Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis
37200 Tours, France
Tél. +33 (0)2 47 36 14 14
polytech.univ-tours.fr

Projet Recherche & Développement

2018-2019

Étude et Résolution d'un problème de mise en place d'un réseau électrique pour bus

Tuteurs académiques
Yannick Kergosien
Jorge E. Mendoza

Étudiant
Pierre Vendé (DI5)

4 avril 2019

Liste des intervenants

Nom	Email	Qualité
Pierre Vendé	pierre.vende@etu.univ-tours.fr	Étudiant DI5
Yannick Kergosien	yannick.kergosien@univ-tours.fr	Tuteur académique, Département Informatique
Jorge E. Mendoza	jorge.mendoza@hec.ca	Tuteur académique, Département Informatique

Avertissement

Ce document a été rédigé par Pierre Vendé susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Yannick Kergosien et Jorge E. Mendoza susnommés les tuteurs académiques.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnait assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnait qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable des tuteurs académiques et de l'entreprise.

L'auteur autorise l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.

Pour citer ce document

Pierre Vendé, *Étude et Résolution d'un problème de mise en place d'un réseau électrique pour bus*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2018-2019.

```
@mastersthesis{
    author={Vendé, Pierre},
    title={Étude et Résolution d'un problème de mise en place d'un réseau électrique pour
      bus},
    type={Projet Recherche \& Développement},
    school={Ecole Polytechnique de l'Université François Rabelais de Tours},
    address={Tours, France},
    year={2018-2019}
}
```

Document mis en forme sous LaTeX
grâce à la classe de document `polytech.cls` V2.2.5 (2017/05/02)

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

output.loa

1 Introduction

1 Acteurs, enjeux et contexte

De 1970 à 2010, les émissions de gaz à effet de serre ont augmenté de 80%. Cette augmentation est due au développement des secteurs liés à la combustion d'énergies fossiles, dans l'industrie ou bien les transports, sur cette période. Les gaz à effet de serre sont présents naturellement dans l'atmosphère, ils permettent de la réchauffer et ainsi de rendre la présence d'eau liquide possible. Cependant, si aucune action n'est faite pour empêcher cette augmentation, le réchauffement climatique attendu d'ici 2100 sera de 4 à 5°C. Cette forte augmentation aurait des conséquences dramatiques comme l'acidification des océans, une perte de la biodiversité, des problèmes alimentaires et de ressources d'eau, une augmentation du nombre de réfugiés climatiques, etc [WWW4].

La conférence sur le climat de Paris, aussi appelée *COP21*, avait pour objectif de trouver des solutions à l'échelle mondiale concernant le problème climatique. Si les États limiteraient leurs émissions de gaz à effet de serre, l'augmentation de température d'ici 2100 ne pourrait être que de 2°C. Cette conférence a donc débouché sur un accord ratifié par 195 pays signataires à travers le monde. Cet accord a pour objectif global de limiter cette augmentation de température et l'un des moyens est de réduire la production de gaz à effet de serre.

Le principal gaz à effet de serre émis, à hauteur de 70%, est le dioxyde de carbone CO_2. De plus, chaque année, le secteur lié au transport représente près de 30% des émissions de CO_2 de l'Union Européenne (comme vu dans le rapport de 2014 concernant le changement climatique [1]). Les États et les collectivités territoriales possèdent donc un premier moyen simple de baisser ces émissions de CO_2 qui est réduire celles liées aux transport et notamment des transports en commun, transports généralement gérés par des entreprises publiques. Ces dites entreprises se sont donc tournées vers l'utilisation de véhicules électriques dans leurs réseaux et la question de l'optimisation des tournées de véhicules électriques s'est donc posée.

C'est dans cette optique qu'a été créé le projet de recherche *e-VRO* (Electric Vehicle Routing Optimization) par l'Agence Nationale de la Recherche. Ce projet permet la collaboration de 4 équipes, 2 françaises, une américaine et une colombienne, sous la coordination de Jorge Mendoza. Le principal objectif de ce projet est de concevoir, implémenter, tester et répandre des algorithmes rapides et efficaces pour résoudre des problèmes de tournées de véhicules électriques [WWW5].

Figure 1 – *Logo du projet* e-VRO *[WWW5]*

Cette problématique liée aux tournées de véhicules électrique est d'autant plus cruciale car le nombre total de bus électrique devrait tripler d'ici 2025 pour atteindre 47% de la flotte de bus urbains utilisés dans le monde (d'après *Bloomberg New Energy Finance* [WWW10]). Cette augmentation est induite par la Chine, possédant le plus grand parc de bus électrique au monde, dans une politique de maintient de la qualité de l'air dans les villes. Cependant, la Chine n'est pas le seul pays à vouloir effectuer sa transition écologique. En France, la RATP (Régie autonome des transports parisiens) a lancé son plan Bus 2025 qui a pour but de convertir 80% des bus en électrique. A terme, ce plan permettra de supprimer les bus diesels, polluants, du réseau routier parisien [WWW3].

2 Objectif

L'objectif du projet est de trouver une modélisation représentant l'électrification d'un réseau de bus urbain à l'aide de bus hybrides. Cette modélisation doit permettre de connaître l'emplacement des différents chargeurs nécessaire au fonctionnement des bus, les portions du circuit parcourues avec le moteur électrique ainsi que les temps de chargement des bus. Pour obtenir ces informations, des méthodes de programmation linéaire en nombres entiers sont utilisées. Ce résultat s'obtient en résolvant deux sous-problèmes, le premier a pour objectif de déterminer les informations liées à l'électrification (emplacement des chargeurs, portions en mode électrique, etc.) et le second vérifie que les chargeurs peuvent effectuer les différents rechargements prévus par le premier problème.

3 Hypothèses

Ce projet est la continuité d'un stage effectué pendant l'été 2018. On suppose donc que l'on possède l'ensemble des implémentations qui fonctionnent effectuées pendant de stage, ce qui inclut la résolution complète du problème pour une unique ligne de bus ainsi que la génération d'instances pour la résolution avec plusieurs lignes. Si on a pas ces sources ou si elles sont inutilisables, il faudra redévelopper la génération d'instance depuis le début.

4 Bases méthodologiques

Afin de mener à bien ce projet, une méthode cycle en V est appliquée. Ce déroulement est contrôlé par un diagramme de Gantt (voir Annexe E) et permet de vérifier l'état d'avancement réel du projet par rapport à l'état prévu. Pour cela, il faut définir à l'avance l'ensemble des besoin et les spécifier. Ensuite, on peut passer à la partie programmation, test et intégration. Les parties programmation et test sont effectuées de manière alternée. Chaque fonctionnalité est implémentée puis testée de manière unitaire. Puis, une fois que toutes les fonctionnalités

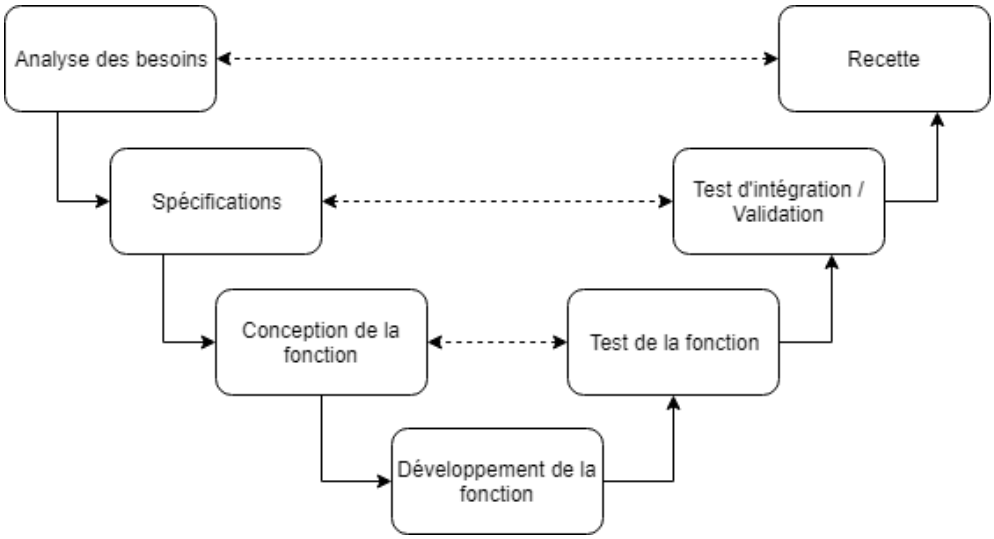


Figure 2 – *Cycle en V utilisé pour le déroulement du projet*

ont été implémentées, l'intégration des différentes fonctionnalités est faite, suivie des tests d'intégration.

Une convention de nommage est aussi adoptée pour la partie programmation. Les noms de type de données (classes, énumérations, etc.) utilisent le *Pascal Case*, un *Camel Case* où la première lettre est en majuscule. Les noms des variables utilisent le *Camel Case*, le nom est un ensemble de mots concaténés et la première lettre de chaque mot est en majuscule à l'exception de la première. Les noms sont choisis afin de pouvoir comprendre l'utilisation de la variable à partir de son nom.

L'ensemble du code est versionné en utilisant la plate-forme Git. Le projet étant la suite d'un autre projet, une nouvelle branche a été crée à partir du projet original. Les branches ne sont fusionnées uniquement à la fin du projet en cas de succès du projet. Sinon, l'ancienne version est conservée.

2 Description générale

1 Environnement du projet

L'environnement du projet est le suivant :

— Langage : C++
— Système d'exploitation : Windows 10
— IDE de développement : Visual Studio 2017

2 Caractéristiques des utilisateurs

Le logiciel est destiné à un utilisateur souhaitant résoudre le problème d'électrification d'un réseau de bus. Cet utilisateur peut être un scientifique voulant des résultats de tests sur une machine particulière, un étudiant travaillant sur le problème ou bien un professionnel devant faire une électrification de réseau de bus. L'application est simple à utiliser, en suivant les instructions du guide d'utilisation, elle peut donc être utilisée par n'importe qui ayant des notions basiques d'informatique, notions nécessaires pour appréhender l'interface.

Table 1 – *Caractéristiques des utilisateurs*

Rôle	Connaissances en informatique	Expérience de l'application	Fréquence d'utilisation
Utilisateur	Basiques	Manuel nécessaire	Occasionnelle

3 Fonctionnalités du système

Le diagramme ci dessous (voir Figure 1) résume l'ensemble des fonctionnalités de l'application utilisables, directement ou non, par un utilisateur. Ces différentes fonctionnalités sont détaillées dans les spécifications fonctionnelles (voir Annexe C).

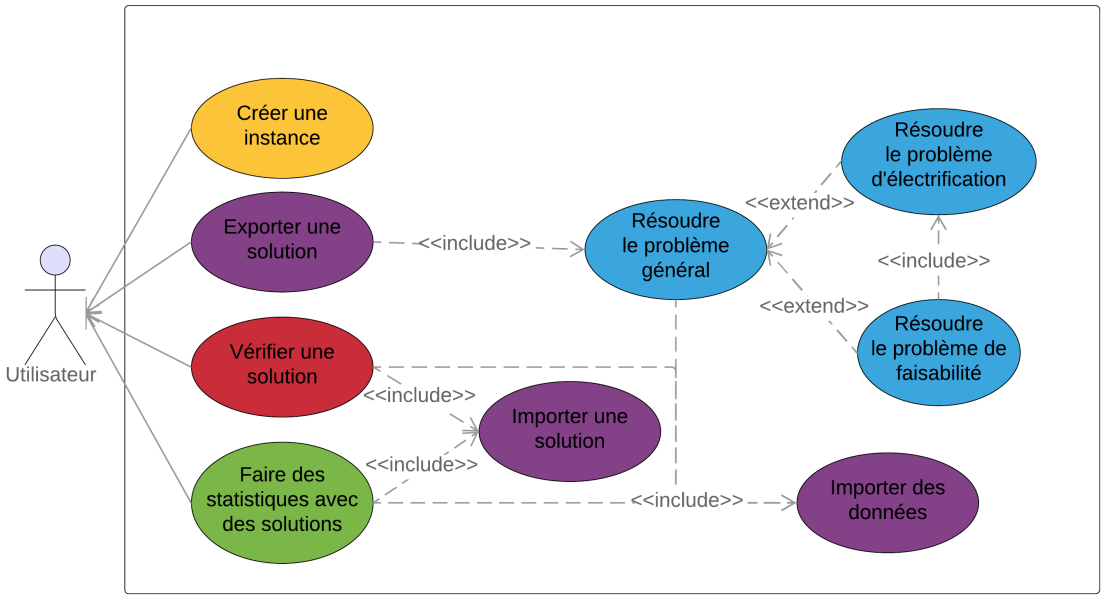


Figure 1 – *Diagramme d'utilisation de l'application*

4 Structure générale du système

La structure générale du système s'appuie principalement sur la structure de données représentant l'instance ainsi que celle représentant la solution. Les modules supplémentaires vont utiliser ces structures principales afin de réaliser les différentes taches. On va distinguer deux types de solution, la solution utilisable par le solveur qui va résoudre le problème et la solution utilisable par le reste du programme. Le solveur utilise des structures de données spécifiques à sa libraire, c'est pourquoi il est préférable d'avoir deux classes représentant la solution. Ainsi, seulement une minorité de classes est en relation avec le solveur, représenté par sa classe principale Ilocplex. Chaque classe en relation avec le solveur représente un des problème à résoudre dans la résolution d'une instance.

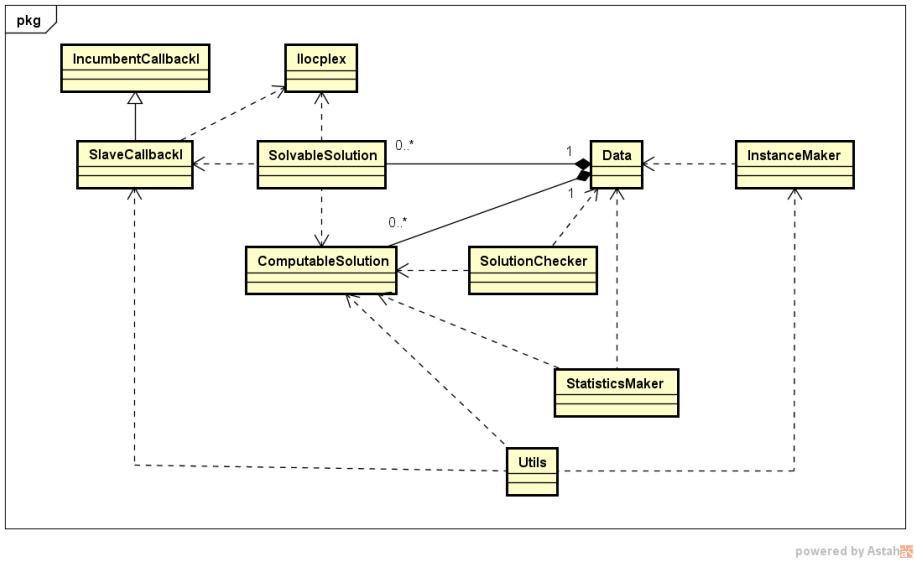


Figure 2 – *Diagramme de classe de l'application*

3 État de l'art

1 Projet précédent

Ce projet est la suite d'un stage[8] effectué pendant l'été 2018 concernant le même sujet.

1.1 Modélisation

Description du problème

Soit le graphe G(V, A) représentant un réseau de transports urbains desservi par des bus hybrides, avec V la liste des sommets (les arrêts de bus) et A la liste des arcs (les trajets entre 2 arrêts). Soit L l'ensemble des lignes composant le réseau et $G_l(V_l, A_l)$, un sous-graphe de G, représentant la ligne de bus $l \in L$, le réseau entier des lignes est donc défini par $G(V, A) = \bigcup_{l \in L} G_l(V_l, A_l)$. Les bus hybrides ont besoin d'électricité, fournie par des chargeurs à placer tout le long des différentes lignes de bus. De plus, la journée est découpée en différentes périodes, chacune représentant ainsi une tranche horaire et permettant ainsi de distinguer les heures de forte affluence et les heures creuses.

L'objectif de cette modélisation est de trouver où placer les différents chargeurs, de savoir combien de temps il faut charger les bus aux différents chargeurs et quels tronçons du parcours il faut parcourir en mode électrique. De plus, certains tronçons du parcours doivent obligatoirement être faits en mode électrique et des temps de recharge maximum ont été définis pour certaines périodes à certains endroits dans un soucis de qualité du service proposé. Pour finir, il faut maximiser la distance parcourue en mode électrique.

Hypothèses de travail

Afin de modéliser le problème, nous avons utilisé les hypothèses de travail suivantes.

- Tous les bus ont les mêmes caractéristiques (consommation électrique, capacité de batterie, modèle, etc.)
- Tous les chargeurs ont les mêmes caractéristiques (puissance électrique, voltage, etc.)
- On ne peut changer de mode de consommation uniquement à un arrêt

Données

- L représente l'ensemble des lignes de bus, composé des différents arrêts et des trajets parcourus
- P représente l'ensemble des périodes, composé des différentes tranches horaires. Il existe 3 types de période : les périodes de forte affluence (lors du début ou de la fin de la journée de travail), les périodes dites normales (dans le courant de la journée) et les périodes de faible affluence (comme la nuit).
- f_{lp} représente la fréquence de passage des bus de la ligne l pendant une période p. Cette fréquence varie en fonction de la période. En effet, il est pas nécessaire de faire passer un nombre important de bus dans une période de faible affluence (comme la nuit), contrairement aux périodes de forte affluence.
- d_{ij} représente la distance entre les arrêts i et j
- c_i représente le coût pour installer un chargeur à l'arrêt i. En effet, bien que le coût de la machine pour charger soit fixe, le coût d'installation varie en fonction de l'endroit où on place le chargeur, à cause des différents aménagements précédemment présents à enlever ou adapter.
- B représente le budget alloué à l'installation de chargeurs
- w_{lpk} représente la durée d'arrêt maximum au $k^{-ème}$ arrêt de la ligne l pendant une période p. Afin d'assurer une bonne qualité de service, les bus ne doivent pas s'arrêter trop longtemps à une station. Cette durée maximale tolérée varie en fonction de l'affluence sur la ligne.
- R représente la vitesse de chargement des batteries, ici l'intensité du courant du chargeur
- Q représente la capacité maximale des batteries
- e_{pij} représente la consommation électrique d'un bus entre les arrêts i et j pendant une période p. Cette consommation est déterminée par une consommation e_p qui peut varier selon certains critères tels que la topologie du circuit, la température ambiante ou bien la vitesse. Dans notre modèle, nous avons choisi de privilégier la vitesse moyenne lors d'une période pour déterminer la consommation électrique. On obtient donc $e_{pij} = e_p \times d_{ij}$.
- $m_{pij} = \begin{cases} 1 & \text{si la section entre les arrêts } i \text{ et } j \text{ doit être} \\ & \text{parcourue en mode électrique pendant la période } p \\ 0 & \text{sinon} \end{cases}$

Variables de décision

- x_i représente le nombre de chargeurs que l'on met à l'arrêt i
- t_{lpk} représente la durée de chargement au $k^{-ème}$ arrêt de la ligne l pendant la période p
- q_{lpk} représente la quantité d'énergie présente dans la batterie en arrivant au $k^{-ème}$ arrêt de la ligne l pendant la période p
- $y_{lpkk'} = \begin{cases} 1 & \text{si la section entre le } k^{-ème} \text{ et le } k'^{-ème} \text{ arrêt de la ligne } l \\ & \text{est parcourue en mode électrique pendant la période } p \\ 0 & \text{sinon} \end{cases}$

Avec $x_i \in \mathbb{N}_+$, $t_{lpi}, q_{lpi} \in \mathbb{R}_+$ et $y_{lpkk'} \in \{0, 1\}$

Contraintes

1. La coût total de l'installation des chargeurs sur les arrêts ne doit pas dépasser le budget total alloué à l'installation des chargeurs.

$$\sum_{i \in \mathrm{V}} c_i \times x_i \leq \mathrm{B} \tag{1}$$

2. Il est nécessaire de mettre un nombre suffisant de chargeur afin de respecter la fréquence de passage et d'éviter des temps d'attente trop importants dans le cas où les bus se rattrapent. Pour finir, il ne peut pas avoir de temps de chargement si il n'y a pas de chargeurs.

$$\max_{l \in \mathrm{L}} \left\lceil \frac{t_{lpk}}{f_{lp}} \right\rceil \leq x_i : \forall\ p \in \mathrm{P} \text{ et } \forall\ i \in \mathrm{V} \text{ tq } k \in \mathrm{V}_l \text{ et } l[k] = i \tag{2}$$

3. Le temps de chargement à chaque arrêt ne doit pas excéder la durée maximale fixée à un arrêt pour une période. En période de forte affluence, le bus ne peut pas se permettre de se recharger car il y a beaucoup de passagers à transporter et peu de temps à perdre. Le temps de chargement doit donc être diminué. Cependant, en période creuse, le bus a un peu plus le temps de se recharger.

$$t_{lpk} \leq w_{lpk} : \forall\ l \in \mathrm{L},\ \forall\ p \in \mathrm{P} \text{ et } \forall\ k \in \mathrm{V}_l \tag{3}$$

4. Le bus doit être en mode électrique dans les portions où c'est obligatoire, si $m_{pij} = 1$ alors $y_{lpkk'} = 1$ (les k et k' positions de l correspondants respectivement à i et j).

$$y_{lpkk'} = 1 : \forall\ l \in \mathrm{L},\ \forall\ p \in \mathrm{P} \text{ et } \forall\ (k, k') \in \mathrm{A}_l \text{ tq } m_{pl[k]l[k']} = 1 \tag{4}$$

5. Il n'est pas possible de charger la batterie d'un bus au delà de sa capacité. La quantité d'énergie présente dans la batterie du bus en arrivant à un arrêt avec la quantité chargée ne doit pas dépasser la capacité totale de la batterie.

$$q_{lpk} + t_{lpk} \times \mathrm{R} \leq \mathrm{Q} : \forall\ l \in \mathrm{L},\ \forall\ p \in \mathrm{P} \text{ et } \forall\ k \in \mathrm{V}_l \tag{5}$$

6. Les bus commencent leur itinéraire avec la batterie chargée.

$$q_{lp0} = \mathrm{Q} : \forall\ l \in \mathrm{L} \text{ et } \forall\ p \in \mathrm{P} \tag{6}$$

7. La quantité d'énergie présente dans la batterie du bus en arrivant à l'arrêt j doit être égale à celle à l'arrêt i avec la quantité chargée en i moins celle consommée pour effectuer a distance entre i et j.

$$q_{lpk} + t_{lpk} \times \mathrm{R} - e_{pl[k]l[k']} \times y_{lpkk'} = q_{lpk'} : \forall\ l \in \mathrm{L},\ \forall\ p \in \mathrm{P} \text{ et } \forall\ (k, k') \in \mathrm{A}_l \tag{7}$$

Fonction objectif

L'objectif est de parcourir la plus grande distance possible en utilisant le moteur électrique.

$$\max z = \sum_{l \in \mathrm{L}} \sum_{p \in \mathrm{P}} \sum_{(k,k') \in \mathrm{A}_l} d_{l[k]l[k']} \times y_{lpkk'} \tag{a}$$

1.2 Commentaires

On peut voir le modèle créé ne prend pas en compte la disponibilité des chargeurs présents sur le réseau. Il est donc à compléter car la solution pourrait réalisable pour chaque ligne indépendemment des autres mais pas d'un point de vue global dans le réseau. De plus, ce modèle considère plusieurs périodes dans la journée mais ne gère pas la transition entre ces périodes. En effet, un point de vue réaliste ferait varier la fréquence de passage des bus de manière continue entre les périodes.

2 Littérature scientifique

2.1 Problème du déploiement d'un système de bus à batterie électrique

Le problème du déploiement d'un système de bus à batterie électrique, ou *Battery Electric Bus System Deployment Problem* (BEBSDP), est un problème relativement récent dans la littérature scientifique. Il s'agit d'une réponse au problème de pollution urbain. Cette pollution peut être stoppée en provoquant une diminution de la production de gaz à effet de serre et une minimisation de l'augmentation de la température sur le long terme. Cet objectif peut être atteint avec la transformation d'un réseau de bus classiques en un réseau de bus électriques. Ceci induit un ajout de stations de chargement sur le réseau afin de permettre le fonctionnement des bus ainsi que des notions de chargement des bus (où, quand, combien de temps, etc.).

Ce problème se compose de deux sous problèmes :

- Problème du déploiement des infrastructures liées au chargement : Ce problème utilise le réseau de bus initial et place les différents chargeurs sur le réseau. De plus, il choisit les différentes technologies utilisées par les bus et les chargeurs. Ce problème est réaliste pour l'électrification des réseaux de bus avec une ligne unique.
- Problème de l'utilisation des infrastructures liées au chargement : Ce problème utilise le réseau de bus avec les chargeurs placés dessus et permet de fixer la durée de chargement de chaque bus. C'est aussi ce problème qui définit l'utilisation des chargeurs (qui, quand et combien). Il permet donc d'étendre le premier problème à un réseau entier.

Ce problème pose tout de même certaines hypothèses de travail permettant de simplifier les différentes modélisations du problème :

- Le réseau des lignes de bus n'est pas modifié
- Chaque station de chargement est fixe
- Le chargement est continu, il ne peut pas être interrompu
- L'énergie est consommée proportionnellement à la distance
- L'énergie est rechargée proportionnellement à la durée de chargement

2.2 Articles scientifiques

Kunith, Mendelevitch, Goehlich (2017)

L'article *Electrification of a city bus network - An optimization model for cost-effective placing of charging infrastructure and battery sizing of fast-charging electric bus systems* a été écrit par Kunith, Mendelevitch et Goehlich en 2017[2]. Il présente un problème d'électrification de réseau de

bus qui a pour but de minimiser les coûts d'installation et d'acquisition des batteries pour des bus électriques. Le modèle utilise une flotte hétérogène sur l'ensemble du réseau (plusieurs types de batterie et plusieurs modèles de bus) cependant la flotte est homogène pour une ligne unique (bus identiques sur chaque ligne). Chaque station de chargement est utilisable pour un unique type de bus et pour une unique ligne (pas de mutualisation) cependant il est possible d'en placer plusieurs à un arrêt pour les autres lignes. Pour finir, on remarque qu'il y a un parcours du réseau avec mémoire, en effet il y a une utilisation de la batterie restante du circuit $n-1$ pour commencer le circuit n. Ainsi, on peut voir que ce modèle fonctionne, cependant la non mutualisation des chargeurs provoque une hausse significative des coûts. Ce modèle a été appliqué sur le système de bus de Berlin.

Wei, Liu, Ou, Fayyaz (2018)

L'article *Optimizing the spatio-temporal deployment of battery electric bus system* a été écrit par Wei, Liu, Ou et Fayyaz en 2018[10]. Il présente un problème d'électrification de réseau de bus qui a pour but de minimiser les coûts d'installation et d'acquisition des batteries pour des bus électriques. Le modèle utilise une flotte hétérogène (électriques et non électriques). De plus, il inclut les dépôts de bus dans le modèle et pas uniquement les chargeurs. L'utilisation de ces chargeurs est partagée entre les lignes. Le temps n'est pas continu mais est séquence en périodes fixes. Il faut une unité de temps pour passer de l'arrêt a à l'arrêt b. Pour finir, il n'y a pas de notion de batterie restante pour les bus mais de kilométrage parcouru (avec un nombre de kilomètre maximum avant recharge pour chaque bus). Ainsi, on peut voir ce séquençage du temps ne semble pas réalisation pour un réseau urbain car il sous-entend aussi que le chargement est immédiat. Ce modèle a été appliqué sur le système de bus de Salt Lake City.

Liu, Song, He (2018)

L'article *Planning of Fast-Charging Stations for a Battery Electric Bus System under Energy Consumption Uncertainty* a été écrit par Liu, Song et He en 2018[6]. Il présente un problème d'électrification de réseau de bus qui a pour but de minimiser les coûts d'installation et d'acquisition des batteries pour des bus électriques. Le modèle prend en compte une flotte hétérogènes (plusieurs types de batterie) ainsi qu'un ensemble de chargeurs hétérogènes (types et puissances différents). Ce modèle est très similaire à celui réalisé dans le projet précédent (voir Section 1) car il y a aussi un chargement en début de ligne et les chargeurs sont supposés partageables. Cependant la mutualisation des chargeurs est laissé en travail ultérieur en supposant que cela se fera avec de l'ordonnancement et une file d'attente. Ce modèle a été appliqué sur le système de bus de Salt Lake City.

Wang, Huang, Xu, Barclay (2017)

L'article *Optimal recharging scheduling for urban electric buses : A case study in Davis* a été écrit par Wang, Huang, Xu et Barclay en 2017[9]. Il présente un problème d'électrification de réseau de bus qui a pour but de minimiser les coûts d'opération du système de rechargement de bus électrique. Le modèle considère une flotte de bus ainsi que des chargeurs homogènes. La particularité de cet article est le chargement des bus. L'utilisation des chargeurs est mutualisé et chaque passage de chaque bus considéré avec une date de début et une date de fin. Le parcours de chaque bus est aussi avec des dates de début et de fin. Toutes ces dates sont des données

car le temps de chargement est ici fixe. Le modèle prend aussi en compte la possibilité d'avoir plusieurs points de chargement à une station de rechargement afin de faciliter le partage de la ressource. Le défaut de ce modèle est la fixation des durées de chargement, le bus pourrait soit trop charger soit pas assez ce qui pourrait provoquer des problèmes de fonctionnement. Ce modèle a été appliqué sur le système de bus de Davis.

Bilan

On peut voir que ce problème est bien un problème récent. La problème a principalement été résolu dans le cas où les chargeurs ne sont pas partagés ce qui représente une grande perte d'argent. Il y a donc un réel vide à ce sujet. Un modèle convenable ainsi qu'une méthode de résolution sont à trouver afin de résoudre le problème.

2.3 Des bus électriques aux bus hybrides

Le problème étudié pour le projet est extrêmement similaire. Il répond aux mêmes problématiques mais avec une technologie différente. Afin de passer aux bus hybrides, il est nécessaire d'adapter quelques éléments de la modélisation utilisant les bus électriques.

- Ajouter une variable de décision indiquant le mode de fonctionnement du bus pour une portion du parcours
- Maximiser la distance parcourue en mode électrique
- Supprimer les contraintes liées au seuil minimal de la batterie du bus
- Transformer la fonction objectif représentant le budget en contrainte (en bornant la fonction) ou passer sur un problème bi-objectif (en ayant deux fonctions objectif)

3 Méthodes de résolution

3.1 Programmation linéaire

La programmation linéaire, ou optimisation linéaire est un champ d'étude qui travaille sur la minimisation d'une combinaison linéaire de variables, aussi appelée fonction objectif. Ces variables sont sujettes à des contraintes, elles aussi décrites par des fonctions linéaires. On distingue deux types de problèmes en programmation linéaires, ceux ne contenant que des variables réelles (programmation linéaire simple ou PLS) et ceux contenant des variables entières (programmation linéaire en nombres entiers ou PLNE). Les PLS sont plus simples à résoudre à cause de l'algorithme utilisé. Il s'agit de l'algorithme du simplexe créé par Dantzig en 1947. L'ensemble des contraintes permettent de définir l'ensemble des solutions possibles comme un polyèdre convexe. Si ce polyèdre est non vide, l'algorithme du simplexe va, à chaque itération, parcourir les différents sommets du polyèdre en suivant une arête jusqu'à minimiser la fonction objectif. Pour les PLNE, il existe aussi plusieurs méthodes mais une des plus connues est celle de la séparation et évaluation (voir Section 3.2).

3.2 Branch and Bound

L'algorithme *Branch and Bound*, ou séparation et évaluation, est un algorithme couramment utilisé dans la résolution de problèmes linéraires à nombres entiers (PLNE). L'objectif est de

partitionner un problème complexe en deux sous-problèmes moins complexes, en rajoutant une contrainte ce qui réduit l'ensemble des solutions admissibles. Cette méthode a été présentée en 1960 par Land et Doig[4] cependant la première appellation *Branch and Bound* n'est apparue qu'en 1963 dans la résolution d'un problème de voyageur de commerce[5].

L'algorithme fonctionne avec un système de bornes. Il y a la borne inférieure du problème (dans le cas d'une maximisation) qui est une solution réalisable du problème d'origine. Il y a la borne supérieure qui est spécifique à chaque sous-problème. Cette borne supérieure est calculée par relaxation du sous-problème étudié. La partition et l'ajout de contrainte permet de créer une arborescence de solutions. Si la borne inférieure est plus petite que la borne supérieure issue du sous-problème, alors le noeud de l'arbre n'est pas conservé. Dans le cas contraire et si il s'agit d'un noeud feuille, on affecte à la borne inférieure le maximum entre la borne supérieure et inférieure, sinon on réitère l'algorithme sur les partitions issues du noeud.

Ainsi, plus borne inférieure choisie au départ est bonne, plus la convergence de l'algorithme est rapide. Il est donc important de trouver une bonne solution réalisable dès le début.

```
1  Liste des noeuds à évaluer ← Problème initial;
2  Borne inférieure ← Valeur de la fonction objectif d'une solution réalisable;
3  tant que Liste des noeuds à évaluer ≠ ∅ faire
4  |  Retirer un noeud n de la liste des noeuds à évaluer;
5  |  Séparer n en plusieurs noeuds fils;
6  |  pour chaque n_f noeud fils de n faire
7  |  |  Borne supérieure de n_f ← Valeur de la fonction objectif du problème n_f relaxé;
8  |  |  si Borne supérieure de n_f > Borne inférieure alors
9  |  |  |  si n_f est un noeud feuille alors
10 |  |  |  |  Borne inférieure = max(Borne inférieure ; Borne supérieure de n_f);
11 |  |  |  sinon
12 |  |  |  |  Ajouter n_f à la liste des noeuds à évaluer;
13 |  |  |  fin
14 |  |  fin
15 |  fin
16 fin
```

Algorithme 1 : Algorithme de *Branch and Bound*

3.3 Branch and Check

L'algorithme *Branch and Check*, ou séparation et vérification, est une variante du *Branch and Bound*. La différence se situe au moment de l'affectation de la nouvelle borne inférieure du système. Le *Branch and Check* vérifie que la solution vérifie un certain nombre de contraintes. Cela se traduit par une résolution d'un problème de faisabilité. Si la solution est réalisable par le problème secondaire alors la borne inférieure est mise à jour sinon le noeud n'est pas conservé.

```
1  si n_f respecte les contraintes du problème de faisabilité alors
2  |  Borne inférieure = max(Borne inférieure ; Borne supérieure de n_f);
3  fin
```

Algorithme 2 : Algorithme de *Branch and Check* (en remplacement de la ligne 10 de Algorithme 1)

4 Analyse

1 Modélisation du problème

1.1 Description générale du problème

Soit le graphe G(V, A) représentant un réseau de transports urbains desservi par des bus hybrides, avec V la liste des sommets (les arrêts de bus) et A la liste des arcs (les trajets entre 2 arrêts). Soit L l'ensemble des lignes composant le réseau et $G_l(V_l, A_l)$, un sous-graphe de G, représentant la ligne de bus $l \in L$, le réseau entier des lignes est donc défini par $G(V, A) = \bigcup_{l \in L} G_l(V_l, A_l)$. Les bus hybrides ont besoin d'électricité, fournie par des chargeurs à placer tout le long des différentes lignes de bus.

La résolution du problème se fait en résolvant 2 sous-problèmes. L'objectif du premier sous-problème, le problème maître, est de trouver où placer les différents chargeurs, de savoir combien de temps il faut charger les bus aux différents chargeurs et quels tronçons du parcours il faut parcourir en mode électrique, afin d'effectuer la plus grande distance possible en mode électrique. Le second sous-problème, le problème esclave dépendant du problème maître, consiste à vérifier la disponibilité de chaque chargeurs lorsque les bus en ont besoin. Les données utilisées dans ce second problème sont en partie calculées par le problème maître. La résolution globale se fait avec l'algorithme *Branch and Check*.

1.2 Problème maître

Description du problème maître

Le problème maître a pour objectif de déterminer les informations relatives aux différents chargeurs sur le réseau de bus, leur emplacement, leur nombre et leur utilisation. Ces informations calculées doivent respecter un certain nombre de contraintes afin que la solution soit réalisable.

Tout d'abord, le placement des chargeurs possède un coût et ce coût ne doit pas dépasser le budget alloué à l'électrification du réseau de bus. Il faut tout de même placer un nombre suffisant de chargeur à un même arrêt de bus afin de respecter la fréquence de passage des bus d'une

ligne. Le chargement d'un bus peut prendre du temps, c'est pourquoi il ne doit pas dépasser une certaine durée, dans un soucis de qualité du service. De plus, les bus commencent leur tournée avec les batteries pleines et ils ne peuvent changer de mode de consommation uniquement lors d'un arrêt. L'énergie chargée doit donc permettre d'effectuer entièrement la distance entre 2 arrêts si le bus est en mode électrique. Pour finir, certaines portions du circuit doivent être parcourues en mode électrique, dans une optique de préservation de l'environnement dans les quartiers chargés en véhicule.

Hypothèses de travail

Afin de modéliser le problème, nous avons utilisé les hypothèses de travail suivantes.

- Tous les bus ont les mêmes caractéristiques (consommation électrique, capacité de batterie, modèle, etc.)
- Tous les chargeurs ont les mêmes caractéristiques (puissance électrique, voltage, etc.)
- On ne peut changer de mode de consommation uniquement à un arrêt.
- La consommation électrique est proportionnelle à la distance parcourue.
- La quantité d'énergie rechargée est proportionnelle au temps de chargement.

Données

- L représente l'ensemble des lignes de bus, composé des différents arrêts et des trajets parcourus
- f_l représente la fréquence de passage des bus de la ligne l
- d_{ij} représente la distance entre les arrêts i et j
- c_i représente le coût pour installer un chargeur à l'arrêt i. En effet, bien que le coût de la machine pour charger soit fixe, le coût d'installation varie en fonction de l'endroit où on place le chargeur, à cause des différents aménagements précédemment présents à enlever ou adapter.
- B représente le budget alloué à l'installation de chargeurs
- w_{li} représente la durée d'arrêt maximum à l'arrêt i sur la ligne l. Afin d'assurer une bonne qualité de service, les bus ne doivent pas s'arrêter trop longtemps à une station. Cette durée maximale tolérée varie en fonction de l'affluence sur la ligne.
- R représente la vitesse de chargement des batteries, ici l'intensité du courant du chargeur
- Q représente la capacité maximale des batteries
- e_{ij} représente la consommation électrique d'un bus entre les arrêts i et j. Cette consommation est déterminée par une consommation e moyenne qui peut varier selon certains critères tels que la topologie du circuit, la température ambiante ou bien la vitesse. Dans notre modèle, nous avons choisi de privilégier la vitesse moyenne pour déterminer la consommation électrique. On obtient donc $e_{ij} = e \times d_{ij}$.
- $m_{ij} = \begin{cases} 1 & \text{si la section entre les arrêts } i \text{ et } j \text{ doit être parcourue en mode électrique} \\ 0 & \text{sinon} \end{cases}$

Variables de décision

- x_i représente le nombre de chargeurs que l'on met à l'arrêt i
- t_{li} représente la durée de chargement à l'arrêt i sur la ligne l

— q_{li} représente la quantité d'énergie présente dans la batterie en arrivant à l'arrêt i sur la ligne l

— $y_{lij} = \begin{cases} 1 & \text{si la section entre l'arrêt } i \text{ et } j \text{ sur la ligne } l \\ & \text{est parcourue en mode électrique} \\ 0 & \text{sinon} \end{cases}$

Avec x_i, $t_{li} \in \mathbb{N}_+$, $q_{li} \in \mathbb{R}_+$ et $y_{lij} \in \{0, 1\}$

Contraintes

1. La coût total de l'installation des chargeurs sur les arrêts ne doit pas dépasser le budget total alloué à l'installation des chargeurs.

$$\sum_{i \in \mathrm{V}} c_i \times x_i \leq \mathrm{B} \tag{1}$$

2. Il est nécessaire de mettre un nombre suffisant de chargeur afin de respecter la fréquence de passage et d'éviter des temps d'attente trop importants dans le cas où les bus se rattrapent. Pour finir, il ne peut pas avoir de temps de chargement si il n'y a pas de chargeurs.

$$\max_{l \in \mathrm{L}} \left\lceil \frac{t_{li}}{f_l} \right\rceil \leq x_i : \forall\ i \in \mathrm{V} \tag{2}$$

3. Le temps de chargement à chaque arrêt ne doit pas excéder la durée maximale fixée à un arrêt.

$$t_{li} \leq w_{li} : \forall\ l \in \mathrm{L} \text{ et } \forall\ i \in \mathrm{V}_l \tag{3}$$

4. Le bus doit être en mode électrique dans les portions où c'est obligatoire, si $m_{ij} = 1$ alors $y_{lij} = 1$.

$$y_{lij} = 1 : \forall\ l \in \mathrm{L} \text{ et } \forall\ (i, j) \in \mathrm{A}_l \text{ tq } m_{ij} = 1 \tag{4}$$

5. Il n'est pas possible de charger la batterie d'un bus au delà de sa capacité. La quantité d'énergie présente dans la batterie du bus en arrivant à un arrêt avec la quantité chargée ne doit pas dépasser la capacité totale de la batterie.

$$q_{li} + t_{li} \times \mathrm{R} \leq \mathrm{Q} : \forall\ l \in \mathrm{L} \text{ et } \forall\ i \in \mathrm{V}_l \tag{5}$$

6. Les bus commencent leur itinéraire avec la batterie chargée.

$$q_{li} = \mathrm{Q} : \forall\ l \in \mathrm{L} \text{ et } \forall\ i \in \mathrm{V}_l \text{ tq } l[0] = i \tag{6}$$

7. La quantité d'énergie présente dans la batterie du bus en arrivant à l'arrêt j doit être égale à celle à l'arrêt i avec la quantité chargée en i moins celle consommée pour effectuer a distance entre i et j.

$$q_{li} + t_{li} \times \mathrm{R} - e_{ij} \times y_{lij} = q_{lj} : \forall\ l \in \mathrm{L} \text{ et } \forall\ (i, j) \in \mathrm{A}_l \tag{7}$$

Fonction objectif

L'objectif est de parcourir la plus grande distance possible en utilisant le moteur électrique.

$$\max z = \sum_{l \in \mathrm{L}} \sum_{(i,j) \in \mathrm{A}_l} d_{ij} \times y_{lij} \tag{a}$$

1.3 Problème esclave

Description du problème esclave

Le problème esclave a pour but de vérifier la disponibilité des chargeurs pour les différents bus voulant utiliser ces chargeurs. Cela permet de confirmer la faisabilité du planning des chargements à l'ensemble des arrêts sur une période donnée. La résolution du problème est appliquée à un ensemble de lignes indépendantes. Un ensemble de lignes est considéré comme indépendant au sein du réseau si aucune ligne ne partage de chargeur avec une ligne d'un autre ensemble indépendant de lignes. Il suffit d'appliquer la résolution à l'ensemble des lignes indépendants du réseau de bus afin d'obtenir la faisabilité pour toutes les lignes de bus. L'objectif du problème est donc de déterminer l'heure de départ de chaque ligne dans une période donnée afin que l'utilisation des chargeurs prévue dans le problème maître soit possible. Ces informations calculées doivent respecter un certain nombre de contraintes afin que la solution soit réalisable.

Pour chaque ligne, l'heure de départ permet de créer des profils pour chaque ligne à un arrêt donné. Le profil indique à quel moment un bus arrive à un chargeur considérant une période de temps donnée. Un ensemble de profils est dit compatible si et seulement si le nombre de lignes utilisant la station à chaque instant est inférieur ou égal au nombre de chargeurs sur la station. Ce problème assigne donc un profil à chaque ligne de bus du sous-réseau en fonction des données fournies par le problème maître. Si on ne peut pas de trouver des profils compatibles pour chaque ligne alors le problème n'est pas réalisable.

Données

- L représente l'ensemble des lignes de bus composant la partie indépendante du réseau
- f_l représente la fréquence de passage des bus de la ligne l
- x_i représente le nombre de chargeurs placés à l'arrêt i
- t_{li} représente la durée de chargement à l'arrêt i sur la ligne l
- tt_{ij} représente le temps de parcours entre les chargeurs i et j
- d^p_{lij} représente la date de début de la $j^{-ème}$ tache de rechargement du profil p d'un bus de la ligne l à l'arrêt i. Afin que toutes les lignes puissent effectuer un cycle complet de rechargement, la période de temps étudiée doit donc être $\text{PPCM}_{l\in \text{L}}(f_l) = \lambda$. Ainsi, chaque profil possède $\frac{\lambda}{f_l}$ taches de rechargement. Les profils sont générés à l'aide des autres données du problème de la manière suivante. A chaque arrêt, la première tache doit être effectuée entre la durée 0 et $f_l - 1$ afin de pouvoir affecter toutes les taches de rechargement. La tache initiale sur le premier arrêt de chaque ligne est effectuée à $d^p_{li_0 0} = p$, où i_0 est le premier arrêt de ma ligne l et p est le profil. Les premières taches de chargement des chargeurs suivants sont définies de la manière suivante $(d^p_{li0p} + t_{li} + tt_{ij}) \mod f_l = d^p_{lj0p}$, où i et j sont des chargeurs successifs utilisés par la ligne l. Les autres taches de rechargement sont définies de la manière suivante $d^p_{lij} + f_l = d^p_{li(j+1)}$.
- M représente un grand nombre qui aura pour effet de désactiver des contraintes sous certaines condition, ici $\text{M} = \lambda$

Variables de décision

- t_{lij} représente la date de début de la $j^{-ème}$ tache de rechargement d'un bus de la ligne l à l'arrêt i après la sélection du profil pour la ligne. Pour une ligne l, à chaque arrêt, on a n_l

taches de rechargement qui se composent de $n_l - 1 = \frac{\lambda}{f_l}$ taches de durée t_{li} et une $n_l^{-ème}$ tache de durée $\max(\lambda - t_{li(n_l-1)}; 0)$. Cette dernière tache est la fin de la $(n_l-1)^{-ème}$ tache qui n'a pas pu être finie car elle s'achète au delà de la période prévue. La durée correspond donc au temps de chargement qui n'a pas pu être effectuée dans la période.

— $y_{lij}^{ck} = \begin{cases} 1 & \text{si la } j^{-ème} \text{ tache de rechargement de la ligne } l \text{ est} \\ & \text{effectuée sur le chargeur } c \text{ de l'arrêt } i \text{ en } k^{-ème} \\ & \text{position, chaque chargeur possède } \sum_{\substack{l \in \mathrm{L} \\ i \in l}} n_l \text{ positions} \\ 0 & \text{sinon ou si } i \notin l \end{cases}$

— $x_l^p = \begin{cases} 1 & \text{si la ligne } l \text{ a le profil } p \\ 0 & \text{sinon} \end{cases}$

Avec $t_{lij} \in \mathbb{R}_+$ et $y_{lij}^{ck}, x_l^p \in \{0, 1\}$

Contraintes

1. Chaque ligne l ne peut avoir qu'un seul profil p.

$$\sum_p x_l^p = 1 : \forall\ l \in \mathrm{L} \tag{8}$$

2. La date réelle de début de la $j^{-ème}$ tache de rechargement de la ligne l correspond à la date de début de la tache de rechargement du profil sélectionné pour la ligne.

$$t_{lij} = \sum_p x_l^p \times d_{lij}^p : \forall\ l \in \mathrm{L}, \forall\ i \in \mathrm{V}_l \text{ et } \forall\ j \in \{1..n_l - 1\} \tag{9}$$

3. La $n_l^{-ème}$ de rechargement de la ligne l à l'arrêt i est placée au début du laps de temps.

$$t_{lin_l} = 0 : \forall\ l \in \mathrm{L} \text{ et } \forall\ i \in \mathrm{V}_l \tag{10}$$

4. Chaque tache d'une ligne l ne peux être effectuée que sur un seul chargeur c de l'arrêt i pendant le laps de temps.

$$\sum_c \sum_k y_{lij}^{ck} = 1 : \forall\ l \in \mathrm{L}, \forall\ i \in \mathrm{V}_l \text{ et } \forall\ j \in \{1..n_l\} \tag{11}$$

5. Chaque position d'un chargeur c ne peut être occupée au maximum que par une tache d'une ligne.

$$\sum_l \sum_j y_{lij}^{ck} \leq 1 : \forall\ i \in \mathrm{V}, \forall\ c \in \{1..x_i\} \text{ et } \forall\ k \in \{1..\sum_{\substack{l \in \mathrm{L} \\ i \in l}} n_l\} \tag{12}$$

6. Une position de rechargement sur le chargeur c à l'arrêt l ne peut être occupée que si la position précédente est aussi occupée.

$$\sum_l \sum_j y_{lij}^{c(k+1)} \leq \sum_{l'} \sum_{j'} y_{l'ij'}^{ck} : \forall\ i \in \mathrm{V}, \forall\ c \in \{1..x_i\} \text{ et } \forall\ k \in \{1..\sum_{\substack{l \in \mathrm{L} \\ i \in l}} n_l - 1\} \tag{13}$$

7. La date de début de la $j'^{-ème}$ tache de la ligne l' sur le chargeur c de l'arrêt i doit être égale à la date de début de la $j^{-ème}$ tache de la ligne l sur le même chargeur plus le temps de chargement de l à l'arrêt. Cette contrainte n'est valide que si l et l' sont sur le chargeur c et

que la $j^{-ème}$ tache de l arrive au chargeur juste avant la $j'^{-ème}$ tache de l', d'où la présence de la quantité M, un nombre grand qui aura pour effet de désactiver la contrainte si les conditions ne sont pas respectées.

$$\begin{aligned} t_{lij} + t_{li} \leq t_{l'ij'} + \mathrm{M} \times [2 - (y_{lij}^{ck} + y_{l'ij'}^{c(k+1)})] : \forall\ (l,l') \in \mathrm{L}^2, \forall\ i \in \mathrm{V}_l \cap \mathrm{V}_{l'}, \\ \forall\ c \in \{1..x_i\}, \forall\ k \in \{1..\sum_{\substack{l \in \mathrm{L} \\ i \in l}} n_l - 1\}, \forall\ j \in \{1..n_l - 1\} \text{ et } \forall\ j' \in \{1..n_l - 1\} \end{aligned} \tag{14}$$

8. La durée restante de la dernière tache de rechargement de la ligne l à l'arrêt i doit s'effectuer avant les taches de rechargement du nouveau cycle de rechargement.

$$\begin{aligned} t_{lin_l} + \lambda - t_{li(n_l-1)} \leq t_{l'ij'} + \mathrm{M} \times [2 - (y_{lin_l}^{ck} + y_{l'ij'}^{c(k+1)})] \\ \forall\ (l,l') \in \mathrm{L}^2, \forall\ i \in \mathrm{V}_l \cap \mathrm{V}_{l'}, \forall\ c \in \{1..x_i\}, \\ \forall\ k \in \{1..\sum_{\substack{l \in \mathrm{L} \\ i \in l}} n_l - 1\} \text{ et } \forall\ j' \in \{1..n_l - 1\} \end{aligned} \tag{15}$$

Fonction objectif

L'objectif de ce modèle est uniquement de vérifier que la solution issue du premier problème permet de donner un ordonnancement des chargements pour chaque arrêt. Il suffit que les variables de décisions respectent les contraintes du modèle. Il n'y a donc pas de fonction objectif dans cette modélisation.

2 Génération des instances

Le projet précédent, Section 1 (Chapitre 3), comportait une partie de génération d'instances afin de pouvoir tester le modèle. Cependant, il existe des différences de contexte et hypothèses entre les deux modélisation. Le format des instances doit être modifié.

Ces fichiers sont nommés *data_nombre*Arrêts*_nombre*Lignes_X, où X est le numéro du fichier pour un nombre d'arrêts donné. Un total de 10 fichiers seront générés par nombre d'arrêts et nombre de lignes différents. Il y aura des fichiers avec 200, 300 et 400 arrêts dans le réseau ainsi que 3, 5 et 7 lignes. Le fichier est composé de la manière suivante :

— |V|, le nombre d'arrêts
— |L|, le nombre de lignes
— $|l|$, le nombre de d'arrêts dans la ligne l, suivi de la composition de la ligne. Tout d'abord, on va affecter |V| – |L| – 1 sommets de G aléatoirement aux différentes lignes. On place ensuite aléatoirement les sommets restants dans plusieurs lignes afin de symboliser les arrêts en commun, de manière à connecter l'ensemble du graphe. Cette procédure permet de limiter les connexions entre les lignes ce qui rend le réseau réaliste. Les nombres générés aléatoirement seront générés selon la loi de probabilité uniforme.
— f_l, la fréquence de passage. La fréquence de passage est de un bus toutes les 10 minutes.
— d_{ij}, la distance entre 2 arrêts compris entre 300 et 600 m (comme vu dans le document [WWW9]). Ces entiers seront générés aléatoirement selon la loi de probabilité uniforme.
— tt_{ij}, la durée de parcours entre 2 arrêts. En utilisant la vitesse moyenne v, on peut calculer $tt_{ij} = \frac{d_{ij}}{v}$. Pour un bus urbain, on considère une vitesse moyenne $v = 18km.h^{-1}$.[2]

— c_i, le prix d'installation des chargeurs à chaque arrêt, composé du prix de base du chargeur, \$500 000 (comme vu dans le document [7]), avec un ajout de 25 à 45% du prix du chargeur comme prix d'installation. Le coût ajouté est généré aléatoirement selon la loi de probabilité uniforme.
— B, le budget défini par $\frac{\sum_{l\in \mathrm{L}} \sum_{(i,j)\in \mathrm{A}_l} e_{ij}}{|\mathrm{L}|\times \mathrm{Q}} \times \bar{c_i} \times \alpha$, où $\bar{c_i} = \frac{\sum_{i\in \mathrm{V}} c_i}{|\mathrm{V}|}$ représente la moyenne des coûts d'installation des chargeurs c_i et α est un coefficient choisi empiriquement.
— w_{li}, le temps de recharge maximum. Cette durée est de 2 minutes.
— R, la vitesse de chargement qui est égale à la puissance [1] du chargeur, 150 kW.[2]
— Q, la capacité de la batterie, 19 kWh pour un bus hybride électrique [2]
— e_{ij}, la consommation électrique des bus. En utilisant la consommation globale e, on peut calculer $e_{ij} = e \times d_{ij}$. En conditions difficiles (période de forte affluence), la consommation est 2.44 $kWh.km^{-1}$, soit 2 fois plus que lors de périodes normales, on considérera donc une consommation $e = 1.22kWh.km^{-1}$.[2]
— m_{ij}, les sections obligatoires, 10% des sections sont obligatoires. Ces sections sont réparties autour de un ou deux points placés aléatoirement sur chaque ligne du circuit, de manière à avoir le nombre de sections électriques obligatoires.

3 Statistiques liées à la résolution

Afin d'analyser les résultats obtenus dans le fichier contenant la solution, il faut effectuer une partie d'analyse statistique. Les données sont agrégées si elles proviennent d'instances avec le même nombre d'arrêts et de lignes de bus. Chaque valeur est donc calculée pour chaque couple $(nombre\ d'arrêts, nombre\ de\ lignes)$.

— Le temps total moyen de chargement sur le circuit, où t_{li} est le temps de chargement de la ligne l à l'arrêt i

$$\frac{1}{nombre\ d'instance} \times \sum_{instances} \sum_{l\in \mathrm{V}} \sum_{i\in \mathrm{A}_l} t_{li}$$

— Le pourcentage moyen d'électrification, où d_{ij} est la distance entre les arrêts i et j et y_{lij} indique si la portion (i,j) de la ligne l est électrifiée

$$\frac{1}{nombre\ d'instance} \times \sum_{instances} \frac{\sum_{i\in \mathrm{L}} \sum_{(i,j)\in \mathrm{A}_l} d_{ij} \times y_{lij}}{\sum_{i\in \mathrm{L}} \sum_{(i,j)\in \mathrm{A}_l} d_{ij}}$$

— Le nombre moyen de chargeurs placés, où x_i est le nombre de chargeurs placés à l'arrêt i

$$\frac{1}{nombre\ d'instance} \times \sum_{instances} \sum_{i\in \mathrm{V}} x_i$$

— Le nombre moyen d'appel au problème esclave
— Le temps moyen de résolution
— Le temps minimum et maximum de résolution

Les valeurs moyennes sont accompagnées de leur écart-type respectif afin de regarder la répartition des valeurs, $\sigma_{\mathrm{X}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2 - \bar{x}^2}$, où $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$.

1. La vitesse de chargement en heures est définie par $t = \frac{\mathrm{C}}{\mathrm{I}}$, où C est la capacité de la batterie en Ah et I est l'intensité du courant en A. De plus, P = U × I et C (en Wh) = U × C (en Ah) $\Rightarrow t = \frac{\mathrm{C}}{\mathrm{P}}$, avec C en W$h$ et P en W. Donc P = $\frac{\mathrm{C}}{t}$ est définie comme la vitesse de chargement.
2. Volvo buses, communication personnelle

5 Conception et mise en oeuvre

1 Données

1.1 Génération

Les données sont générées de la manière présentée dans la Section 2 (Chapitre 4). Une grande partie des données génère uniquement des tableaux de nombres aléatoires, compris dans un intervalle, ou est calculée avec des formules. Cependant, certaines de ces structures sont plus complexes à générer.

Lignes de bus

Pour construire les lignes, on assigne aléatoirement $nbLines - 1$ arrêts aux différentes lignes. On obtient donc des lignes avec des arrêts, déconnectées les unes des autres. La suite va permettre de relier les différentes lignes afin d'obtenir un réseau connexe. On considère deux ensembles de lignes : les lignes dans le réseau et les lignes en dehors. Tant que toutes les lignes ne sont pas dans le réseau, on sélectionne une ligne dans chaque ensemble et leur assigne un sommet non assigné. La ligne anciennement en dehors du réseau fait donc maintenant partie du réseau. Ce processus est décrit dans l'Algorithme 3.

Sections électriques

Une fois que les lignes sont construites, on peut désigner quelles sections sont obligatoirement électriques. Cette désignation se fait ligne par ligne car la proportion d'électrique est définie par ligne et non sur l'ensemble des sections du réseau. On choisit tout d'abord le nombre de sections électriques contiguës ainsi que leur taille. On calcule ensuite le centre de la première section de manière à ce qu'elle loge dans la ligne et passe en mode électrique la section entière. Si il y a une deuxième section électrique, on calcule son centre de manière à ce qu'il soit dans la moitié opposée de celui de la première section, afin d'éviter les recouvrements. On passe aussi la section entière en mode électrique. Si il n'a pas assez de parties électriques au sein de la ligne pour atteindre le seuil requis, on va tout d'abord rajouter des passages électriques à la

```
1  Initialiser la liste des arrêts;
2  Créer nbLines lignes vides;
3  tant que Nombre de arrêts non assignés > nbLines − 1 faire
4  |  Extraire a de la liste des arrêts;
5  |  Assigner a à une ligne du réseau;
6  fin
7  si nbLines > 1 alors
8  |  Lignes dans le réseau ← ∅;
9  |  Lignes en dehors du réseau ← Ensemble des lignes;
10 |  tant que Nombre de lignes en dehors du réseau ≠ 0 faire
11 |  |  Extraire l_o une ligne en dehors du réseau;
12 |  |  si Nombre de lignes dans le réseau = 0 alors
13 |  |  |  Extraire l_i une ligne en dehors du réseau;
14 |  |  sinon
15 |  |  |  Extraire l_i une ligne dans le réseau;
16 |  |  fin
17 |  |  Extraire a de la liste des arrêts;
18 |  |  Assigner a à l_i et l_o;
19 |  |  Insérer l_i et l_o dans la liste des lignes dans le réseau;
20 |  fin
21 fin
```

Algorithme 3 : Algorithme de création des lignes de bus

suite de la dernière section. Ensuite, on va rajouter des passages électriques avant la première section. Pour finir, on va compléter entre les deux sections en partant de la fin de la première. Ce processus est décrit dans l'Algorithme 4.

1.2 Format de fichier

Le format de fichier pour les données est unique au sein de l'application et est utilisé pour charger et sauvegarder les données. Le fichier est composé de la manière suivante (consulter la Figure 5 (Annexe G) pour voir le fichier) :

- Le nombre d'arrêts dans le réseau
- Le nombre de lignes de bus dans le réseau
- Pour chaque ligne de bus, le nombre d'arrêts dans la ligne suivie de la composition de la ligne
- Le vecteur des fréquences de passage de chaque ligne
- La matrice de distance entre tous les arrêts de bus du réseau
- La matrice des temps de parcours entre tous les arrêts de bus du réseau
- Le vecteur du coût d'installation d'un chargeur à un arrêt donné
- Le budget prévu pour l'installation des chargeurs
- La matrice des temps de chargement maximum pour tous les arrêts de chaque ligne
- La vitesse de chargement
- La capacité des batteries de bus
- La matrice des consommations électriques entre tous les arrêts de bus du réseau
- La matrice des sections obligatoirement électriques

```
1  pour chaque Ligne l faire
2      Créer une copie de la ligne l avec uniquement des lignes non électriques;
3      Définir le nombre de sections électriques dans la ligne;
4      Définir le nombre de sections électriques continues;
5      Calculer la taille de la première section continue;
6      si Il y a deux sections alors
7          Taille de la deuxième section ← Nombre de sections électriques dans la ligne -
             Taille de la première section;
8      sinon
9          Taille de la première section ← Nombre de sections électriques dans la ligne;
10     fin
11     Placer le centre de la première section sur la ligne;
12     Mettre en électrique les portions autour du premier centre;
13     si Il y a deux sections alors
14         si Le premier centre est dans la première moitié de la ligne alors
15             Placer le second centre dans la seconde moitié de la ligne;
16         sinon
17             Placer le second centre dans la première moitié de la ligne;
18         fin
19         Mettre en électrique les portions autour du second centre;
20     fin
21     tant que Il n'y a pas assez de sections électriques faire
22         Ajouter des sections électriques après la dernière section jusqu'à atteindre la fin de
             la ligne;
23     fin
24     tant que Il n'y a pas assez de sections électriques faire
25         Ajouter des sections électriques avant la première section jusqu'à atteindre le début
             de la ligne;
26     fin
27     tant que Il n'y a pas assez de sections électriques faire
28         Ajouter des sections électriques entre les sections en commençant par la fin de la
             première section;
29     fin
30     Ajouter la copie de la ligne à l'ensemble définissant les sections électriques;
31 fin
```

Algorithme 4 : Algorithme de création des sections électriques

2 Solution

La partie résolution a été pensée de la même manière que le solveur utilisé, *Cplex*. La résolution fonctionne donc comme une boite noire avec laquelle on interagit pas ou peu. C'est pourquoi nous avons deux types de solution : la solution résolvable et la solution utilisable. Le programme utilise donc la solution résolvable afin de trouver la solution optimale du problème et renvoit son équivalent en solution utilisable, par l'utilisateur.

2.1 Solution résolvable

La solution résolvable, *SolvableSolution*, possède des attributs utilisables par le solveur. Elle implémente le modèle à résoudre et effectue la résolution. La solution est ensuite renvoyée dans

un format utilisable indépendamment de *Cplex*.

Utilisation de CPLEX

Afin d'implémenter le modèle et d'en trouver la solution optimale, nous allons utiliser la librairie *ILOG CPLEX*, version 12.8, de l'entreprise *IBM* en *C++* et coder un programme qui résoudra le problème. Ce programme va utiliser un solveur appelé *CPLEX*, un solveur est un outil permettant de faire des calculs et de résoudre un problème une fois qu'il a été mis sous forme informatique. Dans notre programme, le modèle est une instance de la classe `IloModel` et le solveur est instance de `IloCplex`. Les données peuvent utiliser des structures de données classiques (**int**, `std::vector`, etc.) ou bien leurs équivalences ajoutées par la librairie (`IloInt`, `IloNumArray`, etc.). Les variables de décision sont des instances de `IloNumVar` et sont ajoutées au modèle. Au moment de la déclaration, on peut fournir à la variable des bornes (minimale et maximale), un type (entier, réel, booléen) et un nom. Les contraintes sont des instances de la classe `IloRange` que l'on ajoute aussi au modèle. À la déclaration de la contrainte, on fournit les bornes de la contrainte (ces bornes peuvent être infinies), l'expression à borner et le nom de la contrainte. Pour finir, la fonction objectif est une instance de `IloObjectif`. Lors de la déclaration, on doit fournir l'expression de la fonction, le sens (maximiser ou minimiser) et un nom. Il est conseillé de donner un nom à chaque élément car c'est ce qui est utilisé dans la rédaction du modèle lorsqu'on demande son extraction dans le format *.lp*, format utilisé pour obtenir notre modèle détaillé. Il ne reste plus qu'à appeler le solveur et lire la solution obtenue. On peut noter qu'il est possible de paramétrer le solveur (temps de calcul maximal, nombre de thread utilisé, algorithme utilisé dans la résolution, etc.) afin de le rendre plus efficace selon le problème étudié. Pour finir, la libraire possède un nombre important de fonctions utilisées dans la résolution de problèmes d'optimisation (chronomètre, produit scalaire, somme, arrondi, etc.). Le solveur s'utilise donc comme une boite noire que l'on paramétrise avant de lancer l'outil. Une description plus approfondie de l'utilisation de la libraire est fournie par *IBM* dans le manuel utilisateur [WWW7].

Callbacks

Ainsi, on a pu voir que *cplex* est un outil paramétrable. On peut, à travers ces paramètres, influencer le parcours des solutions par le solveur grâce aux *Callbacks*. Ils permettent d'effectuer des traitements sur la solution courante, comme la rejeter, ou bien d'en obtenir des informations. Il existe un type de *callback* spécifique, appelé `IncumbentCallback`, qui permet d'effectuer des traitements uniquement sur des solutions entières. Il s'agit donc du type de *callback* utilisé afin d'implémenter le problème esclave. On crée donc une classe `SlaveCallbackI` héritée de `IncumbentCallbackI` qui permet d'utiliser un *callback* avec un nombre non fixe de paramètres. Il faut de plus surcharger la méthode `main` dans la classe. Cette méthode est automatiquement appelée par le solveur dans la résolution du problème maître. L'héritage permet d'avoir accès à certaines informations comme la valeur de la meilleure solution ou la valeur de la solution courante. Au sein de cette méthode `main`, on peut y effectuer tout type d'opérations, notamment la résolution d'un autre problème de programmation linéaire comme notre problème esclave.

Sections indépendantes

Avant de résoudre le problème esclave, il faut trouver l'ensemble des sections électriquement indépendantes. Ceci permet de séparer la résolution du problème en problèmes plus petits et plus faciles à résoudre. Tout d'abord, on associe à chaque arrêt les lignes qui utilisent l'arrêt en

question. Ensuite, on parcourt l'ensemble de ces lignes associées et on fusionne les groupes de lignes ayant des lignes en commun. Ainsi, on obtient des groupes contenant chacun des arrêts différents. Ce processus est décrit dans l'Algorithme 5.

```
1  pour chaque Arrêt dans le réseau faire
2  |  Associer à l'arrêt les lignes chargeant à l'arrêt
3  fin
4  pour chaque Groupe g1 de lignes associées faire
5  |  pour chaque Groupe g2 de lignes associées faire
6  |  |  si g1 et g2 ont une ligne en commun alors
7  |  |  |  Fusionner les deux groupes
8  |  |  fin
9  |  fin
10 fin
```

Algorithme 5 : Algorithme de création des sections indépendantes

2.2 Solution utilisable

La solution utilisable, `ComputableSolution`, contient les mêmes attributs que les classes `SolvableSolution` et `SlaveCallbackI` cependant elle utilise des types issus de la bibliothèque standard. Cette classe contient les méthodes de vérification d'une solution ainsi que de création d'indices statistiques.

2.3 Format de fichier

Le format de fichier pour la solution est unique au sein de l'application et est utilisé pour charger et sauvegarder les solutions. Le fichier est composé de la manière suivante (consulter la Figure 6 (Annexe G) pour voir le fichier) :

- La distance parcourue en mode électrique et la durée de résolution
- Le nombre de chargeurs placés à chaque arrêt
- Le temps de chargement pour chaque arrêt pour chaque ligne
- Le niveau de batterie pour chaque arrêt pour chaque ligne
- La matrice des sections parcourues en mode électrique entre chaque arrêt pour chaque ligne du réseau
- Le nombre de sections électriquement indépendantes, pour chaque information suivante, les résultats sont regroupés par sections
- La taille de la section suivie de la composition de la section
- Le nombre d'appel au problème esclave pendant la résolution

3 Création d'instances

La création d'instances est faite par la classe `InstanceMaker` et permet de faire le lien avec la méthode `buildData` de la classe `Data`. La classe crée le dossier *data* dans le système si il n'existe pas et y exporte l'instance générée. Des valeurs par défaut sont utilisées afin de générer les données et ces valeurs correspondent à de vraies informations de la technologie utilisée dans les systèmes de bus. Afin de créer des instances avec des paramètres spécifiques, il faut utiliser directement les méthodes de la classe `Data`.

4 Vérificateur de solution

La vérification d'une solution est faite par la classe `SolutionChecker` et permet faire le lien avec la méthode `check` de la classe `ComputableSolution`. Cette méthode vérifie que chaque contrainte du problème (maître et esclave) soit respectée. La classe contient un attribut initialisé lorsque la vérification échoue, cet attribut permet de savoir la contrainte qui n'est pas valide. Le résultat de la vérification se fait par un affichage, en récupérant la valeur de l'attribut.

5 Calculateur d'indices statistiques

Le calcul des indices statistiques est fait par la classe `StatisticsMaker` et permet de faire le lien avec les différentes méthodes de calcul statistique de la classe `ComputableSolution`. La classe charge chaque donnée et solution à analyser et calcule indépendamment les différents indices. Ces indices sont ensuite agrégés afin d'en calculer une moyenne et un écart-type. Les agrégations sont exportées dans un dossier *stats* créé par la classe.

6 Résultats

Les résultats ont été regroupés par types d'instance (nombre d'arrêts et nombre de lignes), des moyennes et des écarts types ont été calculés sur 10 entrées (voir Section 3 (Chapitre 4)). Ils ont été obtenus par implémentation du modèle en C++ à l'aide de la librairie *ILOG CPLEX* de chez *IBM*. Les temps de chargement correspondent à la durée totale de chargement des bus dans le réseau. Les résolutions ont été effectuées en utilisant un processeur *Intel Core i3-8350k, 4.00GHz, 8.00 Go RAM*.

On obtient les résultats suivants.

Table 1 – *Résultats obtenues avec* $\alpha = 0.7$ *(Partie 1)*

|V|	|L|	Circuit électrifié (%)	Chargeurs placés	Temps de chargement (s)	Nombre d'appel au sous problème
200	3	60.51 ± 0.51	1.0 ± 0.0	200.4 ± 13.2	8.3 ± 0.9
	5	83.68 ± 0.05	0.0 ± 0.0	0.0 ± 0.0	13.0 ± 0.0
	7	97.14 ± 0.30	0.0 ± 0.0	0.0 ± 0.0	6.2 ± 1.0
300	3	45.51 ± 1.50	2.0 ± 0.0	408.0 ± 58.8	17.0 ± 7.3
	5	63.58 ± 0.73	1.0 ± 0.0	240.0 ± 0.0	25.2 ± 1.0
	7	82.07 ± 0.20	0.0 ± 0.0	0.0 ± 0.0	11.3 ± 0.5
400	3	34.71 ± 1.61	2.0 ± 0.0	432.0 ± 58.79	9.4 ± 2.2
	5	48.21 ± 0.01	1.0 ± 0.0	240.0 ± 0.0	18.9 ± 1.4
	7	65.19 ± 0.02	1.0 ± 0.0	238.7 ± 1.2	24.0 ± 4.6

On peut voir dans la Table 1 que la proportion de ligne électrifiée diminue si on augmente le nombre d'arrêt. Cela est dû au fait que la distance totale à parcourir par ligne augmente cependant, la capacité de batterie des véhicules reste constante. De plus, l'augmentation du budget induit par l'augmentation de la taille du réseau n'est pas assez conséquent pour acheter un chargeur supplémentaire. On constate l'effet opposé quand on augmente le nombre de lignes de bus pour des raisons similaires. Une augmentation du nombre de lignes va diminuer la taille globale des lignes.

Table 2 – *Résultats obtenues avec $\alpha = 0.7$ (Partie 2)*

|V|	|L|	Temps CPU (s)		
		Moyenne	Min	Max
200	3	0.467 ± 0.029	0.437	0.531
	5	0.497 ± 0.184	0.297	0.828
	7	0.400 ± 0.028	0.359	0.469
300	3	5.770 ± 3.921	0.875	9.016
	5	820.787 ± 259.578	606.781	1146.440
	7	0.941 ± 0.031	0.891	1.000
400	3	92.638 ± 111.006	1.797	229.766
	5	2165.250 ± 1167.59	300.000	3240.330
	7	1543.500 ± 1439.19	33.813	3278.200

On peut aussi constater que le nombre moyen de chargeurs placés augmente si on augmente le nombre d'arrêt. Cela est dû au fait que la distance totale à parcourir par ligne augmente et que le budget est défini en fonction de la distance à parcourir. On remarque l'effet opposé quand on augmente le nombre de lignes de bus pour des raisons similaires. Une augmentation du nombre de lignes va diminuer la taille globale des lignes ce qui va faire diminuer le budget permettant d'installer des chargeurs.

Certaines instances ne placent pas de chargeurs mais ont cependant une grande partie de leur réseau électrifié. Cela peut s'expliquer par plusieurs facteurs. Tout d'abord, le réseau est peut être trop petit pour pouvoir générer un budget suffisant à l'achat d'un chargeur. Ensuite, dans ces cas là, les bus utilisent l'énergie fournie au début de la tournée.

Dans la Table 2, on peut voir que le temps de résolution est très variable mais il a tout de même tendance à augmenter lorsque les réseaux sont plus gros et qu'on place plus de chargeur. De plus, on constate une tendance à avoir des comportements similaires entre le temps de résolution et le nombre d'appel au sous problème. Cependant, l'augmentation du temps de résolution et du nombre d'appel au sous problème n'est pas similaire, l'augmentation du temps de résolution est bien plus importante. Ceci laisse donc penser que la résolution du problème esclave n'est pas très coûteuse. Plus le temps de résolution est long, plus de solution sont explorées et par conséquent plus de solutions entières. C'est donc l'augmentation du temps de résolution qui provoque cette légère augmentation du nombre d'appel au problème esclave et non l'inverse. On peut tout de même noter que certaines instances (possédant 400 arrêts dans le réseau) ne trouvent pas forcément la solution optimale. Leur résolution est arrêtée par le solveur. La limite de résolution étant fixée à une heure, le programme arrête la résolution en réalité vers 55 minutes de résolution (3300 secondes).

7 Axes d'amélioration

Bien qu'on ait obtenu des résultats satisfaisants, on peut trouver des axes d'amélioration dans les différentes parties où des choix ont été faits : la génération des données, la méthode de résolution et la création des statistiques.

On a pu voir que le budget générait une contrainte assez forte dans le placement des chargeurs et donc dans l'électrification des lignes, il faudrait donc augmenter successivement le paramètre *alpha* afin d'avoir plus grand budget et ainsi étudier la variation de la proportion d'électrique. Cependant, je ne pense pas que le budget va augmenter au point de pouvoir ajouter un nouveau chargeur mais on pourra plutôt placer le chargeur à des endroits plus avantageux. Ensuite, la génération des lignes induit une certaine typologie du réseau. Cette typologie définit indirectement les connections avec les différentes lignes. Dans le cas actuel, il y a assez peu de connexion entre les lignes, ainsi le problème esclave est très simple à résoudre. On pourrait penser dans un premier temps à implémenter différentes stratégies de génération de lignes de bus avec des structures en triangle et d'autres en étoile. Dans un second temps, on pourrait créer un réseau plus réaliste, semblable à celui d'une métropole, avec une forte densité dans le centre et une plus faible sur les extrémités.

En augmentant la complexité des lignes, la durée totale d'exécution du problème esclave va prendre plus de temps. Il pourrait donc être intéressant de limiter l'appel au sous-problème. Un première idée serait d'appliquer le sous problème sur des solutions non entières, en utilisant un autre type de *Callback*, lorsque certaines variables sont définies. Cet appel anticipé pourrait permettre d'éviter de développer plus loin certaines solutions déjà infaisables pour le sous problème. Ensuite, on pourrait aussi essayer de déduire des informations quand à l'infaisabilité de certaines solutions. Ces informations permettraient d'appliquer des coupes supplémentaires sur le modèle du problème maître, afin d'éliminer des solutions infaisables pour le problème esclave.

Pour finir, on peut affiner la présentation des statistiques. Tout d'abord, on peut faire des statistiques que sur des résolutions terminées car on est sûr de l'optimalité. Dans le cap contraire, on affiche la différence entre la borne supérieure du problème et la valeur de la fonction objectif trouvée afin nuancer ces résultats partiels. Pour finir, on peut aussi mesurer la durée totale utilisée pour le sous-problème afin de regarder l'importance qu'il a dans la résolution. On compte aussi le nombre de fois où le sous -problème a été valide et où il a été invalide afin de voir s'il est vraiment efficace.

8 Bilan et conclusion

Pendant cette première partie de PRD, j'ai pu réaliser grâce à l'état de l'art que le problème sur lequel je travaille est assez nouveau. Il y a donc tout un tas de découvertes à effectuer. Je pense qu'avec cette première modélisation, un gros travail a été fait. Cependant, après le test sur des instances générées, il faudrait tester le modèle sur de réels réseaux de bus. On peut tout de même noter que le modèle est fortement orienté par la technologie utilisée (type de bus hybride, type de chargeurs, etc.) et doit donc être utilisé dans un contexte où il peut avoir du sens.

Pour le prochain semestre, il reste à effectuer toutes les parties de programmation (voir Annexe E) puis à analyser les solutions trouver afin de regarder si les résultats sont satisfaisants.

Pendant cette seconde partie de PRD, j'ai pu mettre en oeuvre la résolution de mon problème. Pour cela, j'ai changé le comportement par défaut de *Cplex* en utilisant la technologie des *Callback*. Cela m'a permis d'implémenter mon modèle mathématique de la manière dont il a été théoriquement pensé. J'ai obtenu une application qui me permet de répondre aux besoins que j'avais en début de projet. J'ai donc obtenu des résultats prouvant que le modèle mis en place fonctionne.

Pour la suite du projet, divers améliorations sont possibles et sont exposées dans le Chapitre 7.

Annexes

A Technologie existante relative au problème

Pendant le stage servant de base au projet, une analyse de la technologie existante relative aux bus électriques a été faite. Cette partie a donc été placée en annexe dans la mesure où elle est utile pour comprendre le contexte du projet.

1 Types de bus

1.1 Bus conventionnels

Les bus utilisés jusqu'à maintenant sont des bus avec un moteur à combustion interne (*ICE*) (voir Figure 1). Pour que ce moteur fonctionne, il a besoin de sources d'énergie fossile telle que du gazole ou de l'essence. Par l'utilisation de ces énergies fossiles, les bus conventionnels émettent des gaz de pot d'échappement (des gaz à effet de serre) tels que l'oxyde d'azote NO_x, des hydrocarbures C_nH_m ainsi que du monoxyde de carbone CO. La production de ces gaz provoque une augmentation de la densité du brouillard urbain dans les villes. Cependant, diverses alternatives à ces bus conventionnels existent déjà dont l'objectif est de rendre la conduite plus silencieuse et moins polluante.

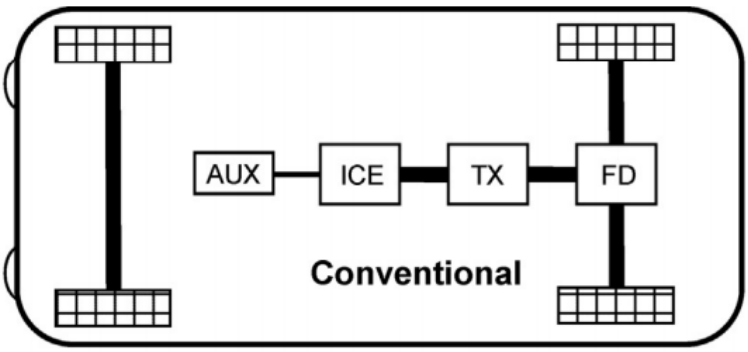


Figure 1 – *Schéma simplifié d'un bus conventionnel (AUX = dispositifs auxiliaires, ICE = moteur à combustion interne, TX = transmission, FD = conducteur final) [3]*

1.2 Bus à batterie électrique

Tout d'abord, il y a les bus à batterie électrique. Ces bus ont la particularité de réduire la consommation de carburant et les émissions de CO_2 de 100%, du fait qu'ils ne possèdent pas d'ICE (voir Figure 2). L'absence du moteur rend la conduite plus lisse et silencieuse. Il y a aussi moins de vibration au sein du véhicule ainsi que moins de pollution. Les bus à batterie électrique présentent donc un avantage pour ceux qui occupent le bus comme pour l'environnement autour.

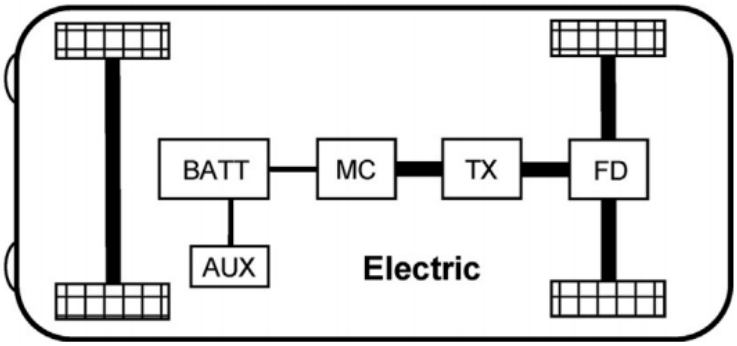


Figure 2 – *Schéma simplifié d'un bus à batterie électrique (AUX = dispositifs auxiliaires, BATT = batterie, MC = moteur/contrôleur, TX = transmission, FD = conducteur final) [3]*

1.3 Bus hybrides

Les bus hybrides sont des bus possédant 2 sources d'énergie. Il peut donc utiliser du diesel (comme un bus conventionnel) ou de l'énergie électrique (comme un bus à batterie électrique). Ces bus ont la particularité de réduire la consommation de carburant et les émissions de CO_2 seulement de 75% à cause de la partie diesel. Il existe 2 types de bus hybrides. Dans la disposition en série, l'ICE n'est pas relié aux roues mais il recharge la batterie électrique à l'aide du générateur (voir Figure 3). C'est donc la batterie qui assure la propulsion du véhicule. Dans la disposition en parallèle, on peut voir (Figure 4) que le moteur à combustion et la batterie électrique sont reliés aux roues. Cette disposition permet de choisir quel mode de fonctionnement utiliser selon la situation.

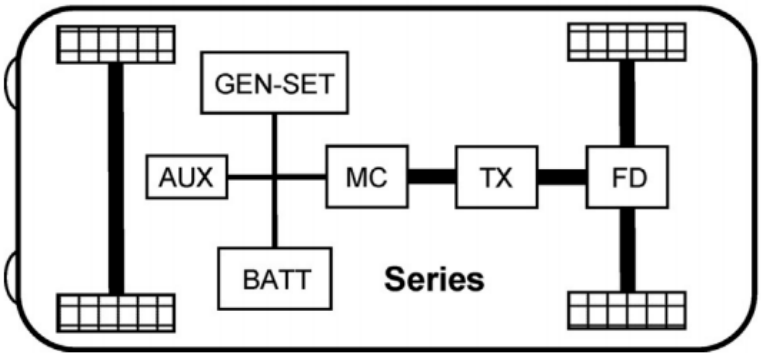


Figure 3 – *Schéma simplifié d'un bus hybride à propulsion en série (AUX = dispositifs auxiliaires, BATT = batterie, GEN-SET = moteur/générateur, MC = moteur/contrôleur, TX = transmission, FD = conducteur final) [3]*

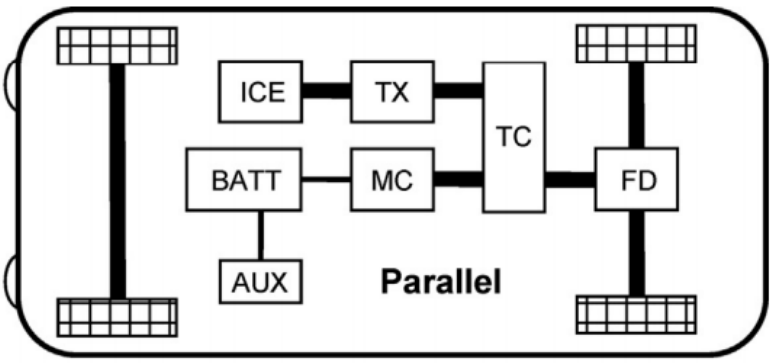


Figure 4 – *Schéma simplifié d'un bus hybride à propulsion parallèle (AUX = dispositifs auxiliaires, BATT = batterie, MC = moteur/contrôleur,ICE = moteur à combustion interne, TX = transmission, TC = coupleur, FD = conducteur final) [3]*

1.4 Bus de pile à combustible

Le bus de pile à combustible ou *fuel cell bus* est un type de bus qui utilise une pile à combustible à hydrogène afin de faire charger continuellement la batterie électrique du bus et ainsi faire avancer le bus. L'utilisation de l'hydrogène implique qu'il n'y a pas d'émission de gaz de pot d'échappement mais seulement de la vapeur d'eau et de la chaleur, chaleur utilisée pour les services du bus tel que le chauffage. L'utilisation d'un gaz au lieu d'un liquide (carburant) fait qu'on peut en stocker énormément en augmentant la pression et donc la distance pouvant être parcourue est très grande. L'hydrogène est stockée dans plusieurs bonbonnes positionnées sur le toit du bus, augmentant la place disponible pour des passagers. De plus, recharger les bonbonnes ne prend seulement que 7 minutes bien qu'il s'agisse d'une technologie en évolution [WWW6], cette durée est donc susceptible d'évoluer à l'avenir. Cependant, ces bus ont besoin de stations de rechargement en hydrogènes (*HRS*) et sont donc limités aux itinéraires courts (dans la limite du réservoir) ou bien avec des HRS. De plus, les techniques pour produire de l'hydrogène ne sont pas faciles à mettre en place d'un point de vue technique. C'est notamment le cas de l'électrolyse de l'eau (séparation en H_2 et O_2 depuis des molécules d'eau à l'aide d'un courant électrique) et du reformage du méthane (des molécules d'hydrocarbures comme le méthane sont brisées en H_2 et CO), qui sont les 2 méthodes les plus répandues de production d'hydrogène. Ainsi, l'hydrogène ne peut être renouvelable que si elle est produite grâce à l'électrolyse de l'eau en utilisant de l'électricité produite écologiquement (barrages, éoliennes, centrales géothermiques, etc.) ce qui est loin d'être majoritairement le cas. Cette technologie n'est donc pas encore confirmée pour s'inscrire durablement dans le domaine des transports.

2 Types de recharge

2.1 Recharge continue

La recharge continue est le maintient constant du véhicule à une source de courant électrique. Les véhicules électriques n'ont donc pas besoin de batterie pour fonctionner. Le courant est fourni par 2 caténaires, des câbles électriques suspendus au dessus de la ligne. Étant donné que le bus ne possède pas de batterie, il doit rester en contact avec les caténaires tout le temps et ne peut ainsi pas dévier de sa trajectoire. Cela rend les itinéraires de bus totalement inflexibles au changement (sous peine de devoir déplacer tous les câbles d'alimentation). L'exemple typique de bus utilisant ce type de recharge est le trolleybus.

Figure 5 – *Trolleybus de Tours [WWW8]*

Tous les autres types de recharge présentés sont dit non continus car les bus possèdent des batteries et ne sont pas continuellement reliés au réseau électrique.

2.2 Recharge nocturne

La recharge nocturne est le fait de recharger totalement la batterie du véhicule pendant une période d'inactivité. Pour les transports, cette période d'inactivité est généralement la nuit d'où le fait que la recharge soit appelée nocturne. Il s'agit d'un type de chargement facile à mettre en place car il ne nécessite qu'un espace de stockage pour que le ou les véhicules où ils puissent se recharger. De plus, les véhicules ont toute la période d'inactivité pour se recharger, il n'y a donc pas besoin d'une grande vitesse de transfert de l'énergie (entre 30 et 60 kW [1]). Le gros avantage de ce type de chargement est que le véhicule s'utilise de la même manière qu'un véhicule à énergie fossile, à la différence qu'il pollue moins. En effet, le bus s'arrête uniquement aux arrêts présents sur l'itinéraire. Cependant, cette grande quantité d'énergie stockée nécessite une grosse batterie qui prend donc beaucoup de place dans le bus.

2.3 Recharge ponctuelle

La recharge ponctuelle est le fait de recharger, partiellement ou totalement, la batterie du véhicule électrique durant la période d'utilisation (la tournée pour les bus). Ce type de recharge nécessite une grande vitesse de transfert de l'énergie (jusqu'à 500 kW[1]) afin de ne pas faire attendre trop longtemps le véhicule et les personnes qu'il transporte. Afin de mettre en place la recharge ponctuelle, il suffit d'installer des infrastructures physiques le long de l'itinéraire du véhicule (des chargeurs). Plus il y aura de chargeurs et moins le bus devra charger d'énergie à chaque fois, en effet, il suffit de charger juste assez d'énergie pour atteindre la prochaine station de chargement. Ainsi le bus restera moins longtemps au même endroit. Afin de charger totalement les 4 batteries du bus Volvo 7900 Electric, qui est un bus totalement électrique, un chargeur possédant une puissance de 300 kW n'a besoin que de 6 minutes[1] (ces chargeurs chargent les batteries 2 par 2). Sachant qu'un bus ne se charge pas totalement mais uniquement jusqu'au chargeur suivant, le temps de chargement est assez court. De plus, si il y a moins d'énergie à charger, il n'est pas nécessaire d'avoir une grosse batterie, ce qui représente un gain de place et de masse conséquent et cela permet donc de placer plus de personnes dans le bus. Cependant, le placement des chargeurs rend la ligne peu flexible aux changements. En cas de changement, il faut soit faire passer la nouvelle ligne par des chargeurs existants, soit ajouter ou déplacer des chargeurs.

1. Volvo buses, communication personnelle

2.4 Changement de batterie

Il existe un dernier type de recharge, il s'agit du changement de batterie. Au cours de l'itinéraire du véhicule d'électrique, la batterie vide est enlevée et remplacée par une nouvelle batterie, chargée. Le "chargement" est donc bien effectué au cours de la période d'utilisation. Cette méthode possède l'avantage de ne pas placer de dispositifs le long du trajet et ainsi le rendre flexible aux changement et économique. Cependant, ayant besoin d'une quantité d'énergie plus importante, la batterie prend plus de place dans le bus.

2.5 Recharge combinée

Bien que la recharge ponctuelle permette d'effectuer une distance infinie sur une route spécifique, on peut combiner les différents type de recharge afin d'optimiser ses coûts. En effet, il suffit de trouver un équilibre entre des batterie pas trop grosses qui coûteront moins cher (pour empêcher les cotés négatifs de la recharge nocturne) et limiter le nombre de chargeurs placés sur la route (pour empêcher les cotés négatifs de la recharge ponctuelle). Ainsi, plusieurs combinaisons ont été pensées. On peut faire une recharge nocturne puis revenir au dépôt dans les heures creuses de la journée afin de recharger de la même manière que la recharge nocturne. Cela permettra de parcourir plus de distance. On peut aussi charger durant la nuit afin de partir avec une batterie pleine puis faire de la recharge ponctuelle le long de l'itinéraire. On utiliserait donc les deux types de recharge mais avec moins de batterie nécessaire.

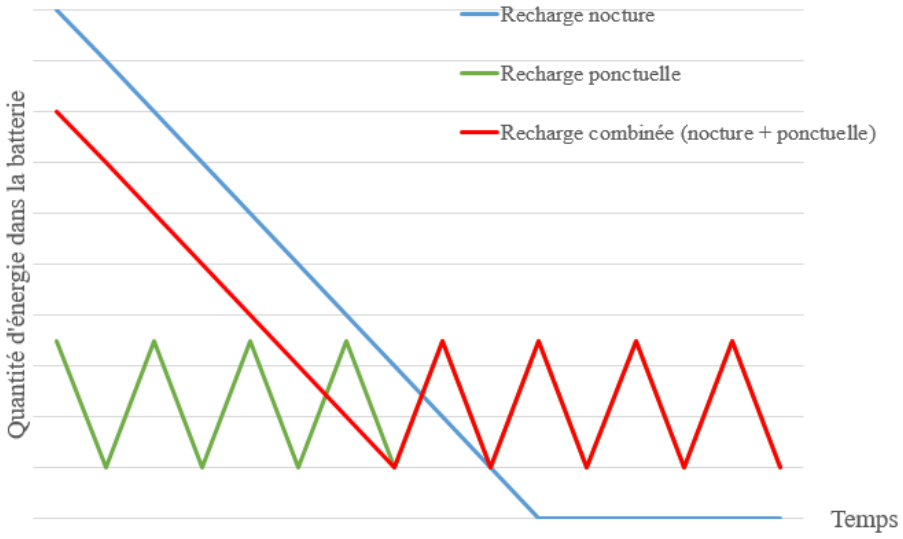
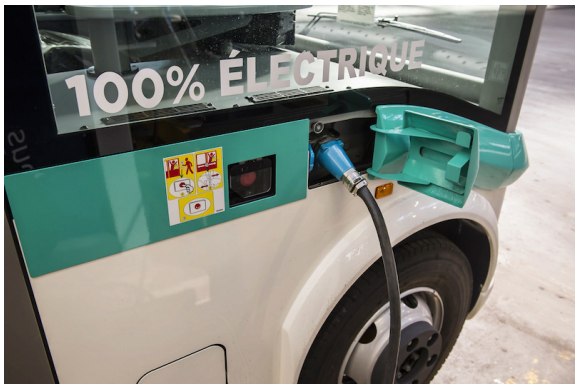


Figure 6 – *Graphe comparatif de la quantité d'énergie dans la batterie au cours du temps (valeurs arbitraires mais représentatives)*

On peut voir, dans la Figure 6, que la recharge nocturne permet d'aller beaucoup moins loin que les recharges utilisant (en entièrement ou non) la recharge ponctuelle. De plus, utiliser sur une plus longue distance la recharge ponctuelle va multiplier l'installation de chargeurs et donc le prix total de l'installation et de l'entretien. En cas d'utilisation de la recharge combinée, il faut donc trouver le pourcentage de nocturne et de ponctuelle adapté aux besoins du véhicule.

2.6 Recharge à conduction

La recharge à conduction est un type de recharge où un contact direct entre la source d'énergie et la batterie est nécessaire afin de transmettre le courant (à l'image d'une prise électrique). Le contact permet d'avoir une grande vitesse de transfert de l'énergie (jusqu'à 300 *k*W) tout en ayant très peu de pertes avec 95% d'efficacité[1]. C'est une technologie mature et fiable ce qui a permis sa démocratisation et son faible coût. Ce type de recharge est devenu le design dominant dans le marché des types de rechargement. La recharge se fait couramment par une pompe à l'image du carburant (Figure 7a) ou bien d'un pantographe mettant la source et la batterie en contact (Figure 7b).



(a) *"Pompe" utilisée pour recharger électriquement les bus [WWW2]*

(b) *Pantographe en utilisation pour charger un bus [WWW1]*

Figure 7 – *Mécanismes utilisant la recharge à conduction*

2.7 Recharge à induction

La recharge à induction est un type de recharge ne nécessitant pas de contact entre la source d'énergie et la batterie afin de transmettre le courant. Cette absence de contact empêche d'avoir une grande vitesse de transfert de l'énergie et la limite entre 50 et 200 *k*W. De plus, l'efficacité n'est que de 80%[1]. Cette technologie est récente et a encore besoin d'améliorations afin de la rendre concurrente sur le marché. Son coût est donc assez élevé comparé à de la recharge à conduction.

B Interfaces du logiciel

1 Interface matériel/logiciel

Le logiciel ne communique pas spécialement avec des périphériques de la machine. Seul le processeur est utilisé afin réaliser les calculs. Un meilleur processeur donne donc de meilleures performances au niveau des temps de calcul, mais pas pour le reste des indicateurs statistiques calculés.

2 Interface homme/machine

L'application est une application console. Les différentes fonctionnalités directement utilisables par l'utilisateur sont appelées à l'aide d'arguments dans la commande. Il n'y a donc pas d'IHM à développer. A chaque évaluation d'un noeud de la procédure de branchement, un message est affiché avec le résultat obtenu. A la fin de la procédure, la valeur finale de la fonction objectif est affichée avec le temps de résolution. Cet affichage est l'affichage par défaut du solveur *Cplex*.

3 Interface logiciel/logiciel

L'application va utiliser la librairie *ILOG CPLEX*, version 12.8, de l'entreprise *IBM* afin d'utiliser les méthodes de résolution de programmation linéaire.

C Spécifications fonctionnelles

1 Création d'une instance

Cette fonction a pour but de créer une instance utilisable par le solveur. La priorité de la fonction est secondaire car on pourrait créer les instances à la main. Cependant, pour des raisons pratiques et spécialement si on veut en créer un grand nombre, il est très utile d'avoir une fonction permettant de créer une instance. La fonction de création prend en entrée le nombre d'arrêts dans le réseau, le nombre de lignes de bus à mettre dans le réseau ainsi que le numéro de l'instance (ce nombre n'est utilisé que pour le nommage du fichier). Elle va produire un fichier texte contenant toutes les données du problème maître suivant les caractéristiques présentées dans la Section 2 (Chapitre 4). Si jamais le nom de fichier créé depuis les paramètres existe déjà dans le répertoire courant, alors le nouveau fichier remplace l'ancien.

2 Résolution du problème général

Cette fonction est la fonction principale du programme et a pour but de résoudre le problème de mise place du réseau électrique. La priorité de la fonction est primordiale car l'algorithme mis en jeu, ici la méthode de séparation et évaluation (voir Algorithme 2 (Chapitre 3)), est difficile à résoudre à la main. La fonction prend en paramètre des données fournies par la fonction d'importation des données. Si l'importation échoue alors la fonction ne s'exécute pas et l'erreur liée à l'importation est renvoyée. Cette fonction renvoie un statut lié à l'état de la solution (*optimal* ou *infaisable*), si le statut est *optimal* alors la solution trouvée par le problème est renvoyée à la fonction exportation d'une solution. Cependant il est peu probablement que le statut soit *infaisable* car il dépend principalement du statut des deux autres fonctions de résolution. Ces deux fonctions sont donc appelées par la fonction principale de résolution suivant l'algorithme utilisé pour la résolution. Cette fonction sert aussi de tampon entre les deux sous problèmes à résoudre et permet de transférer les solutions trouvées de l'une à l'autre. La fonction renvoie une solution envoyée à la fonction d'exportation de solution une fois la résolution effectuée.

3 Résolution du problème d'électrification

Cette fonction a pour but de résoudre le problème d'électrification, le problème maître (voir Section 1.2 (Chapitre 4)). La priorité de la fonction est primordiale car l'algorithme mis en jeu, ici la méthode de séparation et évaluation (voir Algorithme 1 (Chapitre 3)), est difficile à résoudre à la main. La fonction prend en paramètre des données fournies par la fonction de résolution du problème général et lui renvoie la solution trouvée au problème. Cette fonction renvoie un statut lié à l'état de la solution (*optimal* ou *infaisable*), si le statut est *optimal* alors les solutions trouvées par le problème sont aussi renvoyées à la fonction appelant la résolution. Cependant il est peu probablement que le statut soit *infaisable* car le problème est peu contraignant. Il suffit d'électrifier uniquement les portions obligatoires afin d'avoir une solution acceptable.

4 Résolution du problème de faisabilité

Cette fonction a pour but de résoudre le problème de faisabilité, le problème esclave (voir Section 1.3 (Chapitre 4)). La priorité de la fonction est primordiale car l'algorithme mis en jeu, ici la méthode de séparation et évaluation (voir Algorithme 1 (Chapitre 3)), est difficile à résoudre à la main. La fonction prend en paramètre une solution et des données du problème maître. Ces paramètres sont fournis par la fonction de résolution du problème général qui permet de faire le lien avec la fonction de résolution du problème maître. Cette fonction renvoie un statut lié à l'état de la solution (*optimal* ou *infaisable*), si le statut est *optimal* alors les solutions trouvées par le problème sont aussi renvoyées à la fonction appelant la résolution.

5 Vérification d'une solution

Cette fonction a pour but de vérifier que la solution respecte bien l'ensemble des contraintes liées au problème, que ce soit celles du problème maître comme du problème esclave. La priorité de cette fonction est secondaire car on pourrait vérifier en faisant les calculs à la main. Cependant vu le nombre de contraintes à vérifier, il est plus pratique de créer une fonction qui vérifie tout afin de gagner du temps. La fonction prend en paramètre un fichier contenant les données d'un problème et un fichier contenant la solution calculée avec les données. Elle utilise donc les fonctions d'importation des données et d'une solution. Si une contrainte n'est pas respectée alors un message d'erreur est envoyé dans la console indiquant la contrainte non respectée. Sinon un message de validation est envoyé à la fin de la vérification. Si les fichiers ne respectent pas le format attendu, celui produit une erreur de format de fichier incorrect avec un message dans la console indiquant la partie du fichier incorrectement chargée.

6 Création de statistiques liées aux solutions

Cette fonction a pour but de créer des statistiques à partir des résolutions effectuées. La priorité de cette fonction est facultative mais elle permet de comprendre les résultats obtenus. La fonction prend en entrée des fichiers de données ainsi que les fichiers de résultat associés. Cette fonction utilise donc les fonctions d'importation des données et d'une solution. Les résultats obtenus sont écrits dans un fichier texte. Si les fichiers ne respectent pas le format attendu, celui produit une erreur de format de fichier incorrect avec un message dans la console indiquant la partie du fichier incorrectement chargée.

7 Importation des données

Cette fonction a pour but de charger des données dans le programme. La priorité de la fonction est secondaire car on pourrait écrire les données directement dans le programme. Cependant, il est préférable d'utiliser une fonction afin de pouvoir adapter son programme à différentes instances. La fonction prend en entrée un fichier texte contenant des données et renvoie des données du problème. Si le fichier ne respecte pas le format attendu, celui produit une erreur de format de fichier incorrect avec un message dans la console indiquant la partie des données incorrectement chargée.

8 Exportation d'une solution

Cette fonction a pour but d'écrire une solution dans un fichier. La priorité de la fonction est secondaire car on pourrait regarder la solution directement dans le programme en utilisant le mode *Debug*. Cependant, pour des raisons d'utilisation, il est préférable d'utiliser une fonction. la fonction prend en entrée une solution et l'écrit dans une fichier texte. La solution est fournie par la fonction de résolution du problème principal.

9 Importation d'une solution

Cette fonction a pour but de charger une solution dans le programme. La priorité de la fonction est secondaire car elle n'est utilisée que par des fonctions de priorité secondaire. La fonction prend en entrée un fichier texte contenant une solution et renvoie une solution du problème. Si le fichier ne respecte pas le format attendu, celui produit une erreur de format de fichier incorrect avec un message dans la console indiquant la partie de la solution incorrectement chargée.

D Spécifications non fonctionnelles

1 Contraintes de développement et conception

Afin d'implémenter et de résoudre le problème, on utilise un solveur mathématique et *ILOG CPLEX* de l'entreprise *IBM* a été choisi. Ce solveur possède des API dans un certain nombre de langages (*C*, *C++*, *Java*, *Python*, *.NET*, *Matlab*, etc.). Les résolutions doivent se faire le plus rapidement possible, c'est pourquoi le langage *C++* est utilisé pour la partie résolution du projet. Pour faire fonctionner *ILOG CPLEX* en *C++*, il est beaucoup plus pratique d'utiliser l'IDE Visual Studio.

La génération d'instances dans le projet précédent était en Java cependant elle n'est pas dans le même programme que la partie résolution. Si on ne veut qu'un unique livrable, il faut aussi faire la génération d'instances en *C++*.La vérification de l'instance doit pouvoir être appelée juste après la résolution, il est donc plus pratique que cette vérification soit faite dans le même langage que la résolution. La vérification est donc en *C++*. Ainsi, n'ayant pas de nécessité liée à une technologie, toute l'application est en *C++* afin de pouvoir plus facilement faire une unique application.

Les tests unitaires sont effectués avec *Google Test*. Il s'agit d'une librairie produite par *Google* qui permet d'effectuer des tests unitaires. Cette librairie est incluse par défaut dans Visual Studio depuis la version 15.5.

2 Contraintes de fonctionnement et d'exploitation

2.1 Performances

L'application n'a pas de performances limites attendues. Le temps de réponse est le temps de résolution de l'ensemble des instances et cela peut prendre plusieurs minutes.

2.2 Capacités

L'application n'a de limite précise, elle doit juste pouvoir instancier assez de variables afin de représenter la modélisation du système.

2.3 Contrôlabilité

Une fois le modèle instancié, il est écrit dans un fichier au format *.lp* ce qui permet de vérifier, en cas de problème dans la résolution, que le modèle résolu est bien celui attendu. De plus, chaque résultat de séparation puis relaxation est écrit dans la console, il s'agit du comportement par défaut de *Cplex*.

2.4 Sécurité

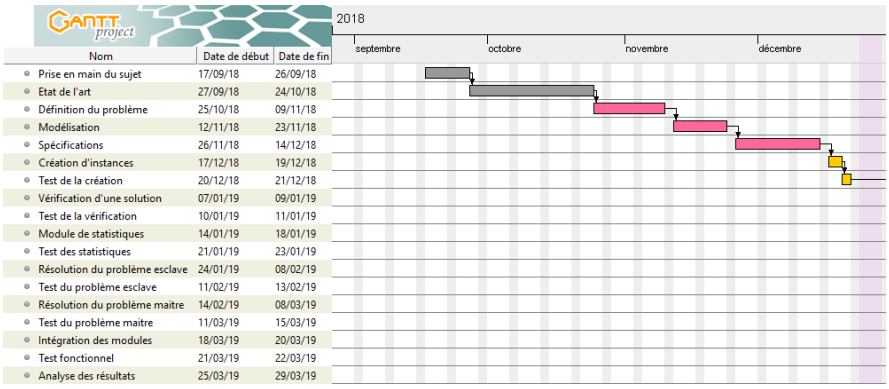
L'application ne contient aucune donnée sensible et ne possède pas de système d'authentification, elle n'est donc soumise à aucune sécurité.

3 Évolution du système

La partie la plus susceptible d'évoluer est la modélisation mathématique. Un changement dans la modélisation va bien entendu modifier la partie résolution mais possiblement aussi le format de des données et des solutions. Une première idée d'évolution serait de passer de la programmation linéaire à la programmation par contrainte, ce qui simplifierait grandement certaines contraintes de la modélisation. Une seconde évolution serait de passer à un problème multi-objectif, où il faudrait maximiser la distance à parcourir et minimiser l'argent dépensé pour la mise en place du réseau. Cette résolution pourrait se faire par la méthode ε-contrainte.

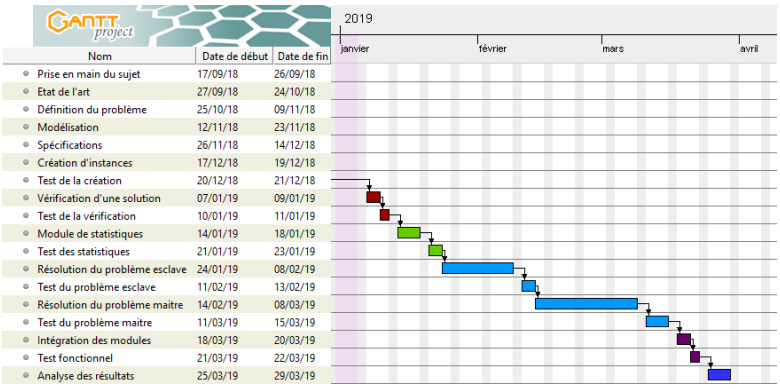
E Planification du projet

Le projet est prévu de se dérouler de la manière suivante (voir Figure 1) pour finir fin Mars 2019. Il faut que noter que les fonctionnalités d'importation et d'exportation des solutions sont développées en même que les fonctionnalités qui dépendent d'elles.



Nom	Date de début	Date de fin
Prise en main du sujet	17/09/18	26/09/18
Etat de l'art	27/09/18	24/10/18
Définition du problème	25/10/18	09/11/18
Modélisation	12/11/18	23/11/18
Spécifications	26/11/18	14/12/18
Création d'instances	17/12/18	19/12/18
Test de la création	20/12/18	21/12/18
Vérification d'une solution	07/01/19	09/01/19
Test de la vérification	10/01/19	11/01/19
Module de statistiques	14/01/19	18/01/19
Test des statistiques	21/01/19	23/01/19
Résolution du problème esclave	24/01/19	08/02/19
Test du problème esclave	11/02/19	13/02/19
Résolution du problème maitre	14/02/19	08/03/19
Test du problème maitre	11/03/19	15/03/19
Intégration des modules	18/03/19	20/03/19
Test fonctionnel	21/03/19	22/03/19
Analyse des résultats	25/03/19	29/03/19

(a) *Planification du projet pour 2018*



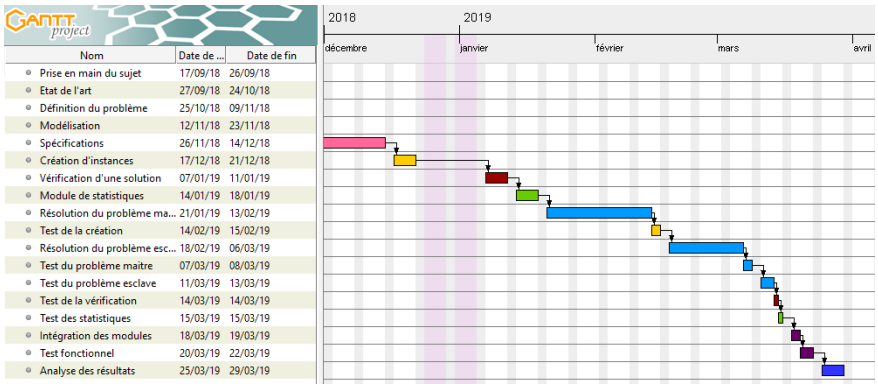
Nom	Date de début	Date de fin
Prise en main du sujet	17/09/18	26/09/18
Etat de l'art	27/09/18	24/10/18
Définition du problème	25/10/18	09/11/18
Modélisation	12/11/18	23/11/18
Spécifications	26/11/18	14/12/18
Création d'instances	17/12/18	19/12/18
Test de la création	20/12/18	21/12/18
Vérification d'une solution	07/01/19	09/01/19
Test de la vérification	10/01/19	11/01/19
Module de statistiques	14/01/19	18/01/19
Test des statistiques	21/01/19	23/01/19
Résolution du problème esclave	24/01/19	08/02/19
Test du problème esclave	11/02/19	13/02/19
Résolution du problème maitre	14/02/19	08/03/19
Test du problème maitre	11/03/19	15/03/19
Intégration des modules	18/03/19	20/03/19
Test fonctionnel	21/03/19	22/03/19
Analyse des résultats	25/03/19	29/03/19

(b) *Planification du projet pour 2019*

Figure 1 – *Diagramme de Gantt illustant la planification prévue du projet*

Pendant le développement, plusieurs étapes ont été déplacées. En effet, la durée de formation à l'outil de test, Google Test, avait mal été estimée et les fonctionnalités liées ont donc été repoussées après la formation. De plus, la recherche de l'implémentation du problème est esclave m'a pris beaucoup plus de temps que prévu. En effet, il s'agit d'une technique peu

répandue et peu documentée. C'est aussi durant cette recherche que j'ai appris que le problème principal devait être implémenté avant. Pour toutes ces raisons, certaines tâches ont été déplacées pendant le projet. Le planning pour cette seconde partie de projet peut être retrouvée dans la Figure 2.



Nom	Date de ...	Date de fin
Prise en main du sujet	17/09/18	26/09/18
Etat de l'art	27/09/18	24/10/18
Définition du problème	25/10/18	09/11/18
Modélisation	12/11/18	23/11/18
Spécifications	26/11/18	14/12/18
Création d'instances	17/12/18	21/12/18
Vérification d'une solution	07/01/19	11/01/19
Module de statistiques	14/01/19	18/01/19
Résolution du problème ma...	21/01/19	13/02/19
Test de la création	14/02/19	15/02/19
Résolution du problème esc...	18/02/19	06/03/19
Test du problème maitre	07/03/19	08/03/19
Test du problème esclave	11/03/19	13/03/19
Test de la vérification	14/03/19	14/03/19
Test des statistiques	15/03/19	15/03/19
Intégration des modules	18/03/19	19/03/19
Test fonctionnel	20/03/19	22/03/19
Analyse des résultats	25/03/19	29/03/19

Figure 2 – *Diagramme de Gantt illustant la planification effective de la seconde partie de projet*

F Test de l'application

Les sections suivantes comportent les fiches de test des différents tests effectués ou à effectuer des différents fonctionnalités développées.

1 Tests de *Utils*

Les tests suivants (Table 1 à Table 4) sont les tests de la classe `Utils`, réalisés lors de la séquence de test `TestUtils`.

Table 1 – *Fiche de test de ObservationPeriod*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestUtils	TestObeservationPeriod
Description	Calcule le plus petit multiple commun de 3 nombres (3, 5 et 7) premiers entre eux deux à deux	
Résultat attendu	Résultat obtenu	Statut
105	105	Validé

Table 2 – *Fiche de test de ObservationPeriod, cas Common*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestUtils	TestObeservationPeriodCommon
Description	Calcule le plus petit multiple commun de 3 nombres (3, 5 et 6), certains n'étant pas premiers entre eux	
Résultat attendu	Résultat obtenu	Statut
30	30	Validé

Table 3 – *Fiche de test de Deviation*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestUtils	TestDeviation
Description	Calcule l'écart type d'une série de 3 valeurs différentes (1, 2 et 3)	
Résultat attendu	Résultat obtenu	Statut
0.816497 ± 0.01	0.816497	Validé

Table 4 – *Fiche de test de Deviation, cas Zero*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestUtils	TestDeviationZero
Description	Calcule l'écart type d'une série de 3 valeurs identiques	
Résultat attendu	Résultat obtenu	Statut
0	0	Validé

2 Tests de *Data*

Les tests suivants (Table 5 à Table 20) sont les tests de la classe `Data`, réalisés lors de la séquence de test `TestData`. Lorsque les tests ont le résultat `SEH Exception`, un test manuel a été réalisé. Le test fonctionnel manuel ne montre pas de différence entre les données attendues et les données obtenues. L'origine de l'exception est donc inconnue.

Table 5 – *Fiche de test de DefaultConstructor*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestData	TestDefaultConstructor
Description	Appel au constructeur par défaut	
Résultat attendu	Résultat obtenu	Statut
$nbStops = 0$ $nbLines = 0$	$nbStops = 0$ $nbLines = 0$	Validé

Table 6 – *Fiche de test de ValueConstructor*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestData	TestValueConstructor
Description	Appel au contructeur avec initialisation des attributs	
Résultat attendu	Résultat obtenu	Statut
$nbStops = 10$ $nbLines = 10$	$nbStops = 2$ $nbLines = 2$	Validé

Table 7 – *Fiche de test de FileConstructor*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestData	TestFileConstructor
Description	Appel à la méthode statique construisant un objet complet, exportation puis importation avec le constructeur, test de la cohérence des données en utilisant la surcharge de l'opérateur ==	
Résultat attendu	Résultat obtenu	Statut
Vrai	Vrai	Validé

Table 8 – *Fiche de test de BuildLines*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestData	TestBuildLines
Description	Construction des lignes de bus pour une instance initialisée et vérification du nombre d'arrêt sur le réseau	
Résultat attendu	Résultat obtenu	Statut
$nbStops + nbLines - 1$	$nbStops + nbLines - 1$	Validé

Table 9 – *Fiche de test de BuildFrequency*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestData	TestBuildFrequency
Description	Construction des fréquences de chaque ligne pour une instance initialisée et vérification de la fréquence de passage de chaque ligne	
Résultat attendu	Résultat obtenu	Statut
$60 \times frequency$	$60 \times frequency$	Validé

3 Tests de *ComputableSolution*

Les tests suivants (Table 21 à Table 26) sont les tests de la classe `ComputableSolution`, à réaliser lors de la séquence de test `TestComputableSolution`.

4 Tests de *InstanceMaker*

Les tests suivants (Table 27 à Table 28) sont les tests de la classe `InstanceMaker`, à réaliser lors de la séquence de test `TestInstanceMaker`.

5 Tests de *SolutionChecker*

Les tests suivants (Table 29 à Table 32) sont les tests de la classe *SolutionChecker*, à réaliser lors de la séquence de test *TestSolutionChecker*.

Table 10 – *Fiche de test de BuildDistance*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestData	TestBuildDistance
Description	Construction des distance entre chaque arrêts pour une instance initialisée et vérification du respect des bornes	
Résultat attendu	Résultat obtenu	Statut
$distance \geq minDistance$ $distance \leq maxDistance$	$distance \geq minDistance$ $distance \leq maxDistance$	Validé

Table 11 – *Fiche de test de BuildDistance, cas Error*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestData	TestBuildDistanceError
Description	Construction des distance entre chaque arrêts pour une instance initialisée avec des bornes incohérentes	
Résultat attendu	Résultat obtenu	Statut
Exception soulevée	Exception soulevée	Validé

Table 12 – *Fiche de test de BuildTimeTravel*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestData	TestBuildTimeTravel
Description	Construction des temps de parcours entre chaque arrêts pour une instance initialisée (dont les distances) et vérification du respect de la formule	
Résultat attendu	Résultat obtenu	Statut
$3.6 \times distance/speed$	$3.6 \times distance/speed$	Validé

6 Tests de *StatisticsMaker*

Le test suivant (Table 33) est le test de la classe `StatisticsMaker`, à réaliser lors de la séquence de test `TestStatisticsMaker`.

Table 13 – *Fiche de test de BuildInstallationCost*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestData	TestBuildInstallationCost
Description	Construction des coûts d'installation à chaque arrêt sur une instance initialisée et vérification du respect des bornes	
Résultat attendu	Résultat obtenu	Statut
$cost \geq basicCost \times (1 + \frac{minIncr}{100})$ $cost \leq basicCost \times (1 + \frac{minIncr}{100})$	$cost \geq basicCost \times (1 + \frac{minIncr}{100})$ $cost \leq basicCost \times (1 + \frac{minIncr}{100})$	Validé

Table 14 – *Fiche de test de BuildInstallationCost, cas Error*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestData	TestBuildInstallationCostError
Description	Construction des coûts d'installation à chaque arrêt sur une instance initialisée avec des variations incohérentes	
Résultat attendu	Résultat obtenu	Statut
Exception soulevée	Exception soulevée	Validé

Table 15 – *Fiche de test de BuildBudget*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestData	TestBuildBudget
Description	Construction du budget sur une instance initialisée (dont les lignes, la capacité, les coûts d'installation, les distances et les consommations) et vérification du respect de la formule	
Résultat attendu	Résultat obtenu	Statut
$\frac{\sum_{l \in L} \sum_{(i,j) \in A_l} e_{ij}}{\lvert L \rvert \times Q} \times \bar{c}_i \times \alpha$	$\frac{\sum_{l \in L} \sum_{(i,j) \in A_l} e_{ij}}{\lvert L \rvert \times Q} \times \bar{c}_i \times \alpha$	Validé

Table 16 – *Fiche de test de BuildMaxLoadingTime*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestData	TestBuildMaxLoadingTime
Description	Construction du temps maximal de chargement sur une instance initialisée et vérification du respect de la formule	
Résultat attendu	Résultat obtenu	Statut
$maxLoadingTime \times 60$	$maxLoadingTime \times 60$	Validé

Table 17 – *Fiche de test de BuildLoadingSpeed*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestData	TestBuildLoadingSpeed
Description	Construction de la vitesse de chargement sur une instance initialisée	
Résultat attendu	Résultat obtenu	Statut
$loadingSpeed$	$loadingSpeed$	Validé

Table 18 – *Fiche de test de BuildCapacity*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestData	TestBuildCapacity
Description	Construction de la capacité des batteries sur une instance initialisée	
Résultat attendu	Résultat obtenu	Statut
$capacity$	$capacity$	Validé

Table 19 – *Fiche de test de BuildElectricalConsumption*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestData	TestBuildElectricalConsumption
Description	Construction de la consommation électrique sur une instance initialisée (dont les distances) et vérification de la cohérence de la formule	
Résultat attendu	Résultat obtenu	Statut
$distance \times \frac{unitElecCons}{1000}$	$distance \times \frac{unitElecCons}{1000}$	Validé

Table 20 – *Fiche de test de BuildElectricalSection*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestData	TestBuildElectricalSection
Description	Construction des sections électriques sur une instance initialisée (dont les lignes) et vérification du nombre de sections électriques	
Résultat attendu	Résultat obtenu	Statut
$\%ageElec \times size(l) \forall l \in L$	$\%ageElec \times size(l) \forall l \in L$	Validé

Table 21 – *Fiche de test de Check*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestComputableSolution	TestCheck
Description	Lecture d'un objet ComputableSolution avec un fichier contenant des valeurs respectant toutes les contraintes	
Résultat attendu	Résultat obtenu	Statut
$Vrai$		Non implémenté

Table 22 – *Fiche de test de Check, cas FailMaster*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestComputableSolution	TestCheckFailMaster
Description	Lecture d'un objet ComputableSolution avec un fichier contenant des valeurs ne respectant pas certaines contraintes du problème maître	
Résultat attendu	Résultat obtenu	Statut
$Faux$ *checkError*		Non implémenté

Table 23 – *Fiche de test de Check, cas FailSlave*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestComputableSolution	TestCheckFailSlave
Description	Lecture d'un objet ComputableSolution avec un fichier contenant des valeurs ne respectant pas certaines contraintes du problème esclave	
Résultat attendu	Résultat obtenu	Statut
$Faux$ *checkError*		Non implémenté

Table 24 – *Fiche de test de ComputeTotalLoadingTime*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestComputableSolution	TestComputeTotalLoadingTime
Description	Lecture d'un objet ComputableSolution avec un fichier et calcul du temps total de chargement	
Résultat attendu	Résultat obtenu	Statut
$\sum_{l \in \mathrm{L}} \sum_{i \in l} t_{li}$		Non implémenté

Table 25 – *Fiche de test de ComputeElectricRatio*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestComputableSolution	TestComputeElectricRatio
Description	Lecture d'un objet ComputableSolution avec un fichier et calcul du ratio parcouru en électrique	
Résultat attendu	Résultat obtenu	Statut
$\frac{\sum_{i \in \mathrm{L}} \sum_{(i,j) \in \mathrm{A}_l} d_{ij} \times y_{lij}}{\sum_{i \in \mathrm{L}} \sum_{(i,j) \in \mathrm{A}_l} d_{ij}}$		Non implémenté

Table 26 – *Fiche de test de ComputeTotalLoaders*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestComputableSolution	TestComputeTotalLoaders
Description	Lecture d'un objet ComputableSolution avec un fichier et calcul du nombre total de chargeur	
Résultat attendu	Résultat obtenu	Statut
$\sum_{i \in V} x_i$		Non implémenté

Table 27 – *Fiche de test de CreateInstance, cas Number*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestInstanceMaker	TestCreateInstanceNumber
Description	Création d'une instance avec la méthode générant le nom du fichier et test de la cohérence des données en utilisant la surcharge de l'opérateur ==	
Résultat attendu	Résultat obtenu	Statut
Vrai	Vrai	Validé

Table 28 – *Fiche de test de CreateInstance, cas Name*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestInstanceMaker	TestCreateInstanceName
Description	Création d'une instance avec la méthode utilisant le nom du fichier et test de la cohérence des données en utilisant la surcharge de l'opérateur ==	
Résultat attendu	Résultat obtenu	Statut
Vrai	Vrai	Validé

Table 29 – *Fiche de test de CheckSolution, cas Number*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestSolutionChecker	TestCheckSolutionNumber
Description	Lecture d'une solution avec la méthode utilisant le numéro du fichier et vérifie les contraintes	
Résultat attendu	Résultat obtenu	Statut
Affichage *valid*		Non implémenté

Table 30 – *Fiche de test de CheckSolution, cas NumberError*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestSolutionChecker	TestCheckSolutionNumberError
Description	Lecture d'une solution avec la méthode utilisant le numéro du fichier et vérifie les contraintes	
Résultat attendu	Résultat obtenu	Statut
Affichage *invalid, error at*		Non implémenté

Table 31 – *Fiche de test de CheckSolution, cas Name*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestSolutionChecker	TestCheckSolutionName
Description	Lecture d'une solution avec la méthode utilisant le nom du fichier et vérifie les contraintes	
Résultat attendu	Résultat obtenu	Statut
Affichage *valid*		Non implémenté

Table 32 – *Fiche de test de CheckSolution, cas NameError*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestSolutionChecker	TestCheckSolutionNameError
Description	Lecture d'une solution avec la méthode utilisant le nom du fichier et vérifie les contraintes	
Résultat attendu	Résultat obtenu	Statut
Affichage *invalid, error at*		Non implémenté

Table 33 – *Fiche de test de BuildStatistics*

Projet	Séquence de test	Identification du test
PRD_MultiLine	TestStatisticsMaker	TestBuildStatistics
Description	Lecture de plusieurs solutions et calcul de statistiques	
Résultat attendu	Résultat obtenu	Statut
Création du fichier		Non implémenté

G Document d'utilisation

1 Document d'utilisation utilisateur

L'application s'utilise en ligne de commande, avec des flags à insérer ainsi que des paramètres afin d'accéder aux différentes fonctionnalités. On peut noter que même si les commandes sont présentées séparément, elles peuvent être écrites au sein d'une unique commande, en respectant les syntaxes respectives. Dans ce cas là, l'ordre pris en compte est la création d'instances, leur résolution, la vérification puis la création de statistiques.

1.1 Création d'une instance

Le flag de création d'instances est *-c*. La commande complète est `PRD_Application.exe [<nbStops> <nbLines> <fileName> [<nbInstances>]] -c`. Lorsque la commande est utilisée sans paramètres, l'application va générer 10 instances de chaque combinaison de nombres d'arrêts (200, 300 et 400 arrêts) et de nombre de lignes (3, 5 et 7 lignes). Lorsque le nombre d'instance à généré n'est pas défini, un seul fichier est généré. Dans le cas contraire, le nom de fichier est utilisé comme un préfixe de nom et est complété par le numéro de l'instance. Les fichiers sont générés dans un dossier *data* qui est créer par l'application et placé à sa racine. Les fichiers générés sont au format texte, l'extension n'a donc pas à être définie dans la ligne de commande.

```
C:\Users\Potter\Desktop\PRD>PRD_Application.exe 300 3 data_ 5 -c
File data/data_0.txt successfully created
File data/data_1.txt successfully created
File data/data_2.txt successfully created
File data/data_3.txt successfully created
File data/data_4.txt successfully created

Execution done
```

Figure 1 – *Exemple d'utilisation de l'application pour créer des instances*

1.2 Résolution d'une instance

Le flag de résolution d'instances est *-r*. La commande complète est `PRD_Application.exe [<nbStops> <nbLines> <dataFileName> <solutionFileName> [<nbInstances>]] -r`. Lorsque la commande est utilisée sans paramètres, l'application va résoudre les instances générées par défaut. Lorsque le nombre d'instance à résoudre n'est pas défini, une seule instance est résolue. Dans le cas contraire, les noms de fichier sont utilisés comme des préfixes de nom et sont complétés par le numéro de l'instance. Les fichiers de données sont cherchés dans un dossier *data* placé à la racine de l'application et le résultat des résolution sont placés dans un dossier *output* à la racine de l'application. Les fichiers de données et de solution sont automatiquement au format texte, l'extension n'a donc pas à être définie dans la ligne de commande.

```
C:\Users\Potter\Desktop\PRD>PRD_Application.exe data_ sol_ 5 -r
CPXPARAM_TimeLimit                               600
Warning: Control callbacks may disable some MIP features.
         Disabling repeat represolve because of lazy constraint/incumbent callback.
Tried aggregator 2 times.
MIP Presolve eliminated 271277 rows and 270942 columns.
MIP Presolve added 6 rows and 0 columns.
MIP Presolve modified 317 coefficients.
Aggregator did 67 substitutions.
Reduced MIP has 798 rows, 1091 columns, and 2514 nonzeros.
Reduced MIP has 563 binaries, 296 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.30 sec. (276.39 ticks)
Probing fixed 0 vars, tightened 1 bounds.
Probing time = 0.01 sec. (8.60 ticks)
Clique table members: 40.
MIP emphasis: balance optimality and feasibility.
MIP search method: traditional branch-and-cut.
Parallel mode: none, using 1 thread.
Root relaxation solution time = 0.02 sec. (5.23 ticks)

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

      0     0    62174.8950     8                62174.8950      131
      0     0    59028.3800    10                  Cuts: 19      215
```

Figure 2 – *Exemple d'utilisation de l'application pour résoudre des instances*

1.3 Vérification d'une solution

Le flag de vérification d'une solution est *-v*. La commande complète est `PRD_Application.exe [<nbStops> <nbLines> <dataFileName> <solutionFileName> [<nbInstances>]] -v`. Lorsque la commande est utilisée sans paramètres, l'application va vérifier les instances générées par défaut avec les résolutions par défaut. Lorsque le nombre d'instance à résoudre n'est pas défini, un seul couple est vérifié. Dans le cas contraire, les noms de fichier sont utilisés comme des préfixes de nom et sont complétés par le numéro de l'instance. Les fichiers de données sont cherchés dans un dossier *data* placé à la racine de l'application et le résultat des résolution sont cherchés dans un dossier *output* à la racine de l'application. Les fichiers de données et de solution sont automatiquement au format texte, l'extension n'a donc pas à être définie dans la ligne de commande. L'application va afficher pour chaque fichier de solution si il est valide ainsi que la contrainte violée dans le cas contraire.

1.4 Calcul de statistiques

Le flag de vérification d'une solution est *-s*. La commande complète est `PRD_Application.exe [<nbStops> <nbLines> <dataFileName> <solutionFileName> solutionFileName> [<nbInstances>]] -s`. Lorsque la commande est utilisée sans paramètres, l'application va faire des statistiques

```
C:\Users\Potter\Desktop\PRD>PRD_Application.exe data_ sol_ 5 -v
Solution file output/sol_0.txt valid
Solution file output/sol_1.txt valid
Solution file output/sol_2.txt valid
Solution file output/sol_3.txt valid
Solution file output/sol_4.txt valid

Execution done
```

Figure 3 – *Exemple d'utilisation de l'application pour vérifier des résolutions*

sur les fichiers générées par défaut. Dans le cas contraire, les noms de fichier sont utilisés comme des préfixes de nom et sont complétés par le numéro de l'instance. Les fichiers de données sont cherchés dans un dossier *data* placé à la racine de l'application et le résultat des résolution sont cherchés dans un dossier *output* à la racine de l'application. Le fichier de statistique produit est placé dans un dossier *stats* à la racine de l'application. Les fichiers de données, de solution et de statistiques sont automatiquement au format texte, l'extension n'a donc pas à être définie dans la ligne de commande.

```
C:\Users\Potter\Desktop\PRD>PRD_Application.exe data_ sol_ stats 5 -s
File stats/stats.txt successfully created

Execution done
```

Figure 4 – *Exemple d'utilisation de l'application pour faire des statistiques*

1.5 Format de fichier de données

Le format utilisé pour exporter et importer des données est fixe pour toute l'application (Figure 5).

data.txt | solution.txt | stats.txt

```
1  Nombre de lignes
2  Nombre de stops
3  Pour chaque ligne, taille de la ligne suivie des stops composant la ligne
4  Pour chaque ligne, fréquence de passage des bus
5  Entre chaque stop du réseau, distance
6  Entre chaque stop du réseau, temps de parcours
7  Pour chaque stop, coût d'installation d'un chargeur
8  Budget
9  Pour chaque ligne
10     Pour chaque stop, temps maximal de chargement
11 Vitesse de chargement
12 Capacité des batteries
13 Entre chaque stop du réseau, consommation électrique
14 Entre chaque stop du réseau, section obligatoirement électrique
15
```

Figure 5 – *Structure du fichier contenant les données*

1.6 Format du fichier de solution

Le format utilisé pour exporter et importer des solutions est fixe pour toute l'application (Figure 6).

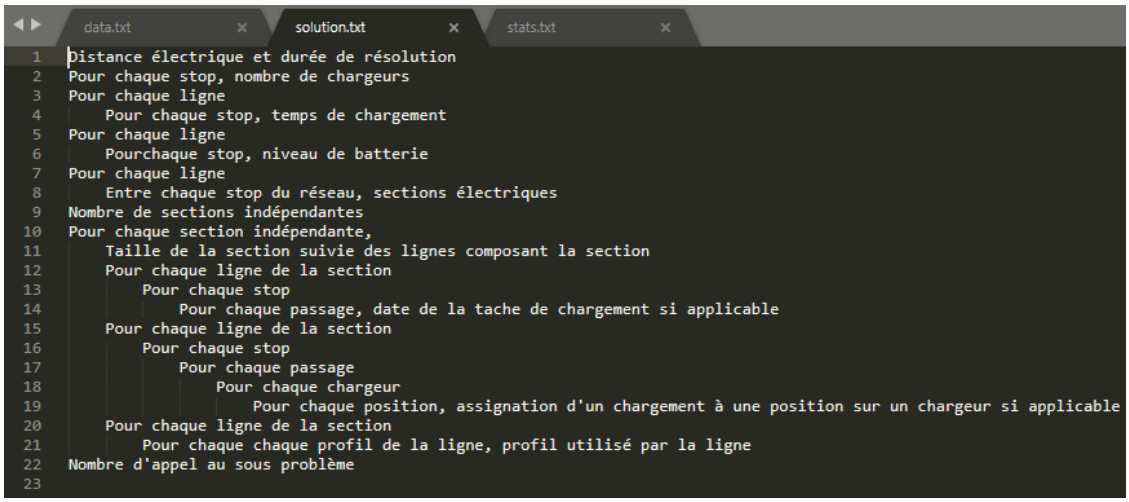


Figure 6 – *Structure du fichier contenant les solutions*

1.7 Format du fichier de statistiques

Le format utilisé pour exporter des statistiques est fixe pour toute l'application (Figure 7).

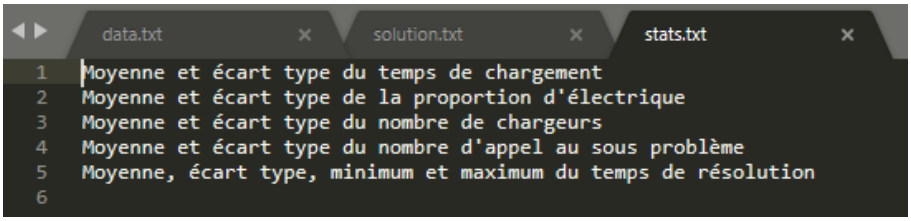


Figure 7 – *Structure du fichier contenant les statistiques*

2 Document d'utilisateur développeur

Le projet contient deux grands types de classe. Il y a les classes de base, liées directement au problème, et les classes d'interfaçage qui utilisent les classes de base. Les classes de base contiennent donc les données, `Data`, les classes liées à la résolution, `SolvableSolution` et `SlaveCallbackI` et les solutions utilisables, `ComputableSolution`. Les classes d'interfaçage, `InstanceMaker`, `Solver`, `SolutionChecker` et `StatisticsMaker`, utilisent principalement des méthodes des classes de base et implémentent les fonctionnalités prévues par l'application. Ces classes ne sont pas obligatoires mais permettent la séparation de l'implémentation des fonctionnalités. Il y a aussi une dernière classe, `Utils`, comportant des fonctions utilitaires telles que la création d'un dossier ou bien le calcul de l'écart type.

H Documents d'installation

1 Document d'installation utilisateur

Afin d'utiliser l'application, il faut tout d'abord télécharger l'application. Elle a besoin de fichiers *.dll* (*msvcp140.dll* et *vcruntime140.dll*) afin d'éxécuter du code C++. Ces fichiers sont nativement fournis avec la librairie *Microsoft Visual C++ Redistributable 2015*. Elle a aussi besoin de *cplex1280.dll*, correspondant à la version utilisée de *CPLEX* (version 12.8), afin de pouvoir exécuter la partie liée au solveur, fourni avec *CPLEX*. L'application est compilée pour fonctionner sous *Windows 10*.

2 Document d'installation développeur

Afin de modifier l'application, il faut tout d'abord posséder les sources dans son projet. Afin d'avoir un environnement fonctionnel, il faut posséder l'IDE *Visual Studio 2017*. Il s'agit de l'environnement utilisé lors du développement utilisé. Il faut aussi posséder une licence *IBM ILOG CPLEX Optimization studio*, version 12.8, avec la librairie d'interfaçage C++ associée, installée sur l'ordinateur. Le projet est fourni avec un fichier *PropertySheet.props.example*. Ce fichier permet à l'IDE de trouver les sources de la libraire *CPLEX* sur la machine. L'édition des liens se fait à l'aide de macros, chaque macro étant un lien vers des dossiers installés par *IBM ILOG CPLEX Optimization studio*. Le fichier de configuration à modifier permet donc d'indiquer vers quels dossier doivent pointer les macros. La première macro, *$(CplexDir)*, dirige vers le dossier *cplex* de la librairie et la seconde, *$(ConcertDir)*, vers le dossier *concert*. Pour finir, *CPLEX* a besoin de la librairie *Microsoft Visual C++ Redistributable 2015*, au minimum, afin de pouvoir fonctionner.

Webographie

[WWW1] *Accord entre Volvo Bus et Siemens sur les autobus électriques*. 2015. URL : http://www.avere-france.org/Site/Article/?article_id=6010 (visité le 22/08/2018).

[WWW2] *Bus2025 : comment la RATP va convertir 3 600 bus à l'électrique*. 2015. URL : http://www.avere-france.org/Site/Article/?article_id=6370 (visité le 22/08/2018)

[WWW3] *Bus2025 : l'ambitieux Plan de la RATP pour un parc 100% propre*. 2018. URL : https://www.ratp.fr/groupe-ratp/newsroom/bus/bus2025-lambitieux-plan-de-la-ratp-pour-un-parc-100-propre (visité le 26/09/2018).

[WWW4] *COP21 (Conférence sur le climat de Paris)*. 2016. URL : https://www.connaissancedesenergies.org/fiche-pedagogique/cop21-conference-sur-le-climat-de-paris (visité le 26/09/2018).

[WWW5] *e-VRO : electric vehicle routing optimization*. 2017. URL : http://www.e-vro.info/ (visité le 26/09/2018).

[WWW6] *Fuel Cell Electric Buses*. URL : https://www.fuelcellbuses.eu/ (visité le 09/07/2018)

[WWW7] IBM. *IBM ILOG CPLEX Optimization Studio CPLEX User's Manual*. 2016. URL : https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.0/ilog.odms.studio.help/pdf/usrcplex.pdf (visité le 18/07/2018).

[WWW8] *L'historique, petite histoire d'un grand réseau*. 2017. URL : https://www.filbleu.fr/reseau-fil-bleu/decouvrir-le-reseau/historique (visité le 22/08/2018).

[WWW9] Private Infrastructure Advisory Facility (PPIAF). *Outils et options pour réformer les systèmes de bus urbains*. 2006. URL : https://ppiaf.org/sites/ppiaf.org/files/documents/toolkits/french_UrbanBusToolkit/site/assets/3/3.1/35(vii)c.html (visité le 12/07/2048).

[WWW10] *Près de la moitié des bus dans le monde seront électriques d'ici à 2025*. 2018. URL : http://www.avere-france.org/Site/Article/?article_id=7227 (visité le 26/09/2018).

Bibliographie

[1] Intergovernmental Panel on Climate Change (IPCC). *Climate Change 2014 : Synthesis Report*. 2014.

[2] Alexander Kunith, Roman Mendelevitch et Dietmar Goehlich. « Electrification of a city bus network — An optimization model for cost-effective placing of charging infrastructure and battery sizing of fastcharging electric bus systems ». In : *International Journal of Sustainable Transportation* 11.10 (2017). DOI : 10.1080/15568318.2017.1310962, p. 707–720.

[3] Antti Lajunen. *Energy consumption and cost-benefit analysis of hybrid and electric city buses*. 2014.

[4] Ailsa Land et Alison Doig. « An Automatic Method of Solving Discrete Programming Problems ». In : *Econometrica* 28.3 (1960). DOI : 10.2307/1910129, p. 497–520.

[5] John Little, Katta Murty, Dura Sweeney et Caroline Karel. « An Algorithm for the Travelling Salesman Problem ». In : *Operations Research* 11.6 (1963), p. 863–1025.

[6] Zhaocai Liu, Ziqi Song et Yi He. « Planning of Fast-Charging Stations for a Battery Electric Bus System under Energy Consumption Uncertainty ». In : *Transportation Research Record* (2018). DOI : 10.1177/0361198118772953.

[7] Transit Cooperative Research Program (TCRP). *Synthesys 130 - Battery Electric Buses — State of the Practice*. 2018.

[8] Pierre Vendé. « Électrification d'une ligne de bus ». Rapport de stage. Tours, France : Ecole Polytechnique de l'Université François Rabelais de Tours, 2017-2018.

[9] Yusheng Wang, Yongxi Huang, Jiuping Xu et Nicole Barclay. « Optimal recharging scheduling for urban electric buses : A case study in Davis ». In : *Transportation Research Part E : Logistics and Transportation Review* 100 (2017). DOI : 10.1016/j.tre.2017.01.001, p. 115–132.

[10] Ran Wei, Xiaoyue Liu, Yi Ou et Kiavash Fayyaz. « Optimizing the spatio-temporal deployment of battery electric bus system ». In : *Journal of Transport Geography* 68 (2018). DOI : 10.1016/j.jtrangeo.2018.03.013, p. 160–198.

Projet Recherche & Développement

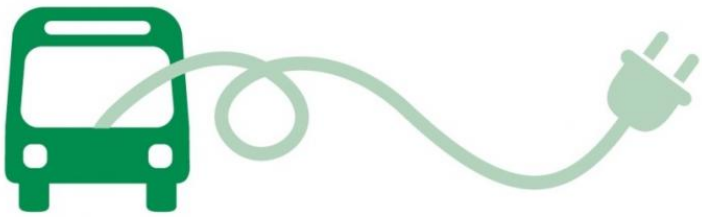
Étude et Résolution d'un problème de mise en place d'un réseau électrique pour bus

Pierre Vendé

Encadrement : Yannick Kergosien et Jorge E. Mendoza

Contexte

Selon la *COP21*, la pollution engendrée par les gaz à effet de serre vont augmenter la température de la planète de 4 à 5°C d'ici 2100. Une solution pour limiter cela est d'utiliser les énergies vertes dans les transports en commun. Ce problème cherche donc à résoudre l'électrification des réseau de bus à travers l'utilisation de bus hybrides.

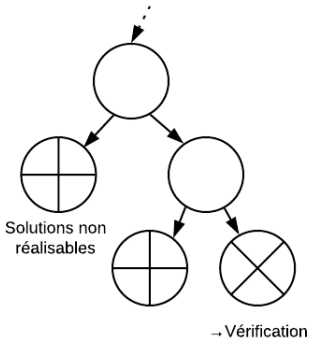


Solution

— Modéliser le problème à l'aide de la programmation linéaire en nombres entiers

— Implémenter la modélisation et obtenir les informations relatives au placement des chargeurs et aux durées de chargement

Logo du solveur utilisé



Procédure de séparation et vérification

Conclusion

Le modèle implémenté et résolu par la méthode de séparation et vérification fonctionne. Des améliorations sur la méthode sont possible afin de limiter le temps de calcul.

Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis, 37200 Tours, France
polytech.univ-tours.fr

POLYTECH TOURS
Informatique

Projet Recherche & Développement

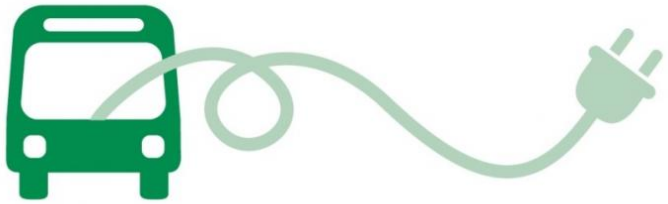
Étude et Résolution d'un problème de mise en place d'un réseau électrique pour bus

Pierre Vendé

Encadrement : Yannick Kergosien et Jorge E. Mendoza

Contexte

Selon la *COP21*, la pollution engendrée par les gaz à effet de serre vont augmenter la température de la planète de 4 à 5°C d'ici 2100. Une solution pour limiter cela est d'utiliser les énergies vertes dans les transports en commun. Ce problème cherche donc à résoudre l'électrification des réseau de bus à travers l'utilisation de bus hybrides.



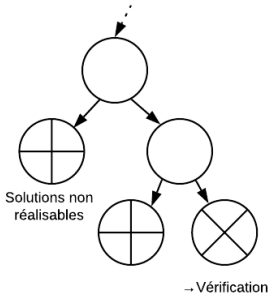
Solution

- Modéliser le problème à l'aide de la programmation linéaire en nombres entiers
- Implémenter la modélisation et obtenir les informations relatives au placement des chargeurs et aux durées de chargement

Logo du solveur utilisé

Conclusion

Le modèle implémenté et résolu par la méthode de séparation et vérification fonctionne. Des améliorations sur la méthode sont possible afin de limiter le temps de calcul.



Procédure de séparation et vérification

Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis, 37200 Tours, France
polytech.univ-tours.fr

POLYTECH TOURS
Informatique

Étude et Résolution d'un problème de mise en place d'un réseau électrique pour bus

Résumé

Ce projet présente une solution au problème de l'électrification d'un réseau de bus. Cette solution est trouvée grâce à la modélisation du problème puis résolue avec la méthode *Branch and Check* implémentée avec la librairie *ILOG CPLEX* de chez *IBM*.

Mots-clés

Recherche, Bus électrique, Bus hybride, Optimisation, Transport, CPLEX, Ordonnancement, Programmation linéaire, Séparation et vérification

Abstract

This project presents a solution for the problem of the electrification of a bus network. This solution is found thanks to the modelisation of the problem then solved avec the *Branch and Check* method implemented with the library *ILOG CPLEX* from *IBM*.

Keywords

Research, Electric bus, Hybrid bus, Optimization, CPLEX, Routing, Scheduling, Linear programming, Branch and check

Tuteurs académiques
Yannick Kergosien
Jorge E. Mendoza

Étudiant
Pierre Vendé (DI5)