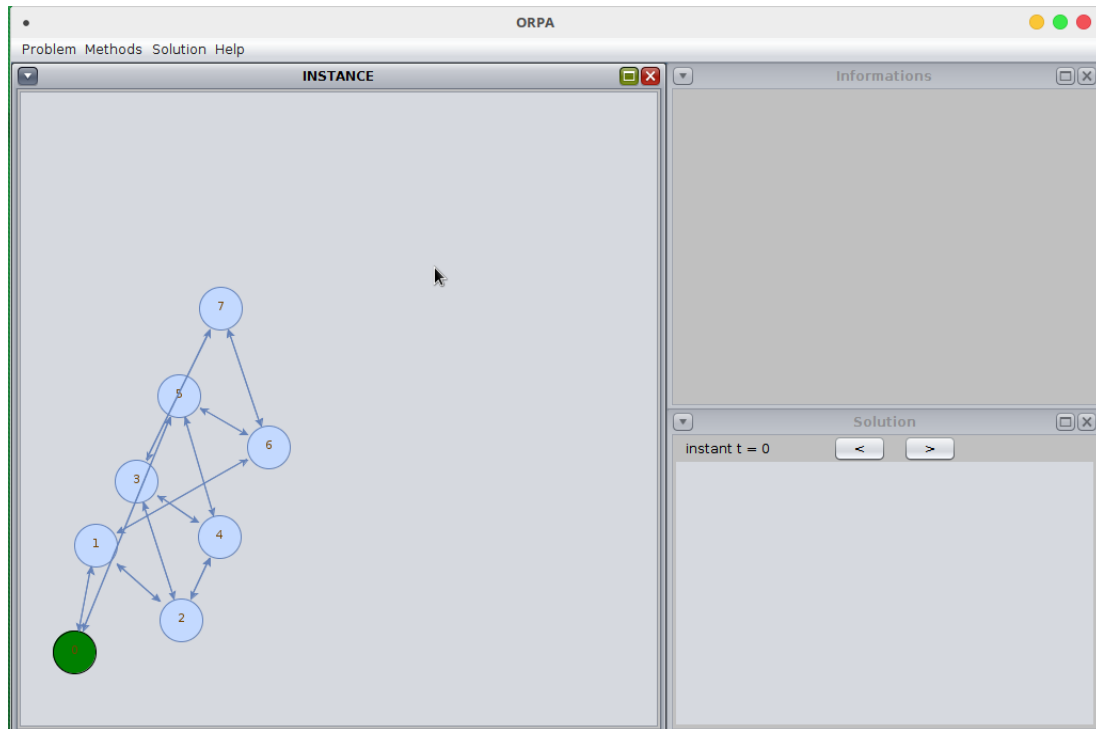


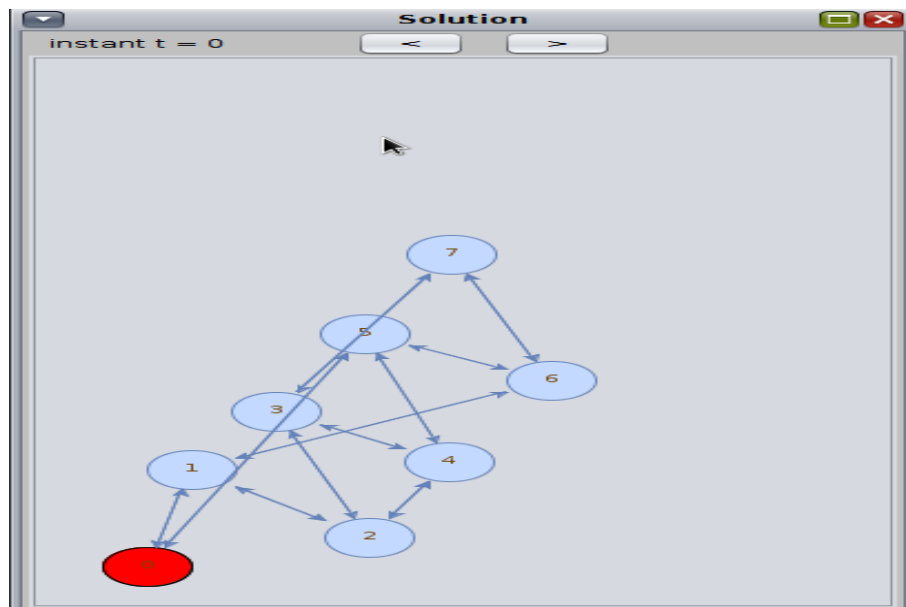
Figure 11. Générateur d'instance.

- affichage d'instances : après chaque génération d'instance, nous avons développé une fonction permettant de générer un graphe représentant l'instance.  
le station de base ou les informations doivent être déposés est représenté par un rond de couleur verte et numéroté toujours par un zéro (0), quand aux autres stations qui elles sont aussi représentées par des ronds de couleur gris sont numéroté de 1 jusqu'au nombre de stations qui compose l'instance moins 1.  
les routes entre chaque station sont elles représentées par une flèche entre les stations.



**Figure 12.** Affichage d'instance.

- affichage des solutions : après sélection d'une méthode de résolution sur le menu "Methods", on peut lancer la résolution sur l'instance générée, ainsi on obtient sur la fenêtre "Solution" un affichage graphique du résultat obtenu  
l'affichage graphique de la solution se fait sur plusieurs graphes qui montre sur quel station le véhicule qui récolte les informations se trouve, en indiquant à quel instant T, il s'est retrouvé sur cette station.



**Figure 13.** solution à l'instant  $t = 0$ .

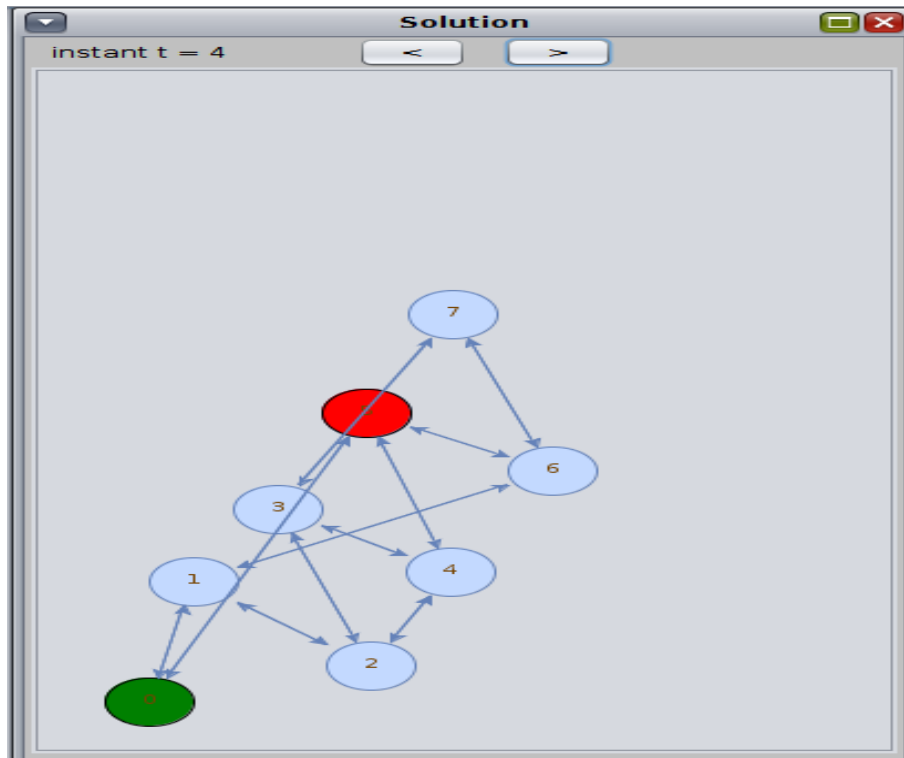


Figure 14. solution à l'instant  $t = 4$ .

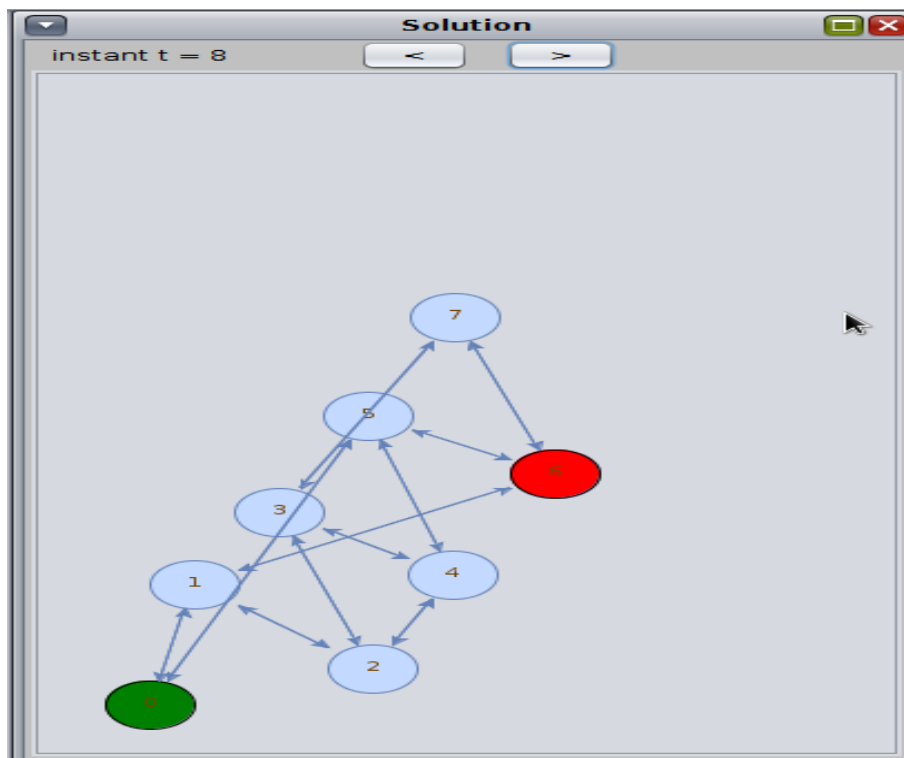


Figure 15. solution à l'instant  $t = 8$ .

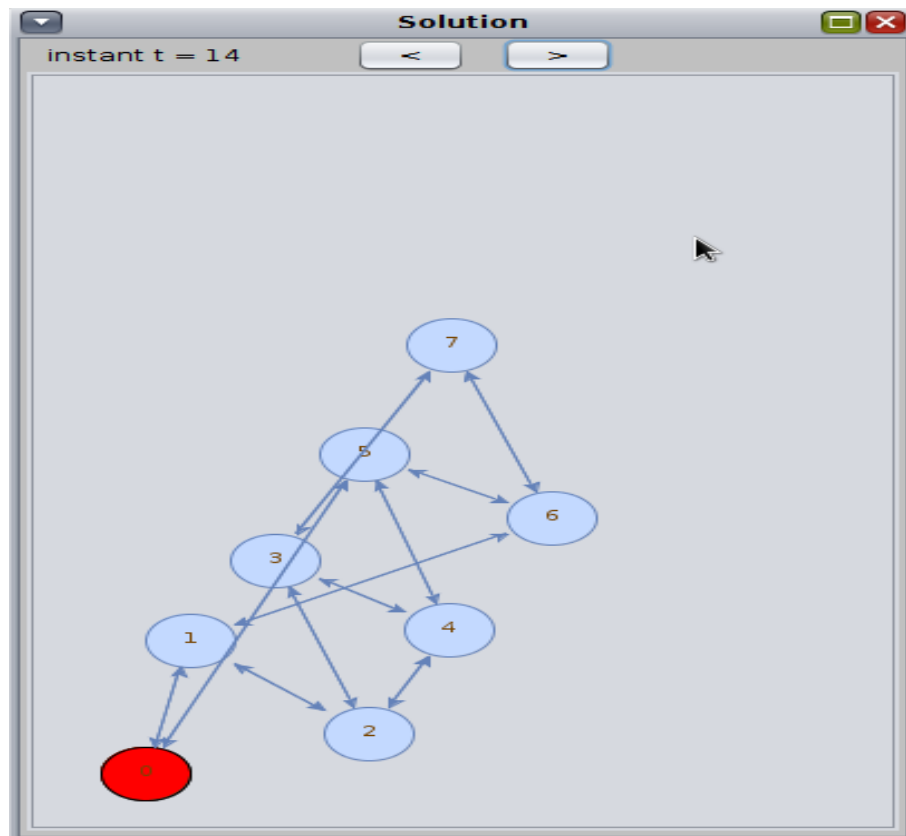


Figure 16. solution à l'instant  $t = 14$ .

## 7.2 Objectifs non atteints :

certains objectifs fixés pour le seconds 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 coordination :**  
Une meilleure synchronisation de notre équipe de projet aurait permis de mieux avancé sur l'implémentation.
- **Correction du modèle :** durant le début du second semestre, nous avons perdu beaucoup de temps sur la correction du modèle, il nous a fallut trouver l'origine des incohérence et amener des solutions.

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

1. **sauvegarde des instance générer :**  
La fonctionnalité permettant de sauvegarder une instance générer dans un fichier n'a pas été implémenté.
2. **chargement des instance sauvegarder :**  
La fonctionnalité permettant de charger une instance depuis un fichier n'a pas été implémenté.
3. **sauvegarde des solutions produites :**  
La fonctionnalité permettant de sauvegarder une solution produite via une des méthodes de résolution dans un fichier n'a pas été implémenté.



## 8 Perspectives

Durant ce second semestre certains objectifs non pas été atteints, et d'autres ne sont pas réalisés comme nous le souhaitons, voici quelques perspectives :

- **gestion d'instance :**

1. une instance doit pouvoir être chargé depuis un fichier.
2. une instance doit pouvoir être crée manuellement avec l'interface graphique.
3. une instance doit pouvoir être sauvegardé dans un fichier.

- **gestion des solutions :**

1. une solution doit pouvoir être sauvegardé dans un fichier.
2. une solution doit pouvoir être chargé depuis un fichier afin de pouvoir la comparé à une autre.

- **Heuristique 2 :**

1. Stratégie pour fixer les valeurs.
2. une solution doit pouvoir être chargé depuis un fichier afin de pouvoir la comparé à une autre.